# Auth node reference

July 4, 2025


AM

Version: latest;latest

# Table of Contents

## Utility nodes

# Basic nodes

## Data Store Decision node

The **Data Store Decision** node checks that the credentials provided during authentication match the ones stored in the configured data store for the realm.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the `realm`, `username`, and `password` properties in the incoming node state.

You can implement the following nodes as inputs to the Data Store Decision node:

#### *Input nodes*

- Zero Page Login Collector node
- PingAM (standalone)
  - Username Collector node
  - Password Collector node
- Advanced Identity Cloud (or Ping Identity Platform)
  - Platform Username node
  - Platform Password node

### Dependencies

The Data Store Decision node is a basic node used in many authentication application types, such as basic, push, OAuth 2.0, and social provider authentication applications.

### Configuration

This node has no configurable properties.

### Outputs

This node copies shared and transient state into the outgoing node state.

### Outcomes

Returns a boolean outcome:

**`True`**

The credentials match those found in the data store.

**`False`**

The credentials do *not* match those found in the data store.

### Errors

The following Data Store Decision node warnings and errors can appear in the logs:

#### *Warnings*

- "invalid password error"
- "invalid username error"

#### *Errors*

- "Exception in data store decision node"

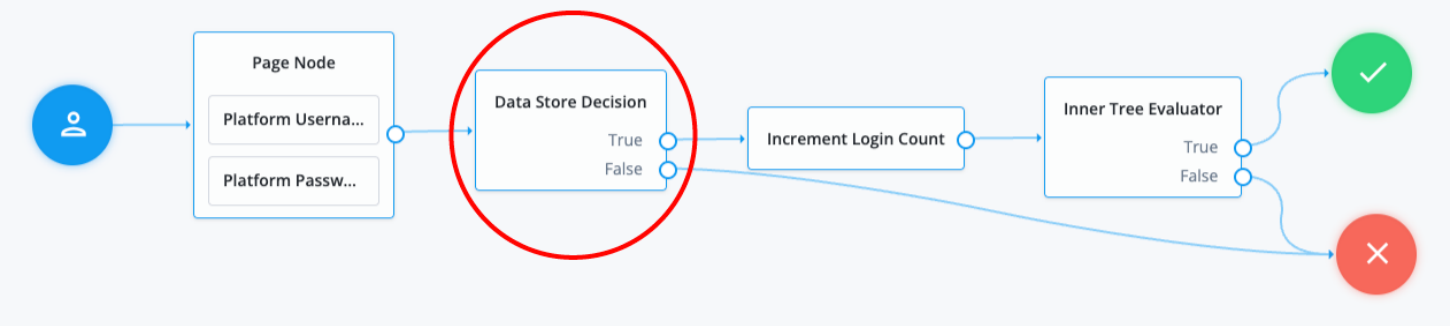
#### Troubleshooting

Review any errors and warnings this node logged.

- If this node logged a warning, fix the credentials and try again.
- If this node logged an error, review the log messages for the transaction to find the reason for the exception.

## Examples

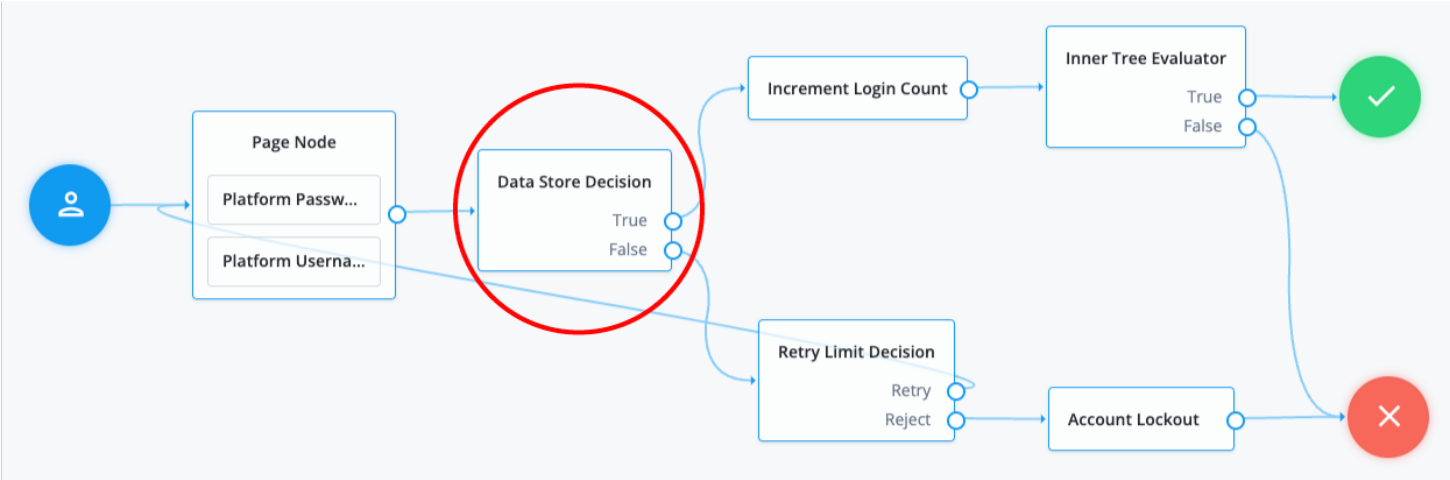
### Example 1: Simple username and password collector nodes with Data Store Decision node




This example illustrates a simple login process. The journey involves a Page node that contains two embedded nodes: Platform Username node and Platform Password node. To enhance user experience, the Page node lets users input their username and password on a single page, instead of splitting them across two different pages.

The Data Store Decision node has two outcomes: `True` or `False`. When the outcome is `True`, it triggers a Login Count Decision node. The Increment Login Count node then moves to an Inner Tree Evaluator node, which performs additional login processes. The `False` outcome connects directly to a failure node, indicating a failed state where the username and/or password provided by the user did not match the information stored in the data store.

### Example 2: Grant the user several attempts to enter their credentials correctly




In the following example, when an authentication attempt fails at the Data Store Decision node, you can direct it to a Retry Limit Decision node. The Retry Limit Decision node determines the number of retries allowed and either retries the login attempt or rejects it. If the journey rejects the login attempt after reaching the configured limit, for example three attempts, the operation results in an account lockout.

## Additional information

The following are alternate nodes that you can use in your journeys depending on your specific use cases:

- The LDAP Decision node supports LDAP Behera Password Policies with separate outcomes for accounts that are locked and passwords that have expired.

- (Advanced Identity Cloud only) The Identity Store Decision node is an enhanced node with additional outcomes. Use this node if your authentication journey needs more functionality than a simple `True` or `False` outcome.

## Kerberos node

Enables desktop single sign-on such that a user who has already authenticated with a Kerberos Key Distribution Center can authenticate to AM without having to provide the login information again.

To achieve this, the user presents a Kerberos token to AM through the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) protocol.

End users may need to set up Integrated Windows Authentication in Internet Explorer or Microsoft Edge to benefit from single sign-on when logged on to a Windows desktop.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `True`
- `False`

Evaluation continues along the `True` path if Windows Desktop SSO is successful; otherwise, evaluation continues along the `False` path.

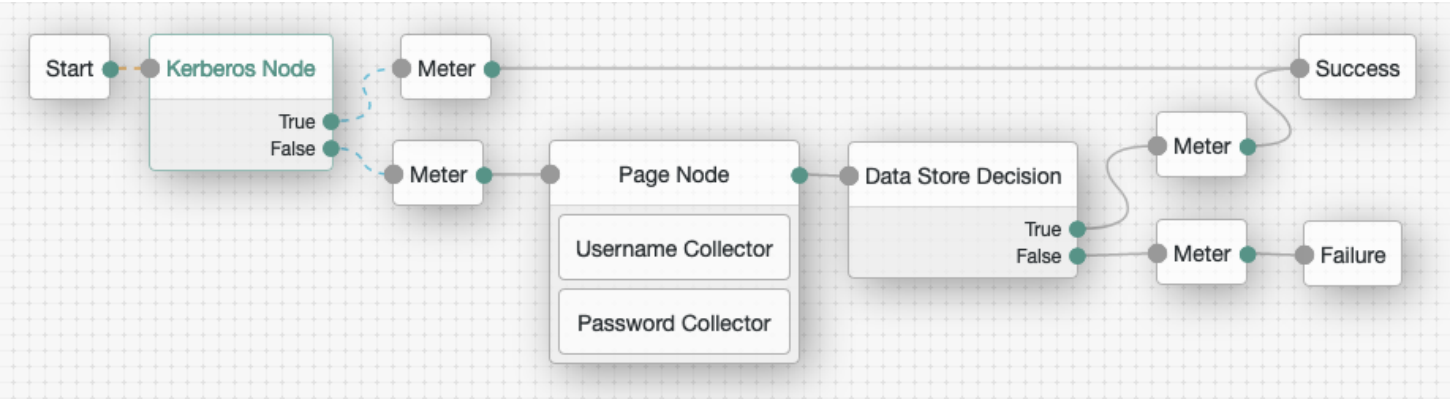
### Properties

| Property | Usage |
|---|---|
| Service Principal | Specifies the Kerberos principal for authentication in the format `HTTP/AM-DOMAIN@AD-DOMAIN`, where *AM-DOMAIN* corresponds to the host and domain names of the AM instance, and *AD-DOMAIN* is the domain name of the Kerberos realm (the FQDN of the Active Directory domain). *AD-DOMAIN* can differ from the domain name for AM.<br>In multi-instance AM deployments, configure *AM-DOMAIN* as the FQDN or IP address of the load balancer in front of the AM instances.<br>For example, `HTTP/AM-LB.example.com@KERBEROSREALM.INTERNAL.COM`. |

| Property | Usage |
| --- | --- |
| Key Tab File Path | Specifies the full, absolute path of the keytab file for the specified Service Principal.<br><br>**Tip**<br>You generate the keytab file using the Windows `ktpass` utility. For example:<br>`C:\> ktpass -out fileName.keytab -princ HTTP/openam.example.com@AD_DOMAIN.COM -pass +rdnPass -maxPass 256 -mapuser amKerberos@frdpcloud.com -crypto AES256-SHA1 -ptype KRB5_NT_PRINCIPAL -kvno 0` |
| Kerberos Realm | Specifies the name of the Kerberos (Active Directory) realm used for authentication.<br>Must be specified in ALL CAPS. |
| Kerberos Server Name | Specifies the fully qualified domain name, or IP address of the Kerberos (Active Directory) server. |
| Trusted Kerberos realms | Specifies a list of trusted Kerberos realms for user Kerberos tickets. If realms are configured, then Kerberos tickets are only accepted if the realm part of the user principal name of the user's Kerberos ticket matches a realm from the list.<br>Each trusted Kerberos realm must be specified in all caps. |
| Return Principal with Domain Name | When enabled, AM returns the fully qualified name of the authenticated user rather than just the username. |
| Lookup User In Realm | Validates the user against the configured data stores. If the user from the Kerberos token is not found, evaluation continues along the `False` path.<br>This search uses the `Alias Search Attribute Name` from the core realm attributes.<br>Find more information about this property in the corresponding documentation for:<br>• PingAM<br>• Advanced Identity Cloud |
| Is Initiator | When enabled ( `true` ), specifies that the node is using *initiator* credentials, which is the default.<br>When disabled ( `false` ), specifies that the node is using *acceptor* credentials. |

## Example

This flow attempts to authenticate the user with Windows Desktop SSO. If unsuccessful, AM requests the username and password for login. Meter nodes are used to track metrics for the various paths through the flow:




# LDAP Decision node

The **LDAP Decision** node verifies that the provided username and password exist in the specified LDAP user data store. The node also checks whether the associated user account has expired or is locked out.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

The node reads the `username` and `password` fields from the node state.

For standalone AM deployments, implement a Username Collector node and a Password Collector node earlier in the journey.

For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node earlier in the journey.

Alternatively, implement a Zero Page Login Collector node.

## Prerequisites

None

## Configuration

| Property | Usage |
| --- | --- |
| **Primary LDAP Server** *(required)* | Specify one or more primary directory servers. Specify each directory server in the following format: `host:port`.<br>For example, `directory_services.example.com:389`. |
| **Secondary LDAP Server** | Specify one or more secondary directory servers. Specify each directory server in the following format: `host:port`.<br>The journey uses the secondary servers when none of the primary servers are available.<br>For example, `directory_services_backup.example.com:389`. |
| **DN to Start User Search** *(required)* | Specify the DN from which to start the user search. More specific DNs, such as `ou=sales,dc=example,dc=com`, result in better search performance.<br>If multiple entries with the same attribute values exist in the directory server, make sure this property is specific enough to return only one entry. |
| **Bind User DN**, **Bind User Password** | The credentials used to connect to the LDAP user data store. |
| **Attribute Used to Retrieve User Profile** *(required)* | The attribute used to retrieve a user profile from the directory server.<br>The user search will have already happened, as specified by the **Attributes Used to Search for a User to be Authenticated** and **User Search Filter** properties. |
| **Attributes Used to Search for a User to be Authenticated** *(required)* | The attributes the node uses to match the credentials provided by the user to an entry in the directory server.<br>For example, a value of `uid` forms the search filter `uid=user`. If you specify multiple values, such as `uid` and `cn`, the node forms a complex search filter `(\|(uid=user)(cn=user))`.<br>Multiple attribute values let the user authenticate with any one of the values. For example, if you set both `uid` and `mail`, then Barbara Jensen can authenticate with either `bjensen` or `bjensen@example.com`.<br>**Note**<br>If you're using account lockout and you set multiple attribute values here, you must add those attributes to the `Alias Search Attribute Name` property in the User profile. Find more information in the corresponding documentation for:<br>• PingAM<br>• Advanced Identity Cloud |
| **User Search Filter** | A filter to append to user searches.<br>For example, if your search attribute is `mail` and you set **User Search Filter** to `(objectClass=inetOrgPerson)`, the node uses `(&(mail=address)(objectClass=inetOrgPerson))` as the resulting search filter. In this example, *address* is the mail address provided by the user. |

| Property | Usage |
| --- | --- |
| **Search Scope** | The extent of the search for users in the directory server:<br>• `OBJECT` : The search extends only to the entry specified by the **DN to Start User Search**.<br>• `ONELEVEL` : The search extends to the entries that are direct children of the **DN to Start User Search**.<br>• `SUBTREE` : The search extends to the **DN to Start User Search** and every entry under it.<br>Default: `SUBTREE` |
| **LDAP Connection Mode** | Specifies whether to use SSL or StartTLS to connect to the directory server. The node must be able to trust the certificates used.<br>Possible values: `LDAP` , `LDAPS` , and `StartTLS`<br>Default: `LDAP` |
| **mTLS Enabled** | Enables mTLS (mutual TLS) between AM and the directory server.<br>This setting applies to *all* configured LDAP servers; that is, AM uses mTLS to authenticate to all LDAP servers configured for this node.<br>When mTLS is enabled, AM ignores the values for **Bind User DN** and **Bind User Password**.<br>If you enable this property, you must:<br>• Set the **LDAP Connection Mode** to `LDAPS`<br>• Provide an **mTLS Secret Label Identifier**<br>Default: Disabled |
| **mTLS Secret Label Identifier** | Identifier used to create a secret label for mapping to the mTLS certificate in the secret store. AM uses this identifier to create a specific secret label for this node. The secret label takes the form `am.authentication.nodes.ldap.decision.mtls.identifier.cert` , where `identifier` is the value of **mTLS Secret Label Identifier**. The identifier can only contain alphanumeric characters ( `a-z` , `A-Z` , `0-9` ) and periods ( `.` ). It can't start or end with a period. All LDAP servers configured for this node share the same secret label.<br>For more security, you should rotate certificates periodically. When you rotate a certificate, update the corresponding mapping in the realm secret store configuration to reflect this label. When you rotate a certificate, AM closes any existing connections using the old certificate. A new connection is selected from the connection pool and no server restart is required. |
| **Return User DN to DataStore** | When enabled, the node returns the DN rather than the User ID. From the DN value, AM uses the RDN to search for the user profile. For example, if a returned DN value is `uid=demo,ou=people,dc=openam,dc=example,dc=org` , AM uses `uid=demo` to search the directory server.<br>Default: Enabled |

| Property | Usage |
| --- | --- |
| **User Creation Attributes** | This list lets you map (external) attribute names from the LDAP directory server to (internal) attribute names used by AM. |
| **Minimum Password Length** | The minimum acceptable password length.<br>Default: `8` |
| **LDAP Behera Password Policy Support** | When enabled, support interoperability with servers that implement the Internet-Draft, Password Policy for LDAP Directories.<br>Default: Enabled |
| **Trust All Server Certificates** | When enabled, the server blindly trusts server certificates, including self-signed test certificates.<br>Default: Disabled |
| **LDAP Connection Heartbeat Interval** | Specifies how often AM should send a heartbeat request to the directory server to ensure that the connection doesn't remain idle.<br>Some network administrators configure firewalls and load balancers to drop connections that are idle for too long. Set the units for the interval in the `LDAP Connection Heartbeat Time Unit` property.<br>**Note**<br>• Setting this property to `0` does *not* disable the heartbeat (keepalive) or load balancer availability checks.<br>• PingAM only: You can only disable these features at the global level.<br>Default: `10` |
| **LDAP Connection Heartbeat Time Unit** | The time unit for the `LDAP Connection Heartbeat Interval`.<br>Default: `seconds` |
| **LDAP Operations Timeout** | The timeout, in seconds, that AM should wait for a response from the directory server.<br>Default: `0` (means no timeout) |
| **Use mixed case for password change messages** | Specifies whether the server returns password change messages in mixed (sentence) case or transforms them to uppercase.<br>By default, the server transforms password reset and password change messages to uppercase. Enable this setting to return messages in sentence case.<br>Default: Disabled |

| Property | Usage |
| --- | --- |
| **LDAP Affinity Level** | Level of affinity used to balance requests across LDAP servers.<br>Affinity-based load balancing means that each request for the same user entry goes to the same DS server. The DS server used for a specific operation is determined by the DN of the identity involved.<br>List the directory server instances that form part of the affinity deployment in the **Primary LDAP Server** and **Secondary LDAP Server** properties.<br>Options are:<br>• `NONE` – no affinity<br>• `BIND` – affinity for BIND requests only<br>• `ALL` – affinity for all requests<br>Default: `NONE` |

### Outcomes

#### `True`

The provided credentials match those found in the LDAP user data store.

#### `False`

The provided credentials don't match those found in the LDAP user data store.

#### `Locked`

The profile associated with the provided credentials is locked.

#### `Cancelled`

The user must change their password. When the journey prompts the user to change their password, the user cancels the password change.

#### `Expired`

The profile is found, but the password has expired.

> **Important**
>
> PingAM only
> The LDAP Decision node *requires* specific user attributes in the LDAP user data store. These required attributes are present by default in PingDS. If you are using an alternative identity store, you might need to modify your LDAP schema to use this node.

## Identity Store Decision node

The **Identity Store Decision** node attempts to match the provided username and password with the credentials stored in the identity store.

If the credentials exist, the node checks the following:

- Is the profile locked?
- Has the provided password expired?
- Has the user cancelled a password reset?

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | No |
| Ping Identity Platform (self-managed) | No |

## Inputs

The node reads the `username` and `password` fields from the node state.

The journey can provide these credentials in a number of ways, for example, with a combination of the Platform Username node and Platform Password node, or by using the Zero Page Login Collector node.

## Dependencies

None

## Configuration

| Property | Usage |
|---|---|
| **Minimum Password Length** | For password change requests, the node rejects passwords that are shorter than this value. If you set this value to `0`, the node doesn't check the password length.<br>Default: `8` |
| **Username as Universal Identifier** | If you enable this property, the `username` property is set to the value of the `uuid`.<br>For example, `"username": "c636b756-ba6b-481d-ab4a-ab8c064cb24b"`.<br>If this property is false, the value of the `username` property remains unchanged.<br>For example, `"username": "bjensen"`.<br>Default: false |
| **Use mixed case for password change messages** | Return password change messages in mixed (sentence) case.<br>By default password reset and password change messages are transformed to upper case. Enable this option to return messages in sentence case.<br>Default: Disabled |

## Outputs

This node copies shared and transient state into the outgoing node state.

## Outcomes

### `True`

The credentials match those found in the identity store.

### `False`

The credentials don't match those found in the identity store.

### `Locked`

The profile associated with the provided credentials is locked.

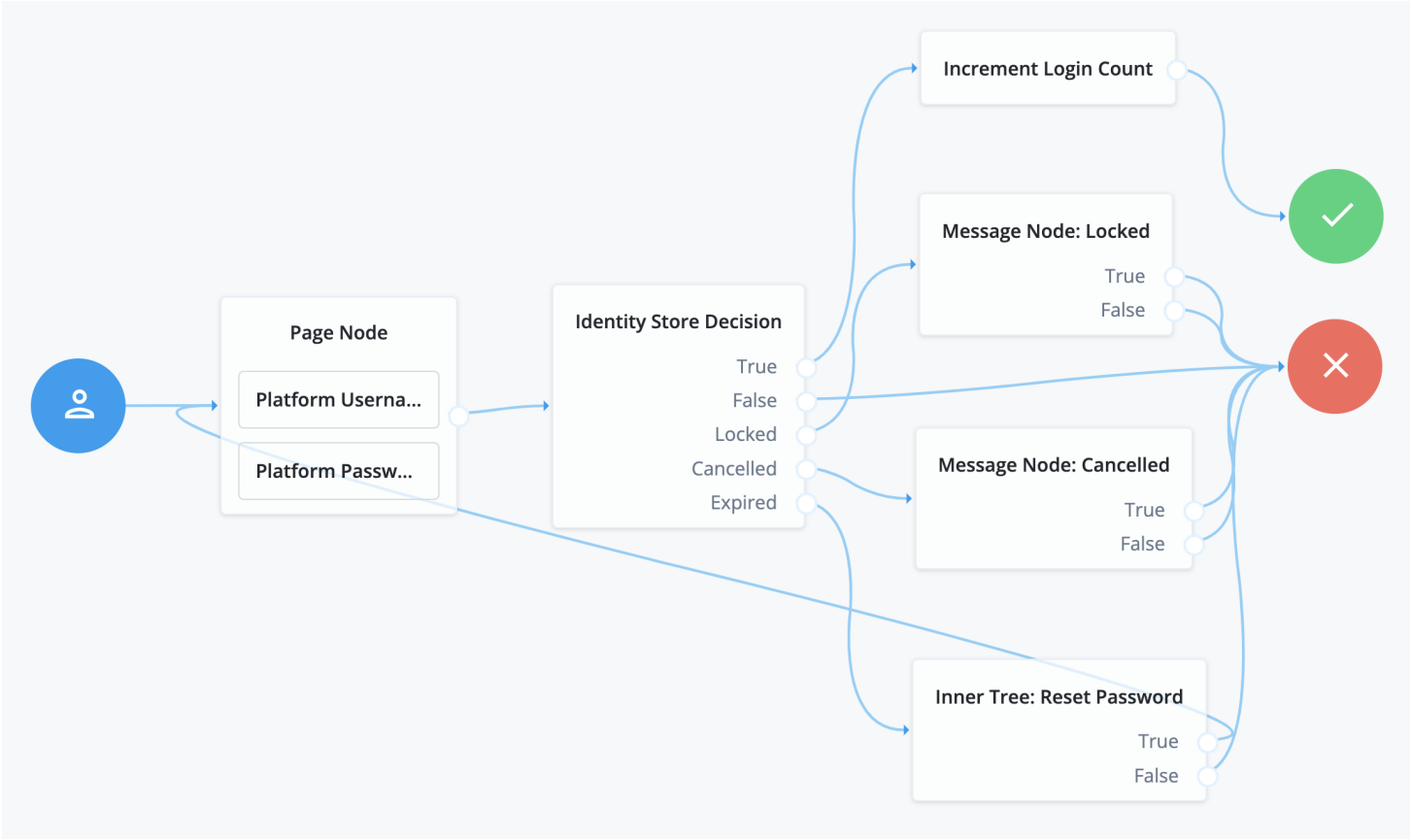
### `Cancelled`

The user cancelled a password change request. The example provides a detailed explanation of this outcome.

### `Expired`

The credentials match those found in the identity store, but the password has expired.

## Example




This example illustrates a simple login process.

- A Page node with the embedded nodes (Platform Username node and Platform Password node) prompts the user for their credentials.
- The Identity Store Decision node assesses the credentials:
  - If it finds the credentials in the data store and the credentials are valid, the journey follows the `True` outcome. An Increment Login Count node increments the login count and the user is authenticated.
  - If the credentials don't exist in the data store, the journey follows the `False` outcome and authentication fails.
  - If the credentials exist in the data store but the account is locked, the journey follows the `Locked` outcome. A Message node displays a custom lockout message and authentication fails.
  - If the credentials exist in the data store but the user must change their password, the node prompts the user to change their password. If the user cancels this change request, the journey follows the `Cancelled` outcome. A Message node displays a custom message and authentication fails.
  - If the credentials exist in the data store but the password has expired, the node follows the `Expired` outcome. The user is routed to an inner tree journey that contains the password reset logic and then routes the user to the start of the journey to authenticate again.

### Alternative nodes

- The Data Store Decision node is a simpler node with only two outcomes, `True` and `False`. Use this node if the flow only requires these outcomes.

## Password Collector node

Prompts the user to enter their password.

The captured password is transient, persisting only until the authentication flow reaches the next node requiring user interaction.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

Single outcome path.

Evaluation continues after capturing the password.

### Properties

This node has no configurable properties.

## Username Collector node

Prompts the user to enter their username.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

Single outcome path.

Evaluation continues after capturing the username.

### Properties

This node has no configurable properties.

## Zero Page Login Collector node

The **Zero Page Login Collector** node verifies the presence of specific HTTP username and password headers in the incoming authentication request. If the headers exist, the node uses their corresponding values as the provided username and password.

The **Zero Page Login Collector** node is commonly used to:

- Connect the `Has Credentials` outcome connector to the input of a Data Store Decision node.
- Connect the `No Credentials` outcome connector to the input of:
  - PingAM deployment: a Username Collector node followed by a Password Collector node
  - Advanced Identity Cloud or Ping Identity Platform deployment: a Platform Username node followed by a Platform Password node
- Then connect into the same Data Store Decision node.

The password collected by this node remains in the node state only until the journey reaches the next node that requires user interaction.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

- HTTP username header
- HTTP password header
- An allowlist of referrers if `Allow Without Referer` property is disabled. When you set the `Allow Without Referer` property to `false`, the request must contain a referrer from the allowlist; otherwise, the journey ends in a failure.

## Dependencies

None.

## Configuration

### Properties

| Property | Usage |
| --- | --- |
| **Username Header name** | Enter the name of the header that contains the username value.<br>Default: **X-OpenAM-Username** |
| **Password Header name** | Enter the name of the header that contains the password value.<br>Default: **X-OpenAM-Password** |
| Allow without referer | If enabled, the node accepts incoming requests that do not contain a `Referer` HTTP header. If a `Referer` HTTP header is present, the value is not checked.<br>If disabled, a `Referer` HTTP header must be present in the incoming request, and the value must appear in the Referer allowlist property.<br>Default: `Enabled` |
| Referer Whitelist | Specify a list of URLs allowed in the `Referer` HTTP header of incoming requests. An incoming request containing a `Referer` HTTP header value not specified in the allowlist causes evaluation to continue along the `No Credentials` outcome path.<br>**Note**<br>You must disable the **Allow Without Referer** property for the referer allowlist property to take effect. |

## Outputs

The collected credentials from the headers.

## Outcomes

- `Has Credentials`
- `No Credentials`

Evaluation continues along the `Has Credentials` outcome path if the specified headers are available in the request, or the `No Credentials` path if the specified headers are not present.

## Errors

### *If more than one header value exists for username and/or password, the node returns the following error message*

"Expecting only one header value for username and/or password but size is {}."

### *If the node can't decode the header values, the node returns the following error message*

"Could not decode username or password header."

## Example

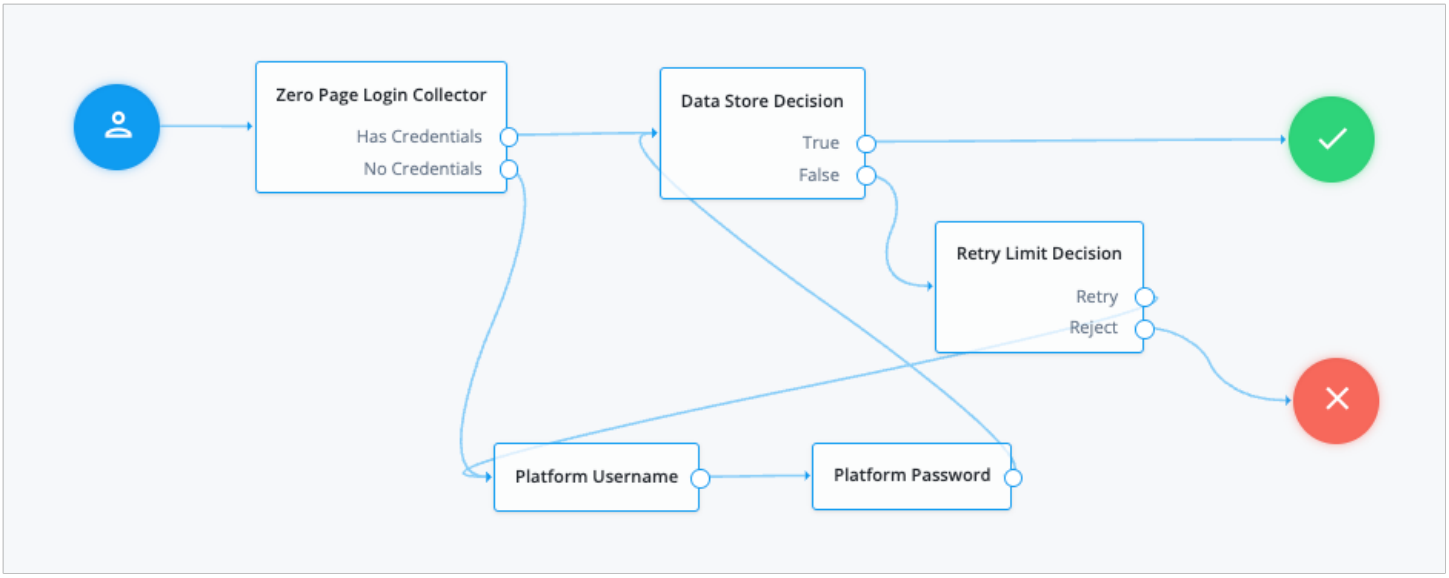



*Figure 1. Advanced Identity Cloud*

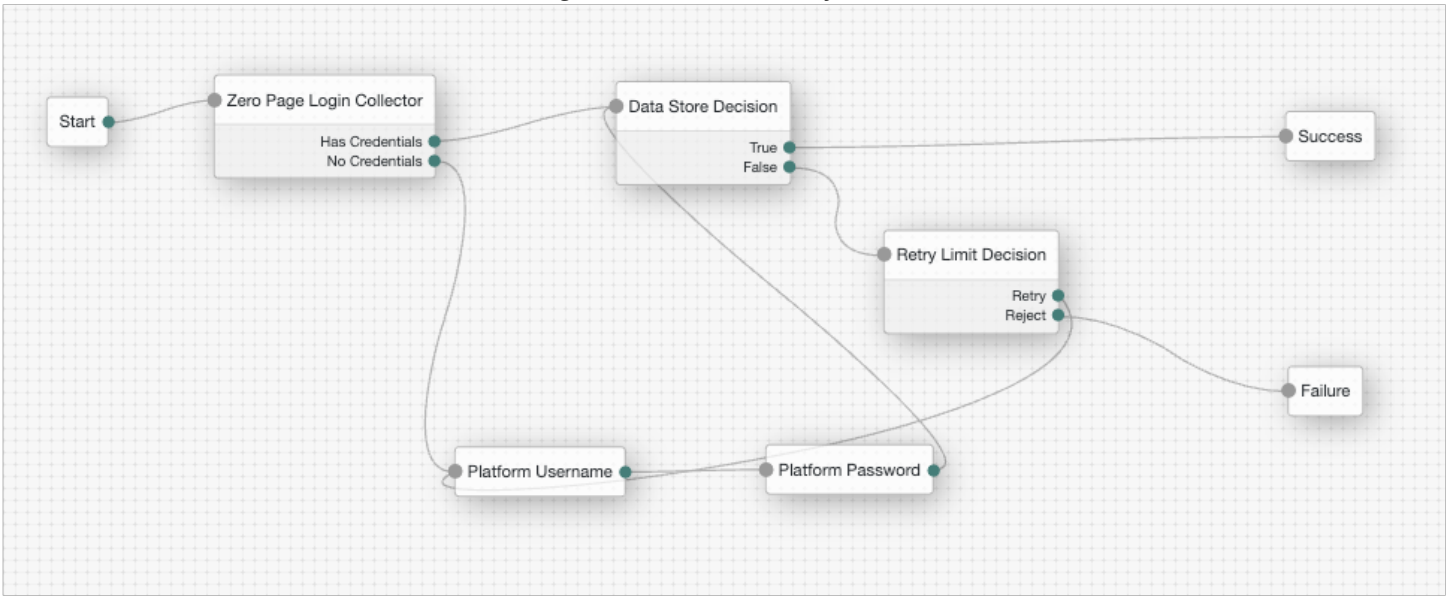



*Figure 2. PingAM*

# Failure node

The **Failure** node is a required element indicating the journey ended in failure.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

The failure outcomes of any preceding nodes.

## Dependencies

None.

## Configuration

This node has no configurable properties.

## Outputs

None. The authentication journey ends in failure.

## Outcomes

The authentication journey completes, ending in failure, and the user is redirected to a failure URL.

Find more information in the corresponding documentation for:

- PingAM
- Advanced Identity Cloud

## Errors

The error depends on the **Authentication > Settings > Account Lockout > Login Failure Lockout Mode** setting for the realm (under **Native Consoles > Access Management**).

Without the setting enabled, by default, the node returns an error with a message such as the following:

```
{"code":401,"reason":"Unauthorized","message":"Login failure"}
```

With the setting enabled, the node checks the invalid attempts property of the user profile and does the following:

- Returns a warning message if the number of failed attempts is equal to or greater than the **Authentication > Settings > Account Lockout > Warn User After N Failures** setting:

```
{
  "code": 401,
  "reason": "Unauthorized",
  "message": "Warning: You will be locked out after 1 more failure(s).",
  "detail": {
    "failureUrl": ""
  }
}
```

- Increments the failure count in the user profile.
- Returns an error message if the account is `Inactive`:

```
{
  "code": 401,
  "reason": "Unauthorized",
  "message": "User Locked Out.",
  "detail": {
    "failureUrl": ""
  }
}
```

To troubleshoot an authentication failure, review the steps in the journey to find what caused the failure.

### Examples

All authentication journeys have a **Failure** node as one of their terminals.

## Success node

The **Success** node is a required element indicating the journey ended successfully.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

The success outcomes of any preceding nodes.

### Dependencies

None.

### Configuration

This node has no configurable properties.

### Outputs

None.

### Outcomes

The authentication journey completes successfully.

The node resets the failure count in the user profile when reached if the **User Status** property is set to `Active`.

### Errors

- Checks the **Status** property of the user profile, when reached, and fails the authentication with an error message if the account is marked as `Inactive`:

```
{
    "code":401,
    "reason":"Unauthorized",
    "message":"User Locked Out.",
    "detail":
    {
        "failureUrl":""
    }
}
```

### Examples

All authentication journeys have a **Success** node as one of their terminals.

# Multi-factor nodes

## Combined MFA Registration node

The **Combined MFA Registration node** lets an authenticated user register a device, such as a mobile phone, for multi-factor authentication with a push notification *and* an OATH one-time password in a single step.

This node can make journeys less complex by combining the functionality of the Push Registration node and OATH Registration node.

The node displays a single QR code that users scan to register their device for both push and OATH authentication. Journeys can then use the Push Sender node to verify possession of a registered device. If push does not succeed, for example, the user's device does not have internet access, the journey can fall back to using the OATH Token Verifier node to request a one-time passcode using OATH.

Learn more about push notifications and OATH one-time passwords in the MFA documentation for:

- PingAM
- Advanced Identity Cloud

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires a `username` in the incoming node state to identify which user is registering for MFA.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

### Dependencies

You must configure the Push Notification service for the realm to use this node. Optionally, also configure the ForgeRock Authenticator (Push) service.

Find more information in the corresponding documentation for:

- Advanced Identity Cloud
- PingAM

Find information on provisioning the credentials used by the service in How To Configure Service Credentials (Push Auth, Docker) in Backstage.

## Configuration

| Property | Usage |
|---|---|
| **Issuer** | An identifier to appear on the user's device, such as a company name, a website, or a realm.<br>The value is displayed by the authenticator application.<br>For example, `Example Inc.` or the name of your application.<br>Default: `ForgeRock` |
| **Account Name** | The profile attribute to display as the username in the authenticator application.<br>If not specified, or if the specified profile attribute is empty, the username is used.<br>Default: `Username` |
| **Background Color** | The background color in hex notation that displays behind the issuer's logo within the authenticator application.<br>Default: `032b75` |
| **Logo Image URL** | The location of an image to download and display as the issuer's logo within the authenticator application.<br><br>**Note**<br>The ForgeRock Authenticator supports logos in JPEG and PNG format only. The application resizes your logo automatically, but a maximum image size of one MByte (or 1024 X 1024 pixels) is recommended.<br><br>Default: none |
| **Generate Recovery Codes** | If enabled, recovery codes are generated and stored in the successful outcome's transient state.<br>Use the Recovery Code Display node to display the codes to the user for safekeeping.<br>Default: true<br><br>**Important**<br>Generating recovery codes overwrites all existing push-specific recovery codes.<br>Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen. |

| Property | Usage |
| --- | --- |
| **QR code message** | A custom, localized message with instructions to scan the QR code to register the device.<br><br>1. Click **Add**.<br>2. Enter the message locale in the **Key** field; for example, `en-gb`.<br>3. Enter the message to display to the user in the **Value** field.<br><br>Default: none |
| **Registration Response Timeout** | The period of time (in seconds) to wait for a response to the registration QR code. If no response is received during this time, evaluation continues along the `Time Out` outcome path.<br>Default: `60` |
| **One Time Password Length** | The length of the generated OTP in digits.<br>This value must be at least `6` and compatible with the hardware/software OTP generators you expect end users to use. For example, Google and ForgeRock authenticators support values of `6` and `8`, respectively.<br>Default: `6` |
| **Minimum Secret Key Length** | Minimum number of hexadecimal characters allowed for the Secret Key.<br>Default: `32` |
| **OATH Algorithm** | The algorithm the device uses to generate the OTP:<br><br>***HOTP***<br>HOTP uses a counter; the counter increments every time a new OTP is generated. When you use this setting, also set the same value in the OATH Token Verifier node.<br>***TOTP***<br>TOTP generates a new OTP every few seconds as specified by the `TOTP Time Step Interval` setting.<br><br>Default: `TOTP` |
| **TOTP Time Step Interval**<br>( `totpTimeInterval` ) | The length of time that an OTP is valid in seconds.<br>For example, if the time step interval is 30 seconds, a new OTP is generated every 30 seconds and is valid for 30 seconds only.<br>Default: `30` seconds |
| **TOTP Hash Algorithm** | The HMAC hash algorithm used to generate the OTP codes. Advanced Identity Cloud and PingAM support SHA1, SHA256, and SHA512.<br>Default: `SHA1` |
| **HOTP Checksum Digit** | Add a digit to the end of the generated OTP to be used as a checksum to verify the OTP was generated correctly. This is in addition to the actual password length. Only set this if the user devices support it.<br>Default: false |

| Property | Usage |
| --- | --- |
| **HOTP Truncation Offset** | An option used by the HOTP algorithm that not all devices support. Leave the default value unless you know user devices use an offset.<br>Default: `-1` |
| **JSON Authenticator Policies** | Policies to apply to the device being registered, in JSON format. Use the following format to apply policies:<br>`{ "policyName" : { "policyParameters" \| "value" } }`<br>**Supported policies**<br>The ForgeRock Authenticator app supports enforcement of the following default policies:<br>**`biometricAvailable`**<br>Parameters: None<br>The device must have a biometric sensor available and enabled in the operating system.<br>**`deviceTampering`**<br>Parameters: `score`<br>The device must not have been tampered with; for example have root access or be jailbroken.<br>This policy applies if the score returned by the device exceeds the provided `score` parameter, which is a number between `0` and `1.0`.<br>**Example**:<br>`{ "biometricAvailable": { }, "deviceTampering": { "score": 0.8 } }` |

## Outputs

- For Push registration, this node updates the shared state with the push device settings, the message ID, and the push challenge.
- For OATH registration, this node records the device profile in the `oathDeviceProfile` shared state attribute and the recovery codes in the `oathEnableRecoveryCode` shared state attribute.

## Outcomes

### *Success*

Device registration succeeded.

### *Failure*

AM encountered an issue when attempting to register the authentication device.

### *Time Out*

The node didn't receive a response from the device within the time specified in the configuration.

## Errors

### *No username found*

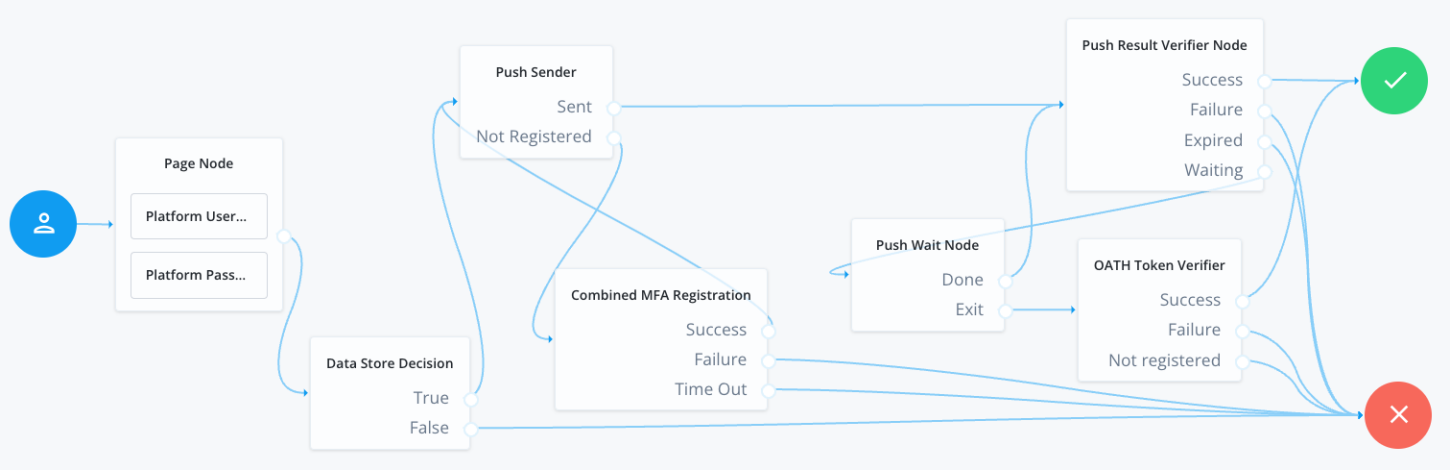
The node failed to read the username from the shared state.

### *Unable to find push message ID in sharedState*

The node failed to read the push message ID from the shared state.

## Examples

The following example shows an implementation of combined multi-factor registration in an authentication journey:




- The Page node with the Platform Username node and the Platform Password node prompts for the user credentials.
- The Data Store Decision node confirms the username-password credentials.
- The Push Sender node determines whether the user has a registered device.
    - If the user has a registered device:
        - The Push Sender node sends a push notification to the device.

- The Push Result Verifier node validate the user's response to the push notification, looping through the Push Wait node until authentication succeeds.
- The Push Wait node lets the user cancel the wait for a push notification. In this case evaluation continues to the OATH Token Verifier node, so the user can enter a one-time password instead.

- If the user **doesn't** have a registered device:
  - The Push Sender node routes the user to the Combined MFA Registration node, which displays a QR code to the user to register a device.
  - After successful registration of a device for both push and OATH authentication, evaluation returns to the Push Sender node and continues with the registered device.

## Device Binding node

Allows users to register one or more devices to their account. A user can bind multiple devices, and each device can be bound to multiple users.

There are many similarities between WebAuthn and device binding and JWS verification. We provide authentication nodes to implement both technologies in your journeys.

Both can be used for usernameless and passwordless authentication, they both use public key cryptography, and both can be used as part of a multi-factor authentication journey.

One major difference is that with device binding, the private key never leaves the device.

With WebAuthn, there is a possibility that the private key is synchronized across client devices because of Passkey support, which may be undesirable for your organization.

For details of the differences, refer to the following table:

*Comparison of WebAuthn and Device Binding/JWS Verification*

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Industry-standards based | ☑ | × | You can refer to the WebAuthn W3C specification. Device binding and JWS verification are proprietary implementations. |
| Public key cryptography | ☑ | ☑ | Both methods use Public key cryptography. |
| Usernameless support | ☑ | ☑ | After registration, the username can be stored in the device and obtained during authentication without the user having to enter their credentials. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Keys are bound to the device | ✕ | ☑ | With WebAuthn, if Passkeys are used, they can be shared across devices. With device binding, the private keys do not leave the device. |
| Sign custom data | ✕ | ☑ | With device binding, you can:<br>• Customize the challenge that the device must sign. For example, you could include details of a transaction, such as the amount in dollars.<br>• Add custom claims to the payload when signing a challenge. This gives additional context that the server can make use of by using a scripted node. Refer to Add custom claims when signing. |
| Format of signed data | WebAuthn authenticator data | JSON Web Signature (JWS) | |
| Integration | ✕ | ☑ | With device binding, after verification, the signed JWT is available in:<br>• Audit Logs<br>• Transient node state<br>This enables the data within to be used for integration into your processes and business logic. |
| Platform support | ☑ Android<br>☑ iOS<br>☑ Web browsers | ☑ Android<br>☑ iOS<br>✕ Web browsers | As it is challenging to store secure data in a browser as a client app, device binding is not supported in web browsers. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Authenticator support | Determined by the platform.<br>Configuration limited to:<br>• Biometric with Fallback to Device Pin | Determined by the authentication node.<br>Full configuration options:<br>• Biometric Authentication<br>• Biometric with Fallback to Device Pin<br>• Application Pin<br>• Silent | With device binding, you can specify what authentication action the user must perform to get access to the private keys.<br>This provides greater flexibility in your security implementation and can reduce authentication friction for your users. |
| Key storage | Web browsers and iOS synchronize to the cloud.<br>Android has the option to synchronize to the cloud. | ***Android***<br>KeyStore<br>***iOS***<br>Secure enclave: hardware-backed and not synchronized to the cloud. | Both technologies store the private keys securely on the client.<br>WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>This can reduce authentication friction for your users but may also increase the risk of a breach. |
| Managing device keys | Managed by the device OS.<br>Apps cannot delete *local* client keys programmatically and do not have a reference to the *remote* server key for deletion. | Managed by the Ping SDKs.<br>Provides an interface to delete local client and remote server keys. | The ability to programmatically delete both client and server keys can greatly simplify the process of registering a new device if an old device is lost or stolen. |
| Passkey support | ☑ | × | WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>Device binding keeps the private key locked in the device. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| App integrity verification | ***Android***<br>Requires an `assetlinks.json` file.<br>***iOS***<br>Requires `apple-app-site-association` file. | Not provided by the device binding or verification nodes.<br>It can be added as part of the journey by using app integrity nodes. | App integrity verification helps ensure your users are only using a supported app rather than a third-party or potentially malicious version. |
| Key attestation | ***Android***<br>SafetyNet<br>***iOS***<br>None | ***Android***<br>Uses hardware-backed key pairs with Key Attestation.<br>***iOS***<br>It can be added as part of the journey by using app integrity nodes to support key attestation. | Key attestation verifies that the private key is valid and correct, is not forged, and was not created in an insecure manner. |
| Complexity | Medium | Low | WebAuthn requires a bit more configuration, for example, creating and uploading the `assetlinks.json` and `apple-app-site-association` files. Device binding only requires the journey and the SDK built into your app. |

You must ensure you authenticate the user and obtain their username in the journey before attempting to bind a device.

Registered devices share device data in the form of a public key and a key ID which AM stores in the user's profile, or you can save it in transient state for processing.

The private key of the keypair is kept safely on the device and secured with biometric security or a PIN.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |

| Product | Available? |
| --- | --- |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Dependencies

You can verify possession of bound devices by using the Device Signing Verifier node.

You can save the device data to the user's profile by using the Device Binding Storage node.

To use Android Key Attestation, you must also configure the Android Key Attestation Service. Learn more about that service for:

- Advanced Identity Cloud
- PingAM

## Configuration

| Property | Usage |
| --- | --- |
| Authentication Type | How the device should secure access to the private key.<br>The available options are:<br><br>**`Biometric only`**<br>Request that the client secures access to the cryptography keys with biometric security, such as a fingerprint.<br>**`Biometric with PIN fallback`**<br>Request that the client secures access to the cryptography keys with biometric security, such as a fingerprint, but allow use of a PIN if biometric is unavailable.<br>**`Application PIN`**<br>Request that the client secures access to the cryptography keys with an application-specific PIN.<br><br>**Important**<br>The PIN is not linked to the user's device PIN and is stored only on the client device.<br>The PIN is not sent to AM, so if the user forgets their PIN, they must bind the device again.<br><br>**None**<br>Allow the client device to create the cryptography keys without securing access to them. |

| Property | Usage |
| --- | --- |
| Application IDs | Specifies a list of Android package names and iOS bundle IDs of applications that are allowed to perform device binding.<br>For example, `com.example.app`. |
| Title | Specifies a title to display to the user when asking them to bind the device. |
| Sub Title | Specifies a subtitle to display to the user when asking them to bind the device. |
| Description | Specifies descriptive text to display to the user when asking them to bind the device. |
| Maximum Saved Devices | Specifies the maximum number of devices stored in the user's profile.<br>Set this property to `0` if you do not want to limit the number of devices.<br>When this property is greater than zero, the `Exceed Device Limit` outcome path becomes available. |
| Timeout | Specify the number of seconds to wait for a response from the client during binding.<br>If the specified time is reached, evaluation continues along the `Timeout` outcome path. |
| Android Key Attestation | Use Android key attestation to increase confidence that the keys used by the bound device are valid, haven't been revoked, and use hardware-backed security storage.<br>The attestation data is also stored in the transient state of the tree, in a variable named `DeviceBindingCallback.ATTESTATION`, so that you can access and parse the data in a scripted node if required.<br>You can find information on the contents of the attestation data JSON response in Attestation certificate in the Android documentation. |

| Property | Usage |
| --- | --- |
| Store Device Data in Transient State | If enabled, the node *does not save* device data in the user's profile when it completes successfully.<br>Instead, the node places the device information into the transient state in a variable named `DeviceBindingNode.DEVICE`. This allows subsequent nodes to use, parse, or alter the information before saving it.<br>Use the Device Binding Storage node to save the device data to the user's profile.<br>**Example device data**<br>{<br>"uuid": "0ea44aa7-ef55-431b-885b-8c3a87e93331",<br>"recoveryCodes": [],<br>"deviceName": "Pixel 7 Pro",<br>"deviceId":<br>"aaddfecd9b8b3e2a-153cae31c23bc51a8db6d71bc3a31423a6aca97d",<br>"createdDate": 1694787036658,<br>"lastAccessDate": 1694787036658,<br>"key": {<br>"kty": "RSA",<br>"kid": "0ea44aa7-ef55-431b-885b-8c3a87e93331",<br>"use": "sig",<br>"alg": "RS512",<br>"n": "n7nn76rmgcOGfuVm8N-wur4GgWW-<br>Iek0edwcQR865L3sjKON3XUCHi210tqMyc-PWlCaY-<br>dHisyy7TxK0jn4poui_aK3lnGYNzJpuyTU1-<br>sunSTRVMW8vDTEJxUNQMZFS086_8hVFiC9OnElkpFllp2jzfgZ7u318bdVMgib2bHlscyMo8<br>CZEwA_MHKteIkSD7CZIHMjm-<br>JlJIrKlaLIJ3lkZTUG29g2J9LvdGTMXyt206ZLQw3kAQ_QczHpiKieAiLd9sHydjB7BqGpgC<br>xjCkmqVi4BEvMl8sEEFnpZG1NzjrCBnGfSWr83dzenr6tbdCh5iew-BIdDXXaDPOXRew",<br>"e": "AQAB"<br>}<br>} |

### Android key attestation

When binding a device running Android N ( `24` ) or newer, you can use Android key attestation to increase confidence that the keys used by the bound device are valid, have not been revoked, and use hardware-backed security storage.

The Ping SDK for Android generates attestation data for the cryptographic keys it uses for device binding. Using information provided by Google, including a certificate revocation status list (CRL) and hardware attestation root certificate, the node can verify that the certificates are trustworthy.

If you enable the **Android key attestation** property and the device is running an earlier Android version, evaluation continues down the `Unsupported` outcome path.

Android key attestation *is not* supported if you select `Application PIN` in the **Authentication Type** property. Evaluation continues down the `Unsupported` outcome path in this case.

The node does not attempt attestation when binding non-Android devices.

### Outputs

If you enable the **Android Key Attestation** property, the node outputs attestation data in a variable named `DeviceBindingCallback.ATTESTATION`.

If you enable the **Store Device Data in Transient State** property, the node outputs device data in a variable named `DeviceBindingNode.DEVICE`.

### Outcomes

- `Success`
- `Failure`
- `Exceed Device Limit`
- `Unsupported` (Client)
- `Abort` (Client)
- `Timeout` (Client)

If the user successfully binds their device, evaluation continues along the `Success` outcome path.

If AM encounters an issue when attempting to register using a device, evaluation continues along the `Failure` outcome path.

If the **Maximum Saved Devices** property is set to an integer greater than zero, and binding a new device would take the number of devices above the specified threshold, then evaluation continues down the `Exceed Device Limit` outcome path. In this case, you need to instruct your users to log in with an existing bound device in order to remove one or more of their registered devices.

If the user's client does not support the requested operation, evaluation continues along the `Unsupported` outcome path. For example, the node is configured to require biometric authentication, but the device does not provide support.

If the user cancels the attempt to bind a device, evaluation continues along the `Abort` outcome path.

If the node does not receive a response from the user's device within the **Timeout** specified in the node configuration, evaluation continues along the `Timeout` outcome path.

## Device Binding Storage node

Persists collected device binding data to a user's profile in the identity store.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

The node uses the variable named `DeviceBindingNode.DEVICE` as input from the state.

The node requires that a Device Binding node has gathered device data previously in the journey.

### Outcomes

#### `True`

Device binding data was successfully stored in the user's profile.

#### `False`

Device binding data was not stored in the user's profile.

### Properties

None.

## Device Signing Verifier node

Verifies possession of a registered bound device.

There are many similarities between WebAuthn and device binding and JWS verification. We provide authentication nodes to implement both technologies in your journeys.

Both can be used for usernameless and passwordless authentication, they both use public key cryptography, and both can be used as part of a multi-factor authentication journey.

One major difference is that with device binding, the private key never leaves the device.

With WebAuthn, there is a possibility that the private key is synchronized across client devices because of Passkey support, which may be undesirable for your organization.

For details of the differences, refer to the following table:

*Comparison of WebAuthn and Device Binding/JWS Verification*

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Industry-standards based | ☑ | × | You can refer to the WebAuthn W3C specification.<br>Device binding and JWS verification are proprietary implementations. |
| Public key cryptography | ☑ | ☑ | Both methods use Public key cryptography. |
| Usernameless support | ☑ | ☑ | After registration, the username can be stored in the device and obtained during authentication without the user having to enter their credentials. |
| Keys are bound to the device | × | ☑ | With WebAuthn, if Passkeys are used, they can be shared across devices.<br>With device binding, the private keys do not leave the device. |
| Sign custom data | × | ☑ | With device binding, you can:<br>• Customize the challenge that the device must sign. For example, you could include details of a transaction, such as the amount in dollars.<br>• Add custom claims to the payload when signing a challenge. This gives additional context that the server can make use of by using a scripted node. Refer to Add custom claims when signing. |
| Format of signed data | WebAuthn authenticator data | JSON Web Signature (JWS) | |
| Integration | × | ☑ | With device binding, after verification, the signed JWT is available in:<br>• Audit Logs<br>• Transient node state<br>This enables the data within to be used for integration into your processes and business logic. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Platform support | ☑ Android<br>☑ iOS<br>☑ Web browsers | ☑ Android<br>☑ iOS<br>✕ Web browsers | As it is challenging to store secure data in a browser as a client app, device binding is not supported in web browsers. |
| Authenticator support | Determined by the platform.<br>Configuration limited to:<br>• Biometric with Fallback to Device Pin | Determined by the authentication node.<br>Full configuration options:<br>• Biometric Authentication<br>• Biometric with Fallback to Device Pin<br>• Application Pin<br>• Silent | With device binding, you can specify what authentication action the user must perform to get access to the private keys.<br>This provides greater flexibility in your security implementation and can reduce authentication friction for your users. |
| Key storage | Web browsers and iOS synchronize to the cloud.<br>Android has the option to synchronize to the cloud. | ***Android***<br>KeyStore<br>***iOS***<br>Secure enclave: hardware-backed and not synchronized to the cloud. | Both technologies store the private keys securely on the client.<br>WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>This can reduce authentication friction for your users but may also increase the risk of a breach. |
| Managing device keys | Managed by the device OS.<br>Apps cannot delete *local* client keys programmatically and do not have a reference to the *remote* server key for deletion. | Managed by the Ping SDKs.<br>Provides an interface to delete local client and remote server keys. | The ability to programmatically delete both client and server keys can greatly simplify the process of registering a new device if an old device is lost or stolen. |
| Passkey support | ☑ | ✕ | WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>Device binding keeps the private key locked in the device. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| App integrity verification | ***Android***<br>Requires an `assetlinks.json` file.<br>***iOS***<br>Requires `apple-app-site-association` file. | Not provided by the device binding or verification nodes.<br>It can be added as part of the journey by using app integrity nodes. | App integrity verification helps ensure your users are only using a supported app rather than a third-party or potentially malicious version. |
| Key attestation | ***Android***<br>SafetyNet<br>***iOS***<br>None | ***Android***<br>Uses hardware-backed key pairs with Key Attestation.<br>***iOS***<br>It can be added as part of the journey by using app integrity nodes to support key attestation. | Key attestation verifies that the private key is valid and correct, is not forged, and was not created in an insecure manner. |
| Complexity | Medium | Low | WebAuthn requires a bit more configuration, for example, creating and uploading the `assetlinks.json` and `apple-app-site-association` files. Device binding only requires the journey and the SDK built into your app. |

The node requires the device to sign a challenge string using the private key that corresponds to a stored public key.

The user might need to unlock their cryptography keys with biometric security — such as a fingerprint — or a PIN.

**Note**

This node can be used in usernameless authentication flows.
The Ping SDKs store and provide the identity when handling the callbacks from this node. If the device has been registered by more than one user, the SDK displays a list of the registered keys to choose from on the client device.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

If you want the device to sign a custom challenge, its value must be available in shared state. Enter the variable name in the **Shared state attribute for Challenge** property.

### Dependencies

This node requires that you have bound devices by using the Device Binding node.

### Configuration

| Property | Usage |
|---|---|
| Sign Random Challenge | Use a random value as the challenge for signing. |
| Shared state attribute for Challenge | Use the value from the named attribute in shared state as the challenge for signing. |
| Application IDs | A list of Android package names and iOS bundle IDs of applications allowed to perform device signing verification.<br>For example, `com.example.app`. |
| Title | A title to display to the user when asking them to bind the device. |
| Sub Title | A secondary or subtitle to display to the user when asking them to bind the device. |
| Description | Descriptive text displayed to the user when asking them to bind the device. |

| Property | Usage |
| --- | --- |
| Capture Failure | When enabled, adds the reason for a failure to the shared node state variable `DeviceSigningVerifierNode.FAILURE` and continues evaluation along the "Failure" outcome.<br>If not enabled, the journey halts with an exception, and the journey *does not* continue along an outcome path.<br>Reasons for failure include:<br>**`INVALID_CLAIM`**<br>Failed to validate one or more claims presented in the token.<br>For example, the challenge claim did not match the value set in the node configuration, or the issuer ( `ISS` ) claim did not match a value in the **Application IDs** list.<br>**`INVALID_SIGNATURE`**<br>Failed to validate the token signature.<br>**`INVALID_USER`**<br>Account does not exist.<br>**`NOT_ACTIVE_USER`**<br>Account is not active or locked out.<br>**`INVALID_SUBJECT`**<br>Failed to validate the token subject. |
| Timeout | Specify the number of seconds to wait for a response from the client during binding.<br>If the specified time is reached, evaluation continues along the `Timeout` outcome path. |

## Outputs

If you enable the **Capture Failure** property, the node outputs a failure reason string in a variable named `DeviceSigningVerifierNode.FAILURE` .

## Outcomes

- `Success`
- `Failure`
- `No Registered Device`
- `Key Not Found`
- `Unsupported` (Client)
- `Abort` (Client)
- `Timeout` (Client)
- `ClientNotRegistered` (Client)

If the response from the device is verified as coming from a bound device, evaluation continues along the `Success` outcome path.

If AM cannot verify that the response was signed by a bound device, evaluation continues along the `Failure` outcome path.

If the user does not have any bound devices, evaluation continues along the `No Registered Device` outcome path. The user is determined either previously in the authentication journey, or by reading the `sub` claim from the response when doing usernameless flows.

If the client device cannot access the cryptography keys, or the key ID that AM requested cannot be located, evaluation continues along the relevant `Key Not Found` outcome path.

If the user's client does not support the requested operation, evaluation continues along the `Unsupported` outcome path.

If the user cancels authentication, evaluation continues along the `Abort` outcome path.

If the node does not receive a response from the user's device within the **Timeout** specified in the node configuration, evaluation continues along the `Timeout` outcome path.

If the client device does not have the keys present to be able to sign the challenge, evaluation continues along the `ClientNotRegistered` outcome path.

## Enable Device Management node

The **Enable Device Management** node controls the restrictions placed on users who want to reset or remove registered multi-factor authentication (MFA) devices.

By default, authenticated users can only remove a registered MFA device if they have authenticated by using a matching device. For example, to delete a device registered for OATH, they must have successfully authenticated by using a journey that includes an OATH Token Verifier node.

You can use this node in a journey to relax or remove this restriction.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

The node requires the `username` of the identity authenticating.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

## Dependencies

This node has no dependencies.

## Configuration

| Property | Usage |
| --- | --- |
| **Device Check Enforcement Strategy** | The MFA authentication method that is required to allow users to remove registered devices. Choose from:<br>**SAME**<br>The user must have authenticated with the same MFA type (WebAuthn, OATH, Device Binding, or PUSH) as the device they want to delete.<br>This matches the default behavior, as if this node were not used.<br>**ANY**<br>The user must have authenticated with any MFA type (WebAuthn, OATH, Device Binding, or PUSH) to be able to delete a device.<br>**NONE**<br>The user does not have to authenticate using WebAuthn, OATH, Device Binding, or PUSH to be able to delete a device.<br>**Caution**<br>If you use this option, ensure you authenticate the user as strongly as possible before allowing them to delete a device.<br>The default is `SAME`. |

## Outputs

This node adds a flag to the auth session the journey creates, depending on the **Device Check Enforcement Strategy** property.

The flag determines which MFA device types, if any, the user can delete.

## Outcomes

### Success

Any of the following situations result in the **Success** outcome from this node:

- The **Device Check Enforcement Strategy** is set to `SAME`.

  This setting relies on existing behavior to update the auth session and makes no changes of its own.

- The **Device Check Enforcement Strategy** is set to `ANY`, and at least one MFA type was used previously in the journey.

- The **Device Check Enforcement Strategy** is set to `NONE`, and the node was able to update the auth session.

### `Failure`

Any of the following situations result in the **Failure** outcome from this node:

- The `username` property of the identity is not available.
- The user is marked as inactive.
- The **Device Check Enforcement Strategy** is set to `ANY`, but the authentication journey didn't perform MFA.
- The node attempted to update the auth session but did not succeed.

## Errors

This node does not output any user-facing error messages.

## Example

The following example journey allows users to reset their registered MFA devices without having to authenticate using MFA.

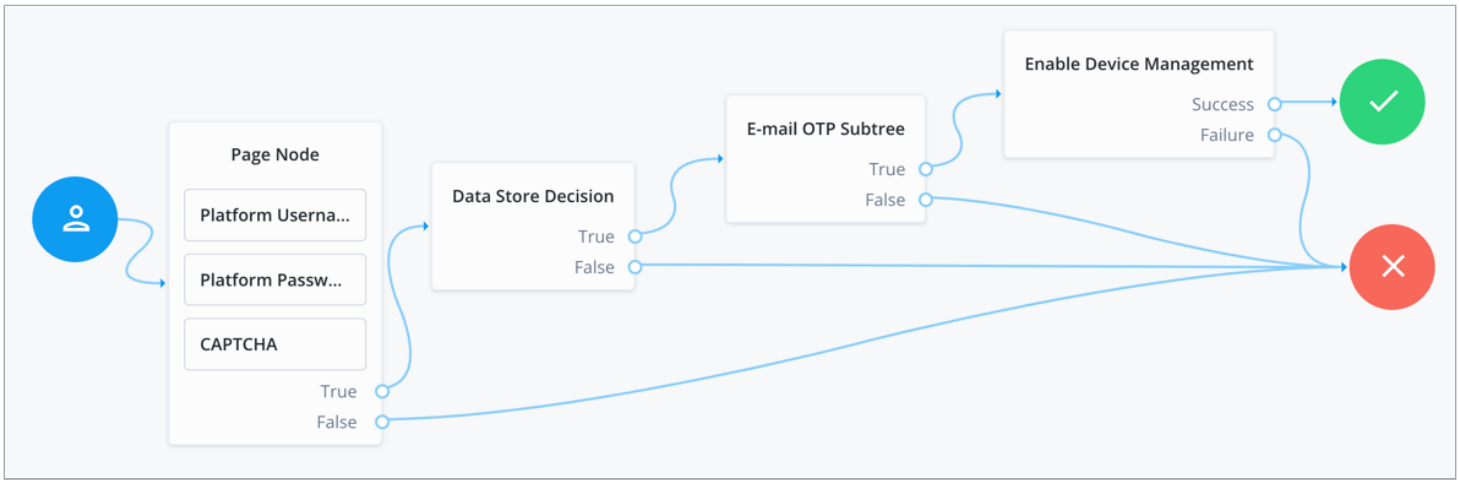



*Figure 1. Example Enable Device Management journey*

- The user enters their credentials, completes the CAPTCHA, and is verified against the identity store.
- A subtree sends a one-time passcode (OTP) to the user's email address to verify they have access to the address they have in their profile.
- The **Enable Device Management** node, with the **Device Check Enforcement Strategy** property set to `NONE`, upgrades the session the user receives to allow them to delete any of their registered MFA devices.

# Get Authenticator App node

The **Get Authenticator App** node displays information to get an authenticator app from the Apple App Store or the Google Play Store.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

None. This node doesn't read shared state data.

## Dependencies

This node has no dependencies.

## Configuration

| Property | Usage |
|---|---|
| **Get App Authenticator Message** | (Optional) Add a custom, localized message to display to the user. You can use the following variables to customize the message:<br>• `{{appleLink}}`<br>• `{{appleLabel}}`<br>• `{{googleLink}}`<br>• `{{googleLabel}}`<br>You can also include HTML elements, for example:<br>`Apple: <a target='_blank' href='{{appleLink}}'>{{appleLabel}}</a>`<br>1. Click **+**.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Get the app from the {{appleLink}} or on {{googleLink}}.` |

| Property | Usage |
| --- | --- |
| **Continue Label** | (Optional) Add custom, localized text to display on the button the user can click to continue.<br><br>1. Click **+**.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br><br>Leave blank to use the default message.[2]<br>Default: `Continue` |
| **Apple App Store URL** | The URL to download your authenticator app from the Apple App Store. The default value points to the ForgeRock Authenticator app for iOS.<br>Default: `https://itunes.apple.com/app/forgerock-authenticator/id1038442926` |
| **Google Play URL** | The URL to download your authenticator app from the Google Play Store. The default value points to the ForgeRock Authenticator app for Android.<br>Default: `https://play.google.com/store/apps/details?id=com.forgerock.authenticator` |

[1] Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

[2] PingAM only: Learn more about customizing and translating default messages in Internationalize nodes.

## Outputs

This node doesn't change the shared state.

## Outcomes

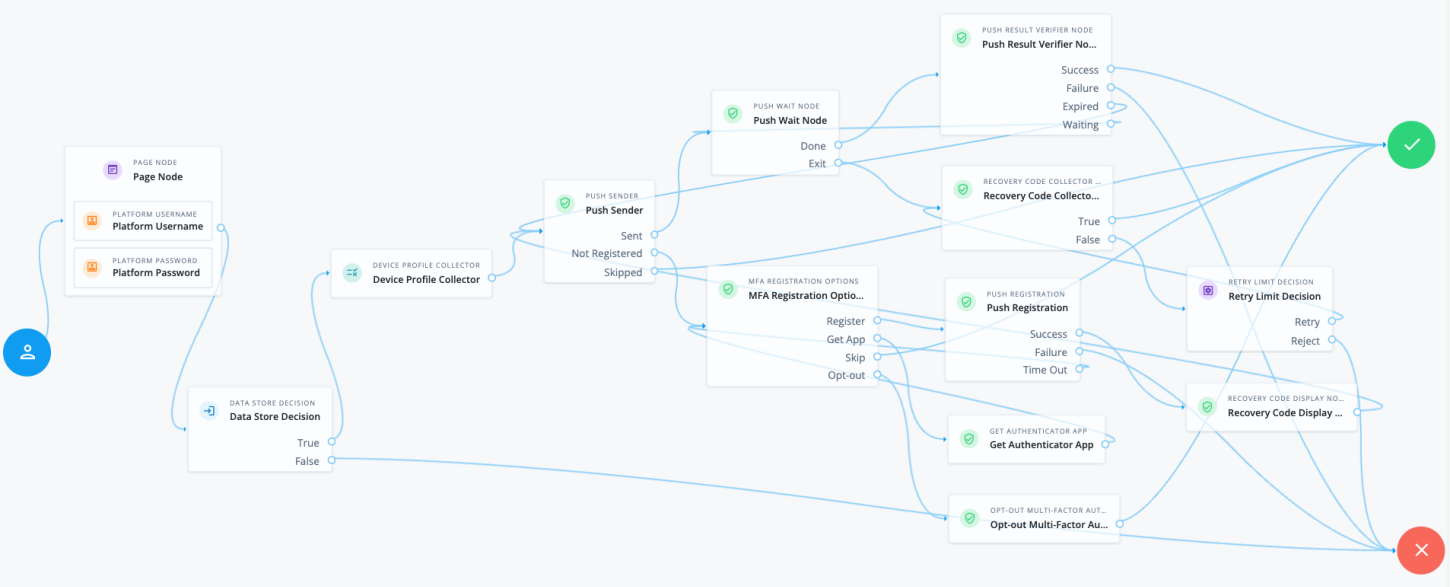
Single outcome path.

## Errors

This node doesn't log any error or warning messages of its own.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |

| Source node | Outcome path | Target node |
|---|---|---|
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ***Skip this step***

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ***Opt-out***

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

## HOTP Generator node

Creates a string of random digits of the specified length for use as a one-time passcode.

Passwords are stored in the `oneTimePassword` transient node state property.

Use this node with these nodes to add one-time passcode verification as an additional factor:

- OTP Email Sender node
- OTP SMS Sender node
- OTP Collector Decision node

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

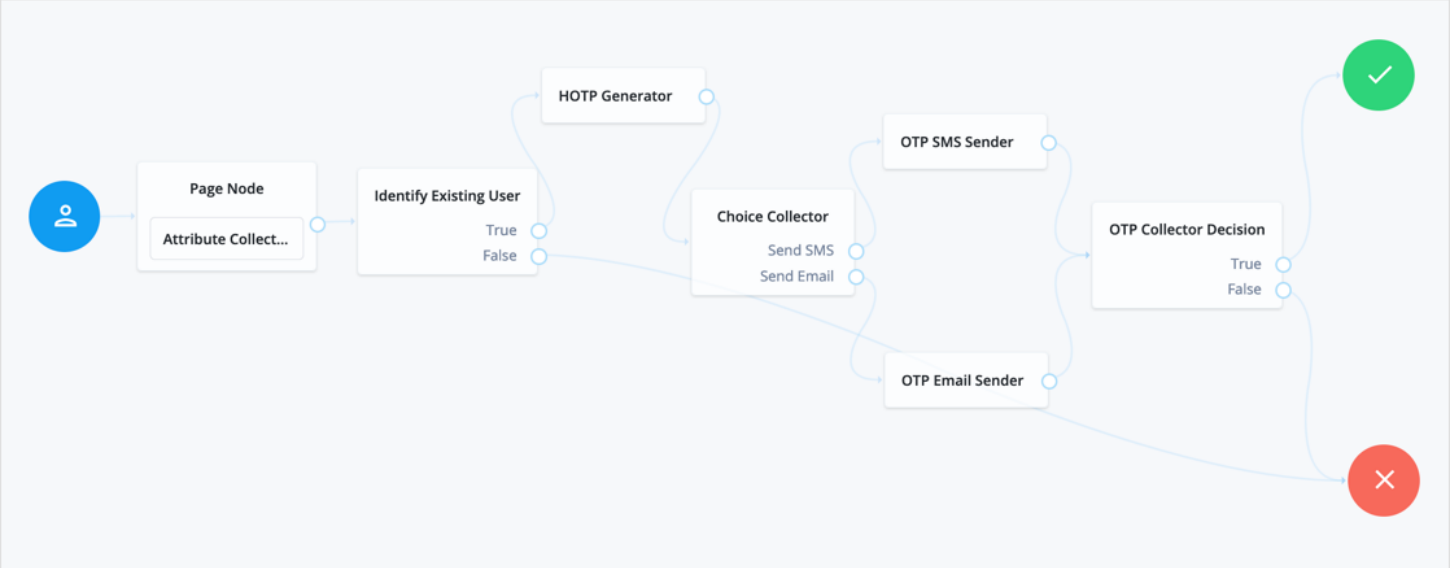
Single outcome path.

### Properties

| Property | Usage |
|---|---|
| One-time password length | Specify the number of digits in the one-time passcode. The minimum number of digits is 6, in accordance with the HOTP specification.<br>Default: 8 |

### Example

The following example uses an HOTP generator as part of multi-factor authentication:




## MFA Registration Options node

The **MFA Registration Options** node lets the user register a multi-factor authentication (MFA) device or skip the registration process.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

- This node requires a `username` in the incoming node state to identify which user to update.

  Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

- This node requires the `mfaMethod` in the incoming state to know what type of MFA device to register:

  - For push authentication, this node requires the `pushMessageId` in the incoming state, which is a unique ID to identify the push notification request.

    Implement a Push Sender node earlier in the journey.

- For OATH authentication, implement the OATH Token Verifier node earlier in the journey.

## Dependencies

This node has no dependencies.

## Configuration

| Property | Usage |
| --- | --- |
| **Remove 'skip' option** | Select this option to make it mandatory for the user to register a device. When selected, the **Skip this Step** and **Opt-out** buttons aren't displayed.<br>When disabled, the user can skip device registration or opt-out if required. |
| **Display Get Authenticator App** | Select this option to display the **Get the App** button. |
| **Message** | (Optional) Add a custom, localized message to display to the user with instructions on interacting with this node:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.(1)<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.(2)<br>Default: `On this page you can choose to register, skip or opt-out the second factor authentication method selected to protect your account. If you "Skip", an MFA method will not be registered now, but you will be prompted again on your next login. Otherwise, if you "Opt out", an MFA method will not be registered now and you will not be asked again. This choice is not recommended.` |
| **Register Device** | (Optional) Add custom, localized text to display on the button the user can click to register their device:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.(1)<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.(2)<br>Default: `Register Device` |

| Property | Usage |
| --- | --- |
| **Get Authenticator App** | (Optional) Add custom, localized text to display on the button the user can click to get the authenticator app:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Get the App` |
| **Skip this Step** | (Optional) Add custom, localized text to display on the button the user can click to skip registering their device:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Skip this step` |
| **Opt-out** | (Optional) Add custom, localized text to display on the button the user can click to opt out of registering their device:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Opt-out`<br>**Note**<br>This node doesn't update the user's profile with the opt-out decision. Use the **Opt-out** outcome in an Opt-out Multi-Factor Authentication node to persist the decision in the user's profile. |

[1] Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

[2] PingAM only: Learn more about customizing and translating default messages in Internationalize nodes.

## Outputs

This node doesn't change the shared state.

## Outcomes

### `Register`

The user chooses to register an MFA device.

### `Get App`

The user chooses to get the authenticator app.

### `Skip`

The user chooses to skip registering an MFA device this time.

### `Opt-out`

The user chooses to opt-out of registering an MFA device.

## Errors

The node can log the following errors:

- `Expected username to be set`

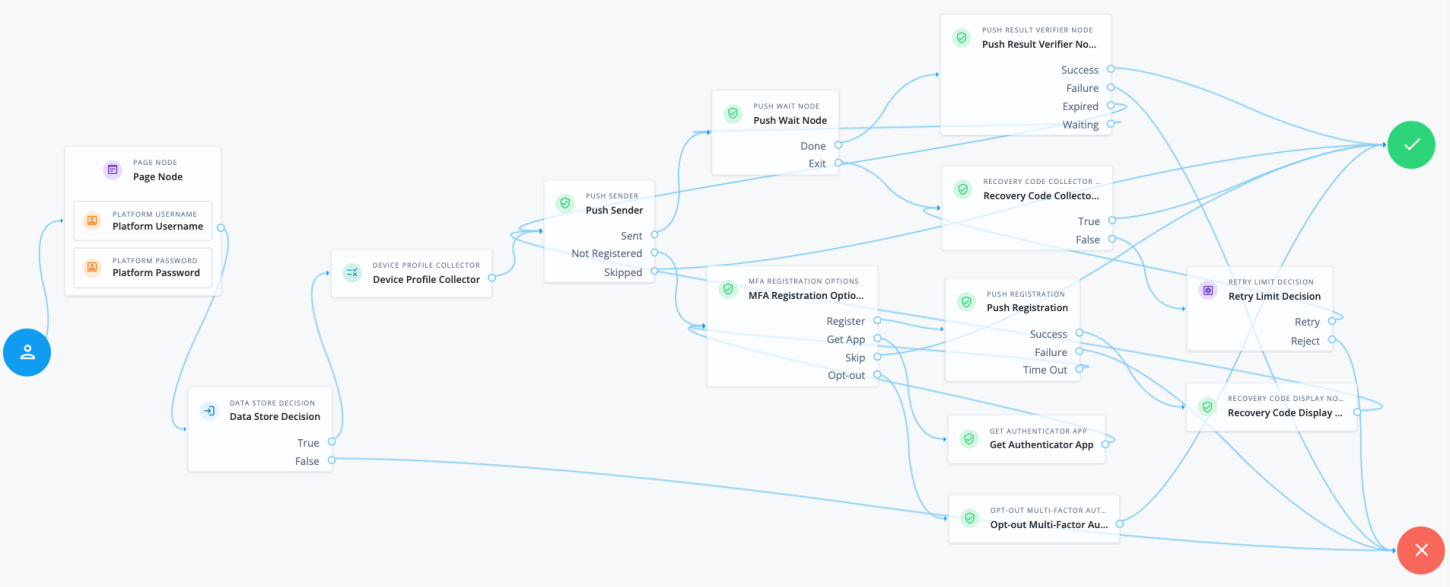
  The node can't identify the user from the shared state.

- `Expected multi-factor authentication method to be set`

  The node can't identify the MFA method from the shared state.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |

| Source node | Outcome path | Target node |
|---|---|---|
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ### *Register Device*

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ### *Get the App*

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ### *Skip this step*

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ### *Opt-out*

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

   After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

# OATH Device Storage node

The **OATH Device Storage** node stores devices in the user profile after an OATH Registration node records them in the shared state.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Authenticators

The OATH-related nodes can integrate with the following authenticator apps:

- The ForgeRock Authenticator app for Android and iOS.
- Third-party authenticator apps that support the following open standards:
  - RFC 4226: HMAC-Based One-Time Password (HOTP)
  - RFC 6238: Time-Based One-Time Password (TOTP)

## Inputs

This node reads the device profile as the value of the shared state attribute `oathDeviceProfile`.

## Dependencies

Precede this node in the flow with an OATH Registration node with its **Store device data in shared state** setting enabled.

## Configuration

This node has no configurable properties.

## Outputs

This node doesn't change the shared state.

## Outcomes

### Success

The node wrote the device profile to the user's account.

### `Failure`

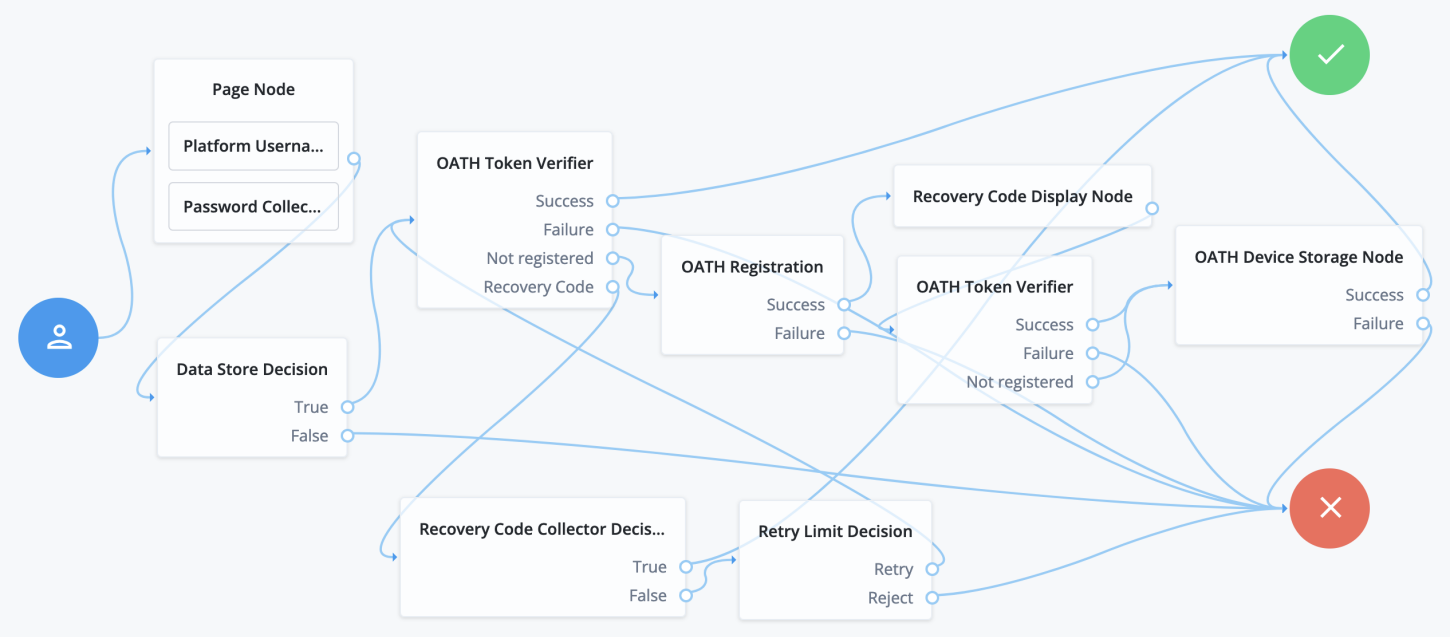
Any other case.

## Errors

This node logs a `No device profile found on shared state` error message if it can't get the device profile from the `oathDeviceProfile` shared state attribute.

## Example

The following journey includes both username-password and one-time passcode authentication:




- The Page node with the Platform Username node and the Platform Password node prompts for the user credentials.
- The Data Store Decision node confirms the username-password credentials.
- The first OATH Token Verifier node prompts for a one-time passcode with an option to use a recovery code.
- The OATH Registration node prompts the user to register a device and includes its profile in the shared state.
- The Recovery Code Display node shows the recovery codes and prompts the user to keep them safe.

- The second OATH Token Verifier node prompts for a one-time passcode using the newly registered device.
- The OATH Device Storage node writes the device profile to the user's account.
- The Recovery Code Collector Decision node prompts for a recovery code.
- The Retry Limit Decision node lets the user retry another code if they enter one incorrectly.

## OATH Registration node

The **OATH Registration** node lets the user register a device for OATH-based multi-factor authentication (MFA). Learn more about OATH in the OATH documentation.

Based on the node settings, the user device displays a QR code that includes all the details required for registration. If registration is successful, the node stores the device data and recovery codes (if enabled), and sets the `skippable` attribute to prevent repeat registration at next login.

> **Tip**
>
> You can use the Combined MFA Registration node to register a device for both push notifications and one-time passcode (OATH) verification in a single step.
> For an example that demonstrates how use to use other MFA nodes to create a complete OATH authentication journey, read the OATH Token Verifier node example.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Authenticators

The OATH-related nodes can integrate with the following authenticator apps:

- The ForgeRock Authenticator app for Android and iOS.
- Third-party authenticator apps that support the following open standards:
  - RFC 4226: HMAC-Based One-Time Password (HOTP)
  - RFC 6238: Time-Based One-Time Password (TOTP)

### Inputs

This node reads the `username` attribute and optionally the `oathDeviceProfile` attribute from the shared state.

## Dependencies

Your journey must confirm the user credentials before letting them register a device.

For example, precede this node with the following nodes earlier in the authentication flow:

PingAM : Username Collector node, Password Collector node, and Data Store Decision node.

Advanced Identity Cloud : Platform Username node, Platform Password node, and Data Store Decision node.

## Properties

| Property | Usage |
|---|---|
| **Issuer** | Specify an identifier to appear on the user's device, such as a company name, a website, or a realm.<br>The authenticator application displays the value.<br>Default: `ForgeRock` |
| **Account Name** | Select the profile attribute to display as the username in the authenticator application.<br>If not specified, or if the specified profile attribute is empty, their username is used.<br>Default: **Username** |
| **Background Color** | The background color in hex notation that displays behind the issuer's logo within the authenticator application.<br>Default: `032b75` |
| **Logo Image URL** | The location of an image to download and display as the issuer's logo within the authenticator application.<br>**Note**<br>The ForgeRock Authenticator supports logos in JPEG and PNG format only. The application resizes your logo automatically, but a maximum image size of one MByte (or 1024 X 1024 pixels) is recommended.<br>Default: none |
| **Generate Recovery Codes** | If enabled, recovery codes are generated and stored in the successful outcome's transient state.<br>Use the Recovery Code Display node to display the codes to the user for safekeeping.<br>Default: true |

| Property | Usage |
| --- | --- |
| **QR code message** | A custom, localized message with instructions to scan the QR code to register the device.<br><br>1. Click **Add**.<br>2. Enter the message locale in the `Key` field. For example, `en-gb`.<br>3. Enter the message to display to the user in the `Value` field.<br><br>Default: none |
| **One Time Password Length** | The length of the generated OTP in digits.<br>This value must be at least `6`. It must also be compatible with the hardware/software OTP generators you expect end users to use. For example, Google and ForgeRock authenticators support values of `6` and `8` respectively.<br>Default: `6` |
| **Minimum Secret Key Length** | Number of hexadecimal characters allowed for the secret key.<br>Default: `32` |
| **OATH Algorithm** | Specify the algorithm the device uses to generate the OTP:<br><br>***HOTP***<br>HOTP uses a counter; the counter increments every time a new OTP is generated. When you use this setting, also set the same value in the OATH Token Verifier node.<br>***TOTP***<br>TOTP generates a new OTP every few seconds as specified by the **TOTP Time Step Interval** setting.<br><br>Default: `TOTP` |
| **TOTP Time Step Interval** | The length of time that an OTP is valid in seconds.<br>For example, if the time step interval is 30 seconds, a new OTP is generated every 30 seconds and is valid for 30 seconds only.<br>Default: `30` seconds |
| **TOTP Hash Algorithm** | The HMAC hash algorithm used to generate the OTP codes. AM supports SHA1, SHA256, and SHA512.<br>Default: `SHA1` |
| **HOTP Checksum Digit** | This adds a digit to the end of the OTP generated to be used as a checksum to verify the OTP was generated correctly. This is in addition to the actual password length.<br>Only set this if the user devices support it.<br>Default: false |
| **HOTP Truncation Offset** | This is an option used by the HOTP algorithm that not all devices support. Leave the default value unless you know user devices use an offset.<br>Default: `-1` |

| Property | Usage |
| --- | --- |
| **Store device data in shared state** | If enabled, the device data isn't stored in the user profile on successful completion of the node.<br>Instead, the node adds the device data as a base64-encoded string to the `oathDeviceProfile` property in the shared node state. This string is decoded as an unescaped plain string representation of a JSON object. For example:<br>In the shared node state:<br>`oathDeviceProfile="eyAidXVpZCI6ICJhNDhiMjUyMS0xYzliLTRiYTct...ja0RyaWZ0U2Vjb25kcyI6IDAgfQ"`<br>Decoded value:<br>`{ "uuid": "a48b2521-1c9b-4ba7-a45c-8dd855c7397c", "recoveryCodes": [], "sharedSecret": "0CF9910A24CAF84E81CEBA71C2086DE4", "deviceName": "OATH Device", "lastLogin": 0, "counter": 0, "checksumDigit": false, "truncationOffset": -1, "clockDriftSeconds": 0 }`<br>Use the OATH Device Storage node to store the device data in the user profile instead.<br>**Note**<br>If you enable this property, recovery codes aren't displayed at the end of the journey. They're stored as part of the `oathDeviceProfile` property in the shared node state.<br>Default: false |

## Outputs

If the **Store device data in shared state** setting is enabled, this node records the device profile in the `oathDeviceProfile` shared state attribute.

If the **Generate Recovery Codes** setting is enabled, this node records the recovery codes in the `oathEnableRecoveryCode` shared state attribute.

## Outcomes

### Success

Device registration succeeded.

### `Failure`

Any other case.

## Errors

This node logs the following error messages:

### `No username found.`

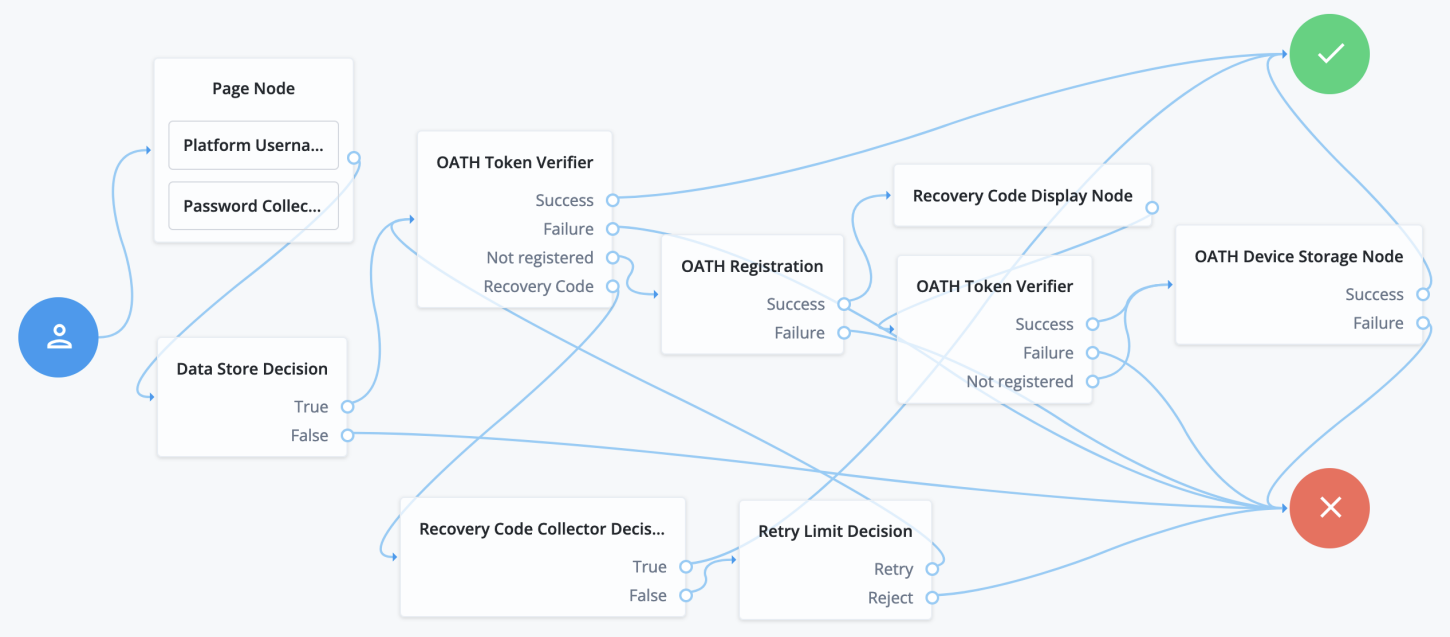
The node failed to read the username from the shared state.

### `No device profile found on shared state`

The node failed to read the device profile from the shared state.

## Example

The following journey includes both username-password and one-time passcode authentication:




- The Page node with the Platform Username node and the Platform Password node prompts for the user credentials.
- The Data Store Decision node confirms the username-password credentials.
- The first OATH Token Verifier node prompts for a one-time passcode with an option to use a recovery code.
- The OATH Registration node prompts the user to register a device and includes its profile in the shared state.
- The Recovery Code Display node shows the recovery codes and prompts the user to keep them safe.
- The second OATH Token Verifier node prompts for a one-time passcode using the newly registered device.
- The OATH Device Storage node writes the device profile to the user's account.

- The Recovery Code Collector Decision node prompts for a recovery code.
- The Retry Limit Decision node lets the user retry another code if they enter one incorrectly.

## OATH Token Verifier node

The **OATH Token Verifier** node requests and verifies a one-time passcode (OTP) generated by a device such as a mobile phone.

The default configuration is time-based OTP (TOTP), but the node also supports HMAC (HOTP).

The node requires prior authentication and a device registered with an OATH Registration node.

> **Note**
>
> You can use the OATH nodes in conjunction with the ForgeRock Authenticator application to register your device, receive notifications, and generate one-time passwords.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

#### Authenticators

The OATH-related nodes can integrate with the following authenticator apps:

- The ForgeRock Authenticator app for Android and iOS.
- Third-party authenticator apps that support the following open standards:
  - RFC 4226: HMAC-Based One-Time Password (HOTP)
  - RFC 6238: Time-Based One-Time Password (TOTP)

### Inputs

This node reads the `username` attribute from the shared state.

### Dependencies

Confirm the user credentials before letting them authenticate with a device.

For standalone AM deployments, implement a Username Collector node and a Password Collector node earlier in the journey.

For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node earlier in the journey.

Also implement a Data Store Decision node

## Configuration

| Property | Usage |
|---|---|
| **OATH Algorithm** | Specify the algorithm the device uses to generate the OTP:<br>***HOTP***<br>HOTP uses a counter; the counter increments every time a new OTP is generated. When you use this setting, also set the same value in the OATH Registration node.<br>***TOTP***<br>TOTP generates a new OTP every few seconds as specified by the **TOTP Time Step Interval** setting.<br>Default: `TOTP` |
| **HOTP Window Size** | Specify how much the OTP device and the server counter can be out of sync.<br>For example, if the window size is 100 and the server's last successful login was at counter value 2, the server accepts an OTP that is generated between counter 3 and 102.<br>Default: `100` |
| **TOTP Time Step Interval** | The length of time an OTP is valid in seconds.<br>For example, if the time step interval is 30 seconds, a new OTP is generated every 30 seconds and is valid for 30 seconds only.<br>Default: `30` seconds |
| **TOTP Time Steps** | Specify how many time steps the OTP can be out of sync.<br>This applies to codes generated before or after the current code. For example, with a time step of 1, the server accepts the previous, current, and next codes.<br>Default: `2` |
| **TOTP Hash Algorithm** | The HMAC hash algorithm used to generate the OTP codes. The ForgeRock Authenticator application supports SHA1, SHA256, and SHA512.<br>Default: `SHA1` |
| **TOTP Maximum Allowed Clock Drift** | Specify how many time steps the authenticator application can be out of sync with the server before manual resynchronization is required.<br>For example, with **TOTP Time Steps** of `3` and a **TOTP Time Step Interval** of 30 (seconds), the server treats codes up to 90 seconds from the current time as belonging to the current time step.<br>The drift for a user's device is calculated each time they enter a new code. If the drift exceeds this value, the outcome is `Failure`.<br>Default: `5` |

| Property | Usage |
| --- | --- |
| **Allow recovery codes** | If enabled, lets users provide a recovery code to authenticate.<br>Default: false |

## Outputs

If the outcome is `Not registered`, this node sets `"mfaMethod": "oath"` in the shared state.

## Outcomes

### `Success`

The user has a registered device and the token code was verified.

### `Failure`

The user was not authenticated, or the collected token code can't be verified.

### `Not registered`

The user account has no registered device profiles.

### `Recovery Code`

**Allow recovery codes** is enabled, and the user chose to provide a recovery code.

## Errors

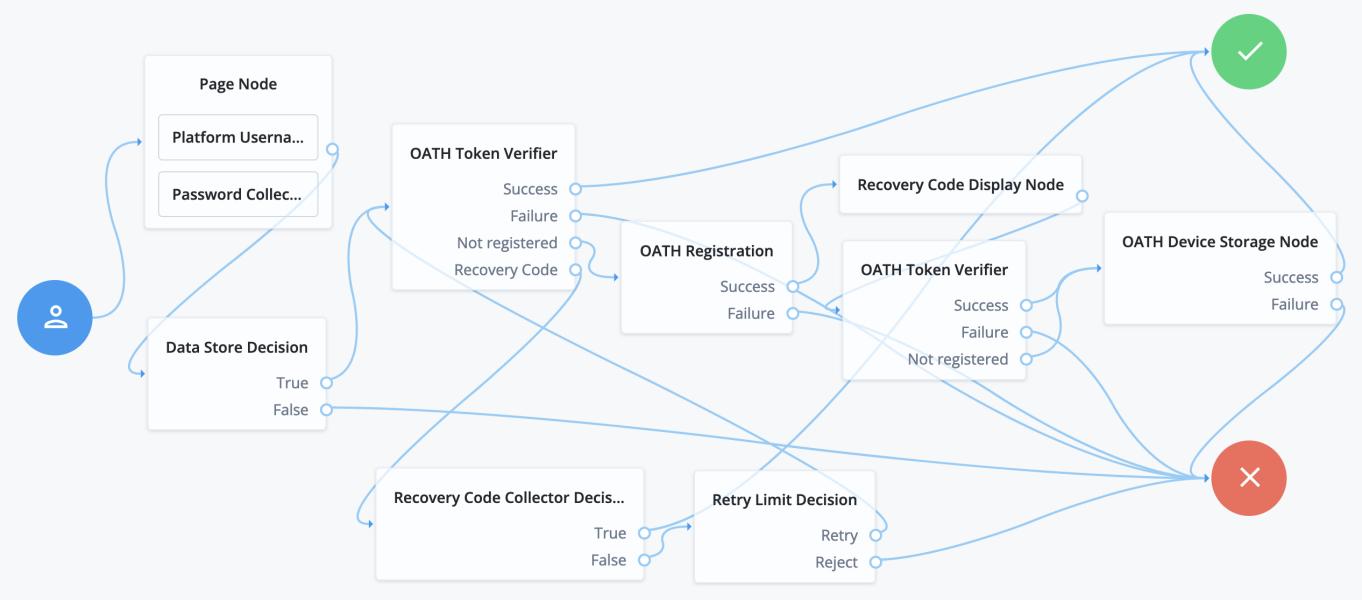
If this node cannot read the username from the shared state, it logs an error message: `Expected username to be set.`

If processing raises an exception, this node logs the detail as an error message.

## Example

### Advanced Identity Cloud

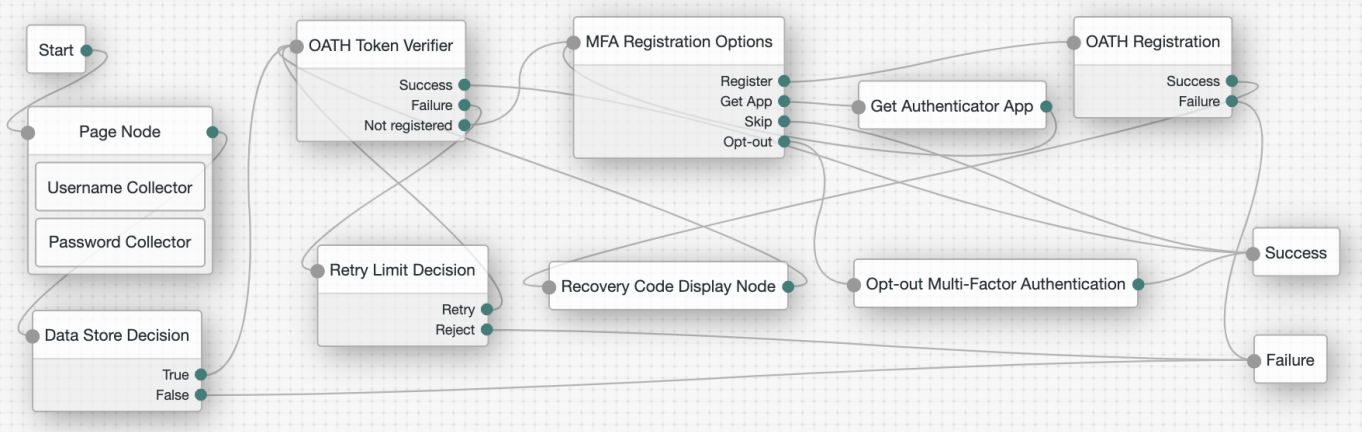
The following journey includes both username-password and one-time passcode authentication:




- The Page node with the Platform Username node and the Platform Password node prompts for the user credentials.
- The Data Store Decision node confirms the username-password credentials.
- The first OATH Token Verifier node prompts for a one-time passcode with an option to use a recovery code.
- The OATH Registration node prompts the user to register a device and includes its profile in the shared state.
- The Recovery Code Display node shows the recovery codes and prompts the user to keep them safe.
- The second OATH Token Verifier node prompts for a one-time passcode using the newly registered device.
- The OATH Device Storage node writes the device profile to the user's account.
- The Recovery Code Collector Decision node prompts for a recovery code.
- The Retry Limit Decision node lets the user retry another code if they enter one incorrectly.

PingAM

The following journey uses this node as part of a flexible multi-factor authentication (MFA) authentication flow:




- The Page node with the Username Collector node and the Password Collector node prompts for the user credentials.
- The Data Store Decision node confirms the username-password credentials.
- The OATH Token Verifier node prompts for a one-time passcode with an option to use a recovery code.
- The Retry Limit Decision node lets the user retry another code if they enter one incorrectly.
- The MFA Registration Options node lets the user choose how to register their device.
- The Get Authenticator App node helps the user install the ForgeRock Authenticator application.
- The OATH Registration node prompts the user to register a device and includes its profile in the shared state.
- The Recovery Code Display node shows the recovery codes and prompts the user to keep them safe.
- The Opt-out Multi-Factor Authentication node lets the user choose to skip the second authentication factor.

## OTP Collector Decision node

Requests and verifies one-time passwords.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `True`
- `False`

Evaluation continues along the `True` outcome path if the one-time passcode is valid; otherwise, evaluation continues along the `False` outcome path.

### Properties

| Property | Usage |
| --- | --- |
| One Time Password Validity Length | Specify the length of time, in minutes, that a one-time passcode remains valid.<br>Default: `5` |

## OTP Email Sender node

The OTP Email Sender node sends an email containing a generated one-time passcode (OTP) to the user.

Send mail requests time out after 10 seconds.

**Tip**

PingAM only

You can change the timeout in the following advanced AM server properties:

- `org.forgerock.openam.smtp.system.connect.timeout`
- `org.forgerock.openam.smtp.system.socket.read.timeout`
- `org.forgerock.openam.smtp.system.socket.write.timeout`

- To configure advanced server properties for all the instances of the AM environment, go to **Configure > Server Defaults > Advanced** in the AM admin UI.
- To configure advanced server properties for a specific instance, go to **Deployment > Servers > *Server Name* > Advanced**.

If the property you want to add or edit is already configured, click the pencil (🖉) button to edit it, then click the checkmark (✓) button.
Save your changes.
For more information, refer to advanced properties.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Prerequisites

- The node requires a configured email provider.

## Inputs

This node requires the following input from the shared state:

- The authenticating user's ID. The node queries the user's entry for an email address.

  Implement an Attribute Collector node node before this node to obtain the user's ID.

- The OTP stored in the `oneTimePassword` transient state property.

  Implement the HOTP Generator node before this node in the journey to obtain the OTP.

## Configuration

| Property | Usage |
|---|---|
| **Mail Server Host Name** *(required)* | The hostname of the SMTP email server. |
| **Mail Server Host Port** | The outgoing mail server port.<br>Common ports are 25, 465 for SSL/TLS, or 587 for StartTLS. |
| **Mail Server Authentication Username** | The username AM uses to connect to the mail server. |
| **Mail Server Authentication Password** | The password AM uses to connect to the mail server.<br>**Note**<br>This property is deprecated. Use the **Mail Server Secret Label Identifier** instead.<br>If you set a **Mail Server Secret Label Identifier**, this password is ignored. |

| Property | Usage |
| --- | --- |
| **Mail Server Secret Label Identifier** | An identifier used to create a *secret label* for mapping to a secret in a secret store. AM uses this identifier to create a specific secret label for this node. The secret label takes the form `am.authentication.nodes.otp.mail.identifier.password` where identifier is the value of **Mail Server Secret Label Identifier**. The identifier can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods (`.`). It can't start or end with a period.<br>If you set a **Mail Server Secret Label Identifier** and AM finds a matching secret in a secret store, the **Mail Server Authentication Password** is ignored. |
| **Email From Address** *(required)* | The email address from which the OTP will appear to have been sent. |
| **Email Attribute Name** | The attribute in the user profile that contains the email address to which the email with the OTP is sent.<br>Default: `mail` |
| **The subject of the email** | Click **Add** to add a new email subject. Enter the locale, such as `en-uk`, in the **Key** field and the subject in the **Value** field. Repeat these steps for each locale that you support. |
| **The content of the email** | Click **Add** to add the content of the email. Enter the locale, such as `en-uk`, in the **Key** field and the email content in the **Value** field. Repeat these steps for each locale that you support. |
| **Mail Server Secure Connection** | Set the connection method to the mail server.<br>If you set a secure method here, AM must trust the server certificate of the mail server.<br>The possible values for this property are:<br>• `NON SSL/TLS`<br>• `SSL/TLS`<br>• `Start TLS`<br>Default: `SSL/TLS` |
| Gateway Implementation Class | The class the node uses to send SMS and email messages.<br>PingAM only: A custom class must implement the `com.sun.identity.authentication.modules.hotp.SMSGateway` interface.<br>Default: `com.sun.identity.authentication.modules.hotp.DefaultSMSGatewayImpl` |

## Outputs

This node copies shared and transient state into the outgoing node state.

## Errors

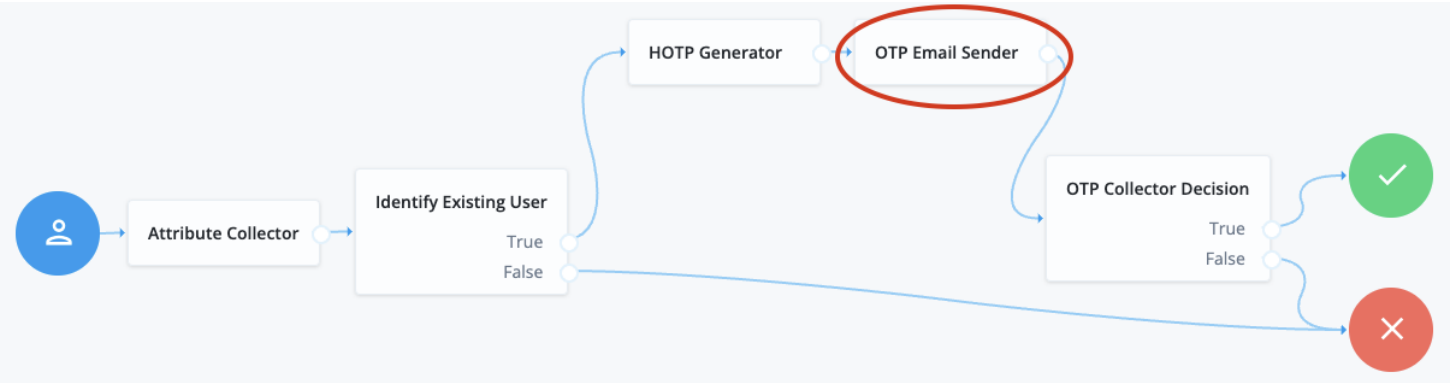
The node throws an `IdRepoException` and an `SSOException` error if it's unable to obtain the user's email address.

### Outcomes

Single outcome path.

Implement an OTP Collector Decision node after this node to continue the authentication journey.

### Example




## OTP SMS Sender node

The OTP SMS Sender node uses an email-to-SMS gateway provider to send an SMS message containing a generated one-time password (OTP) to the user.

The node sends an email to an address formed by joining the following values together:

- The user's telephone number, obtained by querying a specified profile attribute, for example, `telephoneNumber`.
- The `@` character.
- The email-to-SMS gateway domain, obtained by querying the profile attribute specified by the **Mobile Carrier Attribute Name** property.

For example, if configured to use the *TextMagic* email-to-SMS service, the node might send an email through the specified SMTP server to the address: `18005550187@textmagic.com`.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Prerequisites

- The node requires a configured email-to-SMS gateway provider.

## Inputs

This node requires the following input from the shared state:

- The authenticating user's ID. The node queries the user's entry for a telephone number.

  Implement an Attribute Collector node node before this node to obtain the user's ID.

- The OTP stored in the `oneTimePassword` transient state property.

  Implement the HOTP Generator node before this node in the journey to obtain the OTP.

## Configuration

| Property | Usage |
|---|---|
| **Mail Server Host Name** *(required)* | The hostname of the SMTP email server. |
| **Mail Server Host Port** | The outgoing mail server port.<br>Common ports are 25, 465 for SSL/TLS, or 587 for StartTLS. |
| **Mail Server Authentication Username** | The username AM uses to connect to the mail server. |
| **Mail Server Authentication Password** | The password AM uses to connect to the mail server.<br>**Note**<br>This property is deprecated. Use the **Mail Server Secret Label Identifier** instead.<br>If you set a **Mail Server Secret Label Identifier**, this password is ignored. |
| **Mail Server Secret Label Identifier** | An identifier used to create a *secret label* for mapping to a secret in a secret store. AM uses this identifier to create a specific secret label for this node. The secret label takes the form `am.authentication.nodes.otp.sms.identifier.password` where identifier is the value of **Mail Server Secret Label Identifier**. The identifier can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods (`.`). It can't start or end with a period.<br>If you set a **Mail Server Secret Label Identifier** and AM finds a matching secret in a secret store, the **Mail Server Authentication Password** is ignored. |
| **Email From Address** *(required)* | The email address from which the OTP will appear to have been sent. |
| **Mobile Phone Number Attribute Name** | The attribute in the user profile that contains the mobile phone number to which the SMS with the OTP is sent.<br>Default: `telephoneNumber` |

| Property | Usage |
| --- | --- |
| **Mobile Carrier Attribute Name** | The attribute in the user profile that contains the mobile carrier domain for sending SMS messages.<br>By default, an AM user profile doesn't have an attribute for the mobile carrier domain.<br>You can customize the user profile by adding a new attribute to it, then populate that attribute with users' SMS messaging domains.<br>All mobile carriers and bulk SMS messaging services have associated SMS messaging domains. For example, Verizon uses `vtext.com`, T-Mobile uses `tmomail.net`, and the TextMagic service uses `textmagic.com`. If you plan to send text messages internationally, determine whether the messaging service requires a country code.<br>If you leave the **Mobile Carrier Attribute Name** property empty, AM defaults to sending SMS messages using `txt.att.net` for all users. |
| **The subject of the message** | Click **Add** to add a new message subject. Enter the locale, such as `en-uk`, in the **Key** field and the subject in the **Value** field. Repeat these steps for each locale that you support. |
| **The content of the message** | Click **Add** to add the content of the message. Enter the locale, such as `en-uk`, in the **Key** field and the email content in the **Value** field. Repeat these steps for each locale that you support. |
| **Mail Server Secure Connection** | Set the connection method to the mail server.<br>If you set a secure method here, AM must trust the server certificate of the mail server.<br>The possible values for this property are:<br>• `NON SSL/TLS`<br>• `SSL/TLS`<br>• `Start TLS`<br>Default: `SSL/TLS` |
| **Gateway Implementation Class** | The class the node uses to send SMS and email messages.<br>PingAM only: A custom class must implement the `com.sun.identity.authentication.modules.hotp.SMSGateway` interface.<br>Default: `com.sun.identity.authentication.modules.hotp.DefaultSMSGatewayImpl` |

## Outputs

This node copies shared and transient state into the outgoing node state.

## Errors

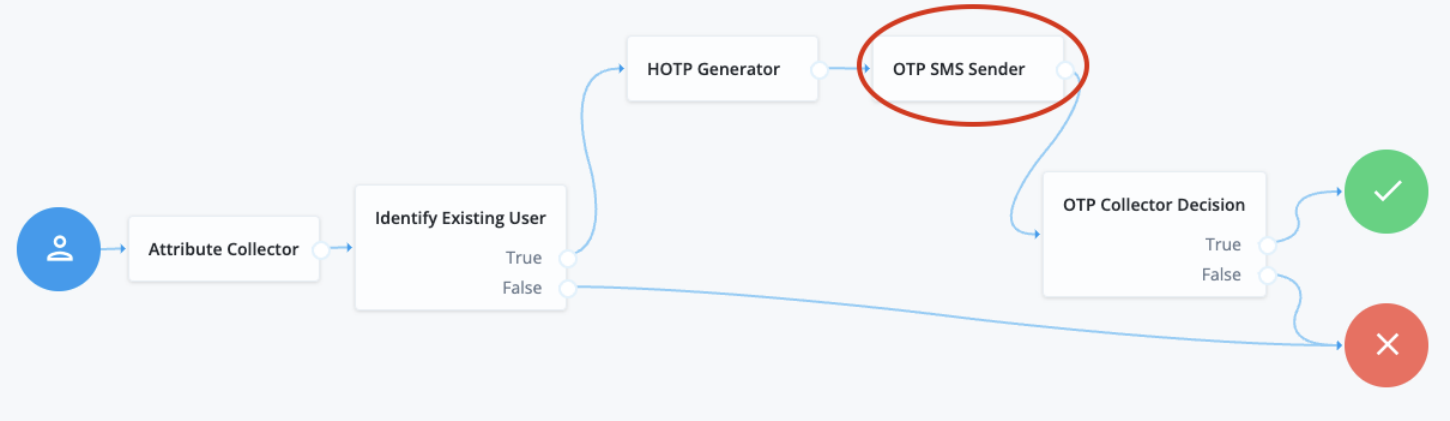
The node throws an `IdRepoException` and an `SSOException` error if it's unable to obtain the user's telephone number.

### Outcomes

Single outcome path.

Implement an OTP Collector Decision node after this node to continue the authentication journey.

### Example




## Opt-out Multi-Factor Authentication node

The **Opt-out Multi-Factor Authentication** node sets an attribute in the user's profile, which records their decision to skip multi-factor authentication (MFA) on the selected device.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

- This node requires the `realm` and `username` properties in the incoming node state.

  Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

- This node requires the `mfaMethod` in the incoming state to know what type of MFA device to update:

  - For push authentication, this node requires the `pushMessageId` in the incoming state, which is a unique ID to identify the push notification request.

Implement a Push Sender node earlier in the journey.

- For OATH authentication, implement the OATH Token Verifier node earlier in the journey.

### Dependencies

This node has no dependencies.

### Configuration

This node has no configurable properties.

### Outputs

This node doesn't change the shared state.

This node updates the user's profile with either the `push2faEnabled` or `oath2faEnabled` attribute to record their decision to opt out.

These are the default attributes but they can be changed in the ForgeRock Authenticator (Push) service or the ForgeRock Authenticator (OATH) service.

### Outcomes

Single outcome path.

### Errors

The node can log the following errors:

- `Expected username to be set`

  The node can't identify the user from the shared state.

- `Expected MFA method to be set`

  The node can't identify the MFA method from the shared state.

- `Unable to set user attribute as skippable`

  The node can't set the relevant attribute for the user's current device.

- `Failed to get the identity object`

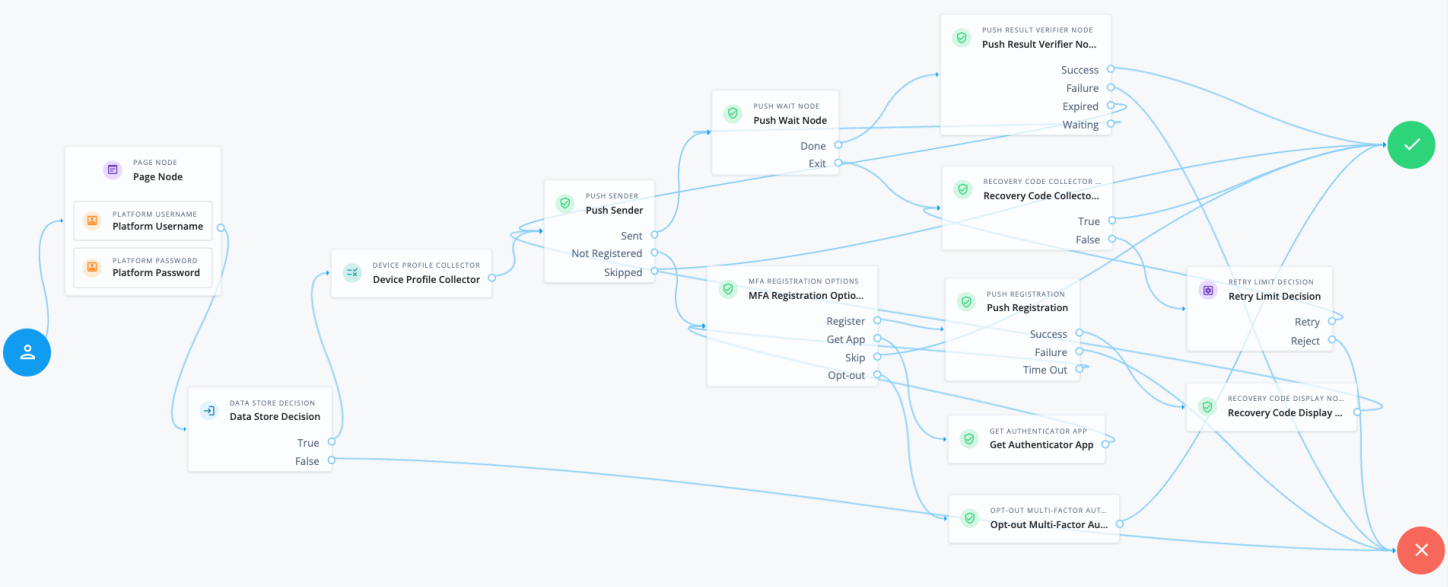
  The node can't read the identity of the account.

- `Unsupported MFA method has been set`

  The node can't retrieve the device profile details or the user has an unsupported MFA method set.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |

| Source node | Outcome path | Target node |
|---|---|---|
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ***Skip this step***

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ***Opt-out***

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

## Push Registration node

The **Push registration** node lets a user register their device, such as a mobile phone for multi-factor authentication (MFA) using push notifications.

Learn more in the Push Authentication documentation for:

- Advanced Identity Cloud
- PingAM

> **Tip**
>
> You can use the Combined MFA Registration node to register a device for use with both push notifications and one-time passcode (OATH) verification in a single step.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Authenticators

The push-related nodes integrate with the ForgeRock Authenticator app for Android and iOS.

Third-party authenticator apps aren't compatible with the push notification functionality.

### Inputs

This node requires the `realm` and `username` properties in the incoming node state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

## Dependencies

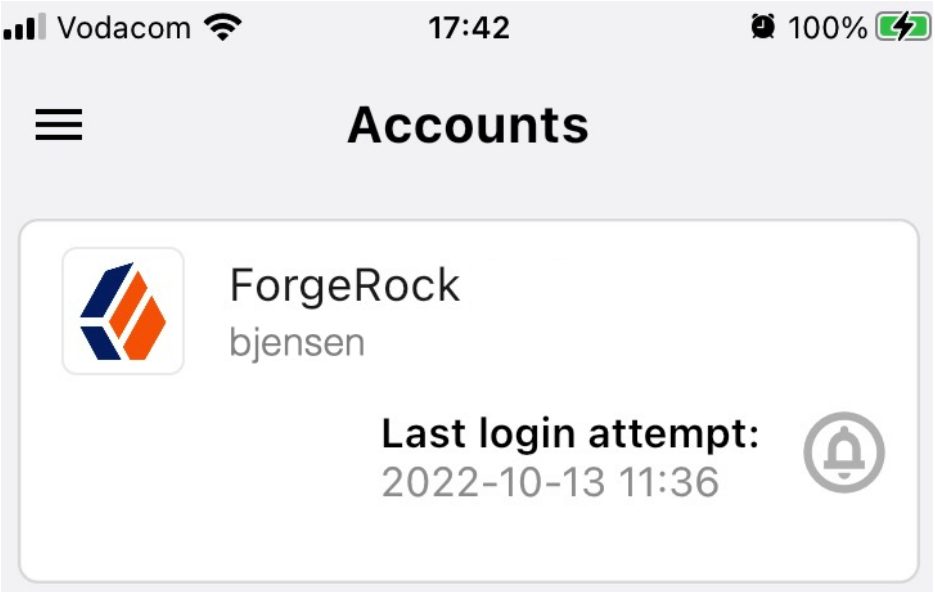
You must configure the Push Notification service for the realm to use this node. Optionally, also configure the ForgeRock Authenticator (Push) service.

Find more information in the corresponding documentation for:

- Advanced Identity Cloud
- PingAM

Find information on provisioning the credentials used by the service in How To Configure Service Credentials (Push Auth, Docker) in Backstage.

## Configuration

| Property | Usage |
|---|---|
| **Issuer** | The name of the issuer or application so the user knows which service their account relates to.<br>This value is displayed in the authenticator app.<br>  |
| **Account Name** | The profile attribute to display as the username in the authenticator app.<br>If not specified, or if the specified profile attribute is empty, their username is used. |
| **Background Color** | The background color, in hex notation, to display behind the issuer's logo within the authenticator app. |
| **Logo Image URL** | The location of an image to download and display as the issuer's logo in the authenticator app. |

| Property | Usage |
| --- | --- |
| **Generate Recovery Codes** | Select this option to generate push-specific recovery codes. When enabled, recovery codes are generated and stored in the transient state if registration is successful.<br>Use the Recovery Code Display node to display the codes to the user for safe keeping.<br><br>**Important**<br>Generating recovery codes overwrites all existing push-specific recovery codes.<br>Only the most recent set of recovery codes can be used for authentication if a device is lost or stolen. |
| **QR code message** | (Optional) Add a custom, localized message to display to the user with instructions to scan the QR code to register the device:<br><br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.(1)<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br><br>Leave blank to use the default message.(2)<br>Default: `Open your Authenticator app and tap the number shown to sign-in` |
| **Registration Response Timeout** | The number of seconds to wait for a response from the authenticator app.<br>As soon as the specified time is reached, evaluation continues along the `Time Out` outcome path. |

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

(2) PingAM only: Learn more about customizing and translating default messages in Internationalize nodes.

## Outputs

- The node adds the `pushDeviceProfiles` attribute to the user's profile with the device details on successful registration.
- The node updates the shared state with the push device settings, the message ID and the push challenge.
- If **Generate Recovery Codes** is enabled, the node records the recovery codes in the `recoveryCodes` shared state attribute. Use the Recovery Code Display node to display the codes to the user for safe keeping.

## Outcomes

### Success

The user successfully registered their authenticator app.

**`Failure`**

An issue occurred during device registration.

**`Time Out`**

A response wasn't received from the user's device within the time specified in the node configuration.

## Errors

The node can log the following errors:

- `Unable to find push message ID`

  The node failed to read the `pushMessageId` from the shared state and can't proceed with registration. Make sure you have implemented a Push Sender node earlier in the journey.

- `Expected username to be set`

  The node can't identify the user from the shared state.

- `Unable to read service addresses for Push Notification Service`

  The node can't retrieve the push service URLs. Check that the Push Notification service is set up correctly in the realm.

- `Could not get messageId`

  The node fails to retrieve the messageId and can't proceed with registration.

- `The push message corresponds to <message type> message type which is not registered in the <realm name> realm`

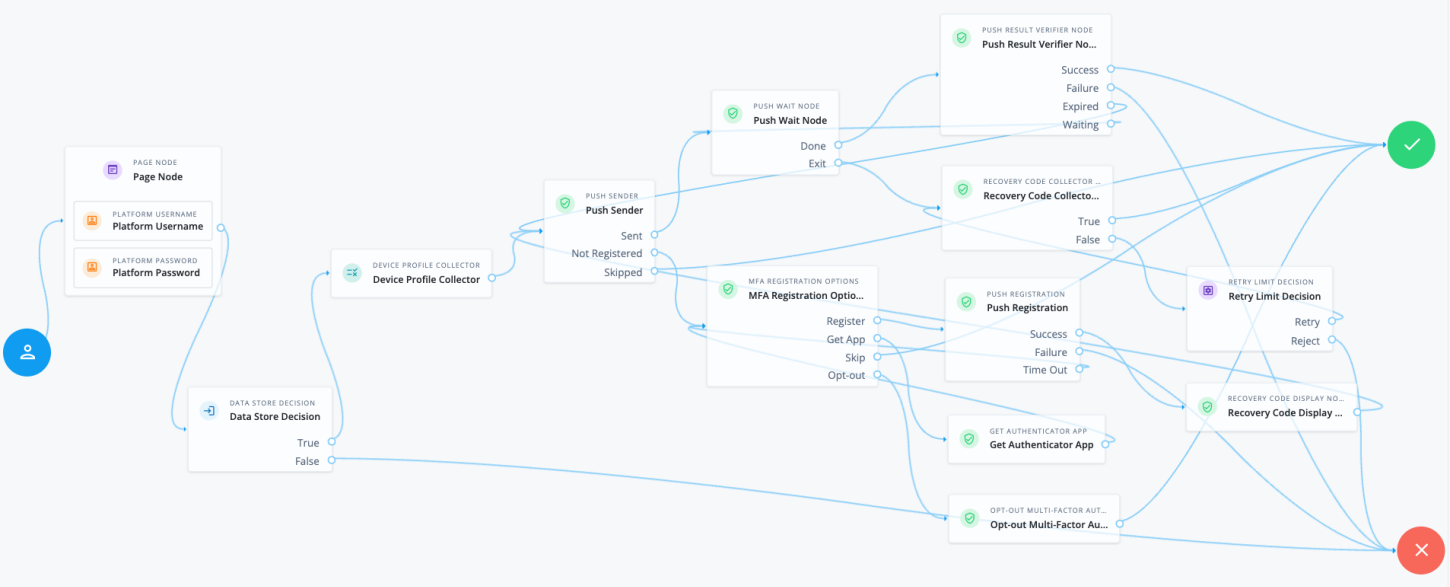
  The node can't start the registration process. Check that the Push Notification service is set up correctly in the realm.

- `Failed to save device to user profile`

  The node can't save the device details to the user's profile.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |

| Source node | Outcome path | Target node |
|---|---|---|
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ***Skip this step***

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ***Opt-out***

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

# Push Result Verifier node

The **Push Result Verifier** node validates the user's response to a previously sent push notification message.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Authenticators

The push-related nodes integrate with the ForgeRock Authenticator app for Android and iOS.

Third-party authenticator apps aren't compatible with the push notification functionality.

## Inputs

- This node requires the `realm` and `username` properties in the incoming node state.

  Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

- This node also requires the `pushMessageId` in the incoming state, which is a unique ID to identify the push notification request.

  Implement a Push Sender node earlier in the journey.

## Dependencies

You must configure the Push Notification service for the realm to use this node. Optionally, also configure the ForgeRock Authenticator (Push) service.

Find more information in the corresponding documentation for:

- Advanced Identity Cloud

- PingAM

Find information on provisioning the credentials used by the service in How To Configure Service Credentials (Push Auth, Docker) in Backstage.

## Configuration

This node has no configurable properties.

## Outputs

This node doesn't change the shared state.

## Outcomes

### `Success`

The user approved the push notification.

### `Failure`

The user rejected the push notification.

### `Expired`

A response wasn't received from the user's device within the time specified in the Push Sender node.

### `Waiting`

A response hasn't been received yet.

## Errors

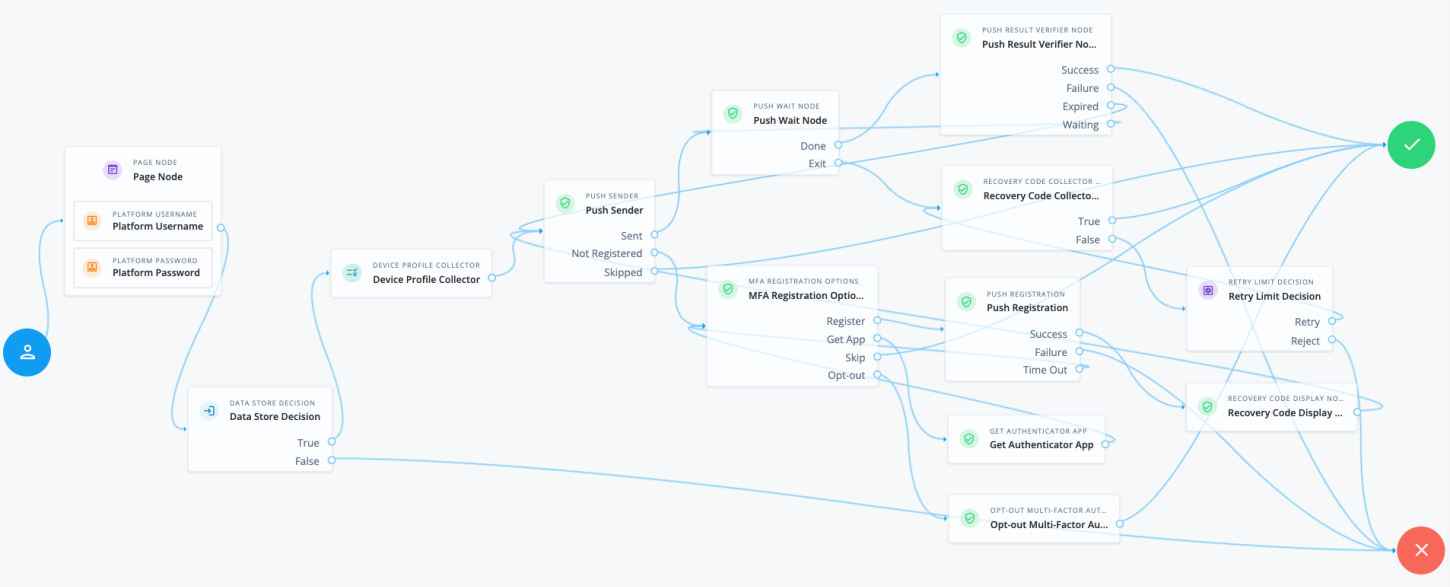
The node can log the following errors:

- `Unable to find push message ID`

  The node failed to read the `pushMessageId` from the shared state and can't verify the user's response. Make sure you have implemented a Push Sender node earlier in the journey.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |

| Source node | Outcome path | Target node |
|---|---|---|
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ***Skip this step***

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ***Opt-out***

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

# Push Sender node

The **Push Sender** node sends push notification messages to a device for multi-factor authentication (MFA).

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Authenticators

The push-related nodes integrate with the ForgeRock Authenticator app for Android and iOS.

Third-party authenticator apps aren't compatible with the push notification functionality.

## Inputs

- This node requires the `realm` and `username` properties in the incoming node state.

  Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

- This node can read the device location from the incoming node state if it exists.

  The device location only exists when a Device Profile Collector node is implemented earlier in the journey with the **Collect Device Location** option selected.

## Dependencies

You must configure the Push Notification service for the realm to use this node. Optionally, also configure the ForgeRock Authenticator (Push) service.

Find more information in the corresponding documentation for:

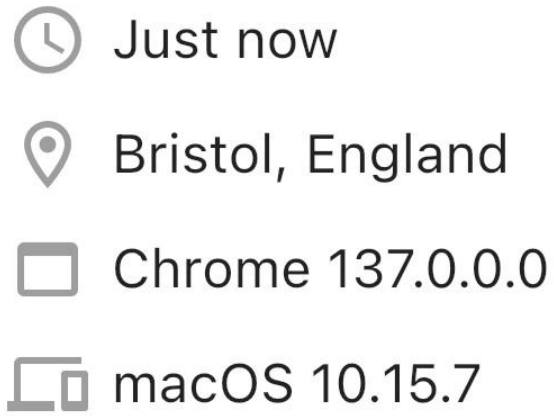
- Advanced Identity Cloud

- PingAM

Find information on provisioning the credentials used by the service in How To Configure Service Credentials (Push Auth, Docker) in Backstage.

## Configuration

| Property | Usage |
|---|---|
| **Message Timeout** | The number of milliseconds that the push notification message remains valid. The Push Result Verifier node rejects responses to push messages that have timed out.<br>Default: `120000` |
| **User Message** | (Optional) Add a custom, localized message to send to the user. You can use the following variables in the **Value** field:<br>**`{{user}}`**<br>This variable is replaced with the username value obtained from the shared state, for example, `bjensen`.<br>**`{{issuer}}`**<br>This variable is replaced with the issuer value read from the device profile metadata, which is stored in the `pushDeviceProfiles` attribute by default.<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.(1)<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.(2)<br>Default: `Login attempt from {{user}} at {{issuer}}` |
| **Remove 'skip' option** | Select this option to make push authentication mandatory. The `Skipped` outcome is not available when this option is selected.<br>When disabled, the user can skip push authentication if required. |

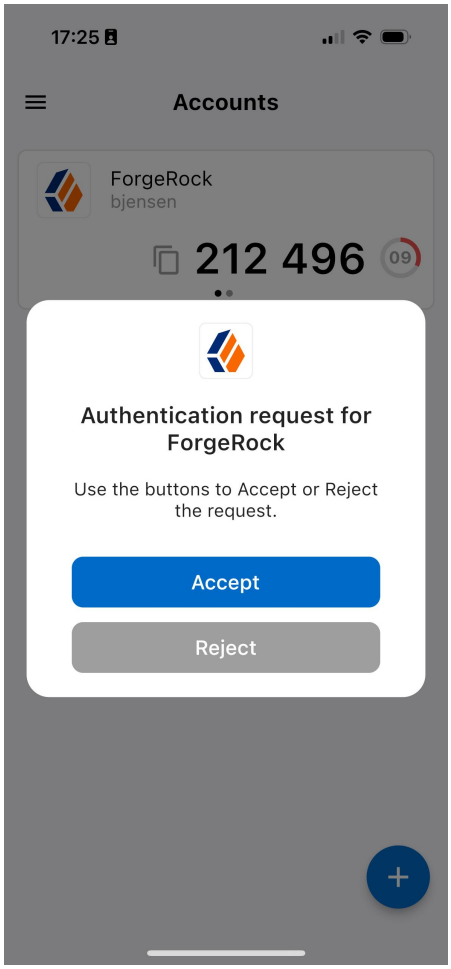
| Property | Usage |
| --- | --- |
| **Share Context Info** | Select this option to include context data such as `remoteIp`, `userAgent`, and `location` in the notification payload.<br><br>`{ "location": { "latitude": 51.454514, "longitude": -2.587910 }, "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36", "remoteIp": "9.9.9.9" }`<br><br>The ForgeRock Authenticator app displays this additional information to the user to help them verify that the request is genuine and initiated by them. For example:<br><br>  <br><br>To include the `location` attribute, the journey must include a Device Profile Collector node with the **Collect Device Location** option selected. |
| **Custom Payload Attributes** | (Optional) Enter the names of the shared state objects to include in the message payload sent to the client. Enter each name separately and press Enter to add it. The size of the payload mustn't exceed 3 Kb. |

**Push Type**

Select the type of push authentication the user must perform on their device to continue the journey.
Possible values are:

**`Tap to Accept` *(default)***

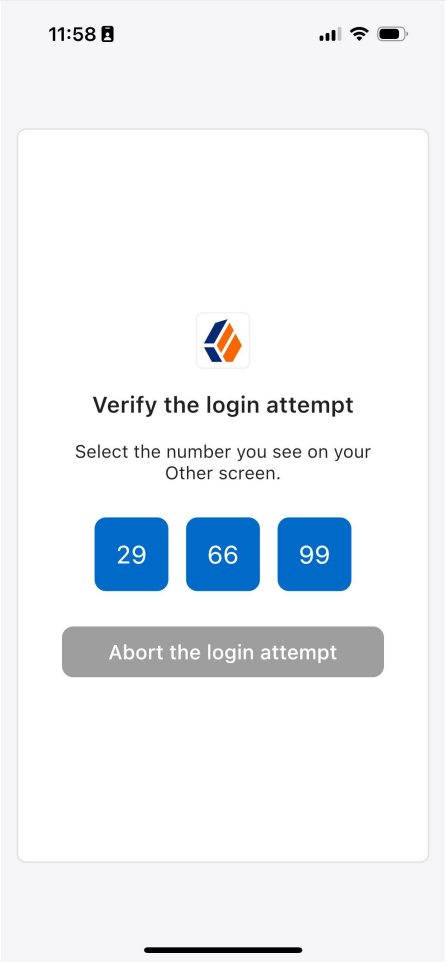
The user must tap `Accept` on their device to verify the request, or tap `Reject`.




> **Note**
> Research shows that users might accept a push authentication without checking it's legitimate. To reduce the chances of a user accepting a malicious push authentication attempt, consider using `Display Challenge Code` or `Use Biometrics to Accept` instead.

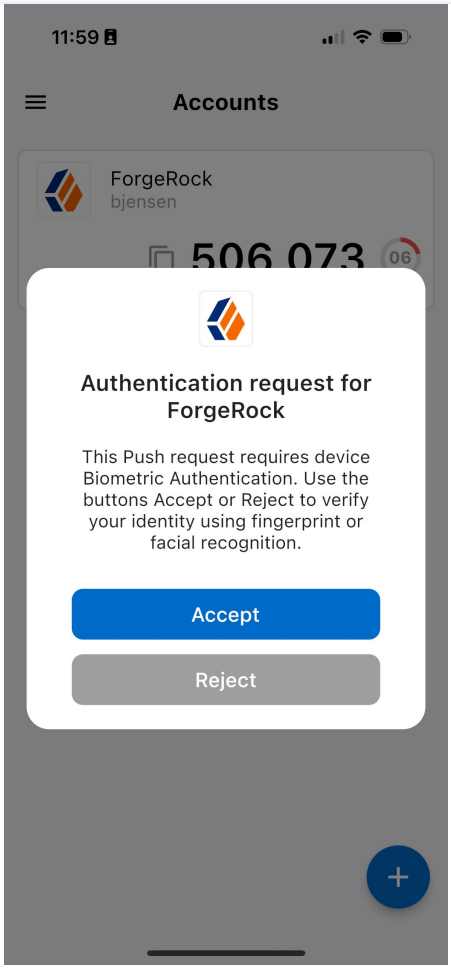
**`Display Challenge Code`**

The user must select one of three numbers displayed on their device. The number they select must match the code displayed in the browser to verify the request.




**`Use Biometrics to Accept`**

The user must tap `Accept` on their device and then authenticate using biometrics to verify the request.

| Property | Usage |
| --- | --- |
| |  |
| **Capture failure** | Select this option to store the failure reason in the `PushAuthFailureReason` shared state attribute when the node fails to send the push notification. The `Failure` outcome is only available when this option is selected. |

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

(2) PingAM only: Learn more about customizing and translating default messages in Internationalize nodes.

## Outputs

- The node adds a unique ID to identify the push notification request to the `pushMessageId` shared state attribute.
- If the outcome is `Not Registered`, this node sets `"mfaMethod": "push"` in the shared state.
- If the node fails to send the push notification and **Capture failure** is enabled, the node adds the failure reason to a property named `PushAuthFailureReason` in the shared state. Other nodes can read this property later in the journey, if required.

Possible failure reasons are:

- `MISSING_USERNAME`
- `SENDER_ALREADY_USED`
- `CTS_ERROR`
- `TRANSMISSION_FAILURE`

## Outcomes

### `Sent`

The push notification was sent successfully to the device.

### `Not Registered`

The user doesn't have a registered device.

### `Skipped`

The user chooses to skip push authentication.

### `Failure`

An error occurred during node execution.

## Errors

The node can log the following errors:

- `Failed to fetch identity`

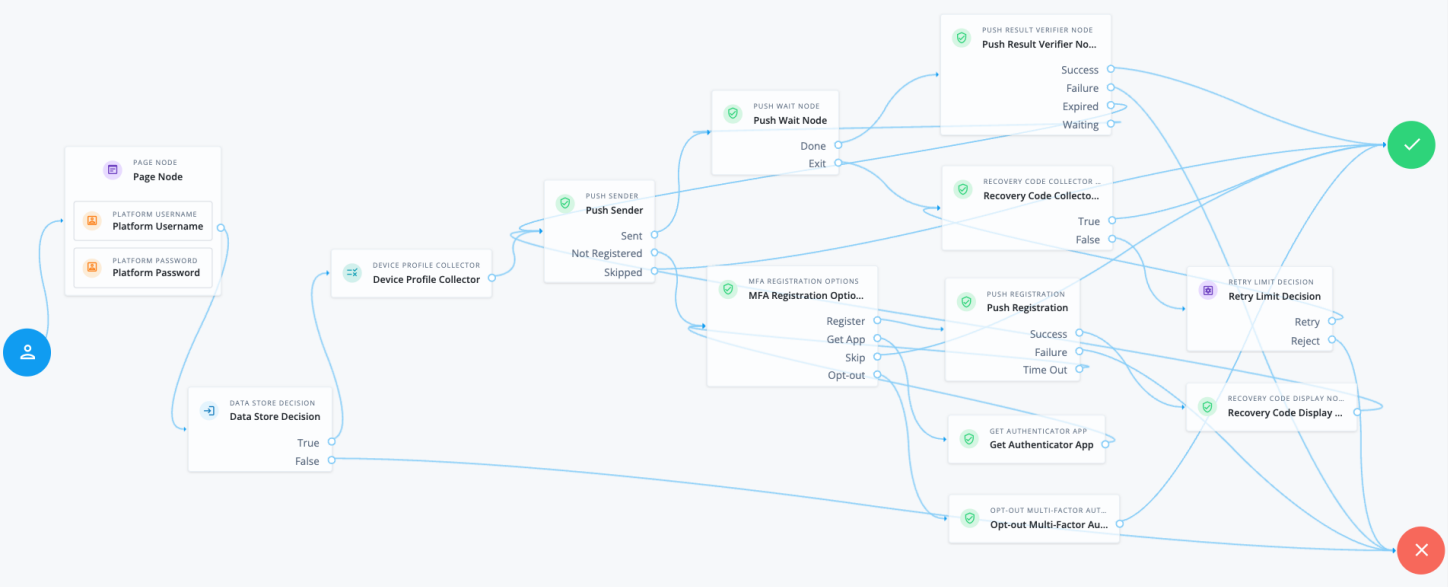
  The node can't identify the user from the shared state.

- `Payload data exceed maximum accepted size`

  The size of the message payload exceeds 3 Kb. Review any custom attributes you've included.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |

| Source node | Outcome path | Target node |
|---|---|---|
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ***Skip this step***

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ***Opt-out***

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

# Push Wait node

The **Push Wait** node pauses authentication for the specified number of seconds during the processing of a push authentication request.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Authenticators

The push-related nodes integrate with the ForgeRock Authenticator app for Android and iOS.

Third-party authenticator apps aren't compatible with the push notification functionality.

## Inputs

If the push type in the Push Sender node is set to `Display Challenge Code`, this node checks for the `pushNumberChallengeKey` in the incoming state. This value populates the `{{challenge}}` variable if it's included in the message.

## Dependencies

Precede this node in the flow with a Push Sender node to send the push authentication request.

## Configuration

| Property | Usage |
|---|---|
| **Seconds To Wait** | The number of seconds to pause authentication.<br>Default: `5` |

| Property | Usage |
| --- | --- |
| **Waiting Message** | (Optional) Add a custom, localized message to display to the user. You can use the `{{time}}` variable in the **Value** field to include the remaining seconds in the message:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Waiting for response…` |
| **Push Challenge Message** | (Optional) Add a custom, localized message to display to the user with the challenge code. You can use the `{{challenge}}` variable in the **Value** field to include the number challenge in the message.<br>This message is displayed only when the **Push Type** in the Push Sender node is set to `Display Challenge Code`:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Open your Authenticator app and tap the number shown to sign-in` |
| **Exit Message** | (Optional) Add custom, localized text to display on the button the user can click to exit the node before the waiting period has elapsed.<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done.<br>Leave blank to use the default message.[2]<br>Default: `Cancel` |

[1] Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

[2] PingAM only: Learn more about customizing and translating default messages in Internationalize nodes.

## Outputs

This node doesn't change the shared state.

## Outcomes

### Done

The waiting time has elapsed.

### `Exit`

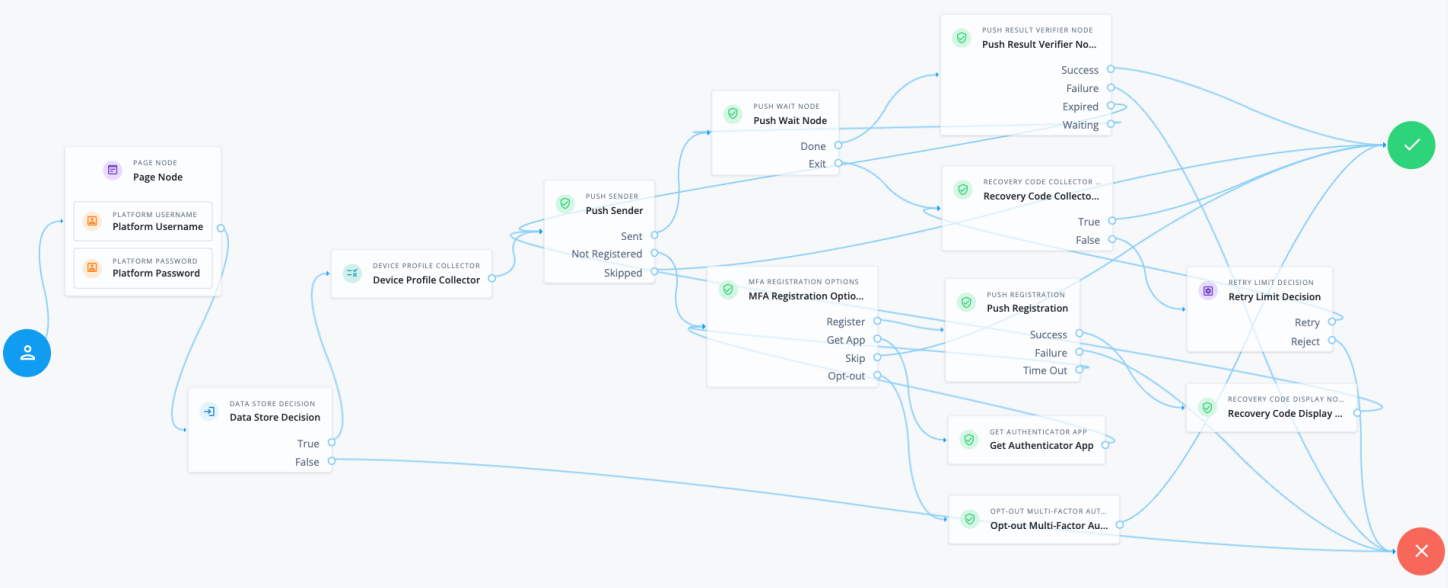
The user clicked the **Cancel** button to exit the node.

## Errors

This node doesn't log any error or warning messages of its own.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |

| Source node | Outcome path | Target node |
|---|---|---|
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |

| Source node | Outcome path | Target node |
|---|---|---|
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

   > **Tip**
   >
   > Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

   A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.
3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

*Skip this step*

Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

*Opt-out*

Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

## Recovery Code Collector Decision node

The **Recovery Code Collector Decision** node lets users authenticate with a recovery code that was generated when they registered a device for multi-factor authentication (MFA).

Use this node in an authentication journey that includes push notifications or one-time passwords. When the user loses their registered device, they can use a recovery code as an alternative method for authentication.

Find more information on viewing recovery codes when registering a device in the *ForgeRock Authenticator* documentation for:

- PingAM
- Advanced Identity Cloud

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the `realm` and `username` properties in the incoming node state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

### Dependencies

This node has no dependencies.

### Configuration

| Property | Usage |
| --- | --- |
| **Recovery Code Type** | Select the type of recovery code the user will submit for verification.<br>Default: `OATH` |

### Outputs

This node doesn't change the shared state.

### Outcomes

#### `True`

The user entered a valid recovery code.

#### `False`

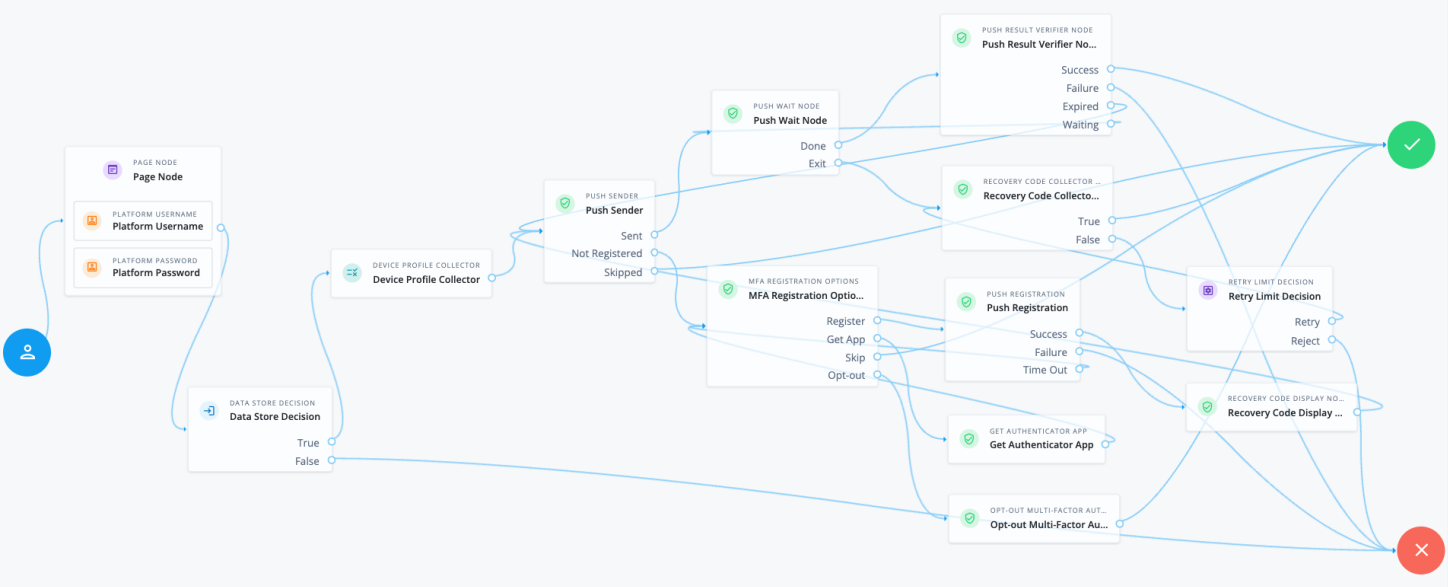
The user didn't enter a valid recovery code.

### Errors

This node doesn't log any error or warning messages of its own.

### Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |

| Source node | Outcome path | Target node |
|---|---|---|
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

> **Tip**
>
> Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.

3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ***Register Device***

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ***Get the App***

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

   ***Skip this step***

   Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

   ***Opt-out***

   Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

## Recovery Code Display node

The **Recovery Code Display** node retrieves the generated recovery codes from the transient state and displays them to the user, for safe-keeping. The codes can be used to authenticate if a registered device is lost or stolen.

> **Important**
>
> The generated recovery codes can't be retrieved from the user's profile because they're one-way encrypted.
> This node provides the *only* opportunity to view and save the recovery codes.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the `recoveryCodes` attribute in the incoming state so that it can display the recovery codes to the user. If there aren't any recovery codes in the transient state, evaluation continues along the only outcome path and nothing is displayed to the user.

Implement an OATH Registration node, a Push Registration node or a WebAuthn Registration node earlier in the journey, and connect the `Success` outcome path to this node to display the codes.

### Dependencies

This node has no dependencies.

### Configuration

This node has no configurable properties.

## Outputs

This node removes the recovery code details from the shared state.

## Outcomes

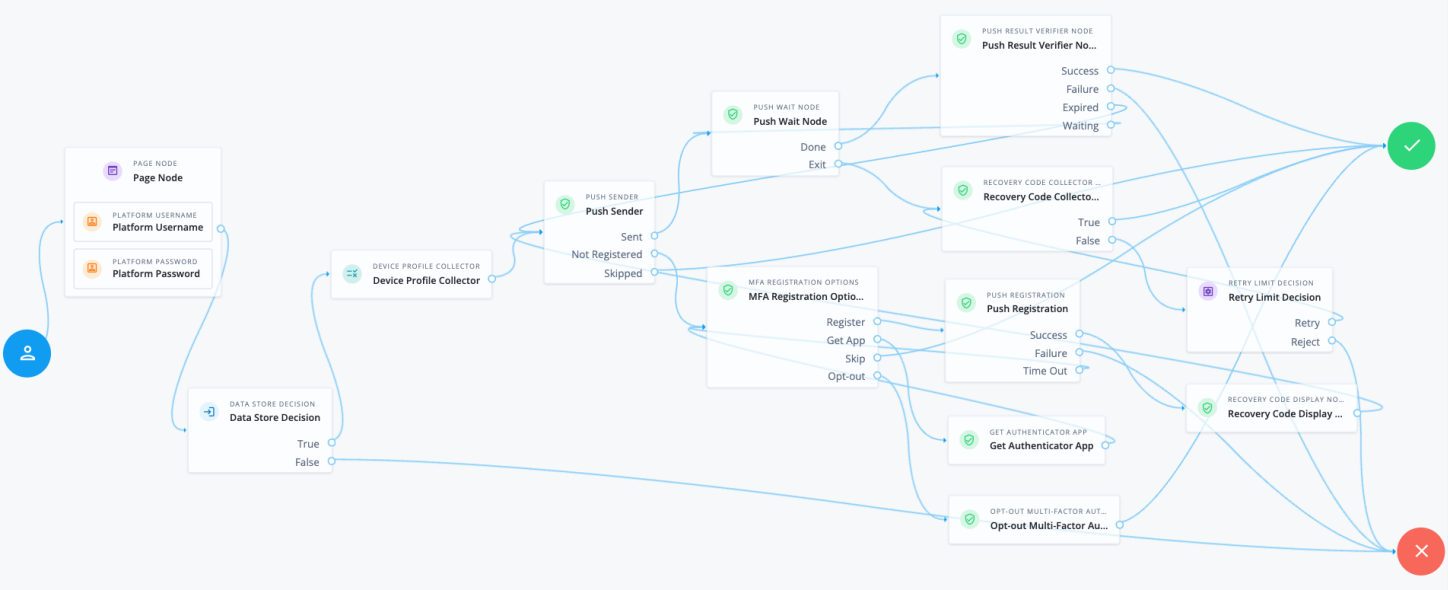
Single outcome path.

## Errors

This node doesn't log any error or warning messages of its own.

## Examples

The following example shows one possible implementation of multi-factor push authentication, which uses this node:




*List of node connections*

| Source node | Outcome path | Target node |
|---|---|---|
| Page Node containing nodes to collect credentials.<br>For standalone AM deployments, implement a Username Collector node and a Password Collector node.<br>For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node. | → | Data Store Decision |
| Data Store Decision | True | Device Profile Collector |

| Source node | Outcome path | Target node |
|---|---|---|
| | False | Failure |
| Device Profile Collector | → | Push Sender |
| Push Sender | Sent | Push Wait |
| | Not Registered | MFA Registration Options |
| | Skipped | Success |
| Push Wait | Done | Push Result Verifier |
| | Exit | Recovery Code Collector Decision |
| Push Result Verifier | Success | Success |
| | Failure | Failure |
| | Expired | Push Sender |
| | Waiting | Push Wait |
| MFA Registration Options | Register | Push Registration |
| | Get App | Get Authenticator App |
| | Skip | Success |
| | Opt-out | Opt-out Multi-Factor Authentication |
| Recovery Code Collector Decision | True | Success |
| | False | Retry Limit Decision |
| Push Registration | Success | Recovery Code Display Node |
| | Failure | Failure |
| | Time Out | MFA Registration Options |
| Get Authenticator App | → | MFA Registration Options |
| Opt-out Multi-Factor Authentication | → | Success |
| Retry Limit Decision | Retry | Recovery Code Collector Decision |
| | Reject | Failure |

| Source node | Outcome path | Target node |
| --- | --- | --- |
| Recovery Code Display Node | → | Push Sender |

After verifying the user's credentials, evaluation continues to the Device Profile Collector node to collect the device's location and then proceeds to the Push Sender node.

**If the user *has* a registered device:**

1. The Push Sender node sends a push notification to their registered device.
2. The Push Wait node pauses authentication for five seconds. During this time, the user can respond to the push notification on their device using the ForgeRock Authenticator app.

   If the user exits the Push Wait node, they're directed to the Recovery Code Collector Decision node, where they can enter a recovery code to authenticate.

   > **Tip**
   >
   > Configure the **Exit Message** property in the Push Wait node with a message, such as `Lost phone? Use a recovery code` for situations like this.

   A Retry Limit Decision node allows three attempts to enter a recovery code before failing the authentication.
3. The Push Result Verifier node verifies the user's response:
   - If the user responds positively, they're authenticated successfully and logged in.
   - If the user responds negatively, authentication fails.
   - If the push notification expires, the Push Sender node sends a new push notification.

     > **Tip**
     >
     > Use a Retry Limit Decision node to constrain the number of times a new code is sent.

   - If the user hasn't yet responded, the flow loops back a step and the Push Wait node pauses authentication for another 5 seconds.

**If the user *doesn't have* a registered device:**

1. The MFA Registration Options node presents the user with the following options:

   ### *Register Device*

   The flow continues to the Push Registration node, which displays a QR code for the user to scan with their authenticator app.

   ### *Get the App*

   Displayed only if the node is configured to display Get Authenticator App. The flow continues to the Get Authenticator App node, which displays links to download the authenticator app.

*Skip this step*

Displayed only if the node is configured to allow users to skip registration. In this example, skipping is linked to the `Success` outcome. However, you could provide an alternative authentication flow using an Inner Tree Evaluator node for example.

*Opt-out*

Displayed only if the node is configured to allow users to skip registration. Evaluation continues to the Opt-out Multi-Factor Authentication node, which updates the user's profile to skip MFA with push in the future. In this example, after updating the profile, the flow continues to the `Success` outcome.

2. The user registers the device with the Push Registration node.

After registration, the Recovery Code Display node displays the recovery codes to the user and the flow returns to the Push Sender node to continue push authentication.

> **Note**
>
> To manage push devices, the user must log in using either the device or a recovery code.
> Find more information in the MFA documentation for:
>
> - PingAM
> - Advanced Identity Cloud

## TypingDNA Decision node

The TypingDNA Decision node handles the authentication logic by communicating with the TypingDNA Authentication API. To perform this, the TypingDNA Decision node uses the API key and API secret from the TypingDNA user account dashboard.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads `TDNA_TEXT_TO_ENTER`, `TDNA_DEVICE_TYPE`, `TDNA_TEXT_ID`, and `TDNA_TYPING_PATTERN` parameters from the shared state.

## Dependencies

Before using the TypingDNA nodes, you must set up PingOne Advanced Identity Cloud integration with TypingDNA as described in the Step-by-step: TypingDNA as a 2FA factor in Ping. This node also requires that the `TypingDNA Recorder` node be configured earlier in the journey.

## Properties

| Property | Usage |
|---|---|
| API URL | The URL for TypingDNA API, for example, https://api.typingdna.com |
| API key | The API key from your TypingDNA account |
| API secret | The API secret from your TypingDNA account |
| Retries | The number of times a user is allowed to retry authentication if it fails.<br>Default: `0`, meaning no retries are allowed. |
| Authentication API Configuration | Two options are available:<br>• `Basic`: Uses the default settings of Authentication API for auto-enroll, minimum number of enrollments, and thresholds for auto-enroll and verification. All requests will use the `/auto` endpoint that is free for all types of Authentication API clients.<br>• `Advanced`: Uses the `/verify` endpoint. The authentication behavior can be managed by using the **API Settings** menu in the **TypingDNA Dashboard for Authentication API**. Don't use the `Advanced` configuration with the free starter account. |
| Hash algorithm | The hash-algorithm used to anonymize user IDs before sending them to the TypingDNA Authentication API. |
| Salt | A string that is used to anonymize the user ID for additional security. For example, username or user email.<br>Default: null. |
| Request identifier | An optional parameter that may be used to identify requests coming from a specific PingOne Advanced Identity Cloud authentication tree. The identifier also appears in the TypingDNA logs.<br>Default: ForgeRock. |
| Request time out | Time in milliseconds after which each request to the TypingDNA Authentication API times out if no response was received.<br>Default: 8000 ms. |

## Outputs

This node doesn't store any output in the shared state.

## Outcomes

### `Enroll`

This occurs if the user's number of saved patterns was lower than the number of enrollments. The newly presented typing pattern will be saved to the profile. In this case, no authentication is actually performed.

For passive enrollment, you need to continue the flow to an alternative authentication node or to success. For active enrollment, you need to link this outcome back to the page node, such as the login page or the short phrase page, where the typing patterns are collected.

### `Initial enrollment complete`

This occurs when the user's number of saved patterns is equal to that needed for enrollment. The minimum number of patterns for initial enrollment can be configured from the **API Settings** menu in the TypingDNA Dashboard for Authentication API.

The **API Settings** menu is available only for paid Authentication API plans.

### `Retry`

This occurs if the authentication fails and the number of times the user has retried is lower than the **Retries** property value configured. The authentication can fail because the `Match` threshold hasn't been reached or because a non-critical error, which could be overcome by trying again, has occurred. To effectively allow retry effectively, link this outcome back to the page node that collects typing patterns.

### `Fail`

This occurs when a critical error occurs, such as if invalid API credentials are entered. This outcome should be linked to an alternative authentication node or `Failure`.

### `Match`

This occurs when the authentication is successful. For this, the net score of the authentication must exceed the Match threshold. This outcome is usually linked to `Success`.

### `No match`

This occurs if the authentication fails and the allowed number of retries has reached. This outcome would be linked to an alternative authentication node.

## Example

Sample journey using this node.

### Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

## TypingDNA Recorder node

This node collects typing behaviors and transforms them into typing patterns.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node does not read any input from the shared state.

### Dependencies

Before using the TypingDNA nodes, you must set up PingOne Advanced Identity Cloud integration with TypingDNA as described in Step-by-step: TypingDNA as a 2FA factor in Ping.

### Properties

| Property | Usage |
|---|---|
| Display Messages | Specify if the node should display messages for retry and enroll. |
| Show Visualizer | A visual representation of how the user types. |
| Disable copy and paste | Prevent the end user from copying and pasting the short phrase, to enforce typing analysis. |
| Recorder Target IDs | The identifiers of the typing patterns recorded by the TypingDNA recorder. |
| Submit button ID | When the user clicks the button, it triggers the TypingDNA Recorder node to record the typing pattern of the user. When the user submits the credentials, the recorded pattern is sent to the backend. |

### Outcomes

Single outcome path.

### Outputs

This node stores the following properties in the shared state:

- `TDNA_TEXT_TO_ENTER`: The short phrase the end user was prompted to type.
- `TDNA_DEVICE_TYPE`: The type of device from which the end user initiated the request
- `TDNA_TEXT_ID`: The identifier of the typing pattern recorded.
- `TDNA_TYPING_PATTERN`: The recorded typing pattern.

### Example

Sample journey using this node.

### Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

## TypingDNA Reset Profile node

Use the **TypingDNA Reset Profile** node to reset an end user's TypingDNA profile. When a user's profile is reset, all the patterns recorded for the user are deleted, and the user is enrolled again the next time they sign on. To reset a profile, you must use the same API key, API secret, and the salt as configured in the TypingDNA Decision node.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads the `username` of the authenticating user from the shared state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

## Dependencies

The dependencies for TypingDNA nodes are explained here.

## Properties

| Property | Usage |
|---|---|
| **API URL** | The URL for TypingDNA API, for example, https://api.typingdna.com. |
| **API key** | The API key from your TypingDNA account. |
| **API secret** | The API secret from your TypingDNA account. |
| **Hash algorithm** | The hashing algorithm that is used for anonymizing usernames before sending them to the TypingDNA Authentication API. |
| **Salt** | A string that is used to anonymize the user ID for additional security. For example, username or user email.<br>Default is null. |
| **Request identifier** | An optional parameter that can be used to identify requests coming from a specific PingOne Advanced Identity Cloud authentication journey. The identifier also appears in the TypingDNA logs.<br>Default: ForgeRock. |
| **Request time out** | Time in milliseconds after which requests to the TypingDNA Authentication API time out if no response is received.<br>Default: 8000 ms |

## Outputs

This node does not store any output to the shared state.

## Outcomes

### Success

This outcome is achieved when the profile was successfully deleted.

### `Error`

This outcome is achieved when a critical error occurs, such as if invalid API credentials are entered. You can link this outcome to an alternative node, such as a `Retry Limit Decision node`, or to the Failure node.

## Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

# TypingDNA Short Phrase Collector node

This node provides an alternative authentication method based on typing biometrics. End users are requested to type a short phrase so that their typing patterns are collected and verified.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node doesn't read any input from the shared state.

## Dependencies

Before using the TypingDNA nodes, you must set up Advanced Identity Cloud integration with TypingDNA as described in Step-by-step: TypingDNA as a 2FA factor in Ping.

## Configuration

| Property | Usage |
|---|---|
| **Text to enter** | The short phrase the end user must type to record their keystrokes. |

## Outputs

This node does not store any output in the shared state.

## Outcomes

### `Next`

Single outcome path.

## Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

# WebAuthn Authentication node

The **WebAuthn Authentication** node lets users on supported clients authenticate using a registered FIDO device.

There are many similarities between WebAuthn and device binding and JWS verification. We provide authentication nodes to implement both technologies in your journeys.

Both can be used for usernameless and passwordless authentication, they both use public key cryptography, and both can be used as part of a multi-factor authentication journey.

One major difference is that with device binding, the private key never leaves the device.

With WebAuthn, there is a possibility that the private key is synchronized across client devices because of Passkey support, which may be undesirable for your organization.

For details of the differences, refer to the following table:

*Comparison of WebAuthn and Device Binding/JWS Verification*

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Industry-standards based | ☑ | × | You can refer to the WebAuthn W3C specification. Device binding and JWS verification are proprietary implementations. |
| Public key cryptography | ☑ | ☑ | Both methods use Public key cryptography. |
| Usernameless support | ☑ | ☑ | After registration, the username can be stored in the device and obtained during authentication without the user having to enter their credentials. |
| Keys are bound to the device | × | ☑ | With WebAuthn, if Passkeys are used, they can be shared across devices. With device binding, the private keys do not leave the device. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Sign custom data | ✕ | ☑ | With device binding, you can:<br>• Customize the challenge that the device must sign. For example, you could include details of a transaction, such as the amount in dollars.<br>• Add custom claims to the payload when signing a challenge. This gives additional context that the server can make use of by using a scripted node. Refer to Add custom claims when signing. |
| Format of signed data | WebAuthn authenticator data | JSON Web Signature (JWS) | |
| Integration | ✕ | ☑ | With device binding, after verification, the signed JWT is available in:<br>• Audit Logs<br>• Transient node state<br>This enables the data within to be used for integration into your processes and business logic. |
| Platform support | ☑ Android<br>☑ iOS<br>☑ Web browsers | ☑ Android<br>☑ iOS<br>✕ Web browsers | As it is challenging to store secure data in a browser as a client app, device binding is not supported in web browsers. |
| Authenticator support | Determined by the platform.<br>Configuration limited to:<br>• Biometric with Fallback to Device Pin | Determined by the authentication node.<br>Full configuration options:<br>• Biometric Authentication<br>• Biometric with Fallback to Device Pin<br>• Application Pin<br>• Silent | With device binding, you can specify what authentication action the user must perform to get access to the private keys.<br>This provides greater flexibility in your security implementation and can reduce authentication friction for your users. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Key storage | Web browsers and iOS synchronize to the cloud.<br>Android has the option to synchronize to the cloud. | ***Android***<br>KeyStore<br>***iOS***<br>Secure enclave: hardware-backed and not synchronized to the cloud. | Both technologies store the private keys securely on the client.<br>WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>This can reduce authentication friction for your users but may also increase the risk of a breach. |
| Managing device keys | Managed by the device OS.<br>Apps cannot delete *local* client keys programmatically and do not have a reference to the *remote* server key for deletion. | Managed by the Ping SDKs.<br>Provides an interface to delete local client and remote server keys. | The ability to programmatically delete both client and server keys can greatly simplify the process of registering a new device if an old device is lost or stolen. |
| Passkey support | ☑ | × | WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>Device binding keeps the private key locked in the device. |
| App integrity verification | ***Android***<br>Requires an `assetlinks.json` file.<br>***iOS***<br>Requires `apple-app-site-association` file. | Not provided by the device binding or verification nodes.<br>It can be added as part of the journey by using app integrity nodes. | App integrity verification helps ensure your users are only using a supported app rather than a third-party or potentially malicious version. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Key attestation | ***Android***<br>SafetyNet<br>***iOS***<br>None | ***Android***<br>Uses hardware-backed key pairs with Key Attestation.<br>***iOS***<br>It can be added as part of the journey by using app integrity nodes to support key attestation. | Key attestation verifies that the private key is valid and correct, is not forged, and was not created in an insecure manner. |
| Complexity | Medium | Low | WebAuthn requires a bit more configuration, for example, creating and uploading the `assetlinks.json` and `apple-app-site-association` files. Device binding only requires the journey and the SDK built into your app. |

**Important**

To authenticate using WebAuthn on an Android or iOS device without an external keystore, screen lock *must* be enabled. Find more information in the respective device documentation for Android and iOS.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

- This node requires a `username` in the incoming node state to assess whether the user has a registered device.

  Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

- Optionally, this node can read the contents of the `webAuthnExtensions` shared state property as input.

You can populate the `webAuthnExtensions` property with any JSON object you choose using a script or in a node that occurs earlier in the journey. If this property is populated, its contents are placed in the `extensions` entry passed to the browser or authenticator. If this property is empty, an empty JSON object is passed to the browser or authenticator.

You can find more information on WebAuthn extensions in WebAuthn Extensions.

## Dependencies

For successful authentication, this node depends on:

- A client that supports web authentication
- A registered FIDO device

## Configuration

| Property | Usage |
|---|---|
| **Relying party identifier** | The domain used as the relying party identifier during web authentication. This is the domain against which to register the device. If you leave this field blank, it defaults to:<br>• PingAM: The domain name of the AM instance, for example, `am.example.com`.<br>Specify an alternative domain if your AM instances are behind a load balancer, for example.<br>• Advanced Identity Cloud: The domain name of your tenant, for example, `openam-example.forgeblocks.com`. |
| **Origin domains** | A list of fully qualified URLs to accept as the origin of the incoming request.<br>If this field is empty, the accepted origin is the incoming request origin. |
| **User verification requirement** | The required user verification level.<br>The available options are:<br>**`REQUIRED`**<br>The authenticator used must verify the user's identity, for example, by using biometrics. Authenticators that don't verify the user's identity are filtered out and can't be selected by the user.<br>**`PREFERRED`**<br>If multiple authenticators are presented, AM prefers those that verify the user's identity. If none are available, AM accepts any authenticator.<br>**`DISCOURAGED`**<br>AM doesn't require an authenticator that verifies the user's identity. Authenticators that don't verify the user's identity are preferred. |

| Property | Usage |
| --- | --- |
| **Allow recovery codes** | If you select this option, AM lets the user enter a recovery code instead of performing an authentication gesture.<br>Enabling this options adds a `Recovery Code` outcome path to the node. The outcome path should lead to a Recovery Code Collector Decision node to collect and verify the recovery code. |
| **Timeout** | The number of seconds to wait for a valid WebAuthn authenticator to be registered before failing.<br>If the specified timeout is reached, evaluation continues along the `Client error` outcome path. AM stores a message in the `WebAuthenticationDOMException` property of the shared state. |
| **Username from device** | Specifies whether AM should get the username from the device.<br>If you enable this option and the device is unable to store or provide usernames, the node fails and evaluation continues along the `Failure` path.<br>You can find information on using this property for usernameless authentication in the documentation for:<br>• PingAM<br>• Advanced Identity Cloud |
| **Return challenge as JavaScript** | Choose how the node should return its challenge for consumption by your front-end user interface:<br>***PingOne Advanced Identity Cloud hosted pages or Ping SDK client apps:***<br>Deselect this option when using either hosted pages in PingOne Advanced Identity Cloud or a client application built using the Ping SDKs as your authentication user interface.<br>When not enabled the node returns the challenge and associated data in a metadata callback.<br>PingOne Advanced Identity Cloud hosted pages and Ping SDK client apps that might not be able to execute JavaScript use the information from the callback to interact with WebAuthn APIs on AM's behalf.<br>***PingAM User UI:***<br>You should only enable this option if you are using the PingAM User UI to authenticate users.<br>Enabling this option causes the node to return its challenge as a fully encapsulated client-side JavaScript that interacts directly with the WebAuthn API. |

| Property | Usage |
| --- | --- |
| **Detect sign count mismatch** | The sign count is useful for detecting potential cloned devices. It's stored in the WebAuthn device profile as `signCount`.<br>If you enable this option, the node compares the authenticator's sign count (signature counter) with the sign count stored in the user's profile.<br>If the authenticator sign count is less than or equal to the stored value, evaluation continues to the Sign Count Mismatch outcome. |

## Outcomes

### `Unsupported`

If the user's client doesn't support web authentication, evaluation continues along the `Unsupported` outcome path. For example, clients connected over the HTTP protocol rather than HTTPS don't support WebAuthn; however, HTTPS may not be required when testing locally on `http://localhost`. Learn more in Is origin potentially trustworthy?.

### `No Device Registered`

If the user doesn't have a registered device, evaluation continues along the `No Device Registered` outcome path.

### `Success`

If the user successfully authenticates with a device of the type determined by the **User verification requirement** property, evaluation continues along the `Success` outcome path.

### `Failure`

If the node encounters an issue when attempting to authenticate the user with the device, evaluation continues along the `Failure` outcome path; for example, if the node can't verify that the response from the authenticator was appropriate for the specific instance of the authentication journey.

### `Client Error`

If the user's client encounters an issue when attempting to authenticate using the device, for example, if the timeout was reached, evaluation continues along the `Client Error` outcome path.

The journey takes this path whenever the client throws a `DOMException`, as required by the Web Authentication: An API for accessing Public Key Credentials Level 1 specification.

### `Recovery Code`

If **Allow recovery code** is enabled, the node gives the user an option to enter a recovery code rather than authenticate using a device. If the user enters a recovery code, evaluation continues along the `Recovery Code` outcome path.

This outcome path must lead to a Recovery Code Collector Decision node to let AM accept and verify the recovery code.

### `Sign Count Mismatch`

If **Detect sign count mismatch** is enabled and the authenticator sign count is less than or equal to the value stored in the user's profile, evaluation continues to the **Sign Count Mismatch** outcome.

This outcome can help to detect cloned or malfunctioning authenticators. A sign count mismatch doesn't necessarily indicate that the operation was performed by a cloned authenticator. Address this situation appropriately for your deployment, for example, by using a scripted decision node.

> **Note**
>
> The node itself is considered to have succeeded even if evaluation leads to this outcome. Therefore, if it suits your deployment, you can design your journey to continue through this outcome.

## Outputs

- If a client error occurs, the node adds the error type and description to a property named `WebAuthenticationDOMException` in the shared state. Other nodes can read this property later in the journey, if required.
- On successful authentication, the node adds the UUID of the device ( `webauthnDeviceUuid` ) and the name of the device ( `webauthnDeviceName` ) to the shared state. [1]

  This lets you track the use of biometric authentication and the device used to authenticate.
- On successful authentication, the node adds a `webauthnAssertionInfo` object to the transient state. [1]

  The `webauthnAssertionInfo` object includes the following information:

```
{
  "authenticatorAttachment": "platform",
  "flags": {
    "UP": true,
    "UV": true,
    "ED": false,
    "AT": false,
    "BE": true,
    "BS": true
  }
}
```

  The `authenticatorAttachment` field is only populated when the UI or client application supports sending it. This field lets the journey identify whether the device used to authenticate was a roaming (cross-platform) device or a client-bound (platform) device. Learn more in Authenticator Attachment Modality in the WebAuthn specification.

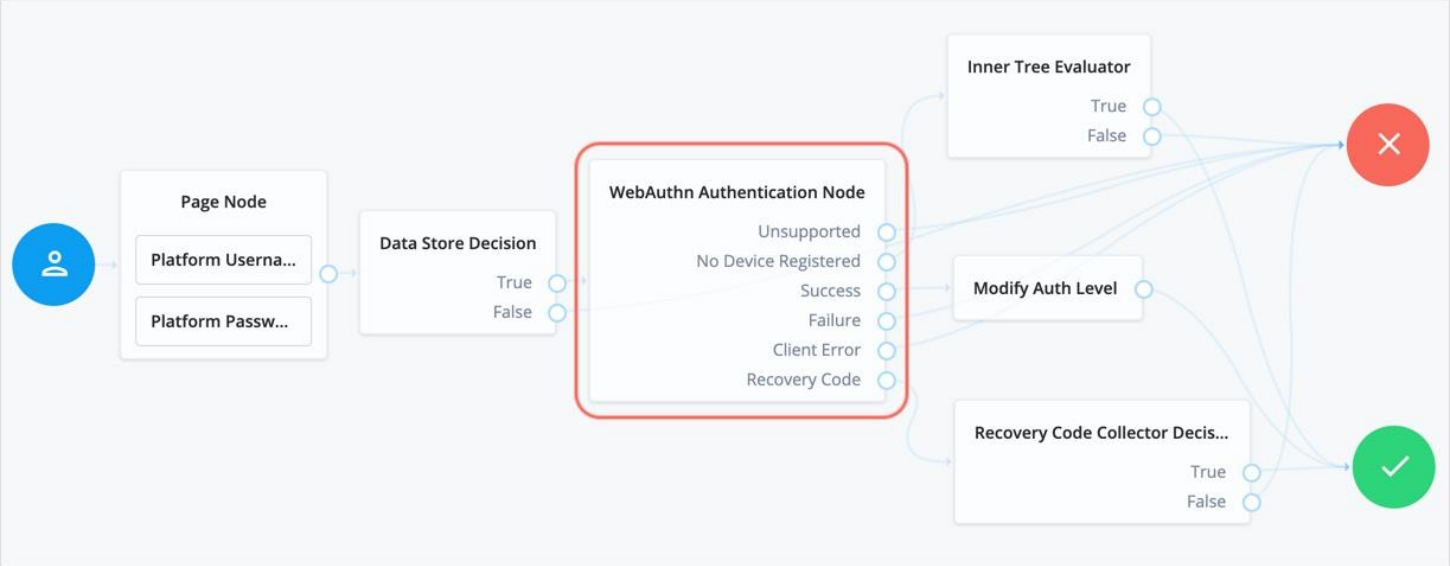
  The `flags` provide additional information about the authenticator. Learn more about these flags in Authenticator Data in the WebAuthn specification.

> **Note**
>
> - With the exception of the `webauthnAssertionInfo` object, the contents of the transient state for this node aren't public. Don't rely on them in your scripts.

[1] Currently available only in the rapid release channel.

## Example

This example shows one possible implementation of the flow for authenticating with WebAuthn devices:

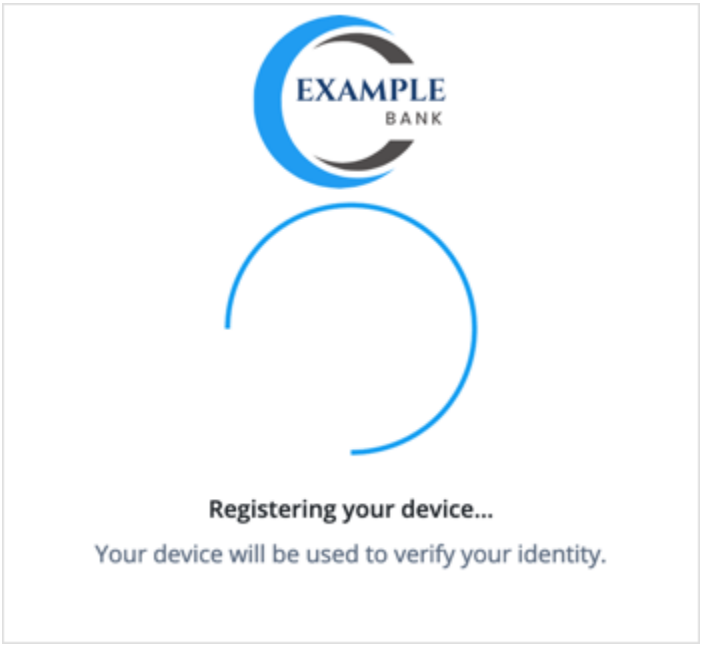



After verifying the users credentials against the configured data store, evaluation continues to the WebAuthn Authentication node.

If the user's client doesn't support WebAuthn, authentication fails and the user doesn't get a session. A more user-friendly approach would be to set a success URL to redirect the user to a page explaining the benefits of multi-factor authentication, and then proceeding to the `Success` node.

If there are no registered WebAuthn devices present in the user's profile, the failure URL is set, pointing to a flow that lets the user register a device. This stage could also be an Inner Tree Evaluator node.

If the user's client supports WebAuthn, and the connection is secured with TLS, the user is prompted to complete an authorization gesture, for example, scanning a fingerprint, or entering a PIN:




The user's browser may present a consent pop-up to allow access to the authenticators available on the client. When consent has been granted, the browser activates the relevant authenticators, ready for authentication.

> **Tip**
>
> The relying party details configured in the node are often included in the consent message to help the user verify the entity requesting access.

The authenticators the client activates for authentication depend on the value of the properties in the node. For example, if the **User verification requirement** property is set to `REQUIRED`, the client **SHOULD** only activate authenticators that verify the identity of the user.

For extra protection, AM **WILL** verify that the response from an authenticator matches the criteria configured for the node, and will reject an authentication attempt by an inappropriate authenticator type by routing it to the `Failure` outcome.

When the user completes an authorization gesture, for example, by scanning a fingerprint or entering a PIN, evaluation continues along the `Success` outcome path. In this example, their authentication level is increased by ten to signify the stronger authentication that has occurred, and the user is taken to their profile page.

If the user clicks the `Use Recovery Code` button, evaluation continues to the Recovery Code Collector Decision node, ready to accept the recovery code. If verified, the user is taken to their profile page.

Any problems encountered during authentication lead to the `Failure` outcome, including a timeout, or to the `Client Error` outcome, resulting in an authentication failure.

## WebAuthn Device Storage node

Writes information about FIDO2 devices to a user's profile. The user can subsequently authenticate using the device.

Use this node to store the device data the WebAuthn Registration node places into the transient node state when its **Store device data in transient state** property is enabled.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `Success`
- `Failure`
- `Exceed Device Limit`

If AM encounters an issue when attempting to save the device data to the user's profile; for example, the user was not identified earlier, then evaluation continues along the `Failure` outcome path.

If the **Maximum Saved Devices** property is set to an integer greater than zero, and registering a new device would take the number of devices above the specified threshold, then evaluation continues down the `Exceed Device Limit` outcome path. In this case, you may need to instruct your users to log in with an existing device in order to remove one or more of their registered devices.

If the node successfully stores the device data to the user's profile, evaluation continues along the `Success` outcome path.

### Properties

| Property | Usage |
|---|---|
| Generate recovery codes | Specify whether WebAuthn device recovery codes should be generated.<br>If enabled, recovery codes are generated and stored in the transient node state, and stored alongside the device profile.<br>Use the Recovery Code Display node to display the codes to the user for safe keeping.<br>**Important**<br>Generating recovery codes overwrites all existing WebAuthn device recovery codes for the device.<br>Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen. |
| Maximum Saved Devices | Specify the maximum number of WebAuthn devices to save in a user's profile.<br>Set this property to `0` if you do not want to limit the number of devices.<br>When this property is greater than zero, the `Exceed Device Limit` outcome path becomes available. |

## WebAuthn Registration node

The **WebAuthn Registration** node lets users of supported clients register FIDO2 devices for use during authentication.

There are many similarities between WebAuthn and device binding and JWS verification. We provide authentication nodes to implement both technologies in your journeys.

Both can be used for usernameless and passwordless authentication, they both use public key cryptography, and both can be used as part of a multi-factor authentication journey.

One major difference is that with device binding, the private key never leaves the device.

With WebAuthn, there is a possibility that the private key is synchronized across client devices because of Passkey support, which may be undesirable for your organization.

For details of the differences, refer to the following table:

*Comparison of WebAuthn and Device Binding/JWS Verification*

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Industry-standards based | ☑ | × | You can refer to the WebAuthn W3C specification.<br>Device binding and JWS verification are proprietary implementations. |
| Public key cryptography | ☑ | ☑ | Both methods use Public key cryptography. |
| Usernameless support | ☑ | ☑ | After registration, the username can be stored in the device and obtained during authentication without the user having to enter their credentials. |
| Keys are bound to the device | × | ☑ | With WebAuthn, if Passkeys are used, they can be shared across devices.<br>With device binding, the private keys do not leave the device. |
| Sign custom data | × | ☑ | With device binding, you can:<br>• Customize the challenge that the device must sign. For example, you could include details of a transaction, such as the amount in dollars.<br>• Add custom claims to the payload when signing a challenge. This gives additional context that the server can make use of by using a scripted node. Refer to Add custom claims when signing. |
| Format of signed data | WebAuthn authenticator data | JSON Web Signature (JWS) | |
| Integration | × | ☑ | With device binding, after verification, the signed JWT is available in:<br>• Audit Logs<br>• Transient node state<br>This enables the data within to be used for integration into your processes and business logic. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| Platform support | ☑ Android<br>☑ iOS<br>☑ Web browsers | ☑ Android<br>☑ iOS<br>✕ Web browsers | As it is challenging to store secure data in a browser as a client app, device binding is not supported in web browsers. |
| Authenticator support | Determined by the platform.<br>Configuration limited to:<br>• Biometric with Fallback to Device Pin | Determined by the authentication node.<br>Full configuration options:<br>• Biometric Authentication<br>• Biometric with Fallback to Device Pin<br>• Application Pin<br>• Silent | With device binding, you can specify what authentication action the user must perform to get access to the private keys.<br>This provides greater flexibility in your security implementation and can reduce authentication friction for your users. |
| Key storage | Web browsers and iOS synchronize to the cloud.<br>Android has the option to synchronize to the cloud. | ***Android***<br>KeyStore<br>***iOS***<br>Secure enclave: hardware-backed and not synchronized to the cloud. | Both technologies store the private keys securely on the client.<br>WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>This can reduce authentication friction for your users but may also increase the risk of a breach. |
| Managing device keys | Managed by the device OS.<br>Apps cannot delete *local* client keys programmatically and do not have a reference to the *remote* server key for deletion. | Managed by the Ping SDKs.<br>Provides an interface to delete local client and remote server keys. | The ability to programmatically delete both client and server keys can greatly simplify the process of registering a new device if an old device is lost or stolen. |
| Passkey support | ☑ | ✕ | WebAuthn supports synchronizing the private keys to the cloud for use on other devices.<br>Device binding keeps the private key locked in the device. |

| Feature | WebAuthn / FIDO | Device Binding / JWS Verifier | Details |
|---|---|---|---|
| App integrity verification | ***Android***<br>Requires an `assetlinks.json` file.<br>***iOS***<br>Requires `apple-app-site-association` file. | Not provided by the device binding or verification nodes.<br>It can be added as part of the journey by using app integrity nodes. | App integrity verification helps ensure your users are only using a supported app rather than a third-party or potentially malicious version. |
| Key attestation | ***Android***<br>SafetyNet<br>***iOS***<br>None | ***Android***<br>Uses hardware-backed key pairs with Key Attestation.<br>***iOS***<br>It can be added as part of the journey by using app integrity nodes to support key attestation. | Key attestation verifies that the private key is valid and correct, is not forged, and was not created in an insecure manner. |
| Complexity | Medium | Low | WebAuthn requires a bit more configuration, for example, creating and uploading the `assetlinks.json` and `apple-app-site-association` files. Device binding only requires the journey and the SDK built into your app. |

AM interacts with FIDO2/WebAuthn capable browsers, such as `Chrome`, `Firefox` and `Microsoft Edge`.

These browsers interact with the following:

- Client to Authenticator Protocol 2 (CTAP2) authenticators, including Universal 2nd Factor (U2F) and FIDO2 Security Keys
- Platforms such as Windows Hello and Apple Touch ID

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |

| Product | Available? |
| --- | --- |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

- This node requires the `username` and `password` properties in the incoming node state.

  For standalone AM deployments, implement a Username Collector node and a Password Collector node earlier in the journey.

  For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node earlier in the journey.

- Optionally, this node can read the contents of the `webAuthnExtensions` shared state property as input.

  You can populate the `webAuthnExtensions` property with any JSON object you choose using a script or in a node that occurs earlier in the journey. If this property is populated, its contents are placed in the `extensions` entry passed to the browser or authenticator. If this property is empty, an empty JSON object is passed to the browser or authenticator.

  You can find more information on WebAuthn extensions in WebAuthn Extensions.

## Dependencies

You must configure the WebAuthn Profile Encryption service to use this node. Find more information in the corresponding documentation for:

- PingAM
- Advanced Identity Cloud

## Configuration

| Property | Usage |
| --- | --- |
| **Relying party** | The name of the relying party entity that registers and authenticates users by using WebAuthn. This could be the name of the organization, realm, and so on. For example, `Example.com` |
| **Relying party identifier** | The domain used as the relying party identifier during WebAuthn. If not specified, AM uses the domain name of the instance, such as `am.example.com`. Specify an alternative domain if your AM instances are behind a load balancer. |
| **Origin domains** | A set of fully-qualified URLs of accepted origins, for example `http://app.example.com:443`. If empty, the accepted origin is the incoming request origin. |

| Property | Usage |
| --- | --- |
| **User verification requirement** | The required level of user verification.<br>The available options are:<br><br>**`REQUIRED`**<br>The authenticator must verify the identity of the user, for example, by using biometrics. AM filters out authenticators that don't verify user identity and the user can't select them.<br>**`PREFERRED`**<br>AM prefers an authenticator that verifies user identity. If none are available, AM accepts any authenticator.<br>**`DISCOURAGED`**<br>The authenticator doesn't need to verify the identity of the user. Authenticators that don't verify user identity are preferred.<br><br>Default: `PREFERRED` |
| **Preferred mode of attestation** | Indicates whether the authenticator must provide attestation statements.<br>The available options are:<br><br>**`NONE`**<br>AM doesn't require the authenticator to provide attestation statements. If the authenticator sends attestation statements, AM doesn't verify them and the process doesn't fail.<br>**`INDIRECT`**<br>AM doesn't require the authenticator to provide attestation statements. However, if the authenticator sends attestation statements, AM verifies them and the process fails if the verification fails.<br>**`DIRECT`**<br>AM requires the authenticator to provide attestation statements and verifies them. The process fails if the attestation statements can't be verified.<br><br>AM supports the following attestation formats:<br><br>• None<br>• Android SafetyNet<br>• Packed<br>• FIDO U2F<br>• TPM<br><br>**Important**<br>If your end users use an authenticator that provides attestation statements in a format other than these supported formats, you must set the **Preferred mode of attestation** property to `NONE`.<br>Specifically, AM *does not* support the Android Key Attestation Statement Format. |

| Property | Usage |
| --- | --- |
| **Accepted signing algorithms** | The algorithms authenticators can use to sign their assertions. |
| **Authentication attachment** | If specified, AM filters out authenticators that don't match the attachment type. There are two attachment types:<br>• A `PLATFORM` authenticator is part of the device, for example, a fingerprint scanner built-in to a phone or laptop.<br>• A `CROSS_PLATFORM` authenticator can be removed from a device and used elsewhere, for example, a USB hardware security key.<br>Available options for this property are:<br>**`UNSPECIFIED`**<br>AM accepts any attachment type.<br>**`PLATFORM`**<br>The authenticator must be a *platform* attachment type. The client shouldn't activate other authenticator types for registration.<br>**`CROSS_PLATFORM`**<br>The authenticator must be a *cross-platform* attachment type. The client shouldn't activate other authenticator types for registration. |
| **Trust Store alias** | The alias of the realm trust store holding the secrets necessary to validate a supplied attestation certificate. The alias can only contain the characters `a-z` and periods ( `.` ).<br>This value is appended to the string `am.authentication.nodes.webauthn.truststore.` to form the dynamic secret label used to map certificate chains. |
| **Enforce revocation check** | Whether to enforce the checking of revocation entries from certificates.<br>If you enable this setting, any attestation certificate's trust chain *must* have a CRL or OCSP entry that AM can verify.<br>If you disable this setting, AM doesn't check presented certificates for revocation.<br>**Note**<br>Certificates downloaded from the FIDO Metadata Service might not have a CRL or OCSP entry. You must remove expired or revoked certificates manually. |
| **Timeout** | The number of seconds to wait for a response from an authenticator.<br>If the specified time is reached, evaluation continues along the `Client error` outcome path and a relevant message is stored in the `WebAuthenticationDOMException` property of the shared state. |
| **Limit registrations** | Indicates whether the same authenticator can be registered multiple times.<br>If you enable this property, the client won't activate an authenticator that's already registered for registration. |

| Property | Usage |
| --- | --- |
| **Generate recovery codes** | Indicates whether AM generates WebAuthn-specific recovery codes. If enabled, AM generates recovery codes and stores them in the transient state if registration is successful.<br>Use the Recovery Code Display node to display the codes to the user for safe-keeping.<br>Don't enable this property if you've enabled the **Store device data in transient state** property (and aren't saving the device data to the user's profile immediately).<br>Enable the **Generate recovery codes** property in the WebAuthn Device Storage node instead.<br>**Important**<br>Generating recovery codes overwrites *all existing* WebAuthn-specific recovery codes.<br>Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen. |
| **Store data in transient state** | If you enable this property, the node stores the following data in transient state:<br>• Information provided by the device is stored in the `webauthnData` property for later analysis by subsequent nodes.<br>• The attestation type achieved ( `BASIC` , `CA` , or `SELF` ) is stored in the `webauthnAttestationType` property.<br>**Warning**<br>The amount of data received from the device can be large. Only enable this option if you intend to analyze it. |
| **Store device data in transient state** | If you enable this property, information about the device required for WebAuthn is stored in transient state instead of saved immediately to the user's profile.<br>Enable this option under the following conditions:<br>• You've enabled the **Store data in transient state** property<br>• You need to make decisions in scripts based on the outcome of the analysis of data in transient state<br>• You don't want to register the device to the user until the analysis is complete<br>**Important**<br>Don't *change* the data while it's in transient state, nor when it's saved to a user's profile.<br>Changing the device data will likely cause the device to be unable to authenticate.<br>If you enable this option, use the WebAuthn Device Storage node to write the device data to the user's profile.<br>If this option is disabled, device data is written automatically to the user's profile when registration is successful. |

| Property | Usage |
| --- | --- |
| **Username to device** | This option specifies that the device should store the user's username.<br>If you enable this option, devices that don't support storing and providing the username won't be able to use this node. If the device can't store or provide usernames, the node fails and the journey follows the Failure outcome.<br>Find more information on using this property for usernameless authentication with ForgeRock Go in the corresponding documentation for:<br>• PingAM<br>• Advanced Identity Cloud |
| **Shared state attribute for display name** | The share state property that contains a display name for the user. For example, their full name, or email address.<br>When **Username to device** is enabled, AM writes the value stored in this property to devices in addition to the username. This helps the user select between the accounts they may have on their devices.<br>If you don't set this property, or if the variable isn't found in shared state, the username is used.<br>Find more information on using this property for usernameless authentication with ForgeRock Go in the corresponding documentation for:<br>• PingAM<br>• Advanced Identity Cloud |
| **Return challenge as JavaScript** | Choose how the node should return its challenge for consumption by your front-end user interface:<br>***PingOne Advanced Identity Cloud hosted pages or Ping SDK client apps:***<br>Deselect this option when using either hosted pages in PingOne Advanced Identity Cloud or a client application built using the Ping SDKs as your authentication user interface.<br>When not enabled the node returns the challenge and associated data in a metadata callback.<br>PingOne Advanced Identity Cloud hosted pages and Ping SDK client apps that might not be able to execute JavaScript use the information from the callback to interact with WebAuthn APIs on AM's behalf.<br>***PingAM User UI:***<br>You should only enable this option if you are using the PingAM User UI to authenticate users.<br>Enabling this option causes the node to return its challenge as a fully encapsulated client-side JavaScript that interacts directly with the WebAuthn API. |

| Property | Usage |
| --- | --- |
| **Maximum Saved Devices** | The maximum number of WebAuthn devices that can be stored in the user's profile.<br>Set this property to `0` if you don't want to limit the number of devices.<br>When this property is greater than zero, the `Exceed Device Limit` outcome path becomes available.<br><br>**Important**<br>You can only limit the number of devices stored in the user's profile.<br>If you enable **Store device data in transient state**, the node can't limit the number of devices and the `Exceed Device Limit` outcome path isn't displayed.<br>In this case, specify the maximum number of saved devices in the WebAuthn Device Storage node. |
| **Validate FIDO-U2F attestation AAGUID** | If enabled, the node validates the Authenticator Attestation Global Unique Identifier (AAGUID) for any FIDO-U2F attestation type. The AAGUID must be 16 bytes, initialized with all zeros. |
| **FIDO Certification Level** | The minimum FIDO certification level that the device's certification status must satisfy during a registration flow.<br>If this setting is `Off` (the default), AM doesn't check the metadata service for the device's certification level.<br>Other options include:<br>• `Self Assertion Submitted`: Use this setting if your authenticator has been submitted for FIDO certification but has not yet been certified.<br>• `FIDO Certified L1` - `FIDO Certified L3+`: Find information on these levels in Certified Authenticator Levels.<br><br>**Note**<br>You must configure the WebAuthn Metadata service for the realm to enforce a FIDO certification check.<br>Find more information in the corresponding documentation for:<br>• PingAM<br>• Advanced Identity Cloud |

## Outputs

- The node passes the contents of the `webAuthnExtensions` property to the browser or authenticator.
- If a timeout is reached, or any other client error occurs, the error type and description are added to the `WebAuthenticationDOMException` shared state property.
- If **Shared state attribute for display name** is set, the node writes the username or display name to shared state as a value of the specified property.

- If **Store data in transient state** is set, the node writes the following data to transient state on successful registration:
  - Information provided by the device is stored in the `webauthnData` property.
  - The attestation type achieved is stored in the `webauthnAttestationType` property.
  - The registered Authenticator Attestation Global Unique Identifier (AAGUID) is stored in the `webauthnDeviceAaguid` property.
  - PingAM Additional attestation information is stored in the `webauthnAttestationInfo` object.

    The `webauthnAttestationInfo` object includes the following information:

    ```
    {
      "authenticatorAttachment": "platform",
      "flags": {
        "UP": true,
        "UV": true,
        "ED": false,
        "AT": false,
        "BE": true,
        "BS": true
      }
    }
    ```

    The `authenticatorAttachment` field is added only if **Return challenge as JavaScript** is enabled. This field lets the journey identify whether the end user registered a roaming (cross-platform) device or a client-bound (platform) device. Learn more in Authenticator Attachment Modality in the WebAuthn specification.

    The `flags` provide additional information about the authenticator. Learn more about these flags in Authenticator Data in the WebAuthn specification.

## Outcomes

### *Unsupported*

If the user's client doesn't support WebAuthn, evaluation continues along the `Unsupported` outcome path. For example, clients connected over the HTTP protocol rather than HTTPS don't support WebAuthn.

### *Success*

If the user successfully registers an authenticator of the correct type as determined by the node's properties, evaluation continues along the `Success` outcome path.

### *Failure*

If AM encounters an issue when attempting to register a user's device, evaluation continues along the `Failure` outcome path. For example, if AM can't verify that the response from the authenticator was appropriate for the specific instance of the authentication ceremony.

### *Client Error*

If the user's client encounters an issue when attempting to register using a device, for example, if the timeout was reached, evaluation continues along the `Client Error` outcome path. The node follows this outcome whenever the client throws a `DOMException`, as required by the WebAuthn specification.

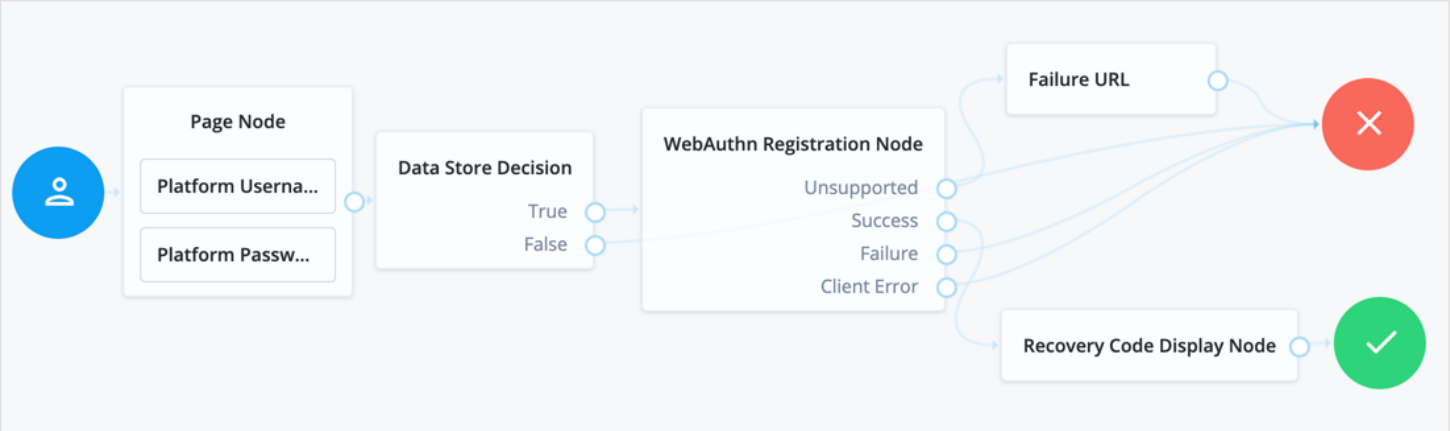
> **Tip**
>
> If a client error occurs, the error type and description are added to the `WebAuthenticationDOMException` shared state property. If required, this property can be read by nodes later in the journey.

### *Exceed Device Limit*

If the **Maximum Saved Devices** property is an integer greater than zero, and registering a new device would take the number of devices above the specified threshold, evaluation continues down the `Exceed Device Limit` outcome path. In this case, you might need to instruct users to log in with an existing device to remove one or more of their registered devices.

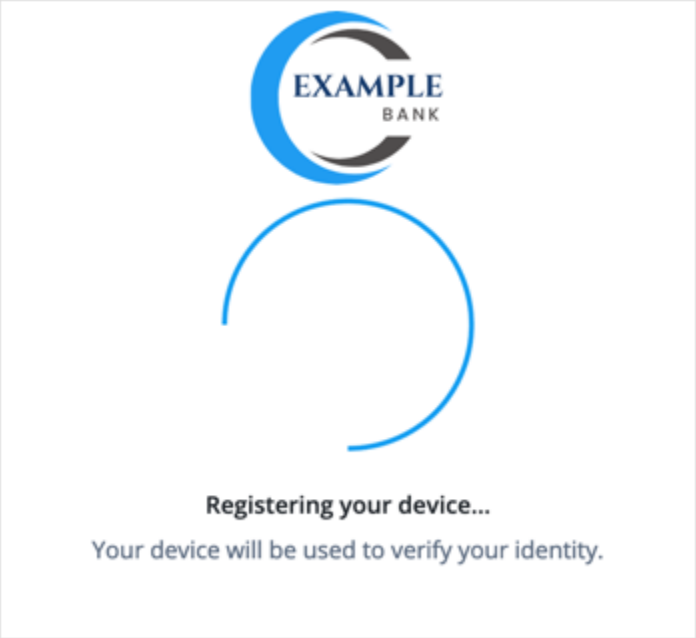
## Example

The following sample journey registers WebAuthn devices:




If the user's client doesn't support WebAuthn, the failure URL is altered, for example to redirect the user to a page explaining which clients and operating systems support WebAuthn.

If the user's client does support WebAuthn, and the connection is secured with TLS, AM prompts the user to register an authenticator:




The user's browser may present a consent pop-up to allow access to the authenticators available on the client. When consent has been granted, the browser activates the relevant authenticators, ready for registration.

> **Tip**
>
> The relying party details configured in the node are often included in the consent message to help the user verify the entity requesting access.

The authenticators the client activates for registration depend on the value of the properties in the node. For example, if the **User verification requirement** property is set to `REQUIRED`, the client would not activate a USB hardware security key for registration.

When the user completes an authorization gesture, for example, by scanning a fingerprint or entering a PIN, the evaluation continues along the `Success` outcome path, and in this example will be taken to their profile page.

The registered authenticator appears on the user's dashboard page, with the label *New Security Key*. To rename the authenticator, click its vertical ellipsis context icon, ⋮, and click Rename.

Any problems encountered during the registration, including a timeout, results in the evaluation continuing to the `Failure` outcome.

# Risk management nodes

## Account Active Decision node

The **Account Active Decision** node determines whether the current account is both active and unlocked, and lets the journey make a decision, based on that check.

An account is considered locked under these conditions:

- The status is inactive.
- The status is active and a duration lockout is set on the account.

An account is considered unlocked under this condition:

- The status is active and no duration lockout is set on the account.

The node determines whether the account has been locked through both persistent (physical) lockout and duration lockout. Find more information in the *Account lockout* documentation for:

- PingAM
- Advanced Identity Cloud

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

The node reads the user's identity from the shared state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

### Dependencies

This node has no dependencies.

## Configuration

This node has no configurable properties.

## Outputs

This node doesn't write anything to the shared state.

## Outcomes

- `True`

  The journey follows this outcome path if the account is assessed to be `active` and `unlocked`.

- `False`

  The journey follows this outcome path if the account is assessed to be `inactive` or `locked`.

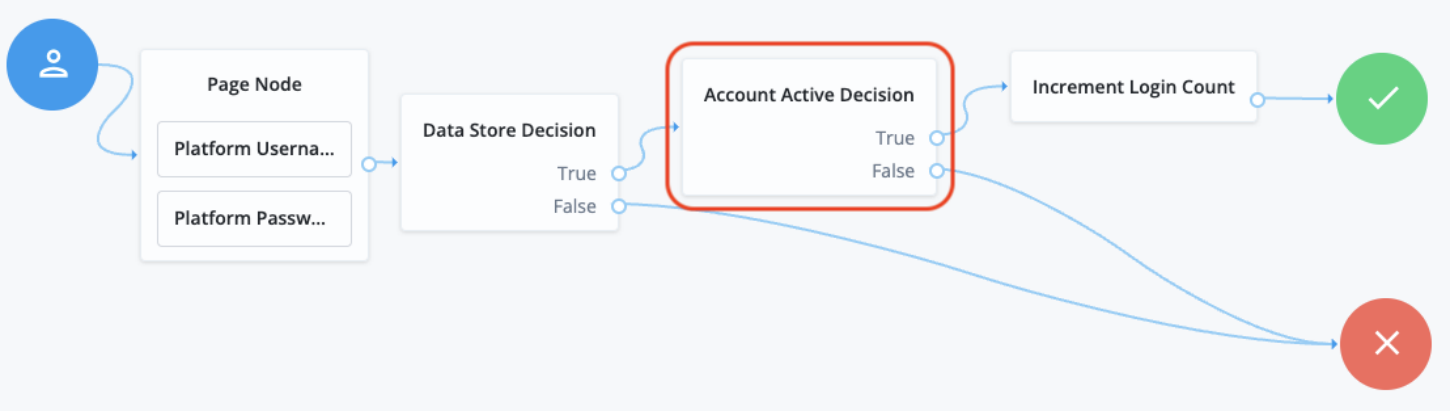
## Errors

If the node cannot read the identity of the account, it throws the following exception:

`Failed to get the identity object`

## Examples

In this simple login journey, authentication fails if the account is assessed to be `inactive` or `locked`.




This example uses the following nodes:

- The Page node prompts the user to input their username and password:
  - The Platform Username node collects the username and stores it in the shared state.
  - The Platform Password node collects the password and stores it in the shared state.
- The Data Store Decision node uses the username and password to determine whether the account exists.
- The **Account Active Decision** node determines whether the account is active and unlocked.

- If the account is active and unlocked, the Increment Login Count node increments the login count and authentication succeeds.
- If the account is inactive or locked, the authentication fails.

## Account Lockout node

The **Account Lockout** node locks or unlocks the authenticating user's account profile.

The node also determines whether the account has been locked through both persistent (physical) lockout and duration lockout. Find more information in the documentation on account lockout for:

- PingAM
- Advanced Identity Cloud

> **Tip**
>
> You can also use the Account Active Decision node to check whether the account is locked at any point in the journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the `username` property in the incoming node state. It uses this information to access the account status in the user profile.

It also requires the `realm` property, which the product using this node sets by default.

### Dependencies

This node depends on the underlying identity service that stores the user profile.

### Configuration

| Property | Usage |
|---|---|
| **Lock Action** | Choose whether to `LOCK` or `UNLOCK` the authenticating user's account profile. |

## Outputs

This node does not change the shared node state.

## Outcomes

Single outcome path; the node updates the account status according to the configured **Lock Action**:

**`LOCK`**

The account is inactive and the user cannot authenticate.

**`UNLOCK`**

The account is active and the user can authenticate.

## Errors

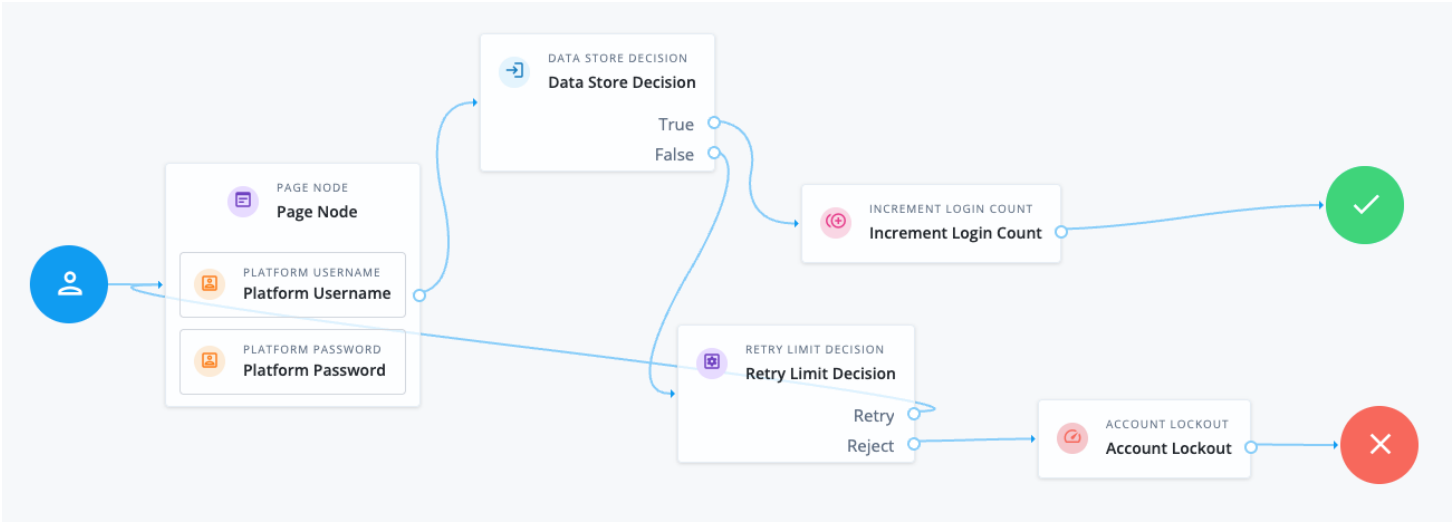
If this node fails to set the account status, it logs a `failed to set the user status inactive` warning.

This node can also throw exceptions with the following messages:

| Message | Notes |
| --- | --- |
| `Could not get a valid username from the context` | Failed to read the `username` from the shared node state |
| `Could not get a valid realm from the context` | Failed to read the `realm` from the shared node state |
| `Could not find the identity based on the information available on context` | Failed to find the account profile with this `username` in this `realm` |
| `An error occurred when trying to lock out the user account` | Failed to update the account status; applies when locking and unlocking the account |

## Example

The following simple example uses this node with the Retry Limit Decision node to lock an account after the set number of invalid attempts:




The Retry Limit Decision node **Retry limit** (default: 3) defines the number of failed attempts before lockout.

Before using a journey like this in deployment, adapt it to reset the retry count on successful authentication.

## Auth Level Decision node

Compares the current authentication level value against a configured value.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `True`
- `False`

### Properties

| Property | Usage |
|---|---|
| Sufficient Authentication Level | Evaluation continues along the `True` path if the current authentication level is equal to or greater than this integer; otherwise, the evaluation continues along the `False` path. |

# CAPTCHA node

The **CAPTCHA** node adds CAPTCHA support by verifying the response token received from the CAPTCHA provider and creating a callback for the UI to interact with.

By default, the node is configured for Google's reCAPTCHA v2.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

> **Note**
>
> This node only supports reCAPTCHA v2 and v3, and hCaptcha.
> For Google reCAPTCHA Enterprise support, use the reCAPTCHA Enterprise node.

## Inputs

None. This node doesn't read shared state data.

## Dependencies

You need to sign up for access to the reCAPTCHA API to get the API key pair required for configuring the node.

## Configuration

| Property | Usage |
|---|---|
| **CAPTCHA Site Key** *(required)* | The CAPTCHA site key supplied by the CAPTCHA provider when you sign up for access to the API. |

| Property | Usage |
| --- | --- |
| **CAPTCHA Secret Key** | The CAPTCHA secret key supplied by the CAPTCHA provider when you sign up for access to the API.<br><br>**Important**<br>This property is deprecated and will be removed in a future release. Use the **CAPTCHA Secret Label Identifier** instead.<br>If you set a **CAPTCHA Secret Label Identifier** and the product using this node finds a matching secret in a secret store, the **CAPTCHA Secret Key** is ignored. |
| **CAPTCHA Secret Label Identifier** | An identifier used to create a *secret label* for mapping to a secret in a secret store.<br>The product using this node uses this identifier to create a specific secret label for this node. The secret label takes the form `am.authentication.nodes.captcha.identifier.secret` where identifier is the value of **CAPTCHA Secret Label Identifier**.<br>The identifier can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods ( `.` ). It can't start or end with a period.<br>If you set a **CAPTCHA Secret Label Identifier** and the product using this node finds a matching secret in a secret store, the **CAPTCHA Secret Key** is ignored. |
| **CAPTCHA Verification URL** | The URL used to verify the CAPTCHA submission.<br><br>• In Advanced Identity Cloud, use `https://www.google.com/recaptcha/api/siteverify`.<br>• In PingAM and Ping Identity Platform, possible values are:<br>◦ Google: `https://www.google.com/recaptcha/api/siteverify`<br>◦ hCaptcha: `https://hcaptcha.com/siteverify` |
| **CAPTCHA API URL** *(required)* | The URL of the JavaScript that loads the CAPTCHA widget.<br><br>• In Advanced Identity Cloud, use `https://www.google.com/recaptcha/api.js`.<br>• In PingAM and Ping Identity Platform, possible values are:<br>◦ Google: `https://www.google.com/recaptcha/api.js`<br>◦ hCaptcha: `https://hcaptcha.com/1/api.js` |
| **Class of CAPTCHA HTML Element** | The class of the HTML element required by the CAPTCHA widget.<br><br>• In Advanced Identity Cloud, use `g-recaptcha`.<br>• In PingAM and Ping Identity Platform, possible values are:<br>◦ Google: `g-recaptcha`<br>◦ hCaptcha: `h-captcha` |

| Property | Usage |
| --- | --- |
| **ReCaptcha V3 node** | If you're using Google reCAPTCHA, specify whether it's v2 or v3. Turn on for v3. |
| **Score Threshold** | If you're using Google reCAPTCHA v3, or hCaptcha (PingAM only), enter a score threshold.<br>The CAPTCHA provider returns a score for each user request, based on observed interaction with your site. CAPTCHA "learns" by observing real site traffic, so scores in a staging environment or in a production deployment that has just been implemented might not be very accurate.<br>A score of 1.0 is likely a good user interaction, while 0.0 is likely to be a bot.<br>The threshold you set here determines whether to allow or deny access, based on the score returned by the CAPTCHA provider.<br>Start with a threshold of 0.5.<br>Learn more about score thresholds in the Google documentation. |
| **Disable submission until verified** | If selected, form submission is disabled until CAPTCHA verification succeeds.<br>Default: Enabled |

## Outputs

None.

## Outcomes

### `True`

The CAPTCHA response was successfully verified.

### `False`

The CAPTCHA response wasn't verified or failed verification.

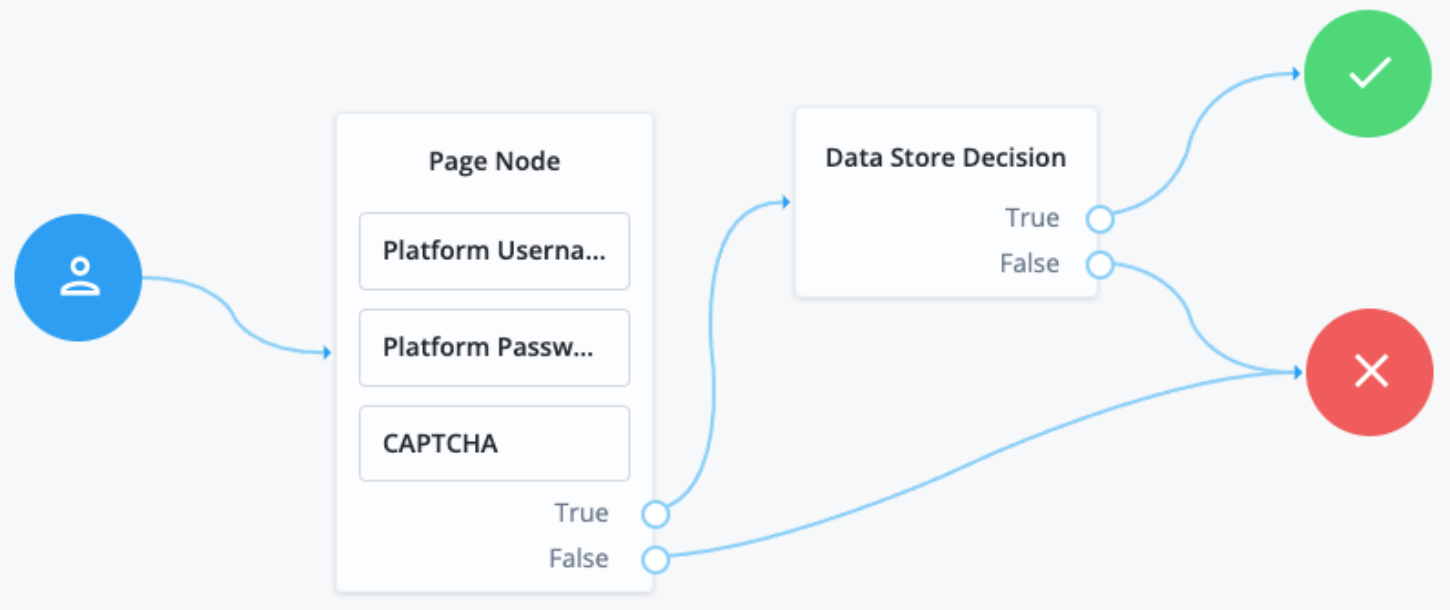
## Errors

This node can throw exceptions with the following messages:

- `CAPTCHA response required for verification`
- `Unable to verify CAPTCHA response`
- `Unable to retrieve state from token response`
- `No secret key found`

## Example

The following journey uses a Page node and a Data Store Decision node to collect and verify the credentials and a CAPTCHA response:




This example uses the following nodes:

- The Page node prompts the user to input their username and password:
  - The Platform Username node collects the username and stores it in the shared state.
  - The Platform Password node collects the password and stores it in the shared state.
  - The CAPTCHA node collects and verifies the CAPTCHA response.
- The Data Store Decision node uses the username and password to determine whether authentication is successful.

## reCAPTCHA Enterprise node

The **reCAPTCHA Enterprise** node adds Google reCAPTCHA Enterprise support to your journeys.

Google reCAPTCHA Enterprise offers improvements over previous versions, including more granular scores, reason codes for events deemed higher risk, Web Application Firewall (WAF) support, and native support for Android and iOS.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

- Only the Ping SDKs support the reCAPTCHA Enterprise functionality this node provides.

The following user interfaces do not support this node:

- PingOne Advanced Identity Cloud hosted pages
- PingAM User UI

- This node only supports Google reCAPTCHA Enterprise.

  For reCAPTCHA v2 and v3 support, and hCaptcha support, use the CAPTCHA node.

## Inputs

This node reads an optional `CaptchaEnterpriseNode.PAYLOAD` variable from shared state.

Use this variable to customize the payload the node sends to the Google reCAPTCHA Enterprise server for assessment.

You can set the value by using a Set State node, or by using a Scripted Decision node, using a script similar to the following:

```
var username = nodeState.get("username");
var customPayload =
  JSON.parse(`{"userInfo": {"accountId": "${username}"}}`);
sharedState.put("CaptchaEnterpriseNode.PAYLOAD", customPayload);
outcome = "true";
```

To learn more about the payload, refer to Project Assessments - Event in the Google Developer documentation.

## Dependencies

You need to sign up for access to the reCAPTCHA API to get the API key pair required to configure the node.

## Configuration

| Property | Usage |
|---|---|
| **Google Cloud project ID** | The ID of the project that has Google reCAPTCHA enabled.<br>You can get the ID of your project in the Google Cloud console.<br>For example, `my-project-65746-07969469388`. |
| **reCAPTCHA Site Key** *(required)* | The ID of the reCAPTCHA key you created in the Google Cloud console.<br>The key can be for any platform type, `Website`, `Android app`, or `iOS app`.<br>Sometimes referred to as a *key ID* in the Google Cloud console and documentation. |

| Property | Usage |
| --- | --- |
| **reCAPTCHA API key secret identifier** | An identifier used to create a secret label for mapping to your Google reCAPTCHA API key in a secret store.<br>Get or create your API key from the **Google Cloud Console** under **APIs and Services > Credentials**.<br>The secret label takes the form `am.authentication.nodes.captchaEnterprise.identifier.secret` where identifier is the value of **reCAPTCHA API key secret identifier**.<br>The identifier can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods (`.`). It can't start or end with a period. |
| **Score Threshold** | The score threshold for determining if a user is likely to be a real person.<br>reCAPTCHA scores are between `0.0` and `1.0`, with higher scores indicating higher confidence that the user is a real person.<br>If the returned score is equal to or greater than the threshold the journey continues along the `true` outcome path.<br>To learn more, refer to Interpret scores in the Google documentation. |
| **Store reCAPTCHA assessment JSON** | Stores the assessment response JSON for future reference within the journey.<br>The node stores the JSON response in the `CaptchaEnterpriseNode.ASSESSMENT_RESULT` variable. |
| **Store reCAPTCHA error messages** | Stores the error messages for future reference within the journey.<br>The node stores the error messages in the `CaptchaEnterpriseNode.FAILURE` variable.<br>The error consists of an error code and description of the error.<br>• `INVALID_TOKEN`<br>• `INVALID_PROJECT_ID`<br>• `CLIENT_ERROR`<br>• `INVALID_SECRET_KEY`<br>• `VALIDATION_ERROR`<br>• `API_ERROR`<br>• `IO_ERROR`<br>• `UNKNOWN` |
| **reCAPTCHA CSS class** | A CSS class to apply to the HTML elements reCAPTCHA adds to JavaScript apps.<br>The default is `g-recaptcha`. |
| **reCAPTCHA Verification URL** | The URL to send the reCAPTCHA to for verification.<br>Only change this if Google updates the URL used for reCAPTCHA verifications.<br>The default is `https://recaptchaenterprise.googleapis.com/v1`. |
| **JavaScript reCAPTCHA API URL** | The URL of the JavaScript file containing the reCAPTCHA API.<br>Only change this if Google releases a new version of the JavaScript reCAPTCHA API.<br>The default is `https://www.google.com/recaptcha/enterprise.js`. |

## Outputs

If you enable the **Store reCAPTCHA assessment JSON** property, the node outputs the reCAPTCHA assessment response JSON in a state variable named `CaptchaEnterpriseNode.ASSESSMENT_RESULT`.

If you enable the **Store reCAPTCHA error messages** property, the node outputs the error response JSON in a state variable named `CaptchaEnterpriseNode.FAILURE`.

## Outcomes

### `True`

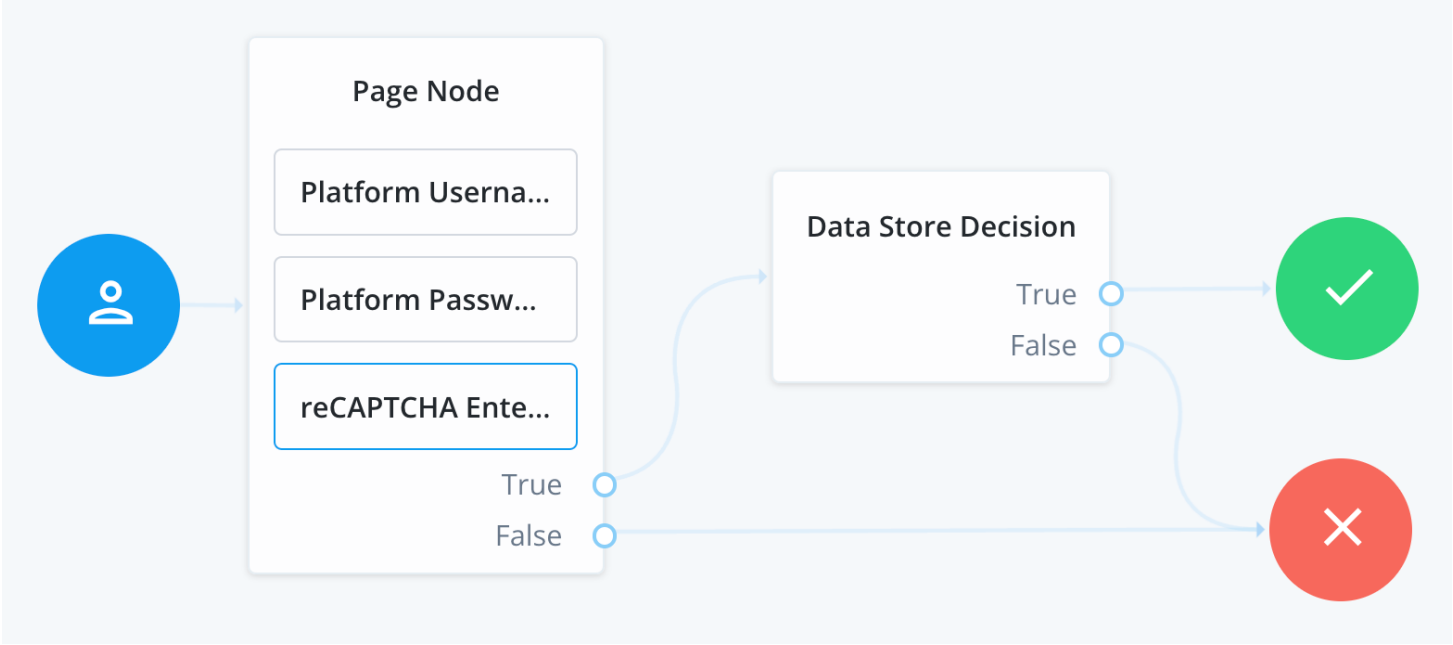
The reCAPTCHA response was successfully verified.

### `False`

The reCAPTCHA response wasn't verified or failed verification.

## Example

The following journey uses a Page node and a Data Store Decision node to collect and verify the credentials and a CAPTCHA response:




This example uses the following nodes:

- The Page node prompts the user to input their username and password:
  - The Platform Username node collects the username and stores it in the shared state.
  - The Platform Password node collects the password and stores it in the shared state.
  - The reCAPTCHA Enterprise node collects and verifies the reCAPTCHA Enterprise response.

- The Data Store Decision node uses the username and password to determine successful authentication.

## Legacy CAPTCHA node

Verifies the response token received from the CAPTCHA verifier, and creates a CAPTCHA callback for the UI to interact with. Default values are for Google ReCAPTCHA.

> **Important**
>
> This node has been superseded by the CAPTCHA node. Use that node instead.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `True` *(success)*
- `False` *(failure)*

### Properties

| Property | Usage |
|---|---|
| CAPTCHA Site Key *(required)* | The CAPTCHA site key supplied by the CAPTCHA provider when you sign up for access to the API. |
| CAPTCHA Secret Key *(required)* | The CAPTCHA secret key supplied by the CAPTCHA provider when you sign up for access to the API. |
| CAPTCHA Verification URL *(required)* | The URL used to verify the CAPTCHA submission.<br>• In Advanced Identity Cloud, use `https://www.google.com/recaptcha/api/siteverify`.<br>• For PingAM and Ping Identity Platform, possible values are:<br>◦ Google: `https://www.google.com/recaptcha/api/siteverify`<br>◦ hCaptcha: `https://hcaptcha.com/siteverify` |

| Property | Usage |
| --- | --- |
| CAPTCHA API URL *(required)* | The URL of the JavaScript that loads the CAPTCHA widget.<br>• In Advanced Identity Cloud, use `https://www.google.com/recaptcha/api.js`.<br>• For PingAM and Ping Identity Platform, possible values are:<br>◦ Google: `https://www.google.com/recaptcha/api.js`<br>◦ hCaptcha: `https://hcaptcha.com/1/api.js` |
| Class of CAPTCHA HTML Element | The class of the HTML element required by the CAPTCHA widget.<br>• In Advanced Identity Cloud, use `g-recaptcha`.<br>• For PingAM and Ping Identity Platform, possible values are:<br>◦ Google: `g-recaptcha`<br>◦ hCaptcha: `h-captcha` |

## Modify Auth Level node

Increases or decreases the current authentication level value.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
| --- | --- |
| Value To Add | Enter a positive integer to increase the current authentication level, or a negative integer to decrease the current authentication level by the specified value. |

# PingOne Protect Evaluation node

The **PingOne Protect Evaluation** node contacts PingOne to calculate the risk level and other risk-related details associated with an event.

Depending on how you configure your risk policies in PingOne, the response could return a risk score, a risk level such as high, medium, or low, and recommended actions to take, such as mitigation against bots.

Learn more in PingOne Protect > How it Works.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node can use shared state variables that contain the PingOne `user.id` and `user.name` as input. If these are not available, the node uses the `UserId` and `Username` variables.

This node requires that you've initialized PingOne Protect in your client application. For example, by using a PingOne Protect Initialization node node previously in the journey or by initializing the SDK within the app itself.

## Dependencies

This node requires a **PingOne Worker Service** configuration so that it can connect to your PingOne instance and send it the necessary data to make risk evaluations.

The client application must be using Ping SDK 4.4.0 or later.

## Configuration

| Property | Usage |
|---|---|
| **PingOne Worker Service ID** | The ID of the PingOne worker service for connecting to PingOne. |
| **Target App ID** | (Optional) If the user is attempting to access a PingOne application through the journey, add its v4 UUID client ID.<br>This correlates the authentication with the application in PingOne, allowing you to filter by the **Resource Id** that matches the entered Target App ID when viewing the audit log in PingOne.<br>For example, `12345678-abcd-4567-abcd-a123b123c123`. |

| Property | Usage |
| --- | --- |
| **Risk Policy Set ID** | The ID of the risk policy in PingOne.<br>To view risk policies in the PingOne admin console, go to **Threat Protection** > **Risk Policies**.<br>If not specified, the environment's default risk policy set is used. |
| **Flow Type** | The type of flow or event for which the risk evaluation is being carried out.<br>Choose from:<br><br>**`REGISTRATION`**<br>Initial registration of an account.<br>**`AUTHENTICATION`**<br>Standard authentication for login or actions such as password change.<br>**`ACCESS`**<br>Verification of whether the user can access the relevant application.<br>**`AUTHORIZATION`**<br>Verification of whether the user is authorized to perform a specific action such as a profile change.<br>**`TRANSACTION`**<br>Authentication carried out in the context of a purchase or other one-time transaction.<br><br>The default is `AUTHENTICATION`. |
| **Device Sharing Type** | Whether the device is shared between users or not.<br>Choose from:<br><br>• `UNSPECIFIED`<br>• `SHARED`<br>• `PRIVATE`<br><br>The default is `SHARED`. |
| **User Type** | The type of user associated with the event.<br>Choose from:<br><br>**`PING_ONE`**<br>User exists within the PingOne environment.<br>**`EXTERNAL`**<br>User exists outside PingOne, such as a federated user.<br><br>The default is `EXTERNAL`. |
| **Score Threshold** | Scoring higher than this value results in evaluation continuing along the `Exceeds Score Threshold` outcome.<br>The default is `300`. |

| Property | Usage |
| --- | --- |
| **Recommended Actions** | A list of recommended actions the risk evaluation could return. Each entry in the list becomes a node outcome.<br>If the evaluation score does not exceed the **Score Threshold** value, and a recommended action is present in the response from PingOne Protect, the journey continues down the matching entry in this list.<br>Possible values are:<br>**`BOT_MITIGATION`**<br>PingOne suspects the client could be automated or a bot. You should route the journey to a CAPTCHA node or similar next step to mitigate against bots.<br>**`AITM_MITIGATION`**<br>PingOne suspects an adversary-in-the-middle (AitM) attack. You should route the journey to the failure node, and consider locking the account, and force a password change to mitigate against these attacks. |
| **Pause Behavioral Data** | After receiving the device signal, instruct the client to pause collecting behavioral data.<br>Default: Selected |
| **Node State Attribute For User ID** | The node state variable that contains the `user.id` as it appears in PingOne.<br>If left blank, the node uses the current context `UserId` as the `user.id`. |
| **Node State Attribute For Username** | The node state variable that contains the `user.name` as it appears in PingOne.<br>If left blank, the node uses the current context `Username` as the `user.name`. |
| **Store Risk Evaluation** | Stores the risk evaluation response in the transient node state under a key named `PingOneProtectEvaluationNode.RISK`.<br>The default is not enabled.<br>**Note**<br>The key is empty if the node is unable to retrieve a risk evaluation from PingOne. |

## Outputs

If you enable the **Store Risk Evaluation** property, the node outputs the risk evaluation response JSON in a state variable named `PingOneProtectEvaluationNode.RISK`.

## Outcomes

### `High`

The risk evaluation level is considered high.

### `Medium`

The risk evaluation level is considered medium.

#### `Low`

The risk evaluation level is considered low.

#### `Exceeds Score Threshold`

The score returned is higher than the configured threshold.

#### `Failure`

The risk evaluation could not be completed.

### *Recommended Actions*

The risk evaluation recommended a mitigation action to take, and it matched a value in the **Recommended Actions** list.

Currently, the only value possible is `BOT_MITIGATION`, which recommends you check for the presence of a human, such as by using a CAPTCHA node.

#### `ClientError`

The client returned an error when attempting to capture the data to perform a risk evaluation.

### Outcome precedence

Evaluation of the journey continues along an outcome based on the response received and which fields are present in it, as follows:

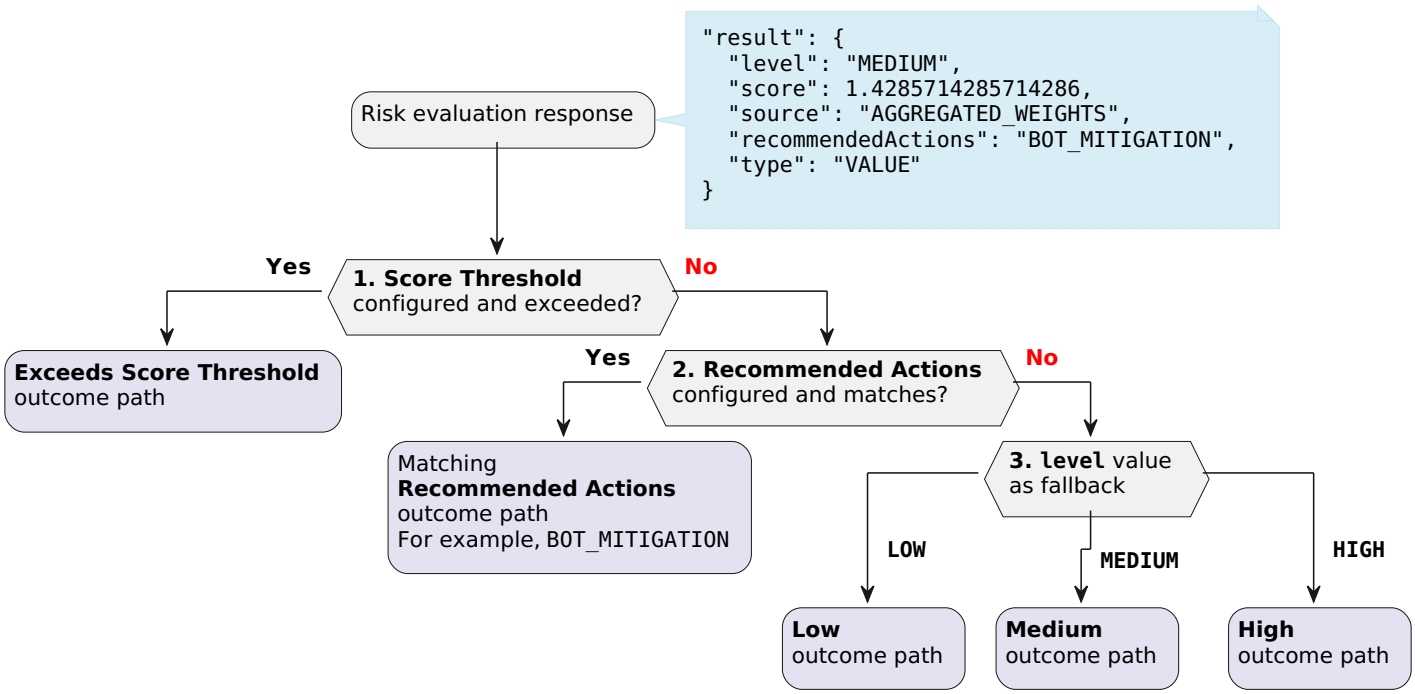



*Figure 1. Risk evaluation outcome path precedence*

1. If you have configured the **Score Threshold** property and the result contains a score that exceeds it, evaluation continues along the `Exceeds Score Threshold` outcome path.

2. If you have *not* configured the **Score Threshold** property, or the score does not exceed it, but *have* added a value in the **Recommended Actions** list that matches one in the response, evaluation continues along the relevant dynamic outcome path. For example, the `BOT_MITIGATION` outcome path.
3. If you have *not* configured the **Score Threshold** property, or the score does not exceed it, and have *not* added a matching value in the **Recommended Actions** list, then evaluation continues along the relevant `level` path, one of `Low`, `Medium`, or `High`.

## Example

The following example journey leverages PingOne Protect functionality to perform a risk evaluation on a client app. The client app is built using the Ping SDKs.

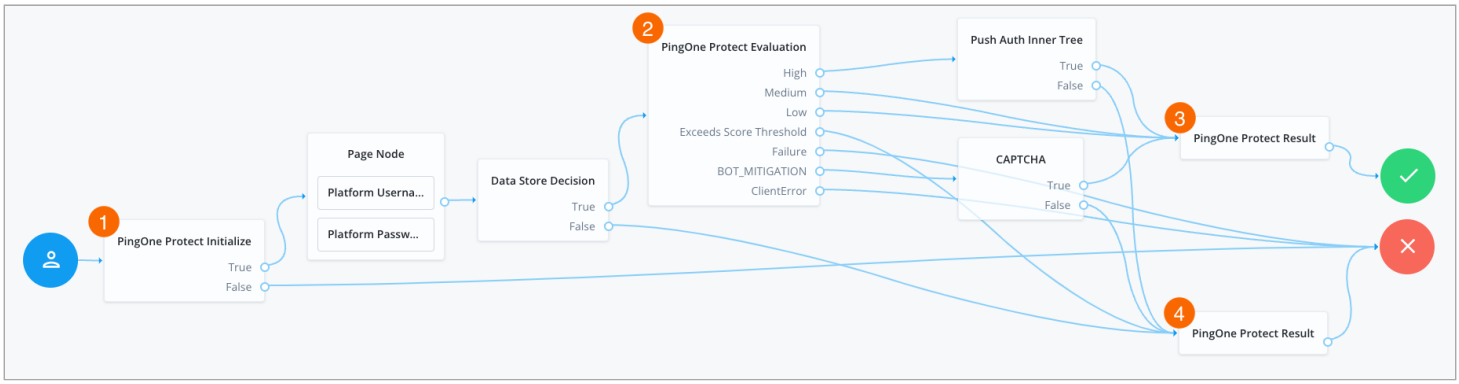



*Figure 2. Example PingOne Protect journey*

- 1 The PingOne Protect Initialization node instructs the SDK to initialize the PingOne Protect Signals API with the configured properties.

> **Tip**
>
> Initialize the PingOne Protect Signals API as early in the journey as possible, before any user interaction. This enables it to gather sufficient contextual data to make an informed risk evaluation.

- The user enters their credentials, which are verified against the identity store.
- 2 The PingOne Protect Evaluation node performs a risk evaluation against a risk policy in PingOne.

The example journey continues depending on the outcome:

**`High`**

The journey requests that the user respond to a push notification.

**`Medium` *or* `Low`**

The risk is not significant, so no further authentication factors are required.

**`Exceeds Score Threshold`**

The score returned is higher than the configured threshold and is considered too risky to complete successfully.

`Failure`

The risk evaluation could not be completed, so the authentication attempt continues to the **Failure** node.

`BOT_MITIGATION`

The risk evaluation returned a recommended action to check for the presence of a human, so the journey continues to a CAPTCHA node.

`AITM_MITIGATION`

The risk evaluation returned a recommended action regarding the possible presence of an adversary-in-the-middle attack, so the journey continues to the Failure node.

`ClientError`

The client returned an error when attempting to capture the data to perform a risk evaluation, so the authentication attempt continues to the **Failure** node.

- 3 An instance of the PingOne Protect Result node returns the `Success` result to PingOne, which can be viewed in the console to help with analysis and risk policy tuning.
- 4 A second instance of the PingOne Protect Result node returns the `Failed` result to PingOne, which can be viewed in the console to help with analysis and risk policy tuning.

## PingOne Protect Initialization node

The **PingOne Protect Initialization** node instructs the SDK to initialize the embedded PingOne Protect SDK on the client device using the configuration provided by the node properties.

Learn more in Threat Protection using PingOne Protect.

You can only initialize the PingOne Protect SDK on the client device once. Attempting to initialize the SDK with a different configuration will not override the initial settings.

> **Tip**
>
> You should initialize the PingOne Protect SDK on the client device as early as possible so that it can gather sufficient contextual information to make risk evaluations.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node has no required predecessor nodes.

It does not read from the shared node state.

## Dependencies

This node requires a **PingOne Worker Service** configuration so that it can connect to your PingOne instance and send it the necessary data to make risk evaluations as part of the journey.

Find information on the configuration properties in the *PingOne Worker service* documentation for:

- PingAM
- Advanced Identity Cloud

The client application must be using Ping SDK 4.4.0 or later.

## Configuration

| Property | Usage |
| --- | --- |
| **PingOne Worker Service ID** | The ID of the PingOne worker service for connecting to PingOne. |
| **Enable SDK Logs** | When enabled, output SDK log messages in the developer console.<br>Default: Disabled |

| Property | Usage |
| --- | --- |
| **Device Attributes To Ignore** | A list of device attributes you want to exclude from the results when collecting device signals. These attributes will not be sent to PingOne to perform evaluations, which might limit its ability to create accurate results.<br>Some examples of attributes the client might obtain from the device include:<br>• `BATTERY_LEVEL`<br>• `CPU_ARCHITECTURE`<br>• `DEVICE_MODEL`<br>• `DEVICE_VENDOR`<br>• `GPS_SUPPORTED`<br>• `HAS_CHROME_APP`<br>• `IS_ACCEPT_COOKIES`<br>• `NAVIGATOR_USER_AGENT`<br>• `OS_NAME`<br>• `OS_VERSION`<br>• `RESOLUTION`<br>• `TOUCH_SUPPORT`<br>**Note**<br>The attributes collected vary depending on the OS of the client.<br>For example, an Android device might provide different attributes to a JavaScript app running on Windows. |
| **Custom Host** | Deprecated. We recommend that you do not change this property. |
| **Lazy Metadata** | When enabled, calculate metadata on demand.<br>When not enabled, metadata is calculated automatically after initialization.<br>Default: Disabled |
| **Collect Behavioral Data** | When enabled, collect behavioral data.<br>When not enabled, behavioral data is not collected.<br>Default: Enabled |
| **Disable Hub** | When selected, the client stores device data in the browser's `localStorage` only.<br>When not selected, an iframe is used.<br>Default: Not selected |
| **Device Key Rsync Intervals (days)** | Number of days that device attestation can rely upon the device fallback key.<br>Default: `14` |
| **Enable Trust** | Tie the device payload to a non-extractable crypto key stored in the browser for content authenticity verification |
| **Disable Tags** | When selected, the client does not collect tag data. Tags are used to record the pages the user visited, forming a browsing history.<br>Default: Not selected |

## Outputs

The node sends a `PingOneProtectInitializeCallback` to the client application.

The Ping SDKs consume this callback and initialize the PingOne Protect functionality so it can start gathering the data it needs to make risk evaluations.

## Outcomes

`True`

The client application confirmed successful receipt of the configuration.

`False`

The client application did not confirm successful receipt of the configuration or returned a client error.

## Example

The following example journey leverages PingOne Protect functionality to perform a risk evaluation on a client app. The client app is built using the Ping SDKs.

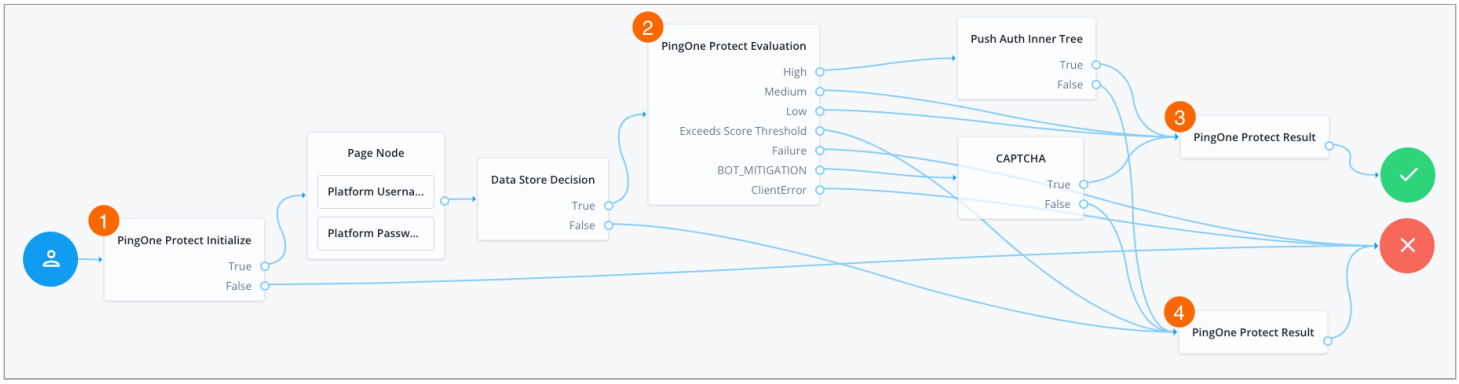



*Figure 1. Example PingOne Protect journey*

- 1 The PingOne Protect Initialization node instructs the SDK to initialize the PingOne Protect Signals API with the configured properties.

> **Tip**
>
> Initialize the PingOne Protect Signals API as early in the journey as possible, before any user interaction. This enables it to gather sufficient contextual data to make an informed risk evaluation.

- The user enters their credentials, which are verified against the identity store.
- 2 The PingOne Protect Evaluation node performs a risk evaluation against a risk policy in PingOne.

The example journey continues depending on the outcome:

**`High`**

The journey requests that the user respond to a push notification.

**`Medium` *or* `Low`**

The risk is not significant, so no further authentication factors are required.

**`Exceeds Score Threshold`**

The score returned is higher than the configured threshold and is considered too risky to complete successfully.

**`Failure`**

The risk evaluation could not be completed, so the authentication attempt continues to the **Failure** node.

**`BOT_MITIGATION`**

The risk evaluation returned a recommended action to check for the presence of a human, so the journey continues to a CAPTCHA node.

**`AITM_MITIGATION`**

The risk evaluation returned a recommended action regarding the possible presence of an adversary-in-the-middle attack, so the journey continues to the Failure node.

**`ClientError`**

The client returned an error when attempting to capture the data to perform a risk evaluation, so the authentication attempt continues to the **Failure** node.

- 3 An instance of the PingOne Protect Result node returns the `Success` result to PingOne, which can be viewed in the console to help with analysis and risk policy tuning.
- 4 A second instance of the PingOne Protect Result node returns the `Failed` result to PingOne, which can be viewed in the console to help with analysis and risk policy tuning.

## PingOne Protect Result node

The **PingOne Protect Result** node updates the risk evaluation configuration or modifies the completion status of the resource while the risk evaluation is still in progress.

You can check the results of the evaluation in the PingOne admin console by filtering for `Risk Evaluation Updated` event types.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node requires that you've initialized PingOne Protect in your client application. For example, by using a PingOne Protect Evaluation node previously in the journey or by initializing the SDK within the app itself.

## Dependencies

This node requires a **PingOne Worker Service** configuration so that it can connect to your PingOne instance and send it the necessary data to make risk evaluations as part of the journey.

## Configuration

| Property | Usage |
| --- | --- |
| **Completion Status** | Report the status of the journey back to PingOne.<br>Choose from:<br>• `FAILED`<br>• `SUCCESS` |

## Outputs

This node does not change the shared node state.

## Outcomes

Single outcome path.

The node attempts to update the PingOne server but continues along the single outcome without confirming the server received the update.

## Example

The following example journey leverages PingOne Protect functionality to perform a risk evaluation on a client app. The client app is built using the Ping SDKs.

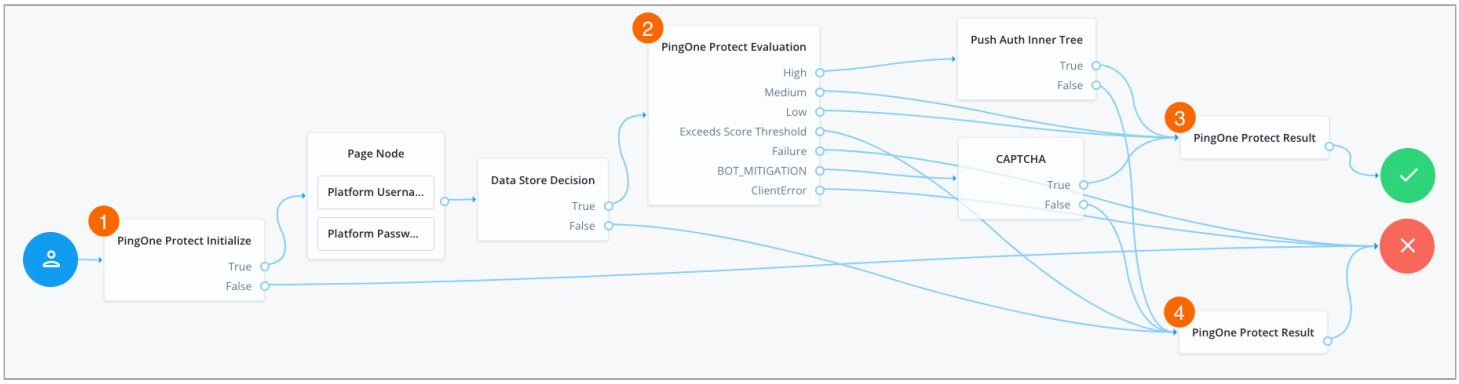



*Figure 1. Example PingOne Protect journey*

- 1 The PingOne Protect Initialization node instructs the SDK to initialize the PingOne Protect Signals API with the configured properties.

> **Tip**
>
> Initialize the PingOne Protect Signals API as early in the journey as possible, before any user interaction. This enables it to gather sufficient contextual data to make an informed risk evaluation.

- The user enters their credentials, which are verified against the identity store.
- 2 The PingOne Protect Evaluation node performs a risk evaluation against a risk policy in PingOne.

The example journey continues depending on the outcome:

**`High`**

The journey requests that the user respond to a push notification.

**`Medium` *or* `Low`**

The risk is not significant, so no further authentication factors are required.

**`Exceeds Score Threshold`**

The score returned is higher than the configured threshold and is considered too risky to complete successfully.

**`Failure`**

The risk evaluation could not be completed, so the authentication attempt continues to the **Failure** node.

**`BOT_MITIGATION`**

The risk evaluation returned a recommended action to check for the presence of a human, so the journey continues to a CAPTCHA node.

**`AITM_MITIGATION`**

The risk evaluation returned a recommended action regarding the possible presence of an adversary-in-the-middle attack, so the journey continues to the Failure node.

`ClientError`

The client returned an error when attempting to capture the data to perform a risk evaluation, so the authentication attempt continues to the **Failure** node.

- 3 An instance of the PingOne Protect Result node returns the `Success` result to PingOne, which can be viewed in the console to help with analysis and risk policy tuning.
- 4 A second instance of the PingOne Protect Result node returns the `Failed` result to PingOne, which can be viewed in the console to help with analysis and risk policy tuning.

# Behavioral nodes

## Increment Login Count node

The **Increment Login Count** node increments the successful login count property of a managed object.

Use the Login Count Decision node to change the flow of the journey based on the count.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

This node's **Identity Attribute** specifies the property it requires in the incoming node state. It uses this property to access the managed object.

### Dependencies

This node depends on the underlying identity service (PingIDM) to store the managed object.

### Configuration

| Property | Usage |
|---|---|
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

### Outputs

This node does not change the shared node state.

## Outcomes

Single outcome path; on success, this node increments the managed object's `loginCount`.

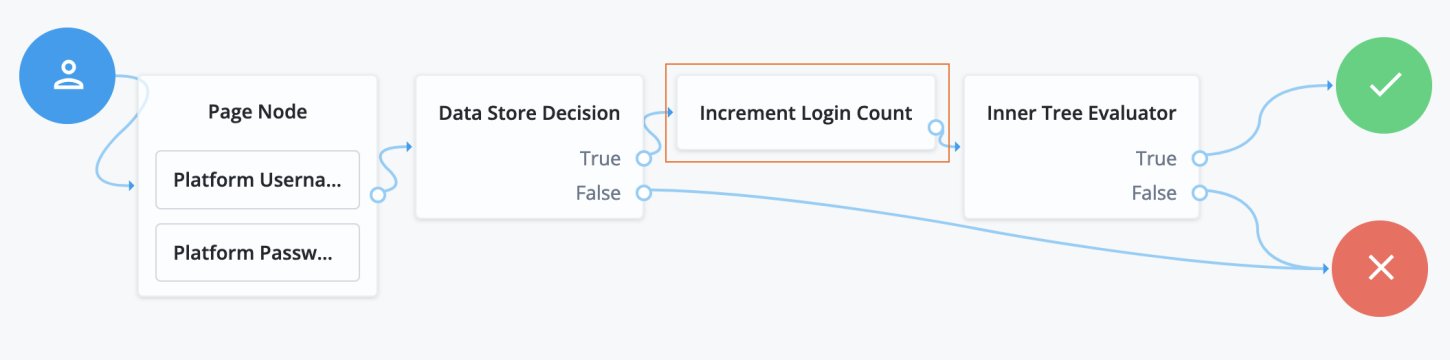
## Errors

If this node fails to access the managed object, it throws an exception with a `No object to increment` message.

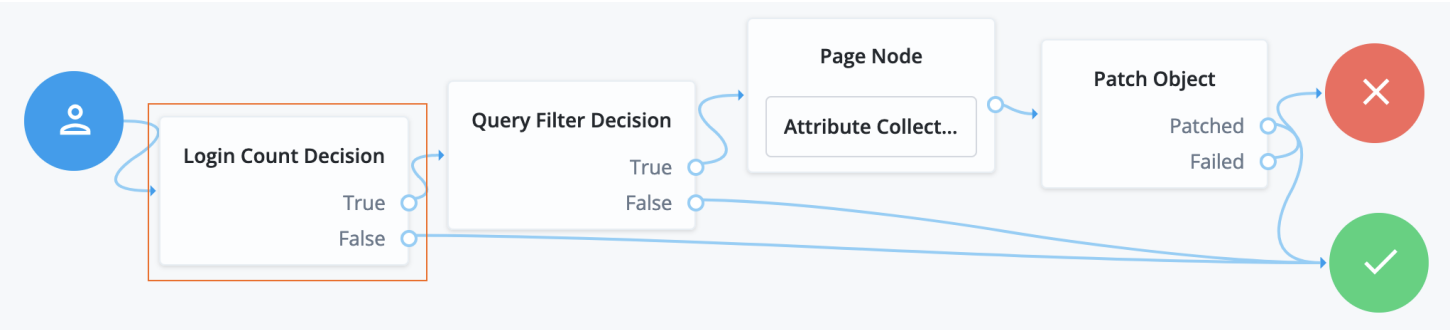
If this node fails to increment the login count, it logs an `Unable to increment login count` warning message.

## Example

The following journey uses the Increment Login Count node to update the login count on successful authentication:




- The Platform Username node injects the `userName` into the shared node state.
- The Data Store Decision node determines whether authentication is successful.
- The Increment Login Count node (outlined in the image) updates the login count.
- The Inner Tree Evaluator node invokes the following nested journey for progressive profiling:




- The Login Count Decision node triggers the rest of the journey depending on the login count and its settings.
- The Query Filter Decision node determines whether managed object profile fields are still missing.
- The Page node requests additional input for the profile.
- The Patch Object node stores the additional input in the managed object profile.

# Login Count Decision node

The **Login Count Decision** node triggers an action when a user's successful login count property reaches a specified number.

Use the Increment Login Count node to set the login count on successful authentication.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

This node's **Identity Attribute** specifies the property it requires in the incoming node state. It uses this property to access the managed object.

## Dependencies

This node depends on the underlying identity service (PingIDM) to store the managed object.

## Configuration

| Property | Usage |
|---|---|
| **Interval** | Trigger the `True` outcome depending on this setting, the **Amount**, and the login count:<br>**AT**<br>Proceed to `True` when the login count matches the **Amount** setting.<br>**EVERY**<br>Proceed to `True` every time the login count reaches a multiple of the **Amount** setting.<br>Default: `AT` |
| **Amount** | The login count to trigger a `True` outcome depending on the **Interval**.<br>Default: 25 |

| Property | Usage |
| --- | --- |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

## Outputs

This node does not change the shared node state.

## Outcomes

### `True`

The login count reached **Amount**, and the **Interval** setting triggered this outome.

### `False`

All other cases.

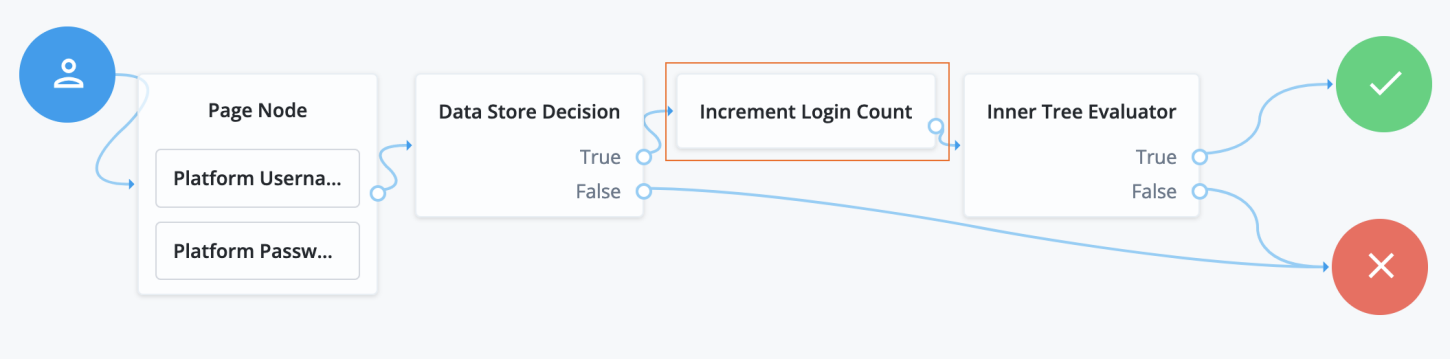
## Errors

This node can throw exceptions with the following messages:

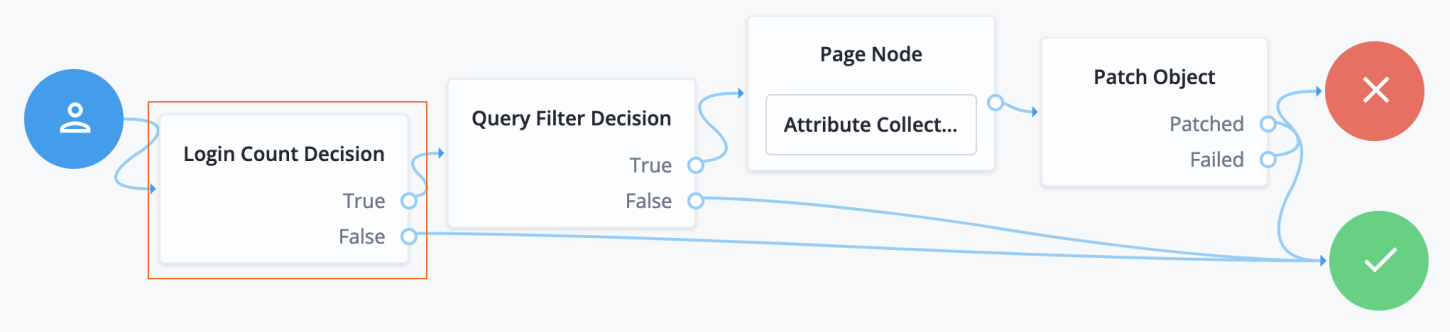
| Message | Notes |
| --- | --- |
| `<Identity Attribute> not present in state` | Failed to read the specified **Identity Attribute** in the shared node state |
| `Failed to retrieve existing object` | Failed to find the managed object using the **Identity Attribute** value from the shared node state |
| `Retrieve login not found` | Failed to read the managed object's login count |

## Example

The following journey uses the Increment Login Count node to update the login count on successful authentication:




- The Platform Username node injects the `userName` into the shared node state.
- The Data Store Decision node determines whether authentication is successful.
- The Increment Login Count node (outlined in the image) updates the login count.
- The Inner Tree Evaluator node invokes the following nested journey for progressive profiling:




- The Login Count Decision node triggers the rest of the journey depending on the login count and its settings.
- The Query Filter Decision node determines whether managed object profile fields are still missing.
- The Page node requests additional input for the profile.
- The Patch Object node stores the additional input in the managed object profile.

# Contextual nodes

## Certificate Collector node

The **Certificate Collector** node collects an X.509 digital certificate from the request. The journey can use the collected certificate as authentication credentials for a user or OAuth 2.0 client.

PingAM accepts certificates in PEM and DER format.

PingOne Advanced Identity Cloud accepts certificates in DER format.

> **Note**
>
> You can't use this node in isolation because it only *collects* the certificate from the request. It doesn't extract or validate the certificate's content. Use a Certificate Validation node to validate the certificate and a Certificate User Extractor node to extract the user details from the certificate.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads the certificate from the request payload or in a request header. It doesn't read anything from the shared state.

### Dependencies

This node has no dependencies.

## Configuration

| Property | Usage |
| --- | --- |
| **Certificate Collection Method** | How the node should collect the certificate from the request. Possible values are:<br>**`Request`**<br>The node locates the certificate in the request. Use this option if TLS termination happens at AM.<br>**`Header`**<br>The node locates the certificate in an HTTP header. Specify the header name in the **HTTP Header Name for the Client Certificate** property. Use this option if TLS termination happens in a proxy or load balancer outside AM.<br>**`Either`**<br>The node attempts to locate the certificate in the request. If there's no certificate in the request, the node attempts to locate the certificate in the HTTP header specified in **HTTP Header Name for the Client Certificate**.<br>Default: `Either` |
| **HTTP Header Name for the Client Certificate** | The name of the HTTP header that contains the certificate. If you set the **Certificate Collection Method** to `Header` or `Either`, you must set a value here.<br>Default: No value |
| **Trusted Remote Hosts** | A list of IP addresses trusted to supply certificates on behalf of the authenticating client, such as load balancers doing TLS termination.<br>If you don't set a value here, AM rejects certificates supplied by remote hosts. If you set a value of `any`, AM trusts certificates supplied by any remote host, on behalf of the authenticating client.<br>Default: No value |

## Outcomes

**`Collected`**

The node was able to collect the certificate.

**`Not Collected`**

The node was unable to collect the certificate.

## Outputs

The node outputs the X509 certificate to the transient state to be consumed by the Certificate Validation node.

## Errors

- If no certificate is provided in the configured location (either header or request), the node logs the following error:

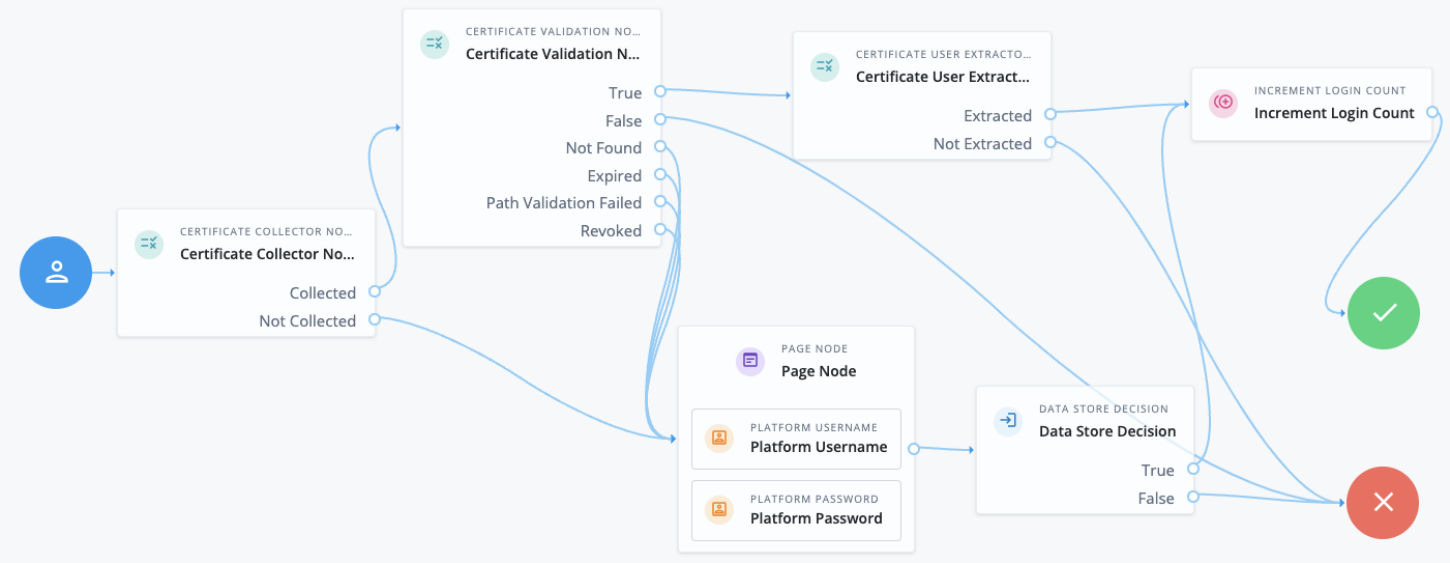
  `Certificate was not successfully collected based on node configuration and client request`

- If there's a problem with the certificate provided in the header or in the request, the node logs the following error:

  `CertificateFromParameter decode failed, possibly invalid Base64 input`

## Example

This example shows an authentication journey using a certificate as credentials.




1. The **Certificate Collector** node attempts to collect the certificate from the request body or the header.
   - If the node can collect the certificate, the journey proceeds to the Certificate Validation node.
   - If the node can't collect the certificate, the journey proceeds to a Page node containing a Platform Username node and a Platform Password node to let the user authenticate with username/password credentials.
2. The **Certificate Validation** node attempts to validate the certificate based on the configuration of that node.
   - If the certificate can be validated, the journey proceeds to the Certificate User Extractor node.
   - If the certificate is invalid, the journey proceeds to the Failure node.
   - In all other cases, the journey proceeds to a Page node containing a Platform Username node and a Platform Password node to let the user authenticate with username/password credentials.
3. The **Certificate User Extractor** node extracts the user ID from the certificate and attempts to find a match in the identity store.
   - If the username can be extracted and a matching user is found in the identity store, the journey increments the login count and authenticates the user.

- If the username can't be extracted or no matching user is found in the identity store, the journey proceeds to the Failure node.

## Certificate User Extractor node

The **Certificate User Extractor** node extracts an identifier from the certificate collected by the Certificate Collector node and searches for that identifier in the identity store. The purpose of this node is to match the collected certificate with a user in the identity store.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads the value of the `X509Certificate` property from the transient state.

Implement the Certificate Collector node as input to this node to obtain the `X509Certificate`.

### Dependencies

This node has no dependencies.

## Configuration

| Property | Usage |
| --- | --- |
| Certificate Field Used to Access User Profile | Specifies the field in the certificate that AM uses to search for the user in the identity store. Possible values are:<br>• `Subject DN`<br>• `Subject CN`<br>• `Subject UID`<br>• `Email Address`<br>• `Other`<br>• `None`<br>If you select `Other`, provide an attribute name in the **Other Certificate Field Used to Access User Profile** property.<br>Select `None` if you want to specify an alternate way of looking up the user profile in the **SubjectAltNameExt Value Type to Access User Profile** property.<br>Default: `Subject CN` |
| Other Certificate Field Used to Access User Profile | Specifies a custom certificate field to use as the base of the user search. |
| SubjectAltNameExt Value Type to Access User Profile | Specifies how to look up the user profile:<br>**None**<br>AM uses the value specified in the **Certificate Field Used to Access User Profile** or the **Other Certificate Field Used to Access User Profile** properties when looking up the user profile.<br>**RFC822Name**<br>AM looks up the user profile using the value of the `RFC822Name` field.<br>**UPN**<br>AM looks up the user profile as the User Principal Name attribute used in Active Directory.<br>Default: `None` |

## Outcomes

### `Extracted`

The node extracted the user ID from the certificate and found a match in the identity store.

### `Not Extracted`

The node couldn't extract the user ID from the certificate or couldn't match the ID to an identity in the identity store.

## Outputs

If the node can extract a value from the certificate, that value is stored in the `username` key in the shared node state.

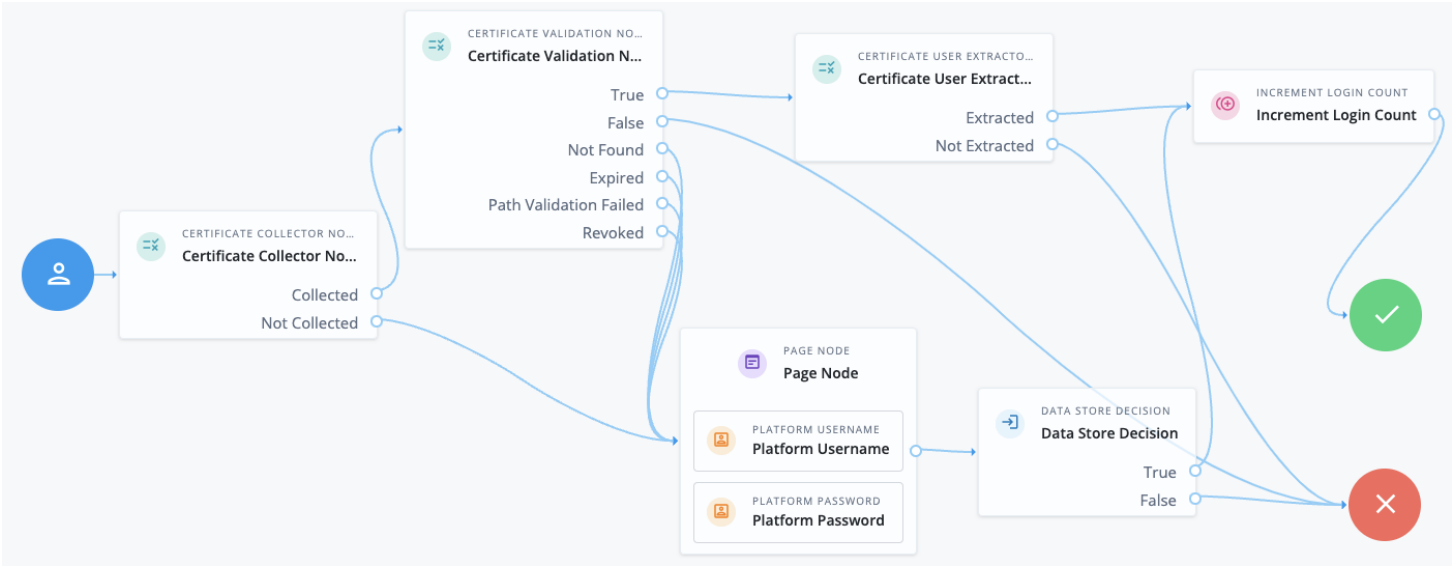
## Errors

If the node can't extract the user ID from the certificate, it logs the following error:

```
Unable to parse user token ID from Certificate
```

## Example

This example shows an authentication journey using a certificate as credentials.




1. The **Certificate Collector** node attempts to collect the certificate from the request body or the header.
    - If the node can collect the certificate, the journey proceeds to the Certificate Validation node.
    - If the node can't collect the certificate, the journey proceeds to a Page node containing a Platform Username node and a Platform Password node to let the user authenticate with username/password credentials.
2. The **Certificate Validation** node attempts to validate the certificate based on the configuration of that node.
    - If the certificate can be validated, the journey proceeds to the Certificate User Extractor node.
    - If the certificate is invalid, the journey proceeds to the Failure node.
    - In all other cases, the journey proceeds to a Page node containing a Platform Username node and a Platform Password node to let the user authenticate with username/password credentials.
3. The **Certificate User Extractor** node extracts the user ID from the certificate and attempts to find a match in the identity store.
    - If the username can be extracted and a matching user is found in the identity store, the journey increments the login count and authenticates the user.

- If the username can't be extracted or no matching user is found in the identity store, the journey proceeds to the Failure node.

## Certificate Validation node

The **Certificate Validation** node validates a digital X.509 certificate collected by the Certificate Collector node.

**Important**

**Certificate validation rules**

- If you add this node to a journey, you *must* configure it. With no configuration, the node returns `True` as the outcome by default, regardless of the validity of the certificate provided in the journey.
- This node validates the *first* certificate in a certificate chain (the user certificate) and ignores the remaining certificates in the chain.
- If the collected user certificate is a self-signed certificate (test environments only), the self-signed user certificate must be present in the truststore for certificate validation to succeed.
- If the collected user certificate is signed by a valid issuer, the issuing certificates (intermediate, or intermediate and root) must be present in the truststore for certificate validation to succeed.
- If the user certificate is signed by a valid issuer and the issuing certificate (intermediate certificate) is *not* present in the truststore, certificate validation fails.
- The node uses the intermediate and user certificates to verify certificate revocation status.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires an `X509Certificate` property in the incoming node state.

Implement the Certificate Collector node as input to the Certificate Validation node.

## Configuration

| Property | Usage |
|---|---|
| **Match Certificate in LDAP** | When enabled, AM matches the collected certificate with a certificate stored in the identity store.<br>Set the **Subject DN Attribute Used to Search LDAP for Certificates** to specify which LDAP property to search for certificate information.<br>Default: Disabled |
| **Check Certificate Expiration** | When enabled, AM checks if the collected certificate has expired.<br>Default: Disabled |
| **Subject DN Attribute Used to Search LDAP for Certificates** | The attribute AM uses to search the identity store for the certificate. The search filter is based on this attribute and the value of the Subject DN as it appears in the certificate.<br>Default: `CN` |
| **Match Certificate to CRL** | When enabled, AM checks if the collected certificate has been revoked according to a Certificate Revocation List (CRL) in the identity store.<br>Define related CRL properties later in the node configuration.<br>Default: Disabled. |
| **Issuer DN Attribute(s) Used to Search LDAP for CRLs** | The name of the attribute or attributes in the issuer certificate that AM uses to locate the CRL in the identity store.<br>• If you specify only one attribute here, the LDAP search filter used is `(attr-name=attr-value-in-subject-DN)`.<br>For example, if the subject DN of the issuer certificate is `C=US, CN=Some CA, serialNumber=123456`, and the attribute specified is `CN`, AM uses a search filter of `(CN=Some CA)` to locate the CRL.<br>• Specify several CRLs for the same CA issuer in a comma-separated list (`,`) where the names are in the same order in which they appear in the subject DN.<br>In this case, the LDAP search filter used is `(attr1=attr1-value-in-subject-DN,attr2=attr2-value-in-subject-DN,…)`, and so on.<br>For example, if the subject DN of the issuer certificate is `C=US, CN=Some CA, serialNumber=123456`, and the attributes specified are `CN,serialNumber`, the LDAP search filter used to find the CRL is `(CN=Some CA,serialNumber=123456)`.<br>Default: `CN` |
| **HTTP Parameters for CRL Update** | Parameters AM includes in any HTTP CRL call to the CA that issued the certificate. If the client or CA certificate includes the `IssuingDistributionPoint` extension, AM uses this information to retrieve the CRL from the distribution point.<br>Add the parameters as key-value pairs in a comma-separated list (`,`). For example, `param1=value1,param2=value2`. |

| Property | Usage |
| --- | --- |
| **Cache CRLs in Memory** | When enabled, AM caches CRLs in memory.<br>If this option is enabled, **Update CA CRLs from CRLDistributionPoint** must also be enabled.<br>Default: Enabled |
| **Update CA CRLs from CRLDistributionPoint** | When enabled, AM fetches new CA CRLs from the CRL Distribution Point and updates them in the identity store. If the CA certificate includes either the `IssuingDistributionPoint` or the `CRLDistributionPoint` extensions, AM attempts to update the CRLs when they're out of date.<br>Default: Enabled |
| **OCSP Validation** | When enabled, AM checks the validity of certificates using the Online Certificate Status Protocol (OCSP).<br>PingAM If you enable this option, the AM instance must be able to connect to the internet. You must also configure OCSP for AM under **Configure > Server Defaults > Security > Online Certificate Status Protocol Check**.<br>Default: Disabled |
| **Certificate Identity Store** | PingAM Select the identity store (configured for the realm) that AM must search for certificates. If you select an identity store here, AM uses the connection details defined for that identity store and ignores all the server settings below this field.<br>PingOne Advanced Identity Cloud Select the default identity store ( `OpenDJ` ) from the list. You *must* select this identity store to let AM search for the certificate. AM ignores all LDAP server settings below this field. |
| PingAM **LDAP Server Where Certificates are Stored** | The LDAP server that holds certificates. Enter the server details in the format `ldap-server:port` .<br>To associate multiple AM servers in a site with corresponding LDAP servers, use the format `am_server\|ldapserver:_port_` . For example, `am.example.com\|ldap1.example.com:636` . |
| PingAM **LDAP Search Start or Base DN** | Valid base DN for the LDAP search, such as `dc=example,dc=com` . To associate AM servers with different search base DNs, use the format `am_server\|base_dn` . For example, `am.example.com\|dc=example,dc=com` `openam1.test.com\|dc=test,dc=com` . |
| PingAM **LDAP Server Authentication User** and **LDAP Server Authentication Password** | The credentials used to connect to the LDAP directory that holds the certificates. If you enable mTLS, the node ignores these credentials.<br>Default Authentication User: `cn=Directory Manager` |

| Property | Usage |
| --- | --- |
| PingAM **mTLS Enabled** | Enables mTLS (mutual TLS) between AM and the directory server.<br>When mTLS is enabled, the node ignores the values for **LDAP Server Authentication User** and **LDAP Server Authentication Password**.<br>If you enable this property, you must:<br>• Enable **Use SSL/TLS for LDAP Access**.<br>• Provide an **mTLS Secret Label Identifier**.<br>Default: Disabled |
| PingAM **mTLS Secret Label Identifier** | An identifier used to create a secret label for mapping to the mTLS certificate in the secret store. AM uses this identifier to create a specific secret label for this node. The secret label takes the form `am.authentication.nodes.certificate.validation.mtls.identifier.cert`, where `identifier` is the value of **mTLS Secret Label Identifier**. The label can only contain alphanumeric characters ( `a-z`, `A-Z`, `0-9` ) and periods ( `.` ). It can't start or end with a period.<br>For greater security, you should rotate certificates periodically. When you rotate a certificate, update the corresponding mapping in the realm secret store configuration to reflect this identifier. When you rotate a certificate, AM closes any existing connections using the old certificate. A new connection is selected from the connection pool and no server restart is required. |
| PingAM Use SSL/TLS for LDAP Access | When enabled, AM uses SSL/TLS to access the LDAP directory. Make sure that AM trusts the certificate from the LDAP server when enabling this option.<br>Default: Disabled |

## Outputs

This node doesn't put anything into the shared state.

## Outcomes

### `True`

The node could validate the certificate.

When the outcome is `True`, add a Certificate User Extractor node to extract the values of the certificate.

### `False`

The node couldn't validate the certificate. The journey follows this path when the node can't validate the certificate and no more specific outcome is available.

### `Not found`

The **Match Certificate in LDAP** property is enabled, but the certificate wasn't found in the LDAP store.

### `Expired`

The **Check Certificate Expiration** property is enabled, and the certificate has expired.

### `Path Validation Failed`

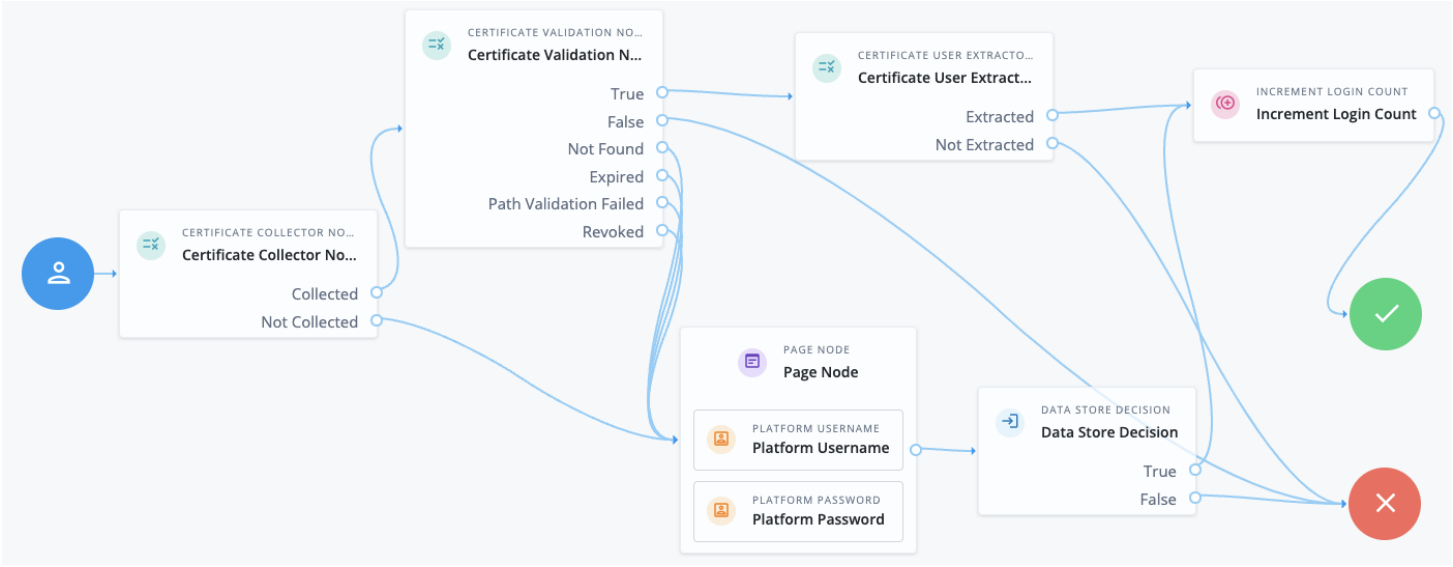
The **Match Certificate to CRL** property is enabled, and the certificate path is invalid.

### Revoked

The **OCSP Validation** property is enabled, and the certificate has been revoked.

## Example

This example shows an authentication journey using a certificate as credentials.




1. The **Certificate Collector** node attempts to collect the certificate from the request body or the header.
    - If the node can collect the certificate, the journey proceeds to the Certificate Validation node.
    - If the node can't collect the certificate, the journey proceeds to a Page node containing a Platform Username node and a Platform Password node to let the user authenticate with username/password credentials.
2. The **Certificate Validation** node attempts to validate the certificate based on the configuration of that node.
    - If the certificate can be validated, the journey proceeds to the Certificate User Extractor node.
    - If the certificate is invalid, the journey proceeds to the Failure node.
    - In all other cases, the journey proceeds to a Page node containing a Platform Username node and a Platform Password node to let the user authenticate with username/password credentials.
3. The **Certificate User Extractor** node extracts the user ID from the certificate and attempts to find a match in the identity store.
    - If the username can be extracted and a matching user is found in the identity store, the journey increments the login count and authenticates the user.

- If the username can't be extracted or no matching user is found in the identity store, the journey proceeds to the Failure node.

## Cookie Presence Decision node

Checks that a named cookie is present in the incoming authentication request.

This node does not check the value of the named cookie, only that it exists.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `True`
- `False`

### Properties

| Property | Usage |
|---|---|
| Name of Cookie *(required)* | Evaluation continues along the `True` path if the named cookie is present in the incoming authentication request; otherwise, evaluation continues along the `False` path. |

## Device Geofencing node

Compares any collected device location metadata with the trusted locations configured in the authentication node.

Use this node with the Device Profile Collector node to determine if the authenticating user's device is located within range of configured, trusted locations.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `Inside`
- `Outside`

Evaluation continues along the `Inside` path if the collected location is within the specified range of a configured trusted location; otherwise, evaluation continues along the `Outside` path.

### Properties

| Property | Usage |
|---|---|
| Trusted Locations *(required)* | Specify the latitude and longitude of at least one trusted location. Separate the values with a comma. For example, `37.7910855,-122.3951663`. |
| Geofence Radius (km) | Specifies the maximum distance, in kilometers, that a device can be from a configured trusted location.<br>The distance is calculated point-to-point. |

## Device Location Match node

Compares any collected device location metadata with that stored in the user's profile.

Use this node with the Device Profile Collector node to determine if the authenticating user's device is located within range of somewhere they have authenticated from, and saved, previously.

You must establish the identity of the user before attempting to match locations.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |

| Product | Available? |
| --- | --- |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `True`
- `False`
- `Unknown Device`

Evaluation continues along the `True` path if the collected location is within the specified range of saved location data; otherwise, evaluation continues along the `False` path.

If the user has no saved device profiles or the identity of the user has not been established, evaluation continues along the `Unknown Device` path.

### Properties

| Property | Usage |
| --- | --- |
| Maximum Radius (km) | Specifies the maximum distance, in kilometers, that a device can be from a previously saved location.<br>The distance is calculated point-to-point. |

## Device Match node

The **Device Match** node compares collected device metadata with that stored in the user's profile.

Use this node with the Device Profile Collector node to check whether the user is authenticating with a previously saved, trusted device.

The Device Match node supports the following methods of comparison:

### *Built-in matching*

The node handles the comparison and matching. You configure the acceptable variance and the maximum age for device profiles.

### *Custom matching*

Create scripts to compare captured device data against trusted device profiles.

For a customizable template script, follow these steps:

Advanced Identity Cloud

1. In the Advanced Identity Cloud admin UI, go to **Scripts** > **Auth Scripts**.
2. Click **Device Profile Match Template - Decision Node Script**.

PingAM

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Scripts**.
2. Click **Device Profile Match Template - Decision Node Script**.

**Tip**

You can find a comprehensive sample script (with usage instructions) and a development toolkit in the GitHub sample repository.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads the `username` from the shared state to look up saved device profiles in the user's account.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

This node also reads collected device metadata from the shared state. Implement a Device Profile Collector node earlier in the journey to collect metadata for the current device.

If **Use Custom Matching Script** is enabled, the inputs depend on the script.

### Uniquely identify devices

The Device Match node looks up a user's stored device profiles using a device identifier as a key.

The client device generates a device identifier as part of the device profile that it returns to the Device Profile Collector node in the JSON payload.

For example:

```
{
  "identifier": "d50cdb5ce8d055a3-86bd35e1b975a14d76b40940112c2380264c8efd",
  ....
}
```

**When can identifiers change?**

If the identifier changes, the Device Match node can't match any stored device profiles.

If this happens, your journey must collect and store a new device profile that contains the new identifier.

This section explains what can cause an identifier to change on each platform.

### Android

In Android, the instance ID is deleted or changes if any of the following occurs:

- An app is restored on a new device.
- The user uninstalls and reinstalls the app.
- The user clears app data.

### iOS

On iOS, the device ID is stored in the Keychain. This means the ID persists when the app is removed.

However, the device ID is deleted or changes if any of the following occurs:

- The user wipes or factory resets the phone.
- The user migrates to a new phone.
- The keychain is programmatically deleted from the phone.
- The device ID is programmatically deleted from the Keychain.
- The keychain identifier in the `forgerock_keychain_access_group` configuration property changes.

JavaScript

In JavaScript, the device ID is deleted or changes if any of the following occurs:

- The browser window creates the device ID while in "private" or "incognito" mode. Closing the browser removes the ID.
- The browser removes the ID when cleaning up old data to accommodate new data.
- The browser is uninstalled and reinstalled. The ID is removed.
- The user removes the device ID by clearing the browser data.

## Dependencies

If **Use Custom Matching Script** is enabled, the dependencies depend on the script.

## Configuration

| Property | Usage |
|---|---|
| **Acceptable Variance** | The maximum number of acceptable device attribute differences for a match.<br>Default: `0` (all attributes must match) |
| **Expiration** | The maximum age in days a saved profile is valid for comparison.<br>The node ignores older device profiles saved to the user's account when comparing device profiles with the collected metadata.<br>Default: `30` (days) |
| **Use Custom Matching Script** | Enable this option to use a custom script instead of built-in matching to compare the collected metadata with saved device profiles.<br>When enabled, the node ignores the **Acceptable Variance** and **Expiration** settings.<br>The script type must be `Journey Decision Node` (Advanced Identity Cloud or Ping Identity Platform) or `Decision node script for authentication trees` (standalone PingAM).<br>Default: false |
| **Custom Matching Script** | Select the custom script to use when **Use Custom Matching Script** is enabled.<br>Only scripts of type `Journey Decision Node` (Advanced Identity Cloud or Ping Identity Platform) or `Decision node script for authentication trees` (standalone PingAM) appear in the list.<br>Default: `Authentication Tree Decision Node Script` |

## Outputs

This node does not change the shared state on its own.

If the node uses a **Custom Matching Script**, the output is determined by the script.

## Outcomes

### `True`

The collected device metadata matches a saved profile within the configured variance.

### `False`

The collected device metadata doesn't match a saved profile, or another error occurred.

### `Unknown Device`

The journey follows this outcome path in the following situations:

- The user has no saved trusted device profiles.
- The user identity hasn't yet been established.
- The acceptable device variance matches, but the device ID no longer matches.

## Errors

This node logs the following warning messages:

### `script outcome error`

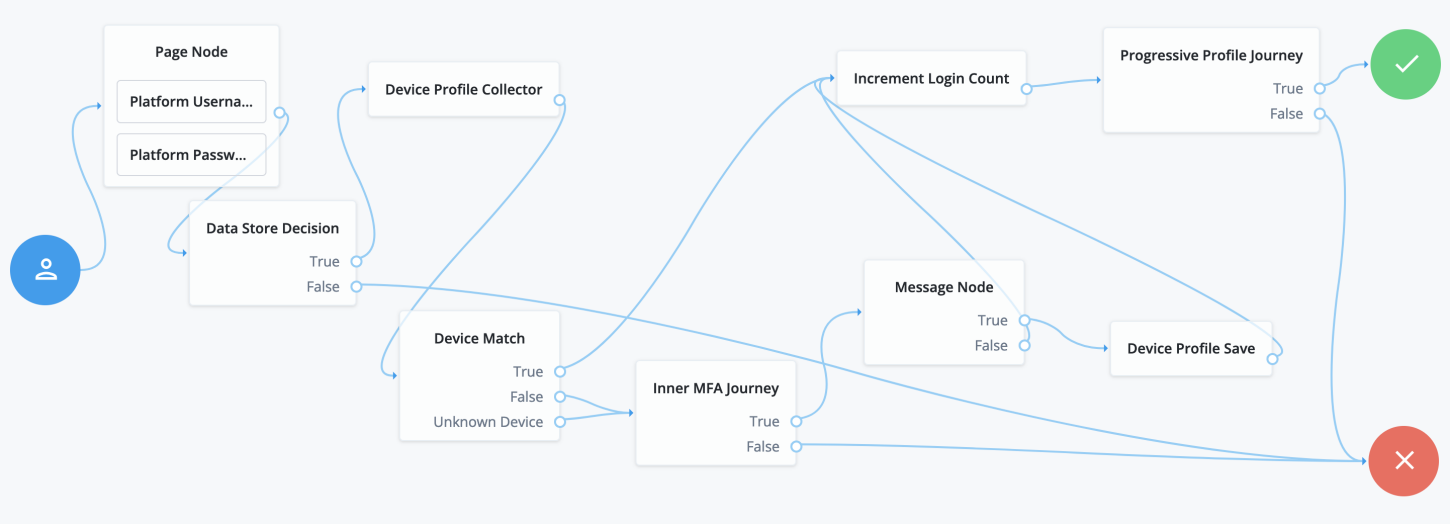
The script failed to set the `outcome` field to a string.

### `error evaluating the script`

The script failed to complete. Refer to the logs for details.

## Example

The following journey authenticates the user and checks whether the current device is trusted. If the device isn't trusted yet, the journey requires an additional authentication factor and lets the user opt to trust the device:




- The Page node with the Platform Username node and Platform Password node prompt the user for their credentials.
- The Data Store Decision node confirms the user's credentials.
- The Device Profile Collector node collects metadata about the current device.
- The **Device Match** node compares saved device profiles with the current device.
- The **Inner MFA Journey**, an Inner Tree Evaluator node, requires an additional authentication factor.
- The Message node prompts the user with an option to trust the current device.
- The Device Profile Save node saves the current device profile.
- The Increment Login Count node updates the number of successful authentications.
- The **Progressive Profile Journey**, an Inner Tree Evaluator node, invokes a journey to collect additional profile data.

## Device Profile Collector node

Gathers metadata about the device used to authenticate.

The node sends a `DeviceProfileCallback` callback.

Find more information in the documentation on interactive callbacks for:

- PingAM
- Advanced Identity Cloud

When used with the Ping SDKs, the node can collect the following:

### *Device Metadata*

Information such as the platform, versions, device name, hardware information, and the brand of the device being used.

The captured data is in JSON format, and stored in the authentication shared state in a variable named `forgeRock.device.profile`.

### *Device Location*

Provides the last known latitude and longitude of the device's location.

The captured data is in JSON format, and stored in the authentication shared state in a variable named `forgeRock.device.location`.

The collection of geographical information requires end-user approval. A browser function drives this process. A pop-up displays, prompting for access to share the geographical location. The browser connection must be secure.

> **Important**
>
> It is up to you what information you collect from users and devices.
> Always use data responsibly and provide your users with appropriate control over data they share with you.
> You are responsible for complying with any regulations or data protection laws.

In addition to the collected metadata, an `identifier` string in the JSON uniquely identifies the device.

Use this node with the Device Profile Save node to create a trusted profile from the collected data. You can use the trusted device profile in subsequent authentication attempts; for example, with the Device Match node and Device Location Match node.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

Single outcome path.

## Properties

| Property | Usage |
|---|---|
| Maximum Profile Size (KB) | Specifies the maximum accepted size, in kilobytes, of a device profile.<br>If the collected profile data exceeds this size, authentication fails.<br>Default: `3` |
| Collect Device Metadata | Specifies whether device metadata is requested. |
| Collect Device Location | Specifies whether device location is requested. |

| Property | Usage |
| --- | --- |
| Message | Specifies an optional message to display to the user while the node collects the requested data.<br>You can provide the message in multiple languages by specifying the locale in the `KEY` field. For example, `en-US`.<br>The locale selected for display is based on the user's locale settings in their browser.<br>Messages provided in the node override the defaults provided by AM. |

## Device Profile Save node

Persists collected device data to a user's profile in the identity store.

Use this node with the Device Profile Collector node to reuse the collected data in future authentications; for example, with the Device Match node and Device Location Match node.

You must establish the identity of the user before attempting to save to their profile.

A user profile can contain multiple device profiles. Use the **Maximum Saved Profiles** property to configure the maximum number of device profiles to persist per user. Saving a device profile with the same identifier as an existing entry overwrites the original record, and does not increment the device profile count.

- In Advanced Identity Cloud and Ping Identity Platform deployments, the end user UI displays saved device profiles to end users.
- In an AM standalone deployment, the PingAM UI **does not** display saved device profiles to end users.

  You can manage device profiles over REST, by using the `/json/users/user/devices/profile` endpoint for the realm.

  Use the AM API Explorer for detailed information about the parameters supported by the `/devices/profile` endpoint and to test it against your deployed AM instance.

  In the AM admin UI, select the **Help** icon, and then go to **API Explorer > /users > /{user} > /devices > /profile**.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Device Name Variable | The name of a variable in the shared node state that contains an alias label for the device profile. |
| Maximum Saved Profiles | The maximum number of device profiles to save in a user's profile.<br>When the maximum is reached, saving a new profile replaces the least-recently used profile. |
| Save Device Metadata | Whether device metadata is saved to the user's profile. |
| Save Device Location | Whether device location metadata is saved to the user's profile. |

## Device Tampering Verification node

Specifies a threshold for deciding if the device has been tampered with; for example, if it has been rooted or jailbroken.

The device scores between zero and one, based on the likelihood that is has been tampered with or may pose a security risk. For example, an emulator scores the maximum of `1`.

Use this node with the Device Profile Collector node to retrieve the tampering score from the device.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- `Not Tampered`
- `Tampered`

Evaluation continues along the `Not Tampered` path if the device scores less than or equal to the configured threshold; otherwise, evaluation continues along the `Tampered` path.

### Properties

| Property | Usage |
| --- | --- |
| Score Threshold | Specifies the score threshold for determining if a device has been tampered with.<br>Enter a decimal fraction, between `0` and `1`. For example, `0.75`.<br>The higher the score returned from the device, the more likely the device is jailbroken, rooted, or is a potential security risk.<br>Emulators score the maximum; `1`. |

## Persistent Cookie Decision node

The **Persistent Cookie Decision** node checks for the existence of a specified persistent cookie (default: `session-jwt`).

If the cookie is present, the node verifies the signature of the JWT stored in the cookie with the configured signing key.

If the configured signing key isn't valid, AM checks the signature against all valid signing keys mapped to the configured secret label.

If the signature is valid, the node decrypts the payload of the JWT using the key pair defined in the active secret mapped to the `am.authentication.nodes.persistentcookie.encryption` secret label.

If there isn't a valid secret label mapping in a secret store, the node uses the key pair specified in the **Persistent Cookie Encryption Certificate Alias** property:

- PingAM: Find this property under **Realms** > ***Realm Name*** > **Authentication** > **Settings** > **Security**, or globally, under **Configure** > **Authentication** > **Core Attributes** > **Security**.
- Advanced Identity Cloud: Find this property under **Native Consoles** > **Access Management** > **Realms** > ***Realm Name*** > **Authentication** > **Settings** > **Security**.

The decrypted JSON payload includes information, such as the UID of the identity and the client IP address. Enable **Enforce Client IP** to verify that the current IP address and the client IP address in the cookie are the same.

> **Note**
>
> This node recreates the specified persistent cookie, updating the value for the idle time property and the JWT `kid` header with the stable ID used to sign the JWT.
> Therefore, the node has cookie creation properties similar to the Set Persistent Cookie node.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node requires the `realm` property, which AM sets by default.

## Dependencies

To authenticate successfully, the tree must have set a persistent cookie using a node such as the Set Persistent Cookie node.

## Configuration

| Property | Usage |
| --- | --- |
| **Idle Timeout** | The maximum idle time allowed before the persistent cookie is invalidated, in hours. If no requests are received and the time is exceeded, the cookie is no longer valid. |
| **Enforce Client IP** | When enabled, ensures that the persistent cookie is only used from the same client IP to which the cookie was issued. |
| **Use Secure Cookie** | When enabled, adds the `Secure` flag to the persistent cookie.<br>If the `Secure` flag is included, the cookie can only be transferred over HTTPS. When a request is made over HTTP, the cookie is not made available to the application. |
| **Use HTTP Only Cookie** | When enabled, adds the `HttpOnly` flag to the persistent cookie.<br>When the `HttpOnly` flag is included, that cookie will not be accessible through JavaScript. According to RFC 6265, the `HttpOnly` flag, "instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (for example, a web browser API that exposes cookies to scripts)." |

| Property | Usage |
| --- | --- |
| **HMAC Signing Key** | The key to use for HMAC signing of the persistent cookie.<br><br>**Note**<br>This property is *deprecated*. Use the **HMAC Signing Key Secret Label Identifier** instead.<br>If you set an **HMAC Signing Key Secret Label Identifier**, this signing key is ignored.<br><br>Values must be base64-encoded and at least 256 bits (32 bytes) long.<br>To generate an HMAC signing key, run one of the following commands:<br>`$ openssl rand -base64 32`<br>or<br>`$ cat /dev/urandom \| LC_ALL=C tr -dc 'a-zA-Z0-9' \| fold -w 32 \| head -n 1\|base64` |
| **HMAC Signing Key Secret Label Identifier** | An identifier used to create a *secret label* for mapping to a secret in a secret store. AM uses this identifier to create a specific secret label for the signing key for this node. The secret label takes the form `am.authentication.nodes.persistentcookie.identifier.signing` where identifier is the value of **HMAC Signing Key Secret Label Identifier**. The identifier can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods (`.`). It can't start or end with a period.<br>If you set an **HMAC Signing Key Secret Label Identifier** and AM finds a matching secret in a secret store, the **HMAC Signing Key** is ignored.<br>If **HMAC Signing Key** is empty, AM uses the value configured for `am.default.authentication.nodes.persistentcookie.signing` for the realm, or at the global level if undefined.<br>For greater security, you should rotate signing keys periodically.<br>Find more information in the *secrets* documentation for:<br>• PingAM<br>When you rotate a key, update the corresponding mapping in the realm secret store configuration to reflect this identifier.<br>• Advanced Identity Cloud<br>Adding new secret versions to the ESV.<br><br>**Important**<br>To read the persistent cookies generated by the Set Persistent Cookie node, ensure the nodes use the same HMAC signing key. |
| **Persistent cookie name** | The name of the persistent cookie to check. |

### Outputs

The node copies shared state into the outgoing node state. It records the user identity and stores the cookie name as a session property.

The node adds the `UpdatePersistentCookieTreeHook`, which runs when the tree completes.

### Outcomes

- `True`
- `False`

Evaluation continues along the `True` outcome path if the persistent cookie is present and all the verification checks are satisfied; otherwise, evaluation continues along the `False` outcome path.

### Errors

The node logs the following warning messages:

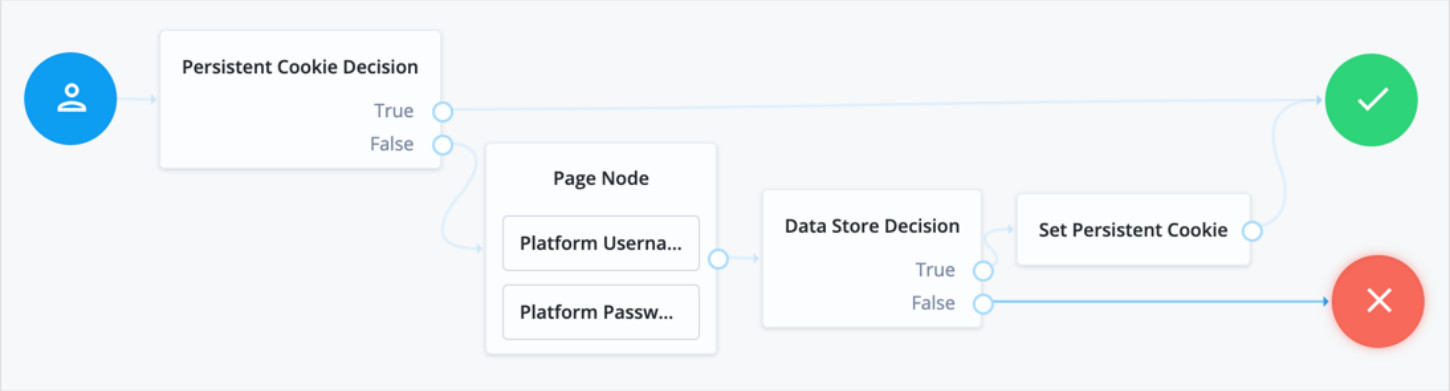
- `Attempt to verify JWT failed, attempting other valid keys`
- `Failed to parse universal id username from claim openam.usr`

The node logs the following error messages:

- `Claims context not found`
- `Failed to find signing key with associated keyID`
- `jwt reconstruction error`
- `Authentication failed. Jwt claim Realm does not match`
- `Authentication failed. Cannot read the user from null claims`
- `Authentication failed. Cannot read the user from empty claims`
- `Failed to parse universal Id from claim: openam.usr`
- `Authentication failed. Client IP is different`

### Example

The following example authenticates the user based on a persistent cookie, if possible:




## Set Custom Cookie node

The **Set Custom Cookie** node lets you store a custom cookie on the client in addition to the session cookie.

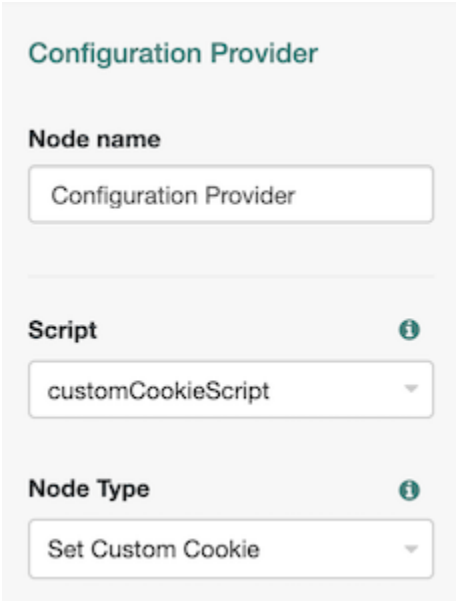
The node uses the specified properties to create a cookie with a custom name and value. It can also set attributes, such as the cookie path, domain, expiry, and security flags.

Use this node with the Configuration Provider node to extend custom capabilities. For example, create a `Config Provider` script to set custom static values or access values from the shared node state.

Include all the attributes in the configuration provider script's `config` map. The following example sets the attributes of the custom cookie to static values:

```
config = {
    "name": "testname",
    "value": "testvalue",
    "maxAge": "60",
    "domain": "am.example.com",
    "path": "/",
    "useSecureCookie": false,
    "useHttpOnlyCookie": false,
    "sameSite": "LAX"
};
```

Reference the script when you create a Configuration Provider node, and set the **Node Type** to `Set Custom Cookie`:




## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads the user data from the shared node state.

It requires a predecessor node that gathers the user data.

## Configuration

| Property | Usage |
|---|---|
| Custom Cookie Name *(required)* | The name of the custom cookie.<br>The cookie name can contain any US-ASCII characters except for: space, tab, control, or a separator character ( `()<>@,;:"/[]?=\{}` ). |
| Custom Cookie Value *(required)* | The value of the custom cookie. |

| Property | Usage |
| --- | --- |
| Max Age | The length of time the custom cookie remains valid, in seconds. If that time is exceeded, the cookie is no longer valid.<br>AM sets the `Max-Age` and `Expires` attributes in the cookie to increase compatibility with different browsers.<br>If omitted, the cookie expires at the end of the current session. The precise implementation of this is determined by the specific browser. Refer to RFC 6265 for details. |
| Custom Cookie Domain | The domain the custom cookie will be sent to. If you specify a value here, AM sets a domain cookie. For example, if you set this property to `am.example.com`, AM sets a cookie on `.am.example.com`. Note the leading `.` indicating a domain cookie rather than a host cookie.<br>If you don't set a value here, AM sets a host level cookie on the FQDN on which the client accessed AM. For example, if the client accesses AM at `https://am.example.com` and this property is empty, AM sets a host cookie on `am.example.com`. |
| Custom Cookie Path | The path of the custom cookie. |
| Use Secure Cookie | When enabled, adds the `Secure` flag to the custom cookie.<br>If you include the `Secure` flag, the cookie can only be transferred over HTTPS. When a request is made over HTTP, the cookie isn't made available to the application. |
| Use HTTP Only Cookie | When enabled, adds the `HttpOnly` flag to the custom cookie.<br>If you include the `HttpOnly` flag, the cookie isn't accessible to scripts. |
| Custom Cookie SameSite attribute | Sets the `SameSite` attribute of the custom cookie.<br>The default value is `LAX`, to align with most modern browsers.<br>PingAM only: Find more information in SameSite cookie rules. |

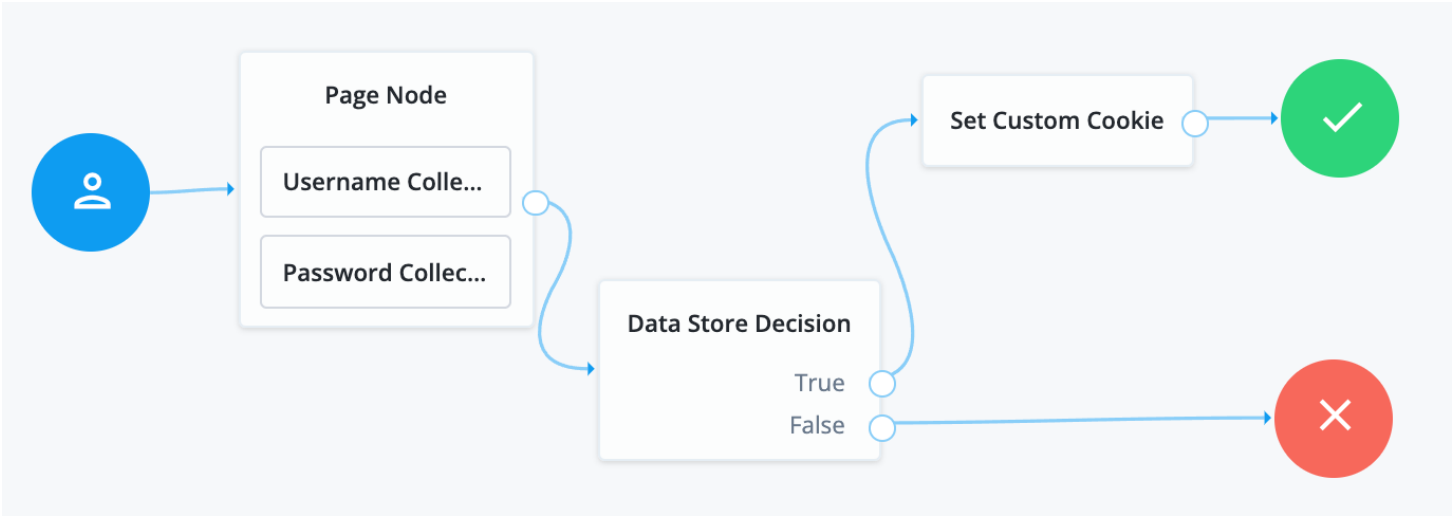
## Outcomes

Single outcome path.

The cookie is created when AM next returns to the client.

## Example

This example uses this node in a login flow. The node sets the custom cookie in the client browser after the user has successfully authenticated:




## Set Persistent Cookie node

Creates the specified persistent cookie, the default being `session-jwt`.

The cookie contains a JWT with a JSON payload including information such as the UID of the identity, and the client IP address.

The node encrypts the payload of the JWT using the key pair defined in the active secret mapped to the `am.authentication.nodes.persistentcookie.encryption` secret label.

If there isn't a valid secret label mapping in a secret store, the node uses the key pair specified in the **Persistent Cookie Encryption Certificate Alias** property:

- PingAM: Find this property under **Realms** > ***Realm Name*** > **Authentication** > **Settings** > **Security**, or globally, under **Configure** > **Authentication** > **Core Attributes** > **Security**.
- Advanced Identity Cloud: Find this property under **Native Consoles** > **Access Management** > **Realms** > ***Realm Name*** > **Authentication** > **Settings** > **Security**.

The node signs the cookie with the HMAC signing key defined in the node properties or the secret store with the mapped secret label. Configure nodes that read the persistent cookie such as the Persistent Cookie Decision node with the same HMAC signing key.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

When the authentication tree completes successfully, the `CreatePersistentCookieTreeHook` treehook for this node uses session properties to create the persistent cookie.

## Dependencies

In PingAM and Ping Identity Platform deployments, you must configure a secret store to store the dynamic secret label mapping to the cookie's signing key.

## Configuration

| Property | Usage |
|---|---|
| **Idle Timeout** | The maximum amount of idle time allowed before the persistent cookie is invalidated, in hours. If no requests are received before the timeout, the cookie is no longer valid. |
| **Max life** | The length of time the persistent cookie remains valid, in hours. After this time has passed, the cookie is no longer valid. |
| **Use Secure Cookie** | When enabled, adds the `Secure` flag to the persistent cookie.<br>If the `Secure` flag is included, the cookie can only be transferred over HTTPS. When a request is made over HTTP, the cookie is not made available to the application. |
| **Use HTTP Only Cookie** | When enabled, adds the `HttpOnly` flag to the persistent cookie.<br>When the `HttpOnly` flag is included, that cookie will not be accessible through JavaScript. According to RFC 6265, the `HttpOnly` flag, "instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (for example, a web browser API that exposes cookies to scripts)." |
| **HMAC Signing Key** | A key to use for HMAC signing of the persistent cookie.<br>**Note**<br>This property is *deprecated*. Use the **HMAC Signing Key Secret Label Identifier** instead. If you set an **HMAC Signing Key Secret Label Identifier**, this signing key is ignored.<br>Values must be base64-encoded and at least 256 bits (32 bytes) long.<br>To generate an HMAC signing key, run one of the following commands:<br>`$ openssl rand -base64 32`<br>or<br>`$ cat /dev/urandom \| LC_ALL=C tr -dc 'a-zA-Z0-9' \| fold -w 32 \| head -n 1\|base64` |

| Property | Usage |
| --- | --- |
| **HMAC Signing Key Secret Label Identifier** | An identifier used to create a *secret label* for mapping to a secret in a secret store.<br>AM uses this identifier to create a specific secret label for the signing key for this node. The secret label takes the form `am.authentication.nodes.persistentcookie.identifier.signing` where identifier is the value of **HMAC Signing Key Secret Label Identifier**. The identifier can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods (`.`). It can't start or end with a period.<br>If you set an **HMAC Signing Key Secret Label Identifier** and AM finds a matching secret in a secret store, the **HMAC Signing Key** is ignored.<br>If **HMAC Signing Key** is empty, AM uses the value configured for `am.default.authentication.nodes.persistentcookie.signing` for the realm, or at the global level if undefined.<br>For greater security, you should rotate signing keys periodically.<br>Find more information in the *secrets* documentation for:<br>• PingAM<br>When you rotate a key, update the corresponding mapping in the realm secret store configuration to reflect this identifier.<br>• Advanced Identity Cloud<br>Adding new secret versions to the ESV.<br>**Important**<br>To read the persistent cookies this node generates, ensure the nodes use the same HMAC signing key. |
| **Persistent Cookie Name** | The name used for the persistent cookie. |

## Outputs

The node stores the cookie name in the session properties.

The node adds the `CreatePersistentCookieTreeHook` treehook, which runs when the tree completes.

## Outcomes

Single outcome path.

## Errors

The node logs the following warning messages:

- `Unable to create signing key from provided configuration.`

The node logs the following error messages:

- `Tree hook creation exception`
- `No signing keys available to sign JWT`

- `Error creating jwt string`

### Example

Refer to the Persistent Cookie Decision node example.

# Federation nodes

## OAuth 2.0 node

Lets AM authenticate users of OAuth 2.0-compliant resource servers.

References in this section are to RFC 6749, The OAuth 2.0 Authorization Framework.

> **Note**
>
> This node and its related services are deprecated.
> Depending on your deployment, you can find information on the current methods for implementing social authentication in:
>
> - Advanced Identity Cloud Social authentication
> - PingAM PingGateway

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

- `Account Exists`
- `No account Exists`

Evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from the social identity provider is found in the user data store; otherwise, evaluation continues along the `No account exists` path.

### Properties

| Property | Usage |
| --- | --- |
| Client ID *(required)* | Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749). |
| Client Secret *(required)* | Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749). |
| Authentication Endpoint URL *(required)* | Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).<br>Example: `https://accounts.google.com/o/oauth2/v2/auth` |
| Access Token Endpoint URL *(required)* | Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).<br>Example: `https://www.googleapis.com/oauth2/v4/token` |
| User Profile Service URL *(required)* | Specifies the user profile URL that returns profile information.<br>Example: `https://www.googleapis.com/oauth2/v3/userinfo` |
| OAuth Scope *(required)* | Specifies a list of user profile attributes that the client application requires, according to The OAuth 2.0 Authorization Framework (RFC 6749).<br>Ensure you use the correct scope delimiter required by the identity provider, including commas or spaces.<br>The list depends on the permissions that the resource owner, such as the end user, grants to the client application. |
| Scope Delimiter *(required)* | Specifies the delimiter used to separate scope values.<br>Some authorization servers use non-standard separators for scopes, for example commas. |
| Redirect URL *(required)* | Specifies the URL the user is redirected to by the social identity provider after authenticating.<br>For authentication trees in AM, set this property to the URL of the UI. For example, `https://am.example.com:8443/am/XUI/`. |
| Social Provider *(required)* | Specifies the name of the social provider for which this module is being set up.<br>Example: `Google` |
| Auth ID Key *(required)* | Specifies the attribute the social identity provider uses to identify an authenticated individual.<br>Example: `id` |
| Use Basic Auth | Specifies that the client uses HTTP Basic authentication when authenticating to the social provider.<br>Default: `true` |

| Property | Usage |
| --- | --- |
| Account Provider *(required)* | Specifies the name of the class that implements the account provider.<br>Default:<br>`org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider` |
| Account Mapper *(required)* | Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from the social identity provider.<br>Provided implementations are:<br>`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`<br>The Account Mapper classes can take two constructor parameters:<br>1. A comma-separated list of attributes<br>2. A prefix to apply to their values.<br>For example, to prefix all received property values with `facebook-` before searching, specify:<br>`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper\|*\|facebook-` |
| Attribute Mapper *(required)* | Specifies the list of fully qualified class names for implementations that map attributes from the OAuth 2.0 authorization server to AM profile attributes.<br>Provided implementations are:<br>`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`<br>The Attribute Mapper classes can take two constructor parameters to help differentiate between the providers:<br>1. A comma-separated list of attributes<br>2. A prefix to apply to their values.<br>For example, to prefix all incoming values with `facebook-`, specify:<br>`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper\|*\|facebook-`<br>To prefix all incoming values use an asterisk (`*`) as the attribute list. This prefixes all values, including email addresses, postal addresses, and so on. |

| Property | Usage |
| --- | --- |
| Account Mapper Configuration | Specifies the attribute configuration used to map the account of the user authenticated in the OAuth 2.0 provider to the local data store in AM.<br>Valid values are in the form `provider-attr=local-attr`.<br>Examples:<br>`email=mail`<br>`id=facebook-id`<br><br>**Tip**<br>When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings using dot notation.<br>For example, given a JSON payload of:<br>`{ "sub" : "12345", "name" : { "first_name" : "Demo", "last_name" : "User" } }`<br>You can create a mapper, such as `name.first_name=cn`. |

| Property | Usage |
| --- | --- |
| Attribute Mapper Configuration | Map of OAuth 2.0 provider user account attributes to local user profile attributes, with values in the form `provider-attr=local-attr`.<br>Examples:<br>`first_name=givenname`<br>`last_name=sn`<br>`name=cn`<br>`email=mail`<br>`id=facebook-id`<br>`first_name=facebook-fname`<br>`last_name=facebook-lname`<br>`email=facebook-email`<br>**Tip**<br>When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings using dot notation.<br>For example, given a JSON payload of:<br>`{ "sub" : "12345", "name" : { "first_name" : "Demo", "last_name" : "User" } }`<br>You can create a mapper, such as `name.first_name=cn`. |
| Save attributes in the session | When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session. |
| OAuth 2.0 Mix-Up Mitigation Enabled | Controls whether the OAuth 2.0 authentication node carries out additional verification steps when it receives the authorization code from the authorization server.<br>Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned as the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.<br>When this is enabled, set the Token Issuer property so that the validation can succeed. The authorization code response contains an issuer value (`iss`) for the client to validate.<br>**Note**<br>Refer to the authorization server's documentation for the value it uses for the issuer field.<br>Learn more in section 4 of OAuth 2.0 Mix-Up Mitigation Draft. |
| Token Issuer | Corresponds to the expected issuer identifier value in the `iss` field of the ID token.<br>Example: `https://accounts.google.com` |

# OpenID Connect node

Lets AM authenticate users of OpenID Connect-compliant resource servers.

As OpenID Connect is an additional layer on top of OAuth 2.0, described in RFC 6749, The OAuth 2.0 Authorization Framework. OpenID Connect is described in the OpenID Connect Core 1.0 incorporating errata set 1 specification.

> **Note**
>
> This node and its related services are deprecated.
> Depending on your deployment, you can find information on the current methods for implementing social authentication in:
>
> - Advanced Identity Cloud Social authentication
> - PingAM PingGateway

The OpenID Connect node implements the Authorization code grant.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

## Outcomes

- `Account Exists`
- `No account Exists`

Evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from the OpenID Connect identity provider is found in the identity store; otherwise, evaluation continues along the `No account exists` path.

## Properties

| Property | Usage |
|---|---|
| Client ID *(required)* | Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749). |
| Client Secret *(required)* | Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749). |

| Property | Usage |
| --- | --- |
| Authentication Endpoint URL *(required)* | Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749). Example: `https://accounts.google.com/o/oauth2/v2/auth` |
| Access Token Endpoint URL *(required)* | Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749). Example: `https://www.googleapis.com/oauth2/v4/token` |
| User Profile Service URL *(required)* | Specifies the user profile URL that returns profile information. If not specified, attributes are mapped from the claims returned by the `id_token`, and no call to a user profile endpoint is made. Example: `https://www.googleapis.com/oauth2/v3/userinfo` |
| OAuth Scope | Specifies a list of user profile attributes that the client application requires, according to The OAuth 2.0 Authorization Framework (RFC 6749). Ensure you use the correct scope delimiter required by the identity provider, including commas or spaces. The list depends on the permissions that the resource owner, such as the end user, grants to the client application. |
| Redirect URL | Specifies the URL the user is redirected to by the social identity provider after authenticating. For authentication trees in AM, set this property to the URL of the UI. For example, `https://am.example.com:8443/am/XUI/`. |
| Social Provider *(required)* | Specifies the name of the OpenID Connect provider for which this node is being set up. Example: `Google` |
| Auth ID Key | Specifies the attribute the social identity provider uses to identify an authenticated individual. Example: `sub` |
| Use Basic Auth | Specifies that the client uses HTTP Basic authentication when authenticating to the social provider. Default: `true` |
| Account Provider | Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider` |

| Property | Usage |
| --- | --- |
| Account Mapper | Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from the social identity provider.<br>The provided implementations is `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper`.<br>The Account Mapper classes can take two constructor parameters:<br>1. A comma-separated list of attributes<br>2. A prefix to apply to their values.<br>For example, to prefix all received property values with `openid-` before searching, specify:<br>`org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper\|*\|openid-` |
| Attribute Mapper | Specifies the list of fully qualified class names for implementations that map attributes from the authorization server to AM profile attributes.<br>The provided implementations is `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper`.<br>The Attribute Mapper classes can take two constructor parameters to help differentiate between the providers:<br>1. A comma-separated list of attributes<br>2. A prefix to apply to their values.<br>For example, to prefix incoming `iplanet-am-user-alias-list` values with `openid-`, specify:<br>`org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` |
| iplanet-am-user-alias-list | openid-<br>To prefix all incoming values use an asterisk (`*`) as the attribute list. This prefixes all values, including email addresses, postal addresses, and so on. |
| Account Mapper Configuration | Specifies the attribute configuration used to map the account of the user authenticated in the provider to the local identity store in AM.<br>To add a mapping, specify the name of the provider attribute as the key, and the local attribute to map to as the value.<br>For example, click **Add**, then specify `sub` in the **Key** field and `iplanet-am-user-alias-list` in the **Value** field, and click +. |

| Property | Usage |
| --- | --- |
| Attribute Mapper Configuration | Specifies how to map provider user attributes to local user profile attributes.<br>To add a mapping, specify the name of the provider attribute as the Key, and the local attribute to map to as the Value.<br>For example, click **Add**, then specify `id` in the **Key** field and `facebook-id` in the **Value** field, and click +.<br>Examples:<br>`first_name=givenname`<br>`last_name=sn`<br>`name=cn`<br>`email=mail`<br>`id=facebook-id`<br>`first_name=facebook-fname`<br>`last_name=facebook-lname`<br>`email=facebook-email` |
| Save attributes in the session | When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session. |
| OAuth 2.0 Mix-Up Mitigation Enabled | Controls whether the authentication node carries out additional verification steps when it receives the authorization code from the authorization server.<br>Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned as the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.<br>When this is enabled, set the Token Issuer property so that the validation can succeed. The authorization code response contains an issuer value ( `iss` ) for the client to validate.<br>**Note**<br>Refer to the authorization server's documentation for the value it uses for the issuer field.<br>Learn more in section 4 of OAuth 2.0 Mix-Up Mitigation Draft. |
| Token Issuer *(required)* | Corresponds to the expected issuer identifier value in the `iss` field of the ID token.<br>Example: `https://accounts.google.com` |

| Property | Usage |
| --- | --- |
| OpenID Connect Validation Type *(required)* | Specifies how to validate the ID token received from the OpenID Connect provider. This ignores keys specified in JWT headers, such as `jku` and `jwe`.<br>The following options are available to validate an incoming OpenID Connect ID token:<br>**`Well Known URL` *(Default)***<br>Retrieves the provider's keys based on the information provided in its OpenID Connect configuration URL.<br>Specify the provider's configuration URL in the OpenID Connect Validation Value field; for example, `https://accounts.google.com/.well-known/openid-configuration`.<br>**`Client Secret`**<br>Validates the ID token signature with a specified client secret key.<br>Specify the key to use in the OpenID Connect Validation Value field.<br>**`JWK URL`**<br>Retrieve the necessary JSON web key from the URL that you specify.<br>Specify the provider's JWK URI in the OpenID Connect Validation Value field; for example, `https://www.googleapis.com/oauth2/v3/certs`. |
| OpenID Connect Validation Value | Provide the URL or secret key used to verify an incoming ID token, depending on the value selected in the OpenID Connect Validation Type property. |

## OIDC ID Token Validator node

The **OIDC ID Token Validator** node lets AM rely on an OIDC provider (OP)'s ID token to authenticate an end user. The node evaluates whether the ID token is valid according to the OIDC specification.

To configure the node, first get an `id_token` from an OIDC client and examine the decoded JWT to view the required claims values.

This example uses an `id_token` from the OAuth 2.0 Playground:

```
{
    "iss": "https://accounts.google.com",
    "azp": "407408718192.apps.googleusercontent.com",
    "aud": "407408718192.apps.googleusercontent.com",
    "sub": "111730983950574648607",
    "at_hash": "kvQJZrGcnNMZqM4w68DFBA",
    "iat": 1677608448,
    "exp": 1677612048
}
```

The `iss`, `azp`, and `aud` claims provide the values for the node's `Token Issuer`, `Authorized parties` and `Audience name` properties respectively.

To use the OIDC ID Token Validator node to authenticate a user, first configure the node to run a transformation script that maps the user attributes from the JWT to local attributes. You can then create a journey with a Scripted Decision node that stores the attributes in the shared node state so that you can authenticate the user with an ID token.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

None.

## Dependencies

A valid OIDC ID token provided in the HTTP request header.

## Configuration

| Property | Usage |
|---|---|
| **OpenID Connect Validation Type** | To validate the ID token from the OP, the node requires either a URL to get the public keys for the provider, or the symmetric key for an ID token signed with an HMAC-based algorithm.<br>Select one of the following options to determine how the node retrieves the required information:<br>• `Well Known URL` (default): Validate with the keys specified in the OP's `/.well-known/openid-configuration` JSON document.<br>• `Client Secret`: Validate the ID token signature with the provided client secret.<br>• `JWK URL`: Validate with the keys retrieved from the URL to the OP's JSON Web Key Set (JWKS). |
| **OpenID Connect Validation Value** | The well-known URL or URL to the JWK location, depending on the value of `OpenID Connect Validation Type`. If the validation type is `Client Secret`, this value is ignored and the `Client Secret Label` is used instead.<br>For example:<br>`https://accounts.google.com/.well-known/openid-configuration` |

| Property | Usage |
| --- | --- |
| **Client Secret Label** | The secret label to which the OIDC client secret should be mapped.<br>Only required if the validation type is `Client Secret`.<br>You can specify an existing label or AM creates a new one dynamically.<br>The label can only contain alphanumeric characters `a-z`, `A-Z`, `0-9`, and periods (`.`). It can't start or end with a period. |
| **ID Token Header Name** | The name of the HTTP request header referencing the ID token.<br>Default: `oidc_id_token` |
| **Token Issuer** | The issuer of the OIDC ID token, which is checked against the `iss` claim in the ID token.<br>For example: `https://accounts.google.com` |
| **Audience name** | The case-sensitive name of the intended audience for this node, which is checked against the `aud` claim in the ID token. |
| **Authorized parties** | The authorized parties from which the node accepts ID tokens, which is checked against the `azp` claim in the ID token.<br>The value can be either a case-sensitive string or a URI. |
| **Transformation Script** | Select a `Social Identity Provider Profile Transformation` script that maps ID token attributes to local attributes.<br>Find examples of transformation (normalization) scripts in the Example or the `*-profile-normalization.js` scripts for:<br>• PingAM<br>• Advanced Identity Cloud |
| **Script Inputs** | A list of state inputs for the script.<br>Default: `*` |
| **Unreasonable Lifetime Limit** | Specify the maximum permitted lifetime of the token in minutes. If the `iat` claim is present, the token must expire within the specified duration.<br>Default: `60` |

## Outputs

The node relies on the transformation script to set profile attributes required later in the journey.

## Outcomes

- `True`
- `False`

Evaluation continues along the `True` path if the ID token is valid; otherwise, evaluation continues along the `False` path.

### Errors

The node logs the following warnings:

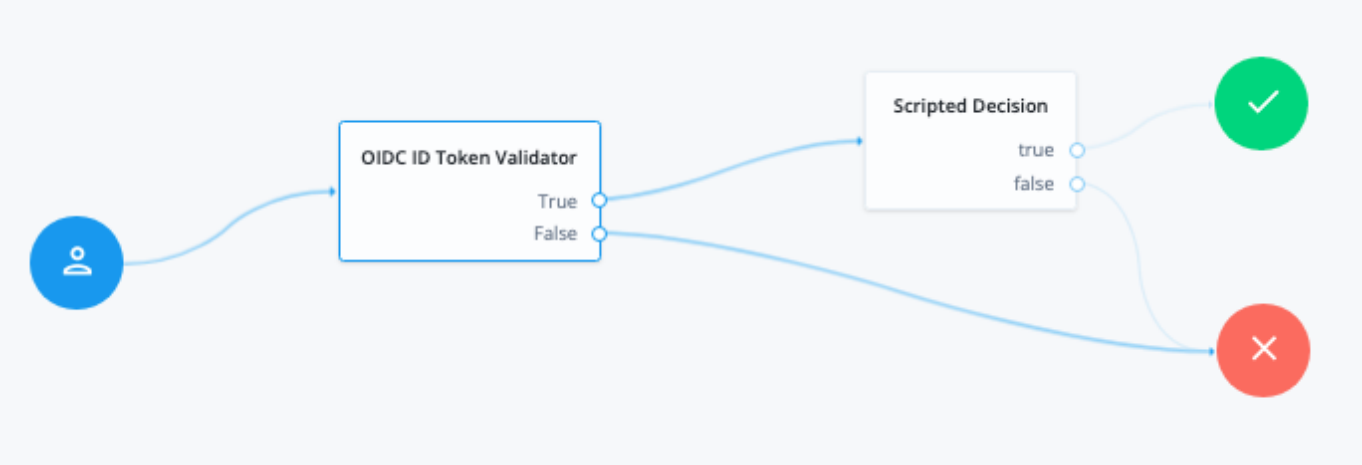
- `No OpenIdConnect ID Token referenced by header value: {}`: if the node can't read the ID token in the HTTP request header.
- `Error evaluating the script`: if there's a problem with the transformation script.

The node logs an error if an `AuthLoginException` occurs during node processing.

### Example

This example demonstrates how to use the OIDC ID Token Validator node as part of a journey to validate an ID token and authenticate the user.

Advanced Identity Cloud




You can access all the provided JWT claims through the `jwtClaims` attribute. This JavaScript, configured as the node's transformation script, retrieves the user ID from the JWT.

```
(function () {
    var fr = JavaImporter( org.forgerock.json.JsonValue);

    var identity = fr.JsonValue.json(fr.JsonValue.object());
    identity.put('uid', jwtClaims.get('sub'));

    return identity;
}());
```

A Scripted Decision node runs this script to find the username from `lookupAttributes` and store it in the shared node state:

Next-generation

```
var attributes = nodeState.get("lookupAttributes");
var userName = attributes.get("uid");

// Add userName in objectAttributes to nodeState for use by Identify Existing User node.
nodeState.putShared("objectAttributes", attributes);

// Add username at the root level so that a session can be created
nodeState.putShared("username", userName);

action.goTo('true');
```

Legacy

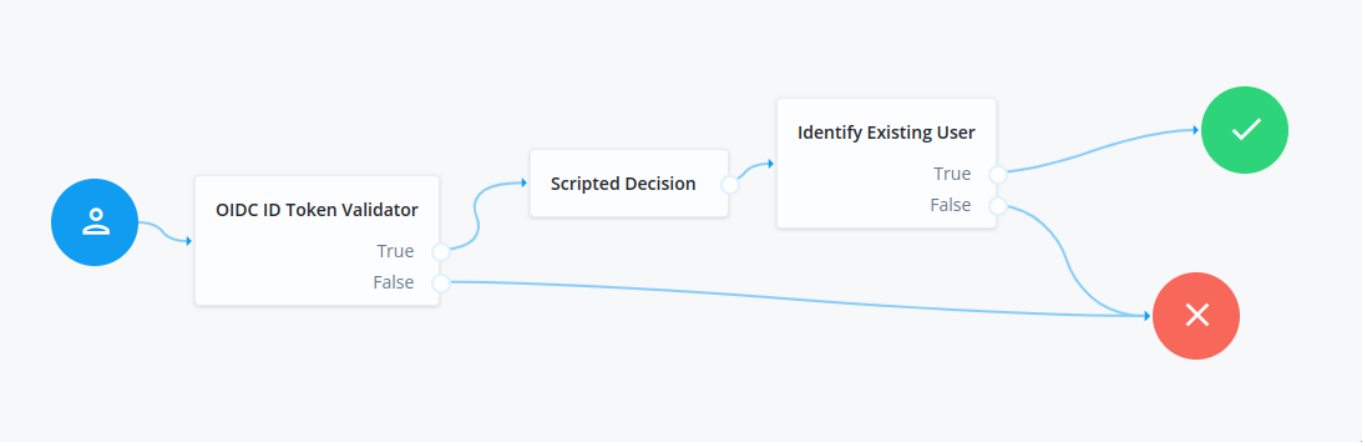
```
var attributes = nodeState.get("lookupAttributes");
var userName = attributes.get("uid").asString();

// Add userName in objectAttributes to nodeState for use by Identify Existing User node.
nodeState.putShared("objectAttributes", attributes);

// Add username at the root level so that a session can be created
nodeState.putShared("username", userName);

outcome = "true";
```

The Identify Existing User node then performs a lookup on IDM with the `_id` attribute using the username saved into shared state by the Scripted Decision node.

The following REST call authenticates the user with the example journey, providing the ID token in the header:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "oidc_id_token: <id_token>" \
"https://<tenant-env-fqdn>/am/json/realms/root/realms/alpha/authenticate?
authIndexType=service&authIndexValue=myJourney"
{
    "tokenId": "AQIC5w…NTcy*",
    "successUrl": "/openam/console",
    "realm": "/alpha"
}
```

PingAM




You can access all the provided JWT claims through the `jwtClaims` attribute. This JavaScript, configured as the node's transformation script, retrieves the user ID from the JWT.

```
(function () {
    var fr = JavaImporter( org.forgerock.json.JsonValue);

    var identity = fr.JsonValue.json(fr.JsonValue.object());
    identity.put('uid', jwtClaims.get('sub'));

    return identity;
}());
```

A Scripted Decision node runs this script to find the user ID from `lookupAttributes` and store it in the shared node state:

Next-generation

```
var attributeName = "uid";
var attributes = nodeState.get("lookupAttributes");
var uid = attributes.get(attributeName);

// get the identity for the sub claim (stored as uid)
var identity = idRepository.getIdentity(uid);

// verify the identity exists
var userName = identity.getAttributeValues(attributeName);

if (!userName.isEmpty()) {
    nodeState.putShared("username", userName[0]);
    action.goTo('true');
} else {
    action.goTo('false');
}
```

Legacy

```
var attributeName = "uid";
var attributes = nodeState.get("lookupAttributes");
var userName = attributes.get(attributeName).asString();
var identity = idRepository.getAttribute(userName, attributeName);

if (!identity.isEmpty()) {
    nodeState.putShared("username", identity.iterator().next());
    outcome = "true";
} else {
    outcome = "false";
}
```

The following REST call authenticates the user with the example tree, providing the ID token in the header:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "oidc_id_token: <id_token>" \
"https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=service&authIndexValue=myJourney"
{
    "tokenId": "AQIC5w…NTcy*",
    "successUrl": "/openam/console",
    "realm": "/alpha"
}
```

## Provision Dynamic Account node

Provision an account following successful authentication by a SAML2 authentication node or the Social Provider Handler node.

Accounts are provisioned using properties defined in the attribute mapper configuration of a social authentication or SAML2 authentication node earlier in the flow.

If a password has been acquired from the user, for example, by using the Password Collector node, it is used when provisioning the account; otherwise, a 20-character random string is used.

In addition to retrieving the password from the node state, the Provision Dynamic Account node gets the `realm` value, and `attributes` and `userNames` from `userInfo` in the shared state. It sets the `username` attribute in the node's shared state.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

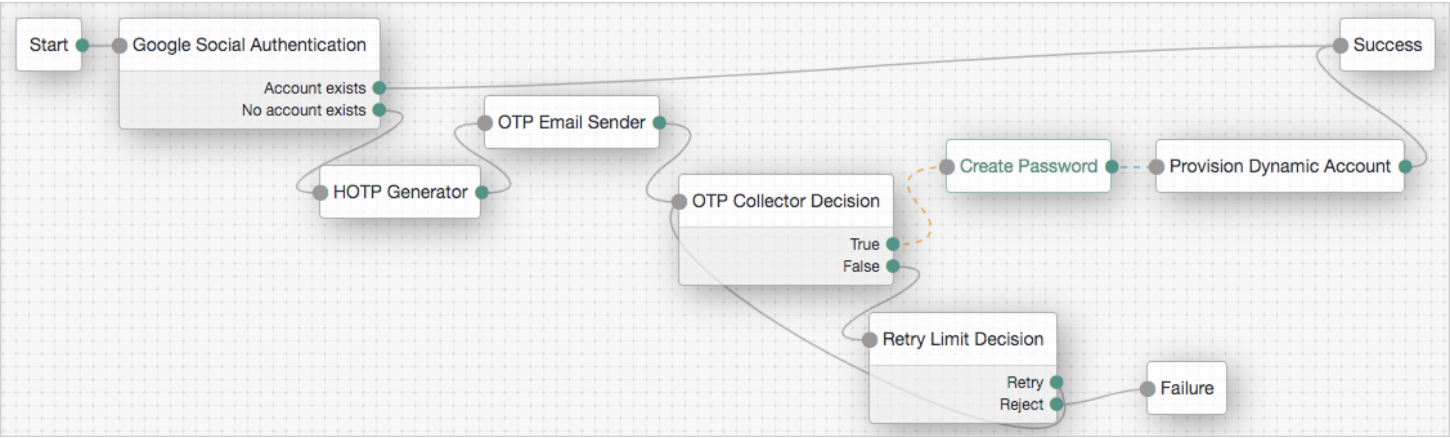
Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Account Provider | Specifies the name of the class that implements the account provider.<br>Default:<br>`org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider` |

### Example

The following example uses this node to let users who have performed social authentication using Google provide a password and provision an account if they do not have a matching existing profile. They must enter a one-time passcode to verify they are the owner of the Google account.




## Provision IDM Account node

Redirects users to an IDM instance to provision an account.

**Note**

This node and its related services are deprecated.
Depending on your deployment, you can find information on the current methods for implementing social authentication in:

- Advanced Identity Cloud Social authentication
- PingAM PingGateway

Ensure you configured the details of the IDM instance in AM, by navigating to **Configure > Global Services > IDM Provisioning**.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

## Outcomes

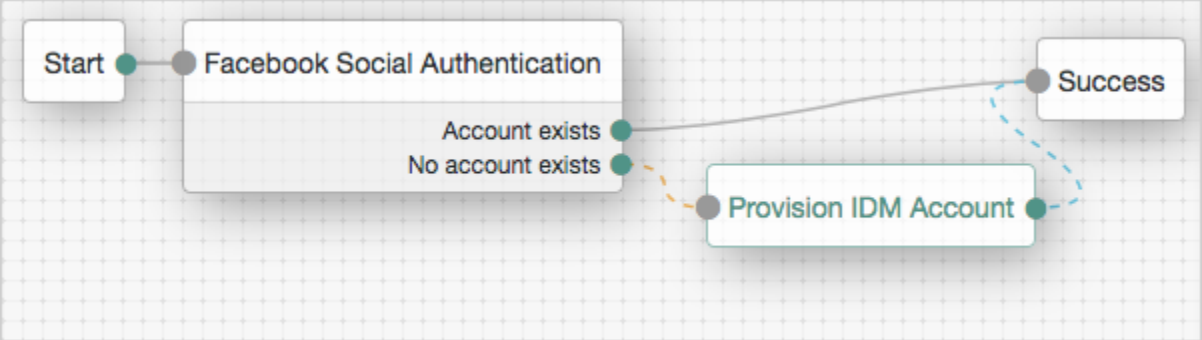
Single outcome path.

## Properties

| Property | Usage |
|---|---|
| Account Provider | Specifies the name of the class that implements the account provider.<br>Default:<br>`org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider` |

## Example

The following example uses this node to let users who have performed social authentication using Facebook provide a password and provision an account if they do not have a matching existing profile:




## SAML2 Authentication node

The **SAML2 Authentication** node integrates SAML 2.0 single sign-on into an authentication flow.

Use this node when deploying SAML 2.0 SSO in integrated mode (SP-initiated SSO only). This node doesn't support single logout (SLO).

Implement the Write Federation Information node after this node in the journey to link the remote account to a local account.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Dependencies

This node requires the following configuration in AM:

- A remote identity provider (IdP) and a hosted service provider (SP) in a circle of trust in the same realm where you're configuring the journey.
- You must configure the service provider for integrated mode.

  Find more information in the *SAML* documentation for:

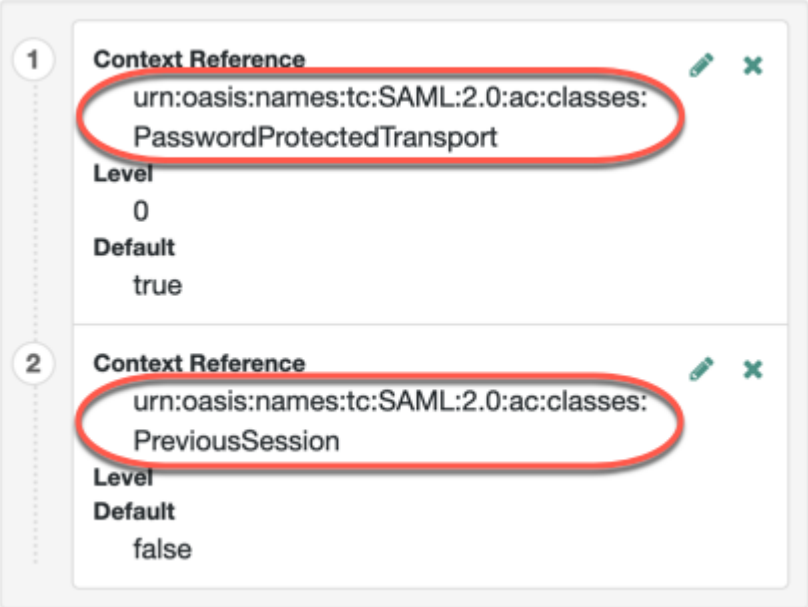
  - PingAM
  - Advanced Identity Cloud

### Inputs

This node sends a SAML Request and processes the incoming SAML assertion.

## Configuration

| Property | Usage |
| --- | --- |
| **IdP Entity ID** | The name of the remote IdP. |
| **SP MetaAlias** | The local alias for the SP in the format `/Realm Name/SP Name`. |
| **Allow IdP to Create NameID** | Enable this option if you want the IdP to create a new identifier for the authenticating user if none exists.<br>Learn more in AllowCreate in the SAML Version 2.0 specification.<br>Default: `Enabled` |
| **Comparison Type** | The comparison method to evaluate authentication context classes or statements. This value overrides the value in the SP configuration:<br>PingAM Under **Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Assertion Content > Authentication Context > Comparison Type**.<br>PingOne Advanced Identity Cloud Under **Native Consoles > Access Management > Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Assertion Content > Authentication Context > Comparison Type**.<br>Valid comparison methods are `exact`, `minimum`, `maximum`, or `better`.<br>Learn more about comparison methods in `Element <RequestedAuthnContext>` in Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.<br>Default: `minimum` |

| Property | Usage |
| --- | --- |
| **Authentication Context Class Reference** | (Optional) Set one or more URIs for authentication context classes to be included in the SAML request.<br>Authentication context classes are unique identifiers for an authentication mechanism. The SAML 2.0 protocol supports a standard set of authentication context classes, defined in Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0.<br>You can specify your own authentication context classes in addition to the standard ones.<br>Any authentication context class you specify here must be supported for the SP. To add authentication context classes to the SP:<br>1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Applications** > **Federation** > **Entity Providers** > ***Service Provider Name*** > **Assertion Content**.<br>2. In the **Authentication Context** section, add the authentication context classes.<br>  <br><br>Use the \| character to separate multiple authentication context classes, for example:<br>`urn:oasis:names:tc:SAML:2.0:ac:classes:Password\|`<br>`urn:oasis:names:tc:SAML:2.0:ac:classes:TimesyncToken` |
| **Authentication Context Declaration Reference** | (Optional) One or more URIs that identify authentication context declarations.<br>Use the \| character to separate multiple URIs.<br>Learn more in the section on the `<RequestedAuthnContext>` element in Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. |
| **Request Binding** | The format of the authentication request that the SP sends to the IdP.<br>Valid values are `HTTP-Redirect` and `HTTP-POST`.<br>Default: `HTTP-Redirect` |

| Property | Usage |
| --- | --- |
| **Response Binding** | The format of the response that the IdP sends to the SP.<br>Valid values are `HTTP-POST` and `HTTP-Artifact`.<br>Default: `HTTP-Artifact` |
| **Force IdP Authentication** | Indicate whether the IdP should force authentication or if it can reuse existing security contexts.<br>Default: Disabled |
| **Passive Authentication** | Indicate whether the IdP should use passive authentication.<br>When this setting is enabled, the IdP can only use authentication methods that don't require user interaction, such as authenticating with an X.509 certificate.<br>Default: Disabled |
| **NameID Format** | The SAML name ID format that's requested in the SAML authentication request.<br>Valid values are:<br>`urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`<br>`urn:oasis:names:tc:SAML:2.0:nameid-format:transient`<br>`urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`<br>Default: `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` |

## Outputs

This node updates the shared state with the SAML 2.0 assertion data as follows:

- It adds a `userInfo` key with two child keys: `attributes` and `userNames`.
- The `attributes` object contains a map of SAML 2.0 attributes, each of which is stored as an array.
- The `attributes` object also stores the SAML 2.0 identity attributes `sun-fm-saml2-nameid-info` and `sun-fm-saml2-nameid-infokey`.

  These attributes are required by the Write Federation Information node.
- The `userNames` object contains the user's UUID.

```
{
    "realm" : "/",
    "authLevel" : 0,
    "username" : "22fe07c3-ac8b-4e84-9016-b55f1c009924",
    "userInfo" : {
      "attributes" : {
        "uid" : [ "bjensen" ],
        "mail" : [ "bjensen@example.com" ],
        "sun-fm-saml2-nameid-info" : [ "serviceprovider2|identityprovider1|sAdI2i7LT2YL0rbJC/QqsRt5SABV|
identityprovider1|urn:oasis:names:tc:SAML:2.0:nameid-format:persistent|null|serviceprovider2|SPRole|false" ],
        "sun-fm-saml2-nameid-infokey" : [ "serviceprovider2|identityprovider1|sAdI2i7LT2YL0rbJC/QqsRt5SABV" ]
      },
      "userNames" : {
        "username" : [ "22fe07c3-ac8b-4e84-9016-b55f1c009924" ],
        "uid" : [ "22fe07c3-ac8b-4e84-9016-b55f1c009924" ]
      }
    },
    "emailAddress" : "bjensen@example.com"
  }
```


**Tip**

You can use a script to read the SAML 2.0 attributes, for example:

```
nodeState.get("userInfo").get("attributes").get("uid").get(0)
```

The updated shared state depends on the node outcome:

- If the outcome is `Account exists`, the shared state is updated with `nodeState.userNames` as follows:

```
userNames={username=[bjensen], uid=[bjensen]}}
```

- If the outcome is `No account exists`, the shared state is updated with `nodeState.userNames` as follows:

```
userNames={username=[null], uid=[null]}}
```

- If the `mail` attribute exists in the `attributes` map, the shared state is updated with `nodeState.emailAddress`.
- If the `RelayState` attribute exists in the `attributes` map, the shared state is updated with `nodeState.successUrl`.
- The `username` is added to the shared state, regardless of outcome.

The node also sets the following session properties:

- `SessionIndex`: the session index
- `NameID`: the NameID of the Assertion XML
- `isTransient`: if the NameId is `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`
- `cacheKey`: for internal use only

### Outcomes

#### `Account exists`

The node found a user account that matches the federated account.

#### `No account exists`

Any other case.

### Errors

This node can log the following errors:

| Message | Notes |
| --- | --- |
| `Unable to complete SAML2 authentication, IDP descriptor not found for entity with id: ID` | The node was unable to find the remote IdP descriptor. |
| `Unable to complete SAML2 authentication, SP descriptor not found for entity with id: ID` | The node was unable to find the SP descriptor. |
| `Unable to retrieve SAML2 state from SFO` | The node was unable to retrieve the SAML 2.0 state from the second-factor authentication. |
| `Unable to complete SAML2 authentication, response data not found` | The node was unable to read the SAML 2.0 response data. |
| `Failed to remove data for responseKey starting with keyname` | The node was unable to remove the SAML 2.0 response data. |
| `AuthConsumer endpoint reported error code: code` | The `AuthConsumer` endpoint (Assertion Consumer Service URL) on the SP reported an error. |

### Example

Read the following sections for examples that show this node in a SAML 2.0 authentication journey:

PingAM SSO and SLO in integrated mode.

PingOne Advanced Identity Cloud Configure Advanced Identity Cloud for integrated mode.

## Social Facebook node

This node duplicates the OAuth 2.0 node but is preconfigured to work with Facebook. You specify only the `Client ID` and `Client Secret`.

> **Note**
>
> This node and its related services are deprecated.
> Depending on your deployment, you can find information on the current methods for implementing social authentication in:
>
> - Advanced Identity Cloud Social authentication
> - PingAM PingGateway

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

## Outcomes

- `Account exists`
- `No account exists`

Evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from Facebook are found in the user data store; otherwise, evaluation continues along the `No account exists` path.

## Properties

| Property | Usage |
|---|---|
| Client ID | Specifies the `client_id` parameter as provided by Facebook. |
| Client Secret | Specifies the `client_secret` parameter as provided by Facebook. |
| Authentication Endpoint URL | Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://www.facebook.com/dialog/oauth` |
| Access Token Endpoint URL | Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://graph.facebook.com/v2.12/oauth/access_token` |
| User Profile Service URL | Specifies the user profile URL that returns profile information. Default: `https://graph.facebook.com/v2.6/me?fields=name%2Cemail%2Cfirst_name%2Clast_name` |

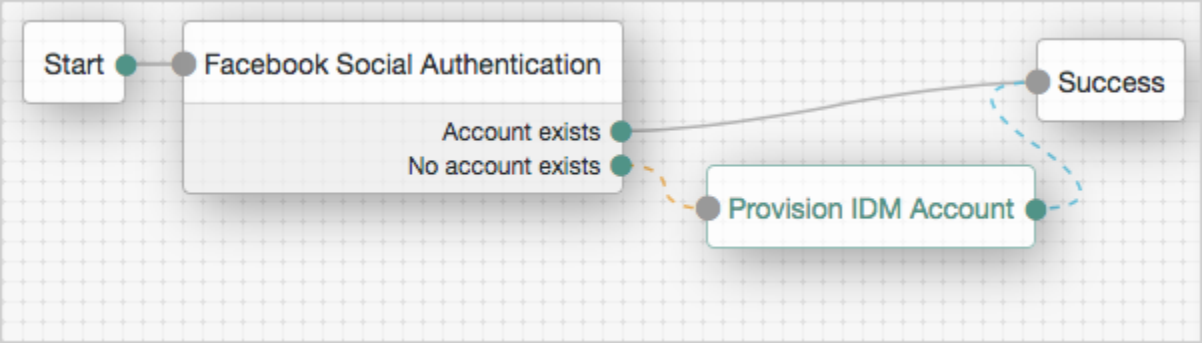
| Property | Usage |
| --- | --- |
| OAuth Scope | Specifies a comma-separated list of user profile attributes the client application requires, according to The OAuth 2.0 Authorization Framework (RFC 6749). The list depends on the permissions the resource owner, such as the end user, grants to the client application. |
| Redirect URL | Specifies the URL the user is redirected to by Facebook after authenticating to continue the flow.<br>Set this property to the URL of the AM UI. For example, `https://am.example.com:8443/am/XUI/`.<br>**Tip**<br>If the tree is not in the Top Level Realm, you can specify the realm in the redirect URL. Use a DNS alias for the realm, or add the realm as a query parameter, for example, `https://am.example.com:8443/am/XUI/?realm=/mySubRealm`.<br>Learn more in Configure DNS aliases to access a realm. |
| Social Provider | Specifies the name of the social provider for which this node is being set up.<br>Default: `facebook` |
| Auth ID Key | Specifies the attribute the social identity provider uses to identify an authenticated individual.<br>Default: `id` |
| Use Basic Auth | Specifies that the client uses HTTP Basic authentication when authenticating to the social provider.<br>Default: `true` |
| Account Provider | Specifies the name of the class that implements the account provider.<br>Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider` |
| Account Mapper | Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from Facebook.<br>Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` |
| Attribute Mapper | Specifies the list of fully qualified class names for implementations that map attributes from Facebook to AM profile attributes.<br>Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper\|uid\|facebook-` |

| Property | Usage |
| --- | --- |
| Account Mapper Configuration | Specifies the attribute configuration used to map the account of the user authenticated in the Social Facebook provider to the local data store in AM. Valid values are in the form `provider-attr=local-attr`.<br>Default: `id=uid`.<br><br>**Tip**<br>When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings using dot notation.<br>For example, given a JSON payload of:<br>`{ "sub" : "12345", "name" : { "first_name" : "Demo", "last_name" : "User" } }`<br>You can create a mapper, such as `name.first_name=cn`. |
| Attribute Mapper Configuration | Map of Facebook user account attributes to local user profile attributes, with values in the form `provider-attr=local-attr`.<br>Default: `name=cn`, `last_name=sn`, `id=uid`, `first_name=givenname`, `email=mail`.<br><br>**Tip**<br>When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings using dot notation.<br>For example, given a JSON payload of:<br>`{ "sub" : "12345", "name" : { "first_name" : "Demo", "last_name" : "User" } }`<br>You can create a mapper, such as `name.first_name=cn`. |
| Save attributes in the session | When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session.<br>Default: `true`. |

| Property | Usage |
| --- | --- |
| OAuth 2.0 Mix-Up Mitigation Enabled | Controls whether the authentication node carries out additional verification steps when it receives the authorization code from the authorization server.<br>Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned as the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.<br>The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response contains an issuer value ( `iss` ) for the client to validate.<br>Learn more in section 4 of OAuth 2.0 Mix-Up Mitigation Draft. |
| Token Issuer | Corresponds to the expected issuer identifier value in the `iss` field of the ID token.<br>Example: `https://graph.facebook.com` |

### Example

The following example shows the node in context:




## Social Google node

This node duplicates the OAuth 2.0 node, but is preconfigured to work with Google. You specify only the `Client ID` and `Client Secret` .

> **Note**
>
> This node and its related services are deprecated.
> Depending on your deployment, you can find information on the current methods for implementing social authentication in:
>
> - Advanced Identity Cloud Social authentication
> - PingAM PingGateway

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

- `Account exists`
- `No account exists`

Evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from Google are found in the user data store; otherwise, evaluation continues along the `No account exists` path.

### Properties

| Property | Usage |
|---|---|
| Client ID *(required)* | Specifies the `client_id` parameter as provided by Google. |
| Client Secret *(required)* | Specifies the `client_secret` parameter as provided by Google. |
| Authentication Endpoint URL | Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://accounts.google.com/o/oauth2/v2/auth` |
| Access Token Endpoint URL | Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://www.googleapis.com/oauth2/v4/token` |
| User Profile Service URL | Specifies the user profile URL that returns profile information. Default: `https://www.googleapis.com/oauth2/v3/userinfo` |
| OAuth Scope | Specifies a space-separated list of user profile attributes the client application requires, according to The OAuth 2.0 Authorization Framework (RFC 6749). The list depends on the permissions the resource owner, such as the end user, grants to the client application. Default: `profile email`. |

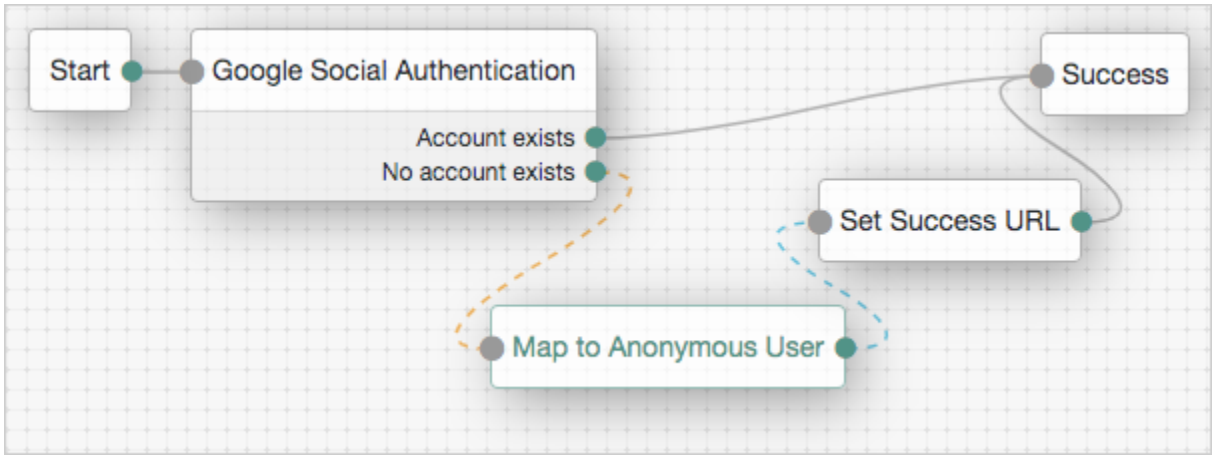
| Property | Usage |
| --- | --- |
| Redirect URL | Specifies the URL the user is redirected to by Google after authenticating to continue the flow.<br>Set this property to the URL of the AM UI. For example, `https://am.example.com:8443/am/XUI/`.<br><br>**Tip**<br>If the tree is not in the Top Level Realm, you can specify the realm in the redirect URL. Use a DNS alias for the realm, or add the realm as a query parameter; for example, `https://am.example.com:8443/am/XUI/?realm=/mySubRealm`.<br>Learn more in Configure DNS aliases to access a realm. |
| Social Provider | Specifies the name of the social provider for which this node is being set up.<br>Default: `google` |
| Auth ID Key | Specifies the attribute the social identity provider uses to identify an authenticated individual.<br>Default: `sub` |
| Use Basic Auth | Specifies that the client uses HTTP Basic authentication when authenticating to Google.<br>Default: `true` |
| Account Provider | Specifies the name of the class that implements the account provider.<br>Default:<br>`org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider` |
| Account Mapper | Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from Google.<br>Default:<br>`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` |
| Attribute Mapper | Specifies the list of fully qualified class names for implementations that map attributes from Google to AM profile attributes.<br>Default:<br>`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper\|iplanet-am-user-alias-list\|google-` |

| Property | Usage |
| --- | --- |
| Account Mapper Configuration | Specifies the attribute configuration used to map the account of the user authenticated in the Social Google provider to the local data store in AM. Valid values are in the form `provider-attr=local-attr`.<br>Default: `sub=uid`.<br><br>**Tip**<br>When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings using dot notation.<br>For example, given a JSON payload of:<br>`{ "sub" : "12345", "name" : { "first_name" : "Demo", "last_name" : "User" } }`<br>You can create a mapper, such as `name.first_name=cn`. |
| Attribute Mapper Configuration | Map of Google user account attributes to local user profile attributes, with values in the form `provider-attr=local-attr`.<br>Default: `sub=uid`, `name=cn`, `given_name=givenName`, `family_name=sn`, `email=mail`.<br><br>**Tip**<br>When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings using dot notation.<br>For example, given a JSON payload of:<br>`{ "sub" : "12345", "name" : { "first_name" : "Demo", "last_name" : "User" } }`<br>You can create a mapper, such as `name.first_name=cn`. |
| Save attributes in the session | When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session.<br>Default: `true`. |

| Property | Usage |
| --- | --- |
| OAuth 2.0 Mix-Up Mitigation Enabled | Controls whether the authentication node carries out additional verification steps when it receives the authorization code from the authorization server.<br>Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned as the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.<br>The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response contains an issuer value ( `iss` ) for the client to validate.<br>Learn more in section 4 of OAuth 2.0 Mix-Up Mitigation Draft. |
| Token Issuer | Corresponds to the expected issuer identifier value in the `iss` field of the ID token.<br>Example: `https://accounts.google.com` |

### Example

The following example shows the node in context:




## Social Ignore Profile node

Specifies whether to ignore a local user profile.

If evaluation flows through this node after successful social authentication, AM issues an SSO token regardless of whether a user profile exists in the data store. AM does not check for whether a user profile is present.

**Note**

This node and its related services are deprecated.
Depending on your deployment, you can find information on the current methods for implementing social authentication in:

- Advanced Identity Cloud Social authentication
- PingAM PingGateway

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

Single outcome path.

### Properties

This node has no configurable properties.

## Social Provider Handler node

The **Social Provider Handler** node attempts to authenticate a user with an identity provider they select in the Select Identity Provider node. The **Social Provider Handler** node collects relevant profile information from the provider, transforms the profile information into the appropriate attributes, and returns the user to the journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads the user's selected social identity provider from shared state.

Implement the Select Identity Provider node before this node to capture the social provider name.

## Dependencies

- The Social Identity Provider service must be configured with the details of at least one social identity provider.
- The user must have selected a social identity provider in a previous node in the journey.

## Configuration

| Property | Usage |
|---|---|
| **Transformation Script** *(required)* | The *normalization* script of each provider maps that provider's attributes to a profile format AM can use.<br>The *transformation* script then transforms the normalized social profile to an identity or managed object.<br>• PingAM<br>Select `Normalized Profile to Identity` or a custom script that transforms the profile to an identity object. Review the sample script (normalized-profile-to-identity.js) for a list of bindings.<br>• Advanced Identity Cloud and Ping Identity Platform<br>Select `Normalized Profile to Managed User` (default) or a custom script to transform the profile to a managed object. Review the sample script (linkhttps://docs.pingidentity.com/pingoneaic/latest/am-scripting/sample-scripts.html#normalized-profile-to-managed-user-js[normalized-profile-to-managed-user.js]) for a list of bindings.<br>**Important**<br>Don't use normalization scripts (`<Identity provider>-profile-normalization.*`) for this purpose. |
| **Username Attribute** | Platform UI<br>The attribute in the underlying identity service (PingIDM) that contains the username for this object. |

| Property | Usage |
| --- | --- |
| **Client Type** | The client type you're using to authenticate to the provider. Select one of the following:<br>• `BROWSER` (default) Select this type for Ping Identity-provided user interfaces or the Ping SDKs for JavaScript.<br>With this setting, the node returns the Redirect callback. Find more information in the `RedirectCallback` documentation for:<br>◦ PingAM<br>◦ Advanced Identity Cloud<br>• `NATIVE` Select this type for the Ping SDKs for Android or iOS.<br>With this setting, the node returns the Idp callback. Find more information in the `IdPCallback` documentation for:<br>◦ PingAM<br>◦ Advanced Identity Cloud |
| **Store Tokens** | When `true`, the node stores access and refresh tokens in the transient state for use by subsequent nodes in the journey.<br>In some cases, the social provider requires these tokens, for example, to revoke user authorization. If you choose to store tokens, you can configure a Scripted Decision node later in the journey to handle your social provider use case.<br>Default: `false` |

## Outputs

- If no profile information is returned from the social provider, the journey follows the `Social auth interrupted` outcome.
- If the node retrieves profile information from the social identity provider, it transforms a normalized version of the profile and stores it in `objectAttributes` in transient state.
- To link existing users, the `aliasList` is updated and saved to `objectAttributes` in transient state.
- The node stores the social identity subject as the `username` both directly in shared state and in its `objectAttributes`.
- The node also updates `socialOAuthData` in transient state with all existing node state, social provider data, and associated tokens.

> **Note**
>
> Make sure you copy required transient data to shared state because all transient data is removed if the node is followed by an interactive page later in the journey.

## Outcomes

### `Account exists`

Social authentication succeeded, and a matching Ping Identity account exists.

### `No account exists`

Social authentication succeeded, but no matching Ping Identity account exists.

> **Note**
>
> PingAM only
>
> To ensure existing users are dynamically linked, complete these additional steps:
>
> 1. Connect the `No account exists` outcome to a Scripted Decision node.
> 2. Write a Scripted Decision node script and use the `idRepository` binding's `get-` and `setAttribute` methods to check for an existing account and add a link by updating the account-linking attribute, `iplanet-am-user-alias-list`.
>
>    For multiple OIDC providers, add links to the existing list. For example:
>
>    ```
>    "iplanet-am-user-alias-list": [
>        "google_IDP-123456789",
>        "amazon_IDP-987654321"
>    ],
>    ```
>
> 3. Connect the Scripted Decision node to a Provision Dynamic Account node to update the account.
>
> Advanced Identity Cloud and Ping Identity Platform deployments only: To ensure existing users are dynamically linked, connect the `No account exists` outcome to an Identify Existing User node followed by a Patch Object node to create the link.

### `Social auth interrupted`

The user interrupted the social authentication journey after the node requested profile information from the social identity provider. This can happen in the following situations:

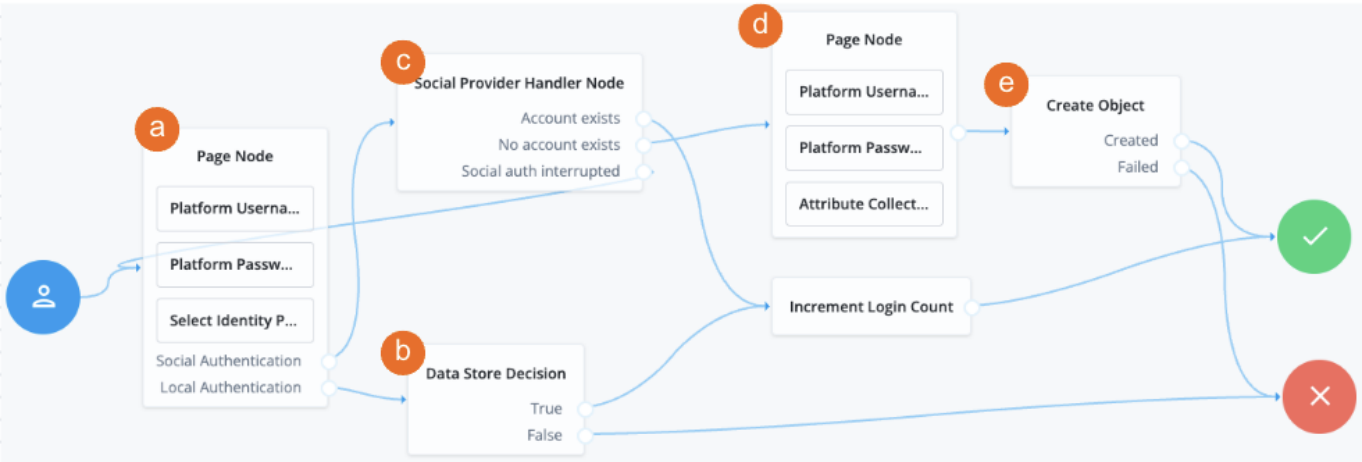
- The user clicks the **Back** button in their browser from the social identity provider's login page
- The user clicks the **Cancel** button on the social identity provider's login page
- The user re-enters the journey URL in the same browser window

In this case, the node routes the user back to the Select Identity Provider node to select a social identity provider again.

## Examples

Advanced Identity Cloud

**Example 1: Social authentication** This example shows the **Social Provider Handler** and **Select Identity Provider** nodes in a social authentication journey.




a The Page node containing the Select Identity Provider node prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c If the user selects social authentication, the Social Provider Handler node does the following:

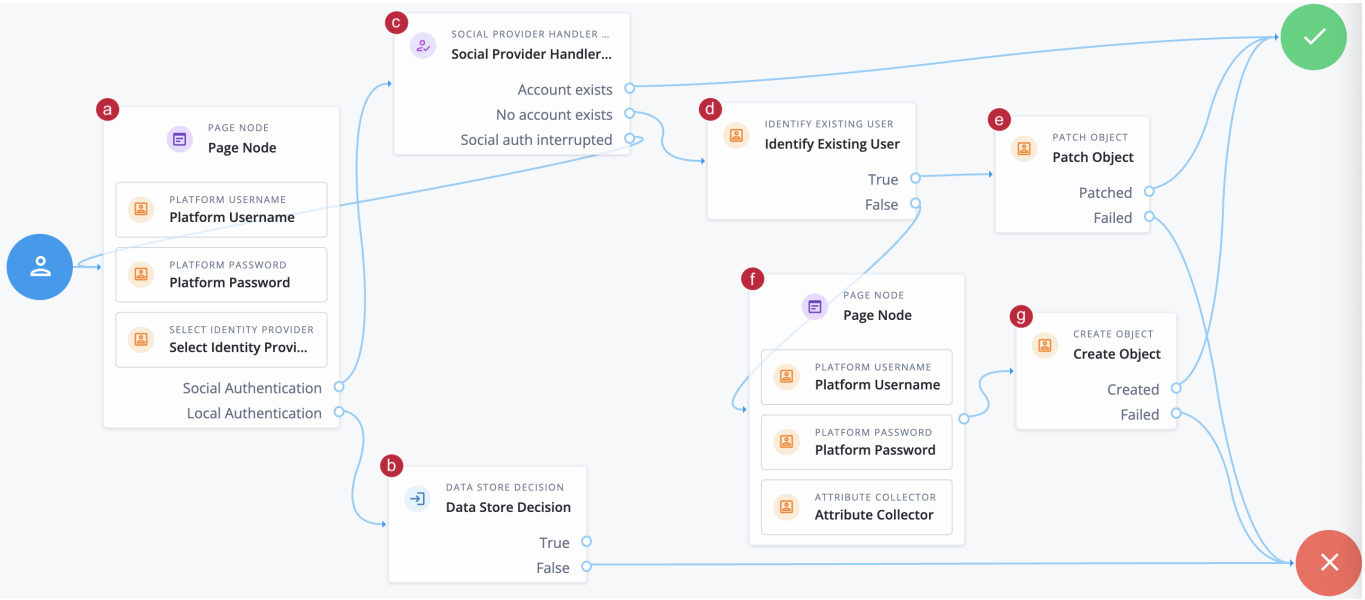
- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information, and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
  - If the user has an existing identity, authenticates that identity
  - If the user doesn't have an identity, routes the user to another page node
  - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

d The nodes on the page node request the information required to *register* a new identity.

e The Create Object node creates the new identity in AM.

**Example 2: Dynamic account linking**

This example shows a social authentication journey with dynamic account linking.




a A Page node contains the Select Identity Provider node node that prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c If the user selects social authentication, the **Social Provider Handler** node does the following:

- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information, and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
  - If the user has an existing identity, authenticates that identity
  - If the user doesn't have an identity, routes the user to the Identify Existing User node
  - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

d The Identify Existing User node checks if the user exists in AM using the **Identity Attribute** specified and does the following:

- If the user exists, routes the user to the Patch Object node
- If the user doesn't exist, routes the user to another page node

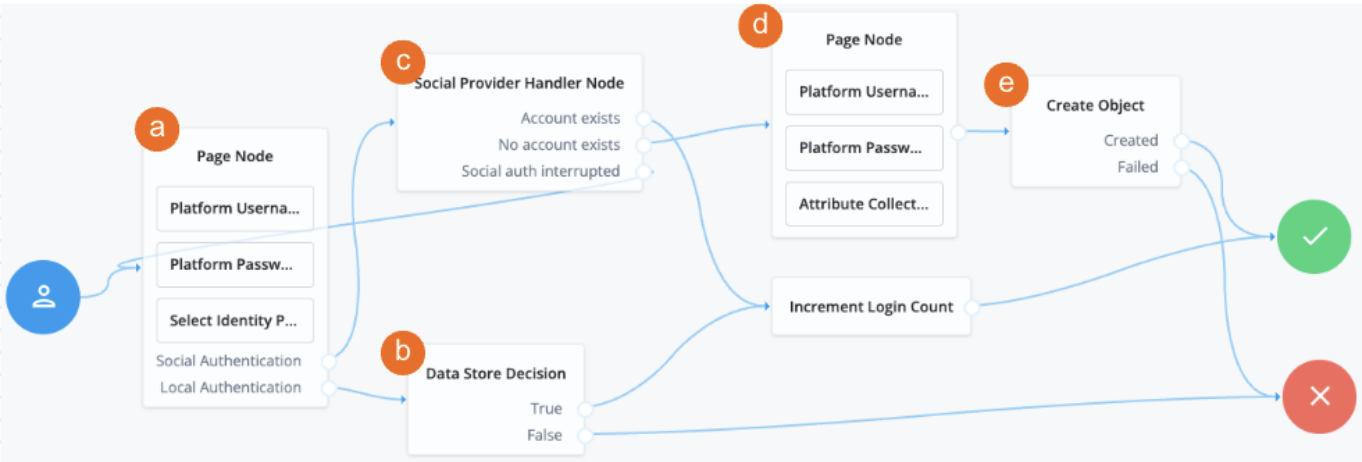
e The Patch Object node updates the existing user to create the link.

f The nodes on the page node request the information required to *register* a new identity.

g The Create Object node creates the new identity in AM.

PingAM

This example shows the **Social Provider Handler** and **Select Identity Provider** nodes in a social authentication journey.




a The Page node containing the Select Identity Provider node prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c If the user selects social authentication, the **Social Provider Handler** node does the following:

- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information, and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
  - If the user has an existing identity, authenticates that identity
  - If the user doesn't have an identity, routes the user to another page node
  - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

d The nodes on the page node request the information required to *register* a new identity.

e The Create Object node creates the new identity in AM.

## Legacy Social Provider Handler node

This legacy node is similar to the newer Social Provider Handler node. It takes a provider selection from the Select Identity Provider node and attempts to authenticate the user. The node collects relevant profile information from the provider, transforms the profile information into the appropriate attributes and returns the user to the journey.

This node remains supported in existing journeys. For new journeys, use the Social Provider Handler node instead.

Implement this node with the Select Identity Provider node to use the Social Identity Provider Service.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

### `Account exists`

Social authentication succeeded, and a matching Advanced Identity Cloud or AM account exists.

### `No account exists`

Social authentication succeeded, but no matching Advanced Identity Cloud or AM account exists.

## Properties

| Property | Usage |
|---|---|
| **Transformation Script** *(required)* | This script is used after the configured provider's *normalization* script has mapped the social identity provider's attributes to a profile format compatible with AM. The *transformation* script then transforms a normalized social profile to an identity or a managed object.<br>• For PingAM select `Normalized Profile to Identity`, or a custom script you've created to transform the profile to an identity object. Find information on the scripts and bindings in normalized-profile-to-identity.js.<br>• For Advanced Identity Cloud and Ping Identity Platform deployments, select `Normalized Profile to Managed User` (default), or a custom script you've created to transform the profile to a managed object. Find information on the scripts and bindings in normalized-profile-to-managed-user.js.<br>Normalization scripts ( `<Identity provider>-profile-normalization.*` ) are not suitable for this purpose. |
| **Username Attribute** | Advanced Identity Cloud and Ping Identity Platform deployments only<br>The attribute in the underlying identity service (PingIDM) that contains the username for this object. |

| Property | Usage |
| --- | --- |
| **Client Type** | Specify the client type you are using to authenticate to the provider.<br>Use the default, `BROWSER`, with ForgeRock-provided user interfaces or the Ping SDK for JavaScript. This causes the node to return the Redirect callback.<br>Find more information in the documentation on this callback for:<br>• Advanced Identity Cloud<br>• PingAM<br><br>Select `NATIVE` with the Ping SDKs for Android or iOS. This causes the node to return the IdP callback.<br>Find more information in the documentation on this callback for:<br>• Advanced Identity Cloud<br>• PingAM |
| **Store Tokens** | When `true`, the node stores access and refresh tokens in the transient state for use by subsequent nodes in the journey.<br>In some cases, the social provider requires these tokens, for example, to revoke user authorization. If you choose to store tokens, you can configure a Scripted Decision node later in the journey to handle your social provider use case.<br>Default: `false` |

## Write Federation Information node

Creates a persistent link between a remote IdP account and a local account in the SP, if none exists yet. If a transient link exists, it is persisted. Existing account links with different IdPs are not lost.

Use this node with the SAML2 Authentication node, and ensure that the NameID Format is `persistent`.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

This node has no configurable properties.

You can find examples in the SAML documentation for:

- PingAM
- Advanced Identity Cloud

# Identity management nodes

## Accept Terms and Conditions node

The **Accept Terms and Conditions** node prompts the user to accept the currently active terms and conditions, configured in the Platform UI.

Find more information in Terms and conditions.

Use this node for registration, or combined with the Terms and Conditions Decision node for progressive profiling or log in.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

None.

### Dependencies

This node depends on the underlying identity service (PingIDM) for the active terms and conditions.

### Configuration

This node has no configurable properties.

### Outputs

The node writes a `termsAccepted` object to the shared node state. The object contains these fields:

- `acceptDate`: A timestamp string indicating when the user accepted the terms.
- `termsVersion`: A string indicating the version of the accepted terms.

### Outcomes

Single outcome path; the user accepted the terms and conditions.

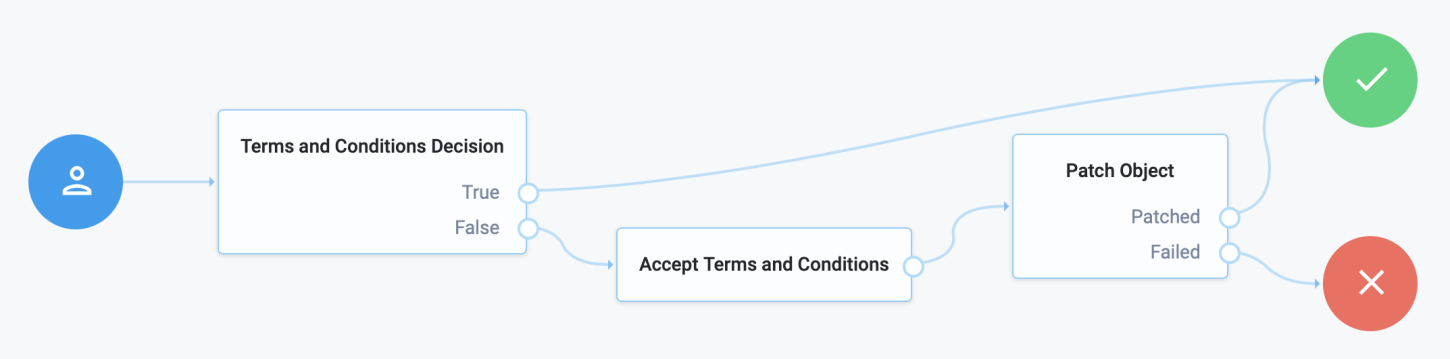
### Errors

This node does not log error or warning messages of its own.

### Example

For progressive profiling, include this node after a Terms and Conditions Decision node. If the user has not accepted the latest version of the terms and conditions, evaluation takes them to a page that requires them to accept the current terms and conditions.

The Patch Object node stores the acceptance response in the underlying identity service (PingIDM) if the user accepts:




## Attribute Collector node

The **Attribute Collector** node collects the values of attributes for use later in the flow; for example, to populate a new account during registration.

This node supports three types of attributes:
`string`
`boolean`
`number`

To request a value, the attribute must be present in the identity schema (PingIDM schema) for the current identity object.

The node lets you configure whether the attributes are required to continue and whether to use a policy filter of the underlying identity service (PingIDM) to validate them.

Use the node alone or within a Page node.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

For validation, this node reads the **Identity Attribute** (default: `userName`) from the shared node state. It uses the value to look up the identity object.

It prompts the user for the attributes to collect.

## Dependencies

This node depends on the underlying identity service (PingIDM) to store the user profile and perform validation.

## Configuration

| Property | Usage |
|---|---|
| **Attributes to Collect** | A list of the attributes to collect based on those in the identity schema (PingIDM schema) for the current identity object.<br>Default: none |
| **All Attributes Required** | When enabled, all attributes collected in this node are required in order to continue.<br>Default: false |

| Property | Usage |
| --- | --- |
| **Validate Input** | When enabled, validate the content against any policies specified in the identity schema (IDM schema) for each collected attribute.<br>Learn more in the documentation on using policies to validate data for:<br>• Advanced Identity Cloud<br>• PingAM<br>If you enable this property, the collected identity attributes must be *User Editable*.<br>To make an attribute user-editable in the IDM admin UI:<br>1. Go to **Configure > Managed Objects > *object-name***.<br>2. Click the pencil (✎) icon, then click **Show advanced options**.<br>3. Select the **User Editable** toggle.<br>Learn more in the documentation on property configuration properties for:<br>• Advanced Identity Cloud<br>• Self-managed platform deployments<br>Default: false |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

## Outputs

The node writes the attributes and their values to the shared node state.

## Outcomes

Single outcome path; on success, downstream nodes can read the attributes from the shared node state.

## Errors

This node does not log error or warning messages of its own.

## Examples

> **Important**
>
> These examples assume an Advanced Identity Cloud tenant or self-managed Ping Identity Platform deployment.

### Add date and datetime fields to a journey

The **Attribute Collector** node lets you add properties (attributes) that follow a date or datetime (date and time of day). The format of the date comes from the locale set in your browser.

The following table displays the differences between date and datetime:

| Display format | Managed object field format | Notes |
|---|---|---|
| Date only | String format<br>Advanced Identity Cloud example: `frIndexedString1` | The format of the date comes from the locale set in your browser.<br>For example, if the locale is English, then the format presented to the end user is `MM-DD-YYYY`. If the local is French, the format is `DD-MM-YYYY`. |
| Date and time | String format (date and time of day)<br>Advanced Identity Cloud example: `frIndexedDate1` | The format of the date comes from the locale set in your browser.<br>For example, if the locale is English (United States), the format presented to the end user is `MM-DD-YYYY`. If the locale is French, the format is `DD-MM-YYYY`.<br>**Important**<br>Advanced Identity Cloud only<br>You can only use the date general purpose extension attributes for datetime. Don't attempt to use the indexed date properties with the date format. |

**Note**

The rendering of the date to the end user changes, depending on the locale set in the browser. However, the server *stores* the date value in UTC format as `YYYY-MM-DDHH:MM:SS`. For example, `2023-09-13T08:01:00Z`.

To render the date or datetime UI element to an end user with the **Attribute Collector** node, you must do the following:

Advanced Identity Cloud

1. Specify the property to use as date or datetime:
   - General purpose extension attributes — Recommended. Use OOTB indexed attributes:
     - Use the `Date` indexed properties, such as `frIndexedDate` for datetime *only*. Don't use it as a date property.
     - Use `String` data types.
   - Create your own custom attributes — Create custom attributes, prefixed with `custom_`.
2. Apply formatting and policies to the property in the IDM admin UI for `Date` or `Datetime`.

Ping Identity Platform

1. Specify the IDM property:
   - In the Ping Identity Platform admin UI, go to **Configure** > **Managed Objects** > *Select object*, for example, **User**.
   - Use an existing `string` property or create your own string property.

     Learn more in Property Configuration Properties in the IDM documentation.
2. Apply formatting and policies to the property in the IDM admin UI for `Date` or `Datetime`.

### Apply formatting and policies

1. In the Advanced Identity Cloud admin console (Platform UI in self-managed deployments) go to **Native Consoles > Identity Management**.
2. Select **Configure > Managed Objects > *object-type***, for example, **User** or **Alpha_user**.
3. Select the property to use.
4. To select the format of the attribute, on the **Details** tab, click the **Format** field, and select one of the following:
   - For date property — `Date`
   - For datetime property — `Datetime`
5. Click **Save**.
6. On the **Validation** tab, click **+ Add Policy**.
7. In the **Policy Id** field, enter one of the following:
   - For date property — `valid-formatted-date`
   - For datetime property — `valid-datetime`

     Find more information in the documentation on applying policies for:
     - Advanced Identity Cloud
     - Ping Identity Platform
8. Click **Add**.
9. In your journey, in the **Attribute Collector** node, add the property name to the **Attributes to Collect** field.

   Advanced Identity Cloud provides the following properties:
   - For date property — `frIndexedString1`
   - For datetime property — `frIndexedDate1`

For an in-depth use case, add the `date` or `datetime` property to an **Attribute Collector** node in a registration flow.

> **Tip**
>
> For Advanced Identity Cloud, you can find more information in User self-registration.

The following video shows an example of a journey collecting the datetime from an end user using the Attribute Collector node:

Your browser does not support the video tag.

## Attribute Present Decision node

The **Attribute Present Decision** node checks whether an attribute is present on an object, including private attributes. There is no need to specify the value of the attribute.

Use this node during an update password flow to check whether the local account has a password, for example.

This node is similar to the Attribute Value Decision node when that node is set to use the `PRESENT` operator, except it can't return the value of the attribute, but can work with private attributes.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

This node reads the **Identity Attribute** from the shared node state. If it can't read the **Identity Attribute**, it reads the `userName` from the shared node state.

It uses the value to look up the identity object.

### Dependencies

This node depends on the underlying identity service (PingIDM) to look up the user object.

## Configuration

| Property | Usage |
| --- | --- |
| **Present Attribute** | The attribute whose presence you want to verify in the the underlying identity service (PingIDM) object. This can be an otherwise private attribute, such as `password`.<br><br>**Note**<br>This field is case-sensitive and must match the the underlying identity service (PingIDM) object attribute. For example, `givenName`, not `givenname`.<br><br>Default: `password` |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

## Outputs

None.

## Outcomes

### `True`

The node found the attribute in the managed identity object.

### `False`

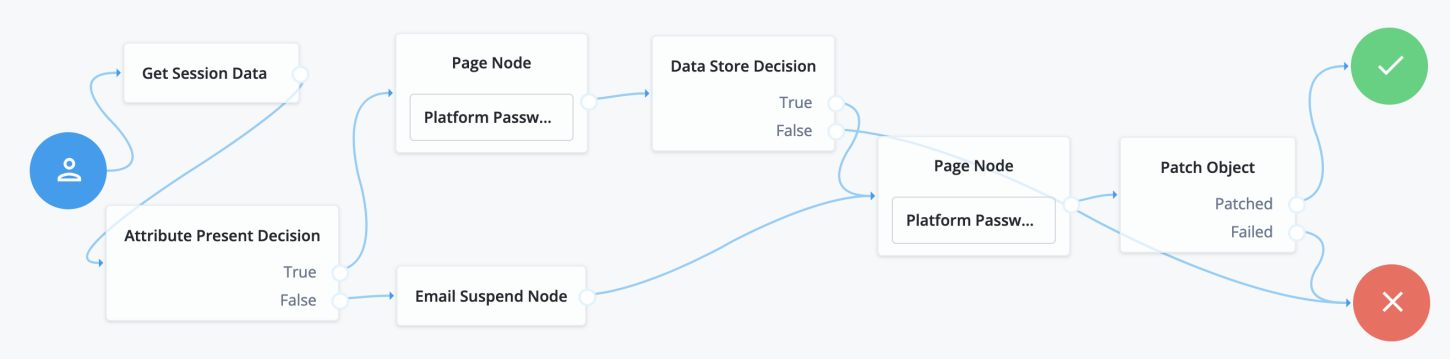
Any other case.

## Errors

This node does not log error or warning messages of its own.

## Example

This journey to update a password uses the **Attribute Present Decision** node to check whether the account has a password:




The user has already authenticated before beginning this journey:

- The Get Session Data node stores the `userName` from the session.
- The **Attribute Present Decision** node checks whether the user object has a password attribute.
- If so, the first Page node with the Platform Password node prompts the user for the current password.
- Otherwise, the Email Suspend node sends an email to the user and suspends the flow until the user follows the link in the message.
- The Data Store Decision node confirms the username-password credentials.
- The second Page node with the Platform Password node prompts the user for the new password.
- The Patch Object node updates the user object with the new password.

## Attribute Value Decision node

Verifies that the specified attribute satisfies a specific condition.

Use this node to check whether an attribute's expected value is equal to a collected attribute value, or to validate that the specified attribute was collected.

Examples:

- To validate that a user provided the country attribute during registration, set the comparison operation to `PRESENT`, and the comparison attribute to `country`.
- To validate that the country attribute is set to the United States, set the comparison operation to `EQUALS`, the comparison attribute to `country`, and the comparison value to `United States`.

Use Attribute Present Decision node instead when you need to check for the presence of a private attribute, such as `password`.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |

| Product | Available? |
|---|---|
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Properties

| Property | Usage |
|---|---|
| Comparison Operation | The operation to perform on the object attribute:<br>**PRESENT**<br>Checks the existence of an attribute regardless of its value.<br>**EQUALS**<br>Checks if the object's attribute value equals the configured comparison value. |
| Comparison Attribute | The object attribute to compare. |
| Comparison Value | When **Comparison Operation** is `EQUALS`, compare this value to the provided attribute value. |
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM). |

## Consent Collector node

Prompts the user for consent to share their profile data.

A consent notice is listed for each identity mapping that has consent enabled. If an identity mapping is not created, or the mappings do not have privacy and consent enabled, the product using this node does not show a consent message to the user.

This node is primarily used in progressive profile and registration flows.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Properties

| Property | Usage |
| --- | --- |
| All Mappings Required | If enabled, all mappings listed by this node require consent in order to move forward. |
| Privacy & Consent Message | Localized message providing the privacy and consent notice. The key is the language, such as `en` or `fr`, and the value is the message to display. |

## Create Object node

The **Create Object node** lets you create a new object in the underlying identity service (PingIDM) based on information collected during authentication, such as user registration.

Any managed object attributes that are marked as required in the underlying identity service (PingIDM) must be collected during authentication in order to create the new object.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

This node requires the managed object attributes marked as required

## Configuration

| Property | Usage |
| --- | --- |
| **Identity Resource** | The type of managed identity resource object that this node creates. It must match the identity resource type for the current flow.<br><br>**Tip**<br>To check for the available managed identity resource types, go to the IDM admin UI, and open the **Manage** drop-down list in the upper right corner of the screen. Identity managed object types are preceded by the icon.<br><br>Default: `managed/user` |

## Outputs

This node doesn't change the shared state.

## Outcomes

This node has the following outcomes:

- Created
- Failed

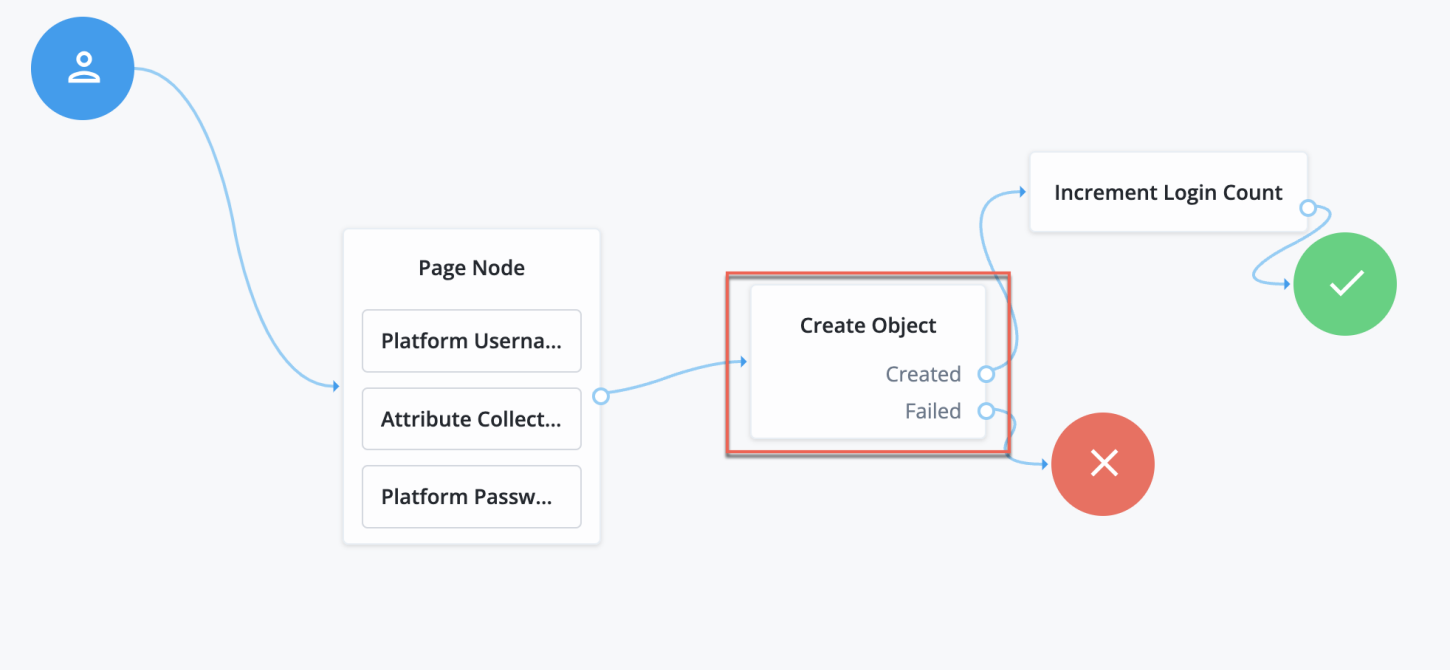
## Errors

This node can log the following warning messages:

| Message | Notes |
| --- | --- |
| Failed to create object | The preceding nodes don't provide all the fields required to create the object. |
| Failed to retrieve object's schema | The node failed to get the list of required attributes from the **Identity Resource** schema. |

## Example

The following example uses this node with the Increment Login Count node to create a new user object.




- The Page node includes various nodes that collect attributes and store them in the shared node state.
- The Create Object node uses these attributes to create the new user.
- The Increment Login Count node resets the retry count on successful authentication of the new user.

## Create Password node

Lets users create a password when provisioning an account.

**Note**

This node and its related services are deprecated.
Depending on your deployment, you can find information on the current methods for implementing social authentication in:

- Advanced Identity Cloud Social authentication
- PingAM PingGateway

Social identity providers do not provide a user's password. Use this node to provide a password to complete the user's credentials before provisioning an account.

**Important**

The flow must provision an account after prompting the user for a password, for example, by using the Provision Dynamic Account node. If no account is provisioned, the flow does not save the password.
Do not place any nodes that request additional input from the user between this node and the provisioning node; otherwise, the password is lost.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

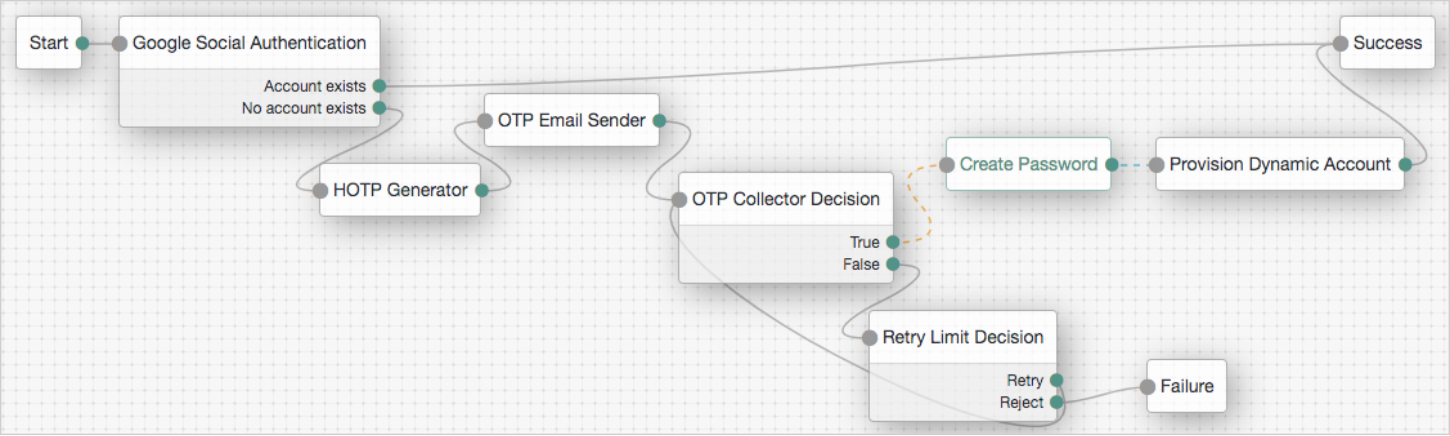
Single outcome path.

### Properties

| Property | Usage |
|---|---|
| minPasswordLength | The minimum number of characters the password must contain. |

### Example

The following example lets users who have performed social authentication using Google provide a password and provision an account when they don't have one. They must enter a one-time passcode to verify they are the owner of the Google account.




## Display Username node

Fetches a username based on a different identifying attribute, such as an email address, then displays it.

To email the username to the user instead, use the Identify Existing User node combined with a Email Suspend node or Email Template node.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Properties

| Property | Usage |
|---|---|
| User Name | The attribute used to identify the username in the managed identity object. |
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>When this node serves to recover a username, the identity attribute should be some other attribute that is unique to a user object, such as the email address.<br>The node raises an exception when more than one value exists for this attribute. Make sure the value of whatever attribute you select is unique for each user. |

## Identify Existing User node

The **Identify Existing User** node verifies if a user exists based on an identifying attribute, such as an email address, then makes the value of a specified attribute available in the shared node state.

Use this node in a forgotten password flow to fetch a username to email to the user. To display the username on the screen, use the Display Username node instead.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

This node reads the **Identity Attribute** (default: `mail`) from the shared node state.

If the **Identity Attribute** is not available, it reads the `userName` from the shared node state.

## Dependencies

This node depends on the underlying identity service (PingIDM) to store the user profile.

## Configuration

| Property | Usage |
| --- | --- |
| **Identifier** | The attribute to collect from a managed identity object.<br>Default: `userName` |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>When this node serves to recover a username, the identity attribute should be some other attribute that is unique to a user object, such as the email address.<br>Default: `mail` |

## Outputs

The node writes the **Identifier** and the user account `_id` to the shared node state.

If the **Identifier** differs from `userName`, this node also writes the `userName` to the shared node state.

## Outcomes

### `True`

The node successfully identified the user and updated the shared node state.

### `False`

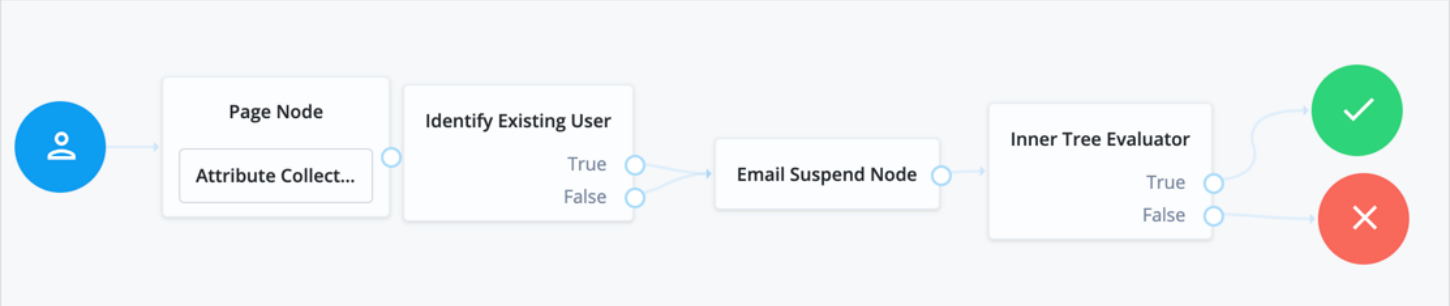
Any other case.

### Errors

This node does not log error or warning messages of its own.

### Example

The following example shows a flow to reset a forgotten password:




- The user enters their email in the Attribute Collector node of the Page node.
- The **Identify Existing User** node uses the email address to look up the username of the user's account. If it finds the user account, it adds the username to the shared node state.
- The Email Suspend node emails the user and suspends authentication.
- Once authentication resumes, the Inner Tree Evaluator node sends the user to a different flow to reset their password.

## KBA Decision node

Checks whether the user account has the required minimum number of KBA questions.

To set the number of KBA questions, edit **Configure > Security Questions > Questions > Number** in the IDM admin UI.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Outcomes

- `True`

- `False`

Evaluation continues along the `True` path if the user profile holds at least the minimum number of KBA questions; otherwise, evaluation continues along the `False` path.

### Properties

| Property | Usage |
|---|---|
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM). |

## KBA Definition node

The **KBA Definition** node collects knowledge-based authentication (KBA) questions and answers.

Use this node when creating or updating a user with KBA enabled.

You can find more information in Security questions.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

None. This node doesn't require any attributes from the shared node state.

### Dependencies

This node depends on the underlying identity service (PingIDM) for the KBA configuration.

## Configuration

| Property | Usage |
| --- | --- |
| **Purpose Message** | A localized message describing the purpose of the data requested from the user.<br>Default: none |
| **Allow User-Defined Questions** | When enabled, users can create their own KBA questions. Disable this setting to restrict users to select from predefined questions only.<br>Default: Enabled |
| **Questions** | Create or modify custom localized questions that the user can choose from when defining security questions.<br>To add a localized security question:<br>1. Click **+** to open the **Add a Security Question** form.<br>2. Select from the list of existing locales or add a new locale, type a question into the text field, and click **Done**.<br>3. Repeat to add further questions, and click **Save** when complete.<br>To edit an existing security question, click the edit icon ✎, make your changes, and click **Save**.<br>Default: `What's your favorite color?` (locale: `en`) |

## Outputs

The node writes the KBA questions and answers in the transient shared node state.

## Outcomes

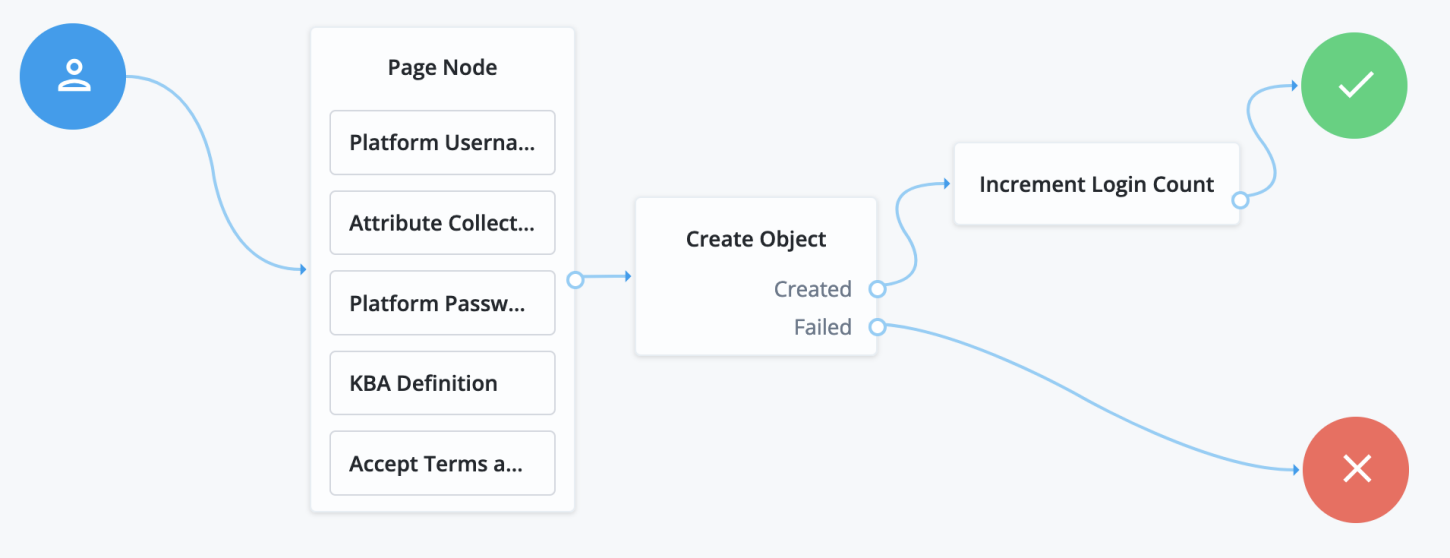
Single outcome path; on success, the transient state holds the questions and answers.

## Errors

This node logs a `Failed to retrieve kba configuration` warning message when it can't read the configuration.

## Example

The following registration journey prompts for questions and answers when creating an account:




- The Page node collects registration information:
  - The Platform Username node prompts for and collects a username for the new account.
  - The Attribute Collector node prompts for a given name, a surname, an email address, and profile preferences.
  - The Platform Password node prompts for and collects a password.
  - The **KBA Definition** node collects questions and answers.
  - The Accept Terms and Conditions node prompts the user to accept the active terms and conditions.
- The Create Object node stores the collected information in the new account object.
- The Increment Login Count node updates the number of successful authentications.

## KBA Verification node

Presents KBA questions to the user, collects answers to those questions, and verifies the input against the user's stored answers.

Use this node for additional authentication when resetting a forgotten password or username.

To set the number of KBA questions, edit **Configure > Security Questions > Questions > Number** in the IDM admin UI.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Properties

| Property | Usage |
|---|---|
| KBA Attribute | The managed object attribute in which KBA questions and answers are stored. |
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM). |

## Pass-through Authentication node

Authenticates an identity through a connector to a third-party service.

This lets you migrate user profiles without forcing users to reset their passwords, or retain a third-party service indefinitely as the canonical store for authentication credentials.

Before you use the node:

- Configure the connector to the third-party service.

  Find more information in the *synchronization* documentation for:

  - PingIDM
  - Advanced Identity Cloud

- If you plan to collect credentials in the identity repository for users, synchronize accounts from the third-party service.

  Find more information in the *synchronization* documentation for:

  - IDM
  - Advanced Identity Cloud

Use this node after collecting the authentication credentials.

For standalone AM deployments, implement a Username Collector node and a Password Collector node earlier in the journey.

For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node earlier in the journey.

Pass the credentials to this node to authenticate the identity against the service.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Connectors that support pass-through authentication

The following connectors support pass-through authentication using the `AuthenticateOp` interface by default:

- LDAP connector
- CSV file connector
- Database Table connector
- Microsoft Graph API Java connector
- Scripted SQL connector

> **Note**
>
> All Scripted Groovy-based connectors are capable of pass-through authentication if the `AuthenticateScript.groovy` script is implemented, but the only default implementation is the ScriptedSQL connector. Learn more in Authenticate script and Authenticate operation.

### Outcomes

- `Authenticated`
- `Missing Input`
- `Failed`

### Properties

| Property | Usage |
|---|---|
| System Endpoint | Required. Name of the connector to the third-party service that performs authentication. |
| Object Type | The OpenICF object type for the object being authenticated. Default: `account` |

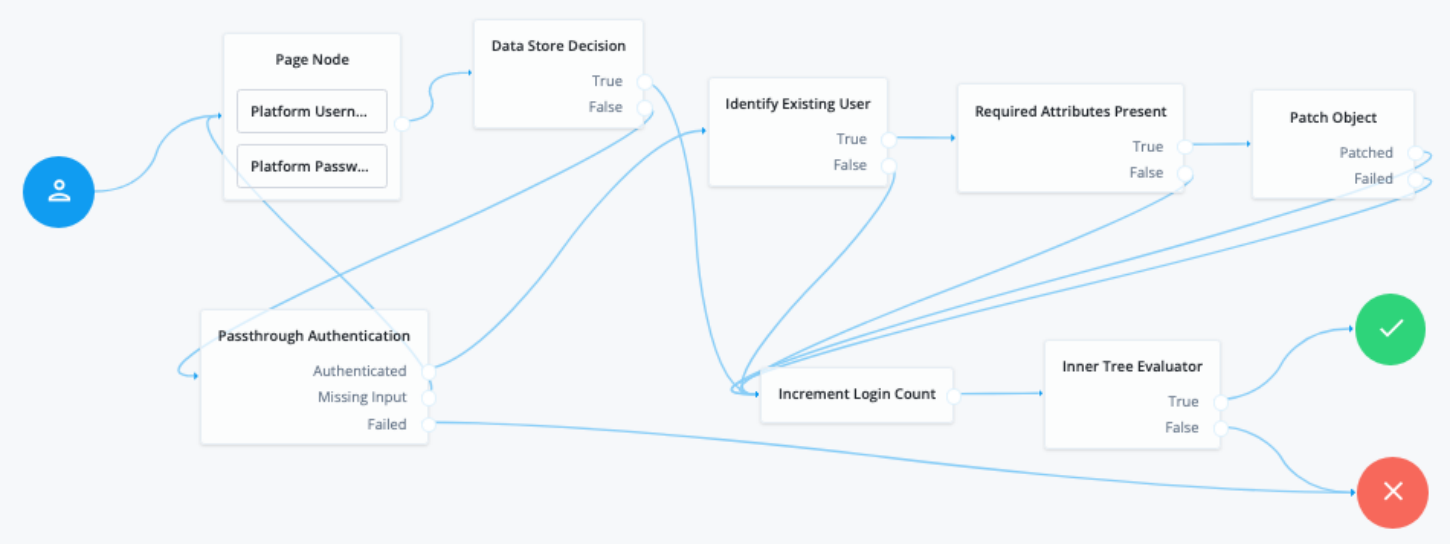
| Property | Usage |
| --- | --- |
| Identity Attribute | The username attribute for authentication.<br>Default: `userName` |
| Password Attribute | The password attribute for authentication.<br>Default: `password` |

## Example

The following example requires a Ping Identity Platform deployment.

Before trying this example, synchronize accounts from the third-party service. The example shows a login flow that tries pass-through authentication when local authentication fails, and stores the user password when authentication with the third-party service succeeds.

In this example, the user enters their credentials with the Platform Username node and Platform Password node. The Data Store Decision node authenticates against the platform directory service. On failure, authentication passes through to the third-party service. If authentication with the third-party service is successful, the Identify Existing User node and Required Attributes Present node check for a valid user profile. The Patch Object node updates the user's profile with the successful password:




*List of node connections*

| Source node | Outcome path | Target node |
| --- | --- | --- |
| Page Node containing:<br>• Platform Username<br>• Platform Password | → | Data Store Decision |
| Data Store Decision | True | Increment Login Count |

| Source node | Outcome path | Target node |
|---|---|---|
| | False | Pass-through Authentication |
| Pass-through Authentication | Authenticated | Identify Existing User |
| | Missing Input | Page Node |
| | Failed | Failure |
| Identify Existing User | True | Required Attributes Present |
| | False | Increment Login Count |
| Required Attributes Present | True | Patch Object |
| | False | Increment Login Count |
| Patch Object | Patched | Increment Login Count |
| | Failed | Increment Login Count |
| Increment Login Count | → | Inner Tree Evaluator |
| Inner Tree Evaluator | True | Success |
| | False | Failure |

## Patch Object node

The **Patch Object** node updates the attributes of an existing managed identity object.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

This node reads the **Identity Attribute** and the managed object fields to patch from the shared node state. If it can't read the **Identity Attribute**, it reads the `userName` from the shared node state.

## Dependencies

This node depends on the underlying identity service (PingIDM) to find and patch the managed object.

## Configuration

| Property | Usage |
|---|---|
| **Patch as Object** | If enabled, update the object as its subject—for example, update a managed user object as the user; otherwise, update the object as the client application.<br>Enable this property to patch fields of the current, authenticated user's account the client application can't update.<br>Default: false |
| **Ignored Fields** | Omit the specified shared state fields from the patch.<br>If no fields are specified, the node attempts to update all the shared state fields as part of the patch.<br>Default: none |
| **Identity Resource** | The type of managed identity resource object this node patches.<br>This must match the identity resource type for the current flow.<br>**Tip**<br>To check for the available managed identity resource types, go to the IDM admin UI, and open the **Manage** drop-down list in the upper right corner of the screen.<br>Identity managed object types are preceded by the icon.<br>Default: `managed/user` |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

## Outputs

None.

## Outcomes

### `Patched`

The node updated the managed object.

### `Failed`

Any other case.

### Errors

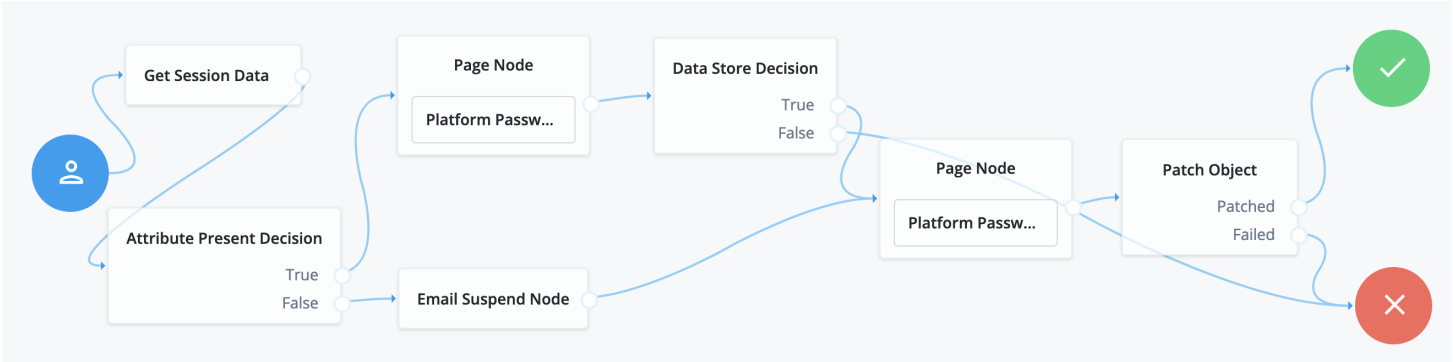
This node logs the following warning messages when an update fails:

- `Failed to create object`
- `Failed to patch object`

Review the logs for additional messages describing the problem.

### Example

This journey uses the **Patch Object** node to update a user's password:




The user has already authenticated before beginning this journey:

- The Get Session Data node stores the `userName` from the session.
- The Attribute Present Decision node checks whether the user object has a password attribute.
- If so, the first Page node with the Platform Password node prompts the user for the current password.
- Otherwise, the Email Suspend node sends an email to the user and suspends the flow until the user follows the link in the message.
- The Data Store Decision node confirms the username-password credentials.
- The second Page node with the Platform Password node prompts the user for the new password.
- The **Patch Object** node updates the user object with the new password.

## PingOne Create User node

The **PingOne Create User** node can create new users in the PingOne platform.

You can configure the node to create a user including their profile data or to create an anonymized user.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads the `username` field from the shared node state to access the user's identity profile.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

You should also verify the user's identity by using a Data Store Decision node (PingAM) or Identity Store Decision node (Advanced Identity Cloud).

## Dependencies

This node requires a **PingOne Worker Service** configuration so that it can authenticate to your PingOne instance.

Find more information on the properties used by the service in the *PingOne Worker service* documentation for:

- PingAM
- Advanced Identity Cloud

## Configuration

| Property | Usage |
|---|---|
| **PingOne Worker Service ID** | The ID of the PingOne worker service for connecting to PingOne. |
| **Population ID** | The ID of the population in PingOne to check for users or provision new ones.<br>If not specified, the node uses the environment's default population ID. |
| **Anonymized user** | When enabled, the node creates a user in PingOne with only a unique identifier and a language attribute.<br>It does not add any other profile attributes, helping prevent any personally identifiable information (PII) from being shared. |

| Property | Usage |
| --- | --- |
| **AM Identity Attribute** | The attribute from the user's AM profile that the node uses as the username for the account created in PingOne.<br><br>**Tip**<br>When creating anonymized users, choose a profile attribute that does not contain PII.<br><br>Default: `uid` |
| **Capture failure** | Capture the details in shared state if a failure occurs.<br>The node stores the details in a variable named `pingOneCreateUserFailureReason`.<br>Default: `False`<br>Example:<br><br>`{`<br>`  "code": "MISSING_ATTRIBUTE_FROM_PROFILE",`<br>`  "message": "Could not get attribute from user profile.",`<br>`  "exception": "",`<br>`}` |

## Outputs

The node is non-interactive and does not send a callback to the client.

If the node was able to create a new user in PingOne it stores the PingOne user identifier in a state variable named `pingOneUserId`. For example `a648aaac-ch15-b357-457b-8d2e714180ff`.

If you select **Capture failure**, the node stores any error response in a shared state variable named `pingOneCreateUserFailureReason`.

## Outcomes

`True`

The node created an account in PingOne.

`False`

The node did not create an account in PingOne.

The journey also uses this outcome if any error occurs. Enable **Capture Failure** to store the details in node state.

## Example

The following example journey leverages PingOne Verify to perform user identity verification.

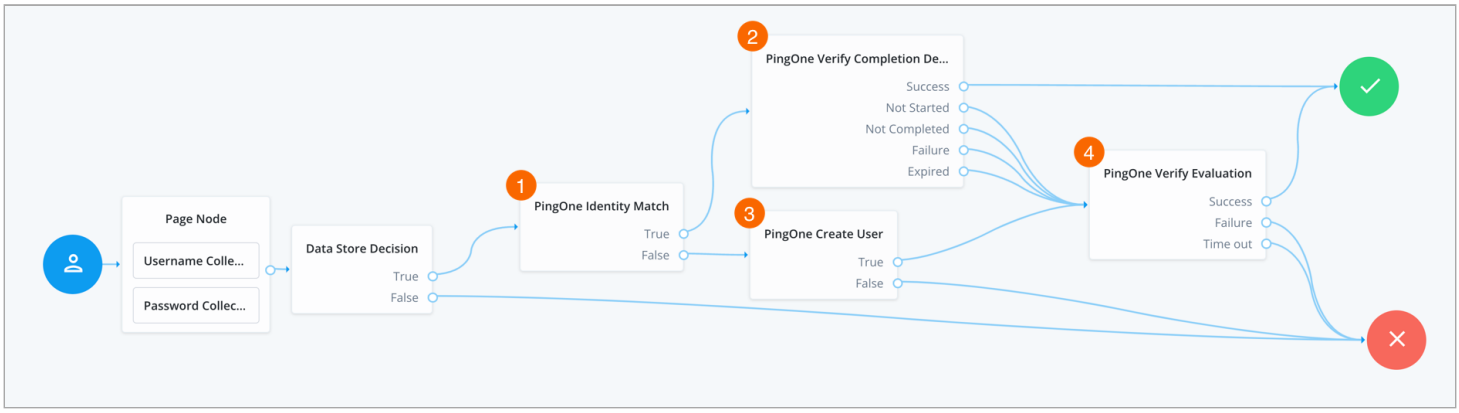



*Figure 1. Example PingOne Verify journey*

- The user enters their credentials, which the Data Store Decision node then verifies against the identity store.
- 1 The PingOne Identity Match node checks PingOne for a matching user.
- 2 If a user is found, the PingOne Verify Completion Decision node checks the user's most recent verification transaction to determine the status:

### *Success*

The user successfully completed the most recent PingOne Verify transaction, so continue directly to the **Success** node, completing the authentication journey.

### *Not Completed*

The user has an existing PingOne Verify transaction in progress, so continue the journey to resume the existing verification transaction.

The node adds the user's existing transaction ID to the shared node state in a variable named `pingOneVerifyTransactionId`.

### *Not Started / Failure / Expired*

The user either does not have an existing transaction (`Not Started`), or did not successfully complete the most recent PingOne Verify transaction, or it expired, so continue the journey to start a new verification transaction.

- 3 If a user is not found, the PingOne Create User node creates a new user in PingOne.
- 4 The PingOne Verify Evaluation node starts a new PingOne Verify evaluation, or continues an existing one if `pingOneVerifyTransactionId` is present in the shared node state, and either completes or fails the journey based on the result.

## PingOne Delete User node

The **PingOne Delete User** node can delete users from the PingOne platform.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node requires the `pingOneUserId` variable from node state. The variable must contain the identifier of the user to delete from PingOne. For example, `a648aaac-ch15-b357-457b-8d2e714180ff`.

Use either of the following nodes to populate the `pingOneUserId` variable in shared state:

- PingOne Identity Match node
- PingOne Create User node

## Dependencies

This node requires a **PingOne Worker Service** configuration so that it can authenticate to your PingOne instance.

You can find information on this service in the corresponding documentation for:

- PingAM
- Advanced Identity Cloud

## Configuration

| Property | Usage |
|---|---|
| **PingOne Worker Service ID** | The ID of the PingOne worker service for connecting to PingOne. |
| **Capture failure** | Capture the details in shared state if a failure occurs.<br>The node stores the details in a variable named `pingOneDeleteUserFailureReason`.<br>Default: `False`<br>Example:<br>`{ "code": "MISSING_PINGONE_USER_ID", "message": "Expected PingOne User ID to be set in sharedState or user's profile.", "exception": "", }` |

### Outputs

The node is non-interactive and does not send a callback to the client.

If you select **Capture failure**, the node stores any error response in a shared state variable named `pingOneDeleteUserFailureReason`.

### Outcomes

`True`

The node deleted the account from PingOne.

`False`

The node did not delete the account from PingOne.

The journey also uses this outcome if any error occurs. Enable **Capture Failure** to store the details in node state.

## PingOne Identity Match node

The **PingOne Identity Match** node checks that users that exist in the ForgeRock platform also exist in the PingOne platform.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads the `username` field from the shared node state to access the user's identity profile.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

You should also verify the user's identity by using a Data Store Decision node (PingAM) or Identity Store Decision node (Advanced Identity Cloud).

### Dependencies

This node requires a **PingOne Worker Service** configuration so that it can authenticate to your PingOne instance.

Find information on the configuration properties in the *PingOne Worker service* documentation for:

- PingAM
- Advanced Identity Cloud

## Configuration

| Property | Usage |
|---|---|
| **PingOne Worker Service ID** | The ID of the PingOne worker service for connecting to PingOne. |
| **Population ID** | The ID of the population in PingOne to check for users or provision new ones.<br>If not specified, the node uses the environment's default population ID. |
| **AM Identity Attribute** | The attribute from the user's ForgeRock profile that the node uses to match their account in PingOne.<br>Default: `uid` |
| **Ping Identity Attribute** | The attribute from the user's PingOne profile that the node uses to search for a matching account.<br>If there are multiple entries with the same attribute value in the PingOne directory server, ensure that this property is specific enough to retrieve only one entry.<br>Default: `username` |
| Capture failure | Capture the details in shared state if a failure occurs.<br>The node stores the details in a variable named `pingOneIdentityMatchFailureReason`.<br>Default: `False`<br>Example:<br>`{ "code": "ACCESS_TOKEN", "message": "Unable to get access token for PingOne Worker.", "exception": "", }` |

## Outputs

The node is non-interactive and does not send a callback to the client.

If the node was able to find a unique match in PingOne it stores the PingOne user identifier in a state variable named `pingOneUserId`. For example `a648aaac-ch15-b357-457b-8d2e714180ff`.

If you select **Capture failure**, the node stores any error response in a shared state variable named `pingOneIdentityMatchFailureReason`.

## Outcomes

`True`

The node found a unique matching account in PingOne.

`False`

The node did not find a unique match in PingOne.

## Example

The following example journey leverages PingOne Verify to perform user identity verification.

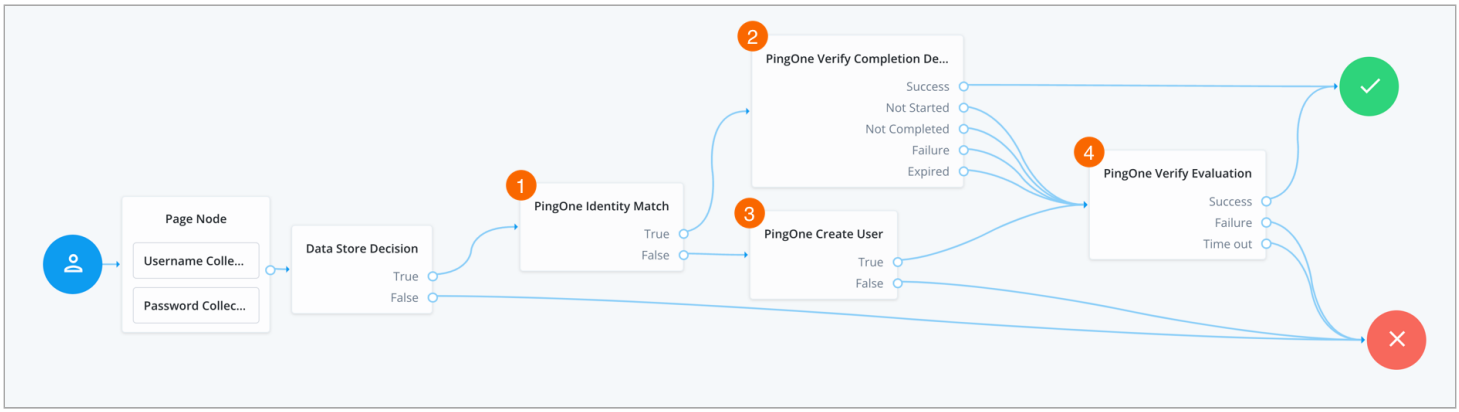



*Figure 1. Example PingOne Verify journey*

- The user enters their credentials, which the Data Store Decision node then verifies against the identity store.
- 1 The PingOne Identity Match node checks PingOne for a matching user.
- 2 If a user is found, the PingOne Verify Completion Decision node checks the user's most recent verification transaction to determine the status:

### *Success*

The user successfully completed the most recent PingOne Verify transaction, so continue directly to the **Success** node, completing the authentication journey.

### *Not Completed*

The user has an existing PingOne Verify transaction in progress, so continue the journey to resume the existing verification transaction.

The node adds the user's existing transaction ID to the shared node state in a variable named `pingOneVerifyTransactionId`.

### *Not Started / Failure / Expired*

The user either does not have an existing transaction (`Not Started`), or did not successfully complete the most recent PingOne Verify transaction, or it expired, so continue the journey to start a new verification transaction.

- 3 If a user is not found, the PingOne Create User node creates a new user in PingOne.
- 4 The PingOne Verify Evaluation node starts a new PingOne Verify evaluation, or continues an existing one if `pingOneVerifyTransactionId` is present in the shared node state, and either completes or fails the journey based on the result.

# PingOne Verify Completion Decision node

The **PingOne Verify Evaluation** node determines the completion status of the most recent identity verification transaction for a PingOne user.

The node checks the previous identity verification transaction for the user and returns an outcome based on the verification status.

You can use this node to prevent users from repeatedly having to verify their identity by checking their most recent verification transaction. You can also determine if a transaction is already in progress, to prevent multiple ongoing transactions.

For further customization of behavior, use a **PingOne Verify Completion Decision** script to evaluate the verification transactions started for the specified PingOne user, and the associated metadata. The script can then add additional context to the journey, or perform additional business logic, dependent on the verification metadata.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node requires that the user has an account in the PingOne environment. It requires that the journey stores the PingOne user ID for the account in the shared state variable named `pingOneUserId`.

Use a PingOne Identity Match node to populate the shared state with the user's PingOne ID.

The shared state data must also include all required **Script Inputs** properties.

You can restrict available inputs using the **Script Inputs** field when configuring the node.

## Dependencies

This node requires a **PingOne Worker Service** configuration so that it can connect to your PingOne instance and send it the necessary data to perform PingOne Verify evaluations as part of the journey.

Find information on the configuration properties in the *PingOne Worker service* documentation for:

- PingAM
- Advanced Identity Cloud

## Configuration

| Property | Usage |
| --- | --- |
| **PingOne Worker service ID** | The ID of the PingOne worker service for connecting to PingOne. |

| | |
|---|---|
| **Use a script to process Verify transactions** | When enabled, use a **PingOne Verify Completion Decision** script to process verification transaction data relating to the user. |

```
{
    "_links": {
        "self": {
            "href": "https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-
a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/verifyTransactions"
        }
    },
    "_embedded": {
        "verifyTransactions": [
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/029ea878-2618-4067-b7e3-922591e6b55f"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "029ea878-2618-4067-b7e3-922591e6b55f",
                "transactionStatus": {
                    "status": "APPROVED_NO_REQUEST"
                },
                "createdAt": "2022-02-17T20:32:22.052Z",
                "updatedAt": "2022-02-17T20:32:58.711Z",
                "expiresAt": "2022-02-17T21:02:58.711Z"
            },
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/cca479e7-d130-4e3c-b888-74ba1920f59a"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "cca479e7-d130-4e3c-b888-74ba1920f59a",
                "transactionStatus": {
                    "status": "REQUESTED"
                },
                "qrUrl": "https://api.pingone.com/v1/idValidations/shortcode/
086645084110/qr",
                "verificationCode": "086645084110",
                "createdAt": "2022-02-17T20:23:58.662Z",
                "updatedAt": "2022-02-17T20:23:58.662Z",
                "expiresAt": "2022-02-17T20:53:58.662Z"
            },
```

```
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/52c9bf9a-0687-4e01-85d1-9caa9bb387ee"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "52c9bf9a-0687-4e01-85d1-9caa9bb387ee",
                "transactionStatus": {
                    "status": "REQUESTED"
                },
                "qrUrl": "https://api.pingone.com/v1/idValidations/shortcode/
008299320746/qr",
                "verificationCode": "008299320746",
                "createdAt": "2022-02-17T20:23:08.887Z",
                "updatedAt": "2022-02-17T20:23:08.887Z",
                "expiresAt": "2022-02-17T20:53:08.887Z"
            },
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/dd5a6d4f-a999-4819-b107-85a0a10138c6"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "dd5a6d4f-a999-4819-b107-85a0a10138c6",
                "transactionStatus": {
                    "status": "REQUESTED"
                },
                "createdAt": "2021-12-09T13:45:34.882Z",
                "updatedAt": "2021-12-09T13:45:34.882Z",
                "expiresAt": "2022-12-09T14:15:34.882Z"
            },
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/bfc414cb-a1b4-46b8-b622-3d806a85002f"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
```

```
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "bfc414cb-a1b4-46b8-b622-3d806a85002f",
                "transactionStatus": {
                    "status": "REQUESTED"
                },
                "createdAt": "2021-12-08T21:34:52.424Z",
                "updatedAt": "2021-12-08T21:34:52.424Z",
                "expiresAt": "2022-12-08T22:04:52.424Z"
            },
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/b21db4c4-01c5-47b5-a2a9-3d8df21d189b"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "b21db4c4-01c5-47b5-a2a9-3d8df21d189b",
                "transactionStatus": {
                    "status": "APPROVED_NO_REQUEST"
                },
                "createdAt": "2021-12-07T21:33:22.088Z",
                "updatedAt": "2021-12-07T21:45:22.944Z",
                "expiresAt": "2022-12-07T22:15:22.944Z"
            },
            {
                "_links": {
                    "self": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/
verifyTransactions/e44ebfe2-6a76-4ffa-ac35-d71ee9365e57"
                    },
                    "environment": {
                        "href": "https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"
                    },
                    "user": {
                        "href": "https://api.pingone.com/v1/users/36ff33da-
cba7-4d46-bedc-8b242889d461"
                    }
                },
                "id": "e44ebfe2-6a76-4ffa-ac35-d71ee9365e57",
                "transactionStatus": {
                    "status": "APPROVED_NO_REQUEST"
                },
                "createdAt": "2021-12-07T19:55:16.630Z",
                "updatedAt": "2021-12-07T21:31:26.835Z",
                "expiresAt": "2022-12-07T22:01:26.835Z"
```

| Property | Usage |
| --- | --- |
| | {<br>"_links": {<br>"self": {<br>"href": "https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/verifyTransactions"<br>},<br>"environment": {<br>"href": "https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"<br>},<br>"user": {<br>"href": "https://api.pingone.com/v1/users/36ff33da-cba7-4d46-bedc-8b242889d461"<br>}<br>},<br>"_embedded": {<br>"verifyTransactions": [<br>{<br>"_links": {<br>"self": {<br>"href": "https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/36ff33da-cba7-4d46-bedc-8b242889d461/verifyTransactions/fc695b11-93a4-48bb-9ec3-ff5738e3818c"<br>},<br>"environment": {<br>"href": "https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"<br>},<br>"user": {<br>"href": "https://api.pingone.com/v1/users/36ff33da-cba7-4d46-bedc-8b242889d461"<br>}<br>},<br>"id": "fc695b11-93a4-48bb-9ec3-ff5738e3818c",<br>"transactionStatus": {<br>"status": "REQUESTED"<br>},<br>"createdAt": "2021-12-07T18:36:48.156Z",<br>"updatedAt": "2021-12-07T18:36:48.156Z",<br>"expiresAt": "2021-12-07T19:06:48.153Z"<br>}<br>]<br>},<br>"size": 8<br>}<br>For more information on verification transaction data, refer to Read All Verify Transactions. |
| **Manage Verify transactions script** | The name of the script to process the JSON containing verification transactions relating to the user.<br>The script must be of type **PingOne Verify Completion Decision**.<br>Refer to Example scripts. |
| **Script Inputs** | Optionally, list the shared state data properties required by the script.<br>Declare each required property, enter `null` for no properties, or set to `*` for access to all shared and transient state data.<br>Default: `*`. |
| **Capture failure** | Capture the details in shared state if a failure occurs.<br>The node stores the details in a variable named `pingOneVerifyCompletionFailureReason`.<br>Default: `False`<br>Example:<br>{<br>"code": "VERIFY_COMPLETION_SCRIPT_ERROR",<br>"message": "Error evaluating the verify completion decision script.",<br>"exception": "",<br>} |

## Outputs

The node is non-interactive and doesn't send a callback to the client.

If the node discovers that the user's most recent Verify transaction isn't yet complete, it adds the ID of the transaction to the shared state, in a variable named `pingOneVerifyTransactionId`. The PingOne Verify Evaluation node can use this value to continue the existing Verify evaluation, rather than start a brand new one.

If you select **Capture failure**, the node stores any error response in a shared state variable named `pingOneVerifyCompletionFailureReason`.

If you enable the **Use a script to process Verify transactions** property, the script you specify can add values to the shared state, as required.

## Outcomes

### `Success`

The user successfully completed their most recent PingOne Verify evaluation.

### `Failure`

The user did not successfully complete their most recent PingOne Verify evaluation, or an error occurred.

### `Expired`

The user's most recent PingOne Verify evaluation timed out. Usually this happens when a user starts a verification transaction, but does not complete it within the time limit.

### `Not Started`

The user's most recent PingOne Verify evaluation has not yet started.

### `Not Completed`

The user's most recent PingOne Verify evaluation has started, but not yet completed.

## Examples

### Example journey

The following example journey leverages PingOne Verify to perform user identity verification.

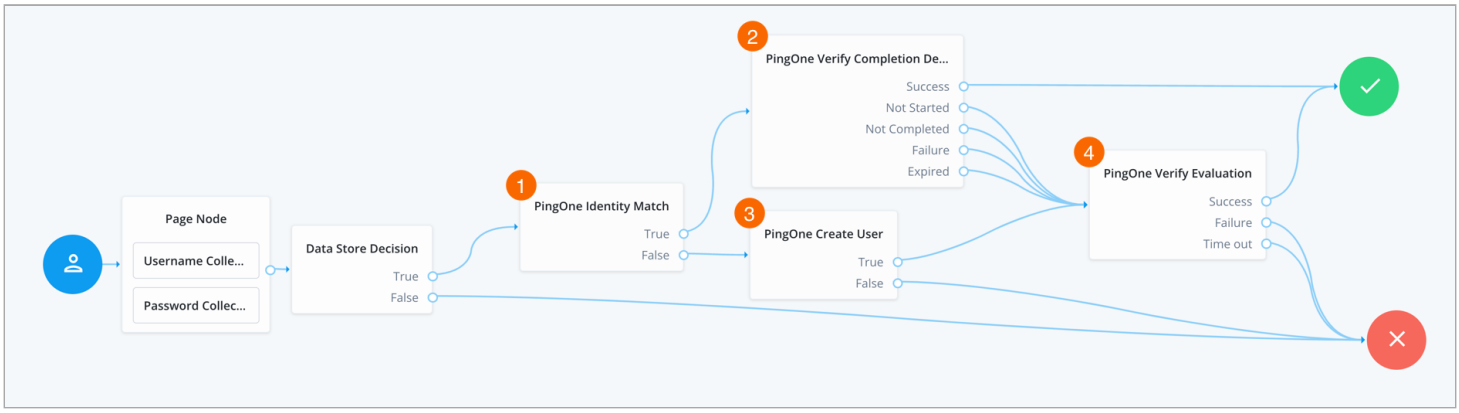



*Figure 1. Example PingOne Verify journey*

- The user enters their credentials, which the Data Store Decision node then verifies against the identity store.
- 1 The PingOne Identity Match node checks PingOne for a matching user.
- 2 If a user is found, the PingOne Verify Completion Decision node checks the user's most recent verification transaction to determine the status:

### *Success*

The user successfully completed the most recent PingOne Verify transaction, so continue directly to the **Success** node, completing the authentication journey.

### *Not Completed*

The user has an existing PingOne Verify transaction in progress, so continue the journey to resume the existing verification transaction.

The node adds the user's existing transaction ID to the shared node state in a variable named `pingOneVerifyTransactionId`.

### *Not Started / Failure / Expired*

The user either does not have an existing transaction (`Not Started`), or did not successfully complete the most recent PingOne Verify transaction, or it expired, so continue the journey to start a new verification transaction.

- 3 If a user is not found, the PingOne Create User node creates a new user in PingOne.
- 4 The PingOne Verify Evaluation node starts a new PingOne Verify evaluation, or continues an existing one if `pingOneVerifyTransactionId` is present in the shared node state, and either completes or fails the journey based on the result.

## Example scripts

These example scripts demonstrate some use cases for the **PingOne Verify Completion Decision** node. They use the `verifyTransactionsHelper` next-generation script binding, which gives access to the data used during a PingOne Verify transaction.

Find more information about this binding in the scripting documentation for:

- PingAM

- Advanced Identity Cloud

### *Example 1*

Check the status of the user's most recent PingOne Verify transaction and that the ID used was a driving license.

```
/** ***************************************************************
 *
 * The following script checks the status of the user's most recent
 * identity check, and ensures the ID used was a driver's license.
 *
 * Global node variables accessible within this scope:
 * 1. `nodeState` provides access to auth tree state attributes
 * 2. `verifyTransactionsHelper` provides access to verify transactions
 * 3. `outcome` variable maps to auth tree node outcomes; values are
 *    'successOutcome', 'notStartedOutcome', 'notCompletedOutcome',
 *    'expiredOutcome', or 'failureOutcome'.
 * ***************************************************************/

// Retrieve the user's latest verified transaction.
var result = verifyTransactionsHelper.getLastVerifyTransaction();
if (result != null) {
    var lastTransaction = JSON.parse(result);
    if (lastTransaction.hasOwnProperty("transactionStatus")) {
        // Get the status of the transaction.
        var status = lastTransaction.transactionStatus.status;
        // Determine the document type and verify if it is a Driver's License.
        var verifiedDocuments = lastTransaction.verifiedDocuments;
        if (status == "SUCCESS" && verifiedDocuments.includes("driver_license")) {
            outcome = "successOutcome";
        } else {
            outcome = "failureOutcome";
        }
    } else {
        outcome = "notStartedOutcome";
    }
}
```

### *Example 2*

Check the ID used in the user's most recent PingOne Verify transaction and that their age is 18 or over.

> **Note**
>
> This data is available for a short timeframe after the verification.
> Usually 30 minutes after PingOne Verify reaches its verification decision.

```
/** ********************************************************************
 *
 * The following script checks the ID used in the user's most recent
 * identity check, and that their age is 18 or over.
 *
 * Note:
 * This data is available for a short timeframe after the verification.
 * Usually 30 minutes after a verification decision is reached.
 *
 * Global node variables accessible within this scope:
 * 1. `nodeState` provides access to auth tree state attributes
 * 2. `verifyTransactionsHelper` provides access to verify transactions
 * 3. `outcome` variable maps to auth tree node outcomes; values are
 *    'successOutcome', 'notStartedOutcome', 'notCompletedOutcome',
 *    'expiredOutcome', or 'failureOutcome'.
 * ********************************************************************/

// Retrieve the user's latest verified transaction.
var result = verifyTransactionsHelper.getLastVerifyTransaction();
if (result != null) {
    var lastTransaction = JSON.parse(result);
    if (lastTransaction.hasOwnProperty("id")) {
        // Obtain the ID of the last transaction.
        var lastTransactionId = lastTransaction.id;
        // Get the verified data for the Government ID provided by the user.
        var verifiedDataResult = verifyTransactionsHelper.getVerifiedDataByType(lastTransactionId,
"GOVERNMENT_ID");
        // Determine the age of the user.
        if (verifiedDataResult != null) {
            var verifiedData = JSON.parse(verifiedDataResult);
            var dateString = verifiedData._embedded.verifiedData[0].data.birthDate;
            if (dateString != null && dateString.trim() != "") {
                var birthDate = new Date(dateString + "T00:00:01");
                var currentDate = new Date().getTime();
                var difference = currentDate - birthDate;
                var currentAge = Math.floor(difference / (1000 * 60 * 60 * 24 * 365.25));
                if (currentAge >= 18) {
                    outcome = "successOutcome";
                } else {
                    outcome = "failureOutcome";
                }
            } else {
                outcome = "failureOutcome";
            }
        } else {
            outcome = "notCompletedOutcome";
        }
    } else {
        outcome = "notStartedOutcome";
    }
} else {
    outcome = "notStartedOutcome";
}
```

### *Example 3*

Check the user's most recent PingOne Verify transaction to obtain the expiration date for the Government ID provided.

The node stores this information in a variable named `governmentIdExpireDate` in the shared state, but you could also store it in a user attribute to determine when to perform the next identity verification.

> **Note**
>
> This data is available for a short timeframe after the verification.
> Usually 30 minutes after PingOne Verify reaches its verification decision.

```
/** ********************************************************************
 *
 * The following script checks the user's most recent Verify transaction
 * to obtain the expiration date for the Government ID provided.
 *
 * This information is stored in a variable named `governmentIdExpireDate`
 * in the shared state, but could also be stored as a user attribute
 * to determine when to perform the next identity verification.
 *
 * Note:
 * This data is available for a short timeframe after the verification.
 * Usually 30 minutes after a verification decision is reached.
 *
 * Global node variables accessible within this scope:
 * 1. `nodeState` provides access to auth tree state attributes
 * 2. `verifyTransactionsHelper` provides access to verify transactions
 * 3. `outcome` variable maps to auth tree node outcomes; values are
 *    'successOutcome', 'notStartedOutcome', 'notCompletedOutcome',
 *    'expiredOutcome', or 'failureOutcome'.
 * *********************************************************************/

// Retrieve the user's latest verified transaction.
var result = verifyTransactionsHelper.getLastVerifyTransaction();
if (result != null) {
    var lastTransaction = JSON.parse(result);
    if (lastTransaction.hasOwnProperty("id")) {
        // Obtain the ID of the last transaction.
        var lastTransactionId = lastTransaction.id;
        // Get the verified data for the Government ID provided by the user.
        var verifiedDataResult = verifyTransactionsHelper.getVerifiedDataByType(lastTransactionId,
"GOVERNMENT_ID");
        // Get the expire date and set on shared state.
        if (verifiedDataResult != null) {
            var verifiedData = JSON.parse(verifiedDataResult);
            var expireDate = verifiedData._embedded.verifiedData[0].data.expirationDate;
            if (expireDate != null && expireDate.trim() != "") {
                nodeState.putShared("governmentIdExpireDate", expireDate);
                outcome = "successOutcome";
            } else {
                outcome = "failureOutcome";
            }
        } else {
            outcome = "notCompletedOutcome";
        }
    } else {
        outcome = "notStartedOutcome";
    }
} else {
    outcome = "notStartedOutcome";
}
```

### *Example 4*

Check that the user has at least one successful identity verification in the past 365 days.

```
/** ********************************************************************
 *
 * The following script checks that the user has at least one successful
 * identity verification in the past 365 days.
 *
 * Global node variables accessible within this scope:
 * 1. `nodeState` provides access to auth tree state attributes
 * 2. `verifyTransactionsHelper` provides access to verify transactions
 * 3. `outcome` variable maps to auth tree node outcomes; values are
 *    'successOutcome', 'notStartedOutcome', 'notCompletedOutcome',
 *    'expiredOutcome', or 'failureOutcome'.
 * ********************************************************************/

// Retrieve all transactions for the user
var result = verifyTransactionsHelper.getAllVerifyTransactions();
if (result != null) {
    var allTransactions = JSON.parse(result);
    if (allTransactions != null) {
        // Loop through the transactions to find a successful verification within the last 12 months.
        var verifyTransactions = allTransactions._embedded.verifyTransactions;
        var found = false;
        for (var i = 0; i < verifyTransactions.length; i++) {
            var transaction = verifyTransactions[i];
            // Get the status of the transaction.
            var status = transaction.transactionStatus.status;
            // If status is success, verify if it is still valid
            if (status == "SUCCESS") {
                found = true;
                // Calculate the number of days
                var dateString = transaction.createdAt;
                var createdDate = new Date(dateString);
                var currentDate = new Date().getTime();
                var difference = Math.abs(createdDate - currentDate);
                var numDaysBetween = difference / (1000 * 60 * 60 * 24);
                if (numDaysBetween <= 365) {
                    outcome = "successOutcome";
                    break;
                } else {
                    outcome = "expiredOutcome";
                }
            }
        }
        // No successful transaction found
        if (found == false) {
            outcome = "failureOutcome";
        }
    } else {
        outcome = "notStartedOutcome";
    }
} else {
    outcome = "notStartedOutcome";
}
```

### *Example 5*

Check the user's most recent PingOne Verify transaction to review the biographic match results.

The script uses the **Failure** outcome if the match confidence for any attribute is anything other than `HIGH`.

```
/** ********************************************************************
 *
 * The following script checks the user's last identity verification to
 * review the "Biographic Match" results.
 *
 * The script uses the failure outcome if the match confidence for any
 * attribute is below `HIGH`.
 *
 * Global node variables accessible within this scope:
 * 1. `nodeState` provides access to auth tree state attributes
 * 2. `verifyTransactionsHelper` provides access to verify transactions
 * 3. `outcome` variable maps to auth tree node outcomes; values are
 *    'successOutcome', 'notStartedOutcome', 'notCompletedOutcome',
 *    'expiredOutcome', or 'failureOutcome'.
 * *********************************************************************/

// Retrieve the user's latest verified transaction.
var result = verifyTransactionsHelper.getLastVerifyTransaction();
if (result != null) {
    var lastTransaction = JSON.parse(result);
    if (lastTransaction.hasOwnProperty("id")) {
        // Obtain the ID of the last transaction.
        var lastTransactionId = lastTransaction.id;
        // Get all the metadata.
        var allMetadataResult = verifyTransactionsHelper.getAllMetadata(lastTransactionId);
        // Loop through the metadata to find the biographic match results.
        var biographicMatchResults;
        if (allMetadataResult != null) {
            var allMetadataJson = JSON.parse(allMetadataResult);
                        var allMetadata = allMetadataJson._embedded.metaData;
            for (var i = 0; i < allMetadata.length; i++) {
                var metadata = allMetadata[i];
                var type = metadata.type;
                var status = metadata.status;
                if (type == "BIOGRAPHIC_MATCH" && status == "SUCCESS") {
                    biographicMatchResults = metadata.data.biographic_match_results;
                    break;
                }
            }
        } else {
            outcome = "failureOutcome";
        }
        // Validate the biographic match results
        if (biographicMatchResults != null && biographicMatchResults.length > 0) {
            var success = true;
            for (var j = 0; j < biographicMatchResults.length; j++) {
                var match = biographicMatchResults[j].match;
                if (match != "HIGH") {
                                        success = false;
                                        break;
                }
            }
            if (success) {
                outcome = "successOutcome";
            } else {
                outcome = "failureOutcome";
            }
        } else {
            outcome = "failureOutcome";
        }
```

```
    } else {
        outcome = "notStartedOutcome";
    }
} else {
    outcome = "notStartedOutcome";
}
```

## PingOne Verify Evaluation node

The **PingOne Verify Evaluation** node leverages the PingOne Verify Service to initiate a new or continue an existing verification transaction.

It offers a range of delivery methods, such as a QR code, email, or SMS to start the identity verification process.

You can customize the verification types users can perform in the PingOne Verification Policy.

Learn more in Verify policies.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires that the user has an account in the PingOne environment. It requires that the journey stored the PingOne user ID for the account in a shared state variable named `pingOneUserId`.

Use a PingOne Identity Match node to populate the shared state with the user's PingOne ID.

If there's a transaction ID in the shared state variable named `pingOneVerifyTransactionId`, this node continues that evaluation, rather than starting a new one.

Use a PingOne Verify Completion Decision node to determine the status of any previous transactions and populate the shared state with an in-progress transaction ID.

### Dependencies

This node requires a **PingOne Worker Service** configuration so that it can connect to your PingOne instance and send it the necessary data to perform PingOne Verify evaluations as part of the journey.

Find information on the configuration properties in the *PingOne Worker service* documentation for:

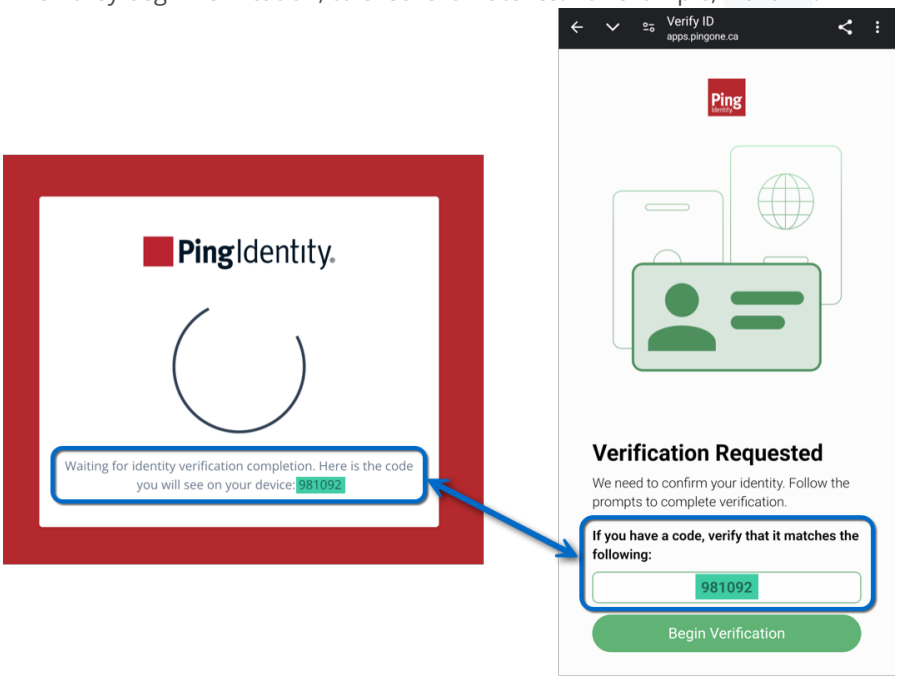
- PingAM

- Advanced Identity Cloud

## Configuration

| Property | Usage |
|---|---|
| **PingOne Worker Service ID** | The ID of the PingOne worker service for connecting to PingOne. |
| **Verify Policy ID** | The ID of the policy to use for the PingOne Verify evaluation.<br>If not specified, the node uses the environment's default Verify policy. |
| **Verify URL delivery method** | How the user will receive the URL they need to start a PingOne Verify evaluation.<br>Choose from:<br>**`QR Code`**<br>Display the URL as a QR code.<br>**`Email`**<br>Send an email containing the URL to the email address in the user's AM identity profile.<br>**`SMS`**<br>Send an SMS containing the URL to the phone number in the user's AM identity profile.<br>**`Redirect`**<br>Redirect the user to the PingOne Verify web app for identity verification.<br>On completion, redirect the user back to AM to continue the authentication journey.<br>Default: `QR Code` |
| **Allow user to choose the URL delivery method** | When enabled, the node prompts the user to choose the URL delivery method. |
| **Delivery method message** | Add the text per locale to display when prompting the user to choose their delivery method:<br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[1]<br>If the incoming HTTP request does not include the header or the preferred locales do not match any configured locales, the node uses the first text in the list.<br>3. In the **Value** field, enter the text to display to the user.<br>If you leave this blank, the node displays a localized version of the following text:<br>`Select the delivery method to start the identity verification process.`<br>To edit an entry, click its pencil icon (✎).<br>To remove an entry, click its delete icon (🗑). |

| Property | Usage |
| --- | --- |
| **QR code message** | Add the text per locale to display when you select QR code as the delivery method:<br><br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[1]<br>If the incoming HTTP request does not include the header or the preferred locales do not match any configured locales, the node uses the first text in the list.<br>3. In the **Value** field, enter the text to display to the user.<br>If you leave this blank, the node displays a localized version of the following text: `Scan the QR code to initiate the identity verification process.`<br><br>To edit an entry, click its pencil icon (✎).<br>To remove an entry, click its delete icon (🗑). |
| **Redirect message** | Add the text per locale to display when you select **Redirect** as the delivery method, and the node redirects the user back to AM to continue the journey:<br><br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[1]<br>If the incoming HTTP request does not include the header or the preferred locales do not match any configured locales, the node uses the first text in the list.<br>3. In the **Value** field, enter the text to display to the user.<br><br>To edit an entry, click its pencil icon (✎).<br>To remove an entry, click its delete icon (🗑). |

| Property | Usage |
| --- | --- |
| **Waiting message** | Add the text per locale to display while waiting for the user to respond to the Verify transaction, when using the SMS or Email delivery methods:<br><br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[(1)]<br>If the incoming HTTP request does not include the header or the preferred locales do not match any configured locales, the node uses the first text in the list.<br>3. In the **Value** field, enter the text to display to the user.<br>You can use the following variable in the `Value` field:<br>**`{{verificationCode}}`**<br>Replaced with a 6-digit code that the user can compare with the code displayed when they begin verification, to ensure it matches. For example, `981092`.<br><br>  <br><br>***Figure 1. Compare the code displayed on screen with the code presented during verification.***<br>If you leave this blank, the node displays a localized version of the following text:<br>`Waiting for identity verification completion. Here is the code you will see on your device: {{verificationCode}}`<br><br>To edit an entry, click its pencil icon (✎).<br>To remove an entry, click its delete icon (🗑). |

| Property | Usage |
| --- | --- |
| **Biographic Matching** | Require that the specified data obtained from the user's identity documents match the paired attribute in the user's profile.<br>To create a pairing:<br>1. Click **Add**.<br>2. In the **Key** field, enter the biographic matching requirement.<br>One of:<br>**`referenceSelfie`**<br>The photo the user took of themselves, in base64 encoded data form.<br>**`phone`**<br>The phone number obtained from the user's identification.<br>**`email`**<br>The email address obtained from the user's identification.<br>**`given_name`**<br>The first, or given name obtained from the user's identification. For example, `Babs`.<br>**`family_name`**<br>The last, surname, or family name obtained from the user's identification. For example, `Jensen`.<br>**`name`**<br>The full name obtained from the user's identification. For example, `Babs Jensen`.<br>**`address`**<br>The address obtained from the user's identification, as a single string. For example, `123 Any Street, London, United Kingdom, CH15 1EE`.<br>**`birth_date`**<br>The date of birth obtained from the user's identification.<br>3. In the **Value** field, enter the attribute in the user's AM profile that should match. For example, you could pair the `family_name` biographic key to the `sn` profile attribute.<br>To edit an entry, click its pencil icon (✎).<br>To remove an entry, click its delete icon (🗑). |

| | |
|---|---|
| **Store Verification Metadata** | When enabled, store the verification metadata returned from PingOne Verify in shared state under a key named `pingOneVerifyMetadata`. |

```
{
    "_links":{
        "self":{
            "href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-
a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/verifyTransactions/
7668563d-0226-4ca5-8401-03f6dc5bcdc6/metaData"
        },
        "user":{
            "href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-
a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94"
        },
        "environment":{
            "href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-
a5d9-80ad5c98f9f6"
        },
        "verifyTransaction":{
            "href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-
a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/verifyTransactions/
7668563d-0226-4ca5-8401-03f6dc5bcdc6"
        }
    },
    "_embedded":{
        "metaData":[
            {
                "_links":{
                    "self":{
                        "href":"https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/
verifyTransactions/7668563d-0226-4ca5-8401-03f6dc5bcdc6/metaData/
4ebb9165-4e5c-4270-94e4-d50d7b17ecb4"
                    }
                },
                "id":"4ebb9165-4e5c-4270-94e4-d50d7b17ecb4",
                "provider":"IDRND",
                "type":"LIVENESS",
                "status":"SUCCESS",
                "data":{
                    "score":6.4909873,
                    "probability":0.99848527,
                    "quality":0.94462675
                },
                "retry":{
                    "attempt":2
                }
            },
            {
                "_links":{
                    "self":{
                        "href":"https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/
verifyTransactions/7668563d-0226-4ca5-8401-03f6dc5bcdc6/metaData/546d3a8e-
f606-4078-92f1-96a5c2d003e9"
                    }
                },
                "id":"546d3a8e-f606-4078-92f1-96a5c2d003e9",
                "provider":"AMAZON",
                "type":"FACIAL_COMPARISON",
                "status":"SUCCESS",
                "data":{
                    "similarity":99.37002,
```

```
                    "confidence":99.99767,
                    "quality":{
                        "brightness":36.77353,
                        "sharpness":20.92731
                    }
                }
            },
            {
                "_links":{
                    "self":{
                        "href":"https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/
verifyTransactions/7668563d-0226-4ca5-8401-03f6dc5bcdc6/metaData/96315a69-
fb46-4d28-9b0d-c79927e59df1"
                    }
                },
                "id":"96315a69-fb46-4d28-9b0d-c79927e59df1",
                "provider":"BIOGRAPHIC_MATCHER",
                "type":"BIOGRAPHIC_MATCH",
                "status":"SUCCESS",
                "data":{
                    "biographic_match_results":[
                        {
                            "identifier":"address",
                            "match":"NOT_APPLICABLE"
                        },
                        {
                            "identifier":"given_name",
                            "match":"NONE"
                        },
                        {
                            "identifier":"family_name",
                            "match":"HIGH"
                        },
                        {
                            "identifier":"birth_date",
                            "match":"HIGH"
                        }
                    ]
                }
            },
            {
                "_links":{
                    "self":{
                        "href":"https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/
verifyTransactions/7668563d-0226-4ca5-8401-03f6dc5bcdc6/metaData/
fba13756-8c24-49ff-9b42-ff1a3661d0ae"
                    }
                },
                "id":"fba13756-8c24-49ff-9b42-ff1a3661d0ae",
                "provider":"MITEK",
                "type":"DOCUMENT_AUTHENTICATION",
                "status":"SUCCESS",
                "data":{
                    "mitekVerifications":[
                        {
                            "name":"Document Ensemble Authenticator",
                            "judgement":"Authentic",
                            "verificationType":202,
                            "probability":753,
```

```
            "version":"3.47.0.7114",
            "documentId":"048f28f1-a7fe-42a5-9722-f10977606719"
        },
        {
            "name":"Black And White Copy",
            "judgement":"Authentic",
            "verificationType":102,
            "probability":717,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"Image Classification",
            "judgement":"Authentic",
            "verificationType":105,
            "probability":1000,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"Data Comparison",
            "judgement":"Authentic",
            "verificationType":700,
            "probability":1000,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"Ensemble Authenticator",
            "judgement":"Authentic",
            "verificationType":201,
            "probability":753,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"ID Document Blacklist",
            "judgement":"Authentic",
            "verificationType":101,
            "probability":1000,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"Generic Font",
            "judgement":"Authentic",
            "verificationType":104,
            "probability":926,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"MRZ Check Digit",
            "judgement":"Authentic",
            "verificationType":601,
            "probability":1000,
            "version":"3.47.0.7114",
            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
        },
        {
            "name":"MRZ Font Type Authentication",
```

```
                            "judgement":"Authentic",
                            "verificationType":600,
                            "probability":1000,
                            "version":"3.47.0.7114",
                            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
                        },
                        {
                            "name":"Image Processing",
                            "judgement":"Authentic",
                            "verificationType":710,
                            "probability":1000,
                            "version":"1.0",
                            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
                        },
                        {
                            "name":"Document Liveness",
                            "judgement":"Authentic",
                            "verificationType":108,
                            "probability":999,
                            "version":"1.0",
                            "documentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856"
                        }
                    ],
                    "frontImageDocumentId":"e290d74d-bf9c-4116-9fe7-9b6fb909c856",
                    "documentEvidenceId":"048f28f1-a7fe-42a5-9722-f10977606719",
                    "retry":{
                        "attempt":1
                    }
                }
            }
        ]
    },
    "previousAttempts":[
        {
            "_links":{
                "self":{
                    "href":"https://api.pingone.com/v1/environments/
abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/03df72b1-b80b-4449-8eef-ee8f85f48d94/
verifyTransactions/7668563d-0226-4ca5-8401-03f6dc5bcdc6/metaData/
06aebfbd-0053-4860-8b59-4f3cb7371dcb"
                }
            },
            "id":"06aebfbd-0053-4860-8b59-4f3cb7371dcb",
            "provider":"IDRND",
            "type":"LIVENESS",
            "status":"FAIL",
            "data":{
                "score":2.4509223,
                "probability":0.40062885,
                "quality":0.40874674
            },
            "retry":{
                "attempt":1
            }
        }
    ],
    "size":4
}
```

| Property | Usage |
| --- | --- |
| | **Note**<br>The key is empty if the node is unable to retrieve the verification metadata from PingOne.<br><br>Default: Disabled |

| Property | Usage |
| --- | --- |
| **Store Verified Data** | When enabled, store a list of the verified data submitted by the user in shared state under a key named `pingOneVerifyVerifiedData`.<br><br>{<br>"_links":{<br>"self":{<br>"href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/a27dec16-1e80-4f10-a261-2cac46a12b78/verifyTransactions/0e2ed48f-6c3a-46c4-bcb5-3a6bd791348b/verifiedData/34613a50-672c-428f-8db9-c67fe09fc4cc"<br>},<br>"environment":{<br>"href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6"<br>},<br>"user":{<br>"href":"https://api.pingone.com/v1/users/a27dec16-1e80-4f10-a261-2cac46a12b78"<br>},<br>"transaction":{<br>"href":"https://api.pingone.com/v1/environments/abfba8f6-49eb-49f5-a5d9-80ad5c98f9f6/users/a27dec16-1e80-4f10-a261-2cac46a12b78/verifyTransactions/0e2ed48f-6c3a-46c4-bcb5-3a6bd791348b"<br>}<br>},<br>"id":"84170421-62c6-49a5-b343-496bee93c206",<br>"type":"GOVERNMENT_ID",<br>"createdAt":"2022-02-23T15:51:01.603Z",<br>"data":{<br>"addressCity":"this city",<br>"addressState":"this state",<br>"addressZip":"11111",<br>"birthDate":"1970-01-01",<br>"country":"USA",<br>"expirationDate":"1970-01-01",<br>"firstName":"given",<br>"gender":"",<br>"idNumber":"11111",<br>"issueDate":"1970-01-01",<br>"issuingCountry":"",<br>"lastName":"surname",<br>"nationality":"",<br>"weight":""<br>},<br>"retry":{<br>"attempt":1<br>}<br>}<br><br>**Note**<br>The key is empty if the node is unable to retrieve the verified data from PingOne.<br><br>Default: Disabled |

| Property | Usage |
| --- | --- |
| **Capture failure** | Capture the details in shared state if a failure occurs.<br>The node stores the details in a variable named `pingOneVerifyEvaluationFailureReason`.<br>Learn more about the errors this node can return in Errors.<br>Default: `False` |

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

## Outputs

- If **Allow user to choose the URL delivery method** is selected, the node sends the following callbacks:

  **`TextOutputCallback`**

  Contains the **Delivery method message**.

  **`ConfirmationCallback`**

  Contains the options available to the client application.

- When using the `QR Code` URL delivery method, the node sends the following callbacks:

  **`TextOutputCallback`**

  Contains the **QR Code message**.

  **`ScriptTextOutputCallback`**

  Contains JavaScript script to run to display the QR code.

  **`HiddenValueCallback`**

  Contains the actual URL to start the verification. The client might display this to users on a mobile device that cannot scan a QR code, or to render their own QR code, for example.

  **`PollingWaitCallback`**

  Waits for the user to complete the verification, and the **Waiting message**.

- When using the `Email` or `SMS` URL delivery method, the node sends the following callbacks:

  **`PollingWaitCallback`**

  Waits for the user to complete the verification, and contains the **Waiting message**.

- When using the `Redirect` delivery method, the node sends the following callbacks:

  **`RedirectCallback`**

  Contains the URI to redirect the user to for identity verification, using the PingOne Verify web application.

- If you select **Store Verification Metadata**, the node outputs the verification metadata JSON in a state variable named `pingOneVerifyMetadata`.

  To learn more about verification metadata, refer to Read All Verification Metadata.

- If you select **Store Verified Data**, the node outputs the verified information gathered from the user's ID in a state variable named `pingOneVerifyVerifiedData`.

  To learn more about verified data, refer to Read One User Verified Data.

- If you select **Capture failure**, the node stores any error response in a shared state variable named `pingOneVerifyEvaluationFailureReason`.

## Outcomes

### Success

The user successfully completed the PingOne Verify evaluation.

### Failure

The user did not successfully complete the PingOne Verify evaluation, or an error occurred.

### Time Out

The node did not receive a response from the user performing the verification before the timeout specified in the **Verify Transaction Timeout** property.

## Errors

This node can output the errors in the following table.

**Tip**

Enable **Capture failure** to store errors in shared state in a variable named `pingOneVerifyEvaluationFailureReason`. The node stores errors in this format:

```
{
  "code": "{{Error code}}",
  "message": "{{Error message}}",
  "exception": "",
}
```

*PingOne Verify Evaluation node error codes and messages*

| Error code | Error message |
|---|---|
| `ACCESS_TOKEN` | Unable to get access token for PingOne Worker. |
| `IDENTITY_NOT_FOUND` | Could not find the identity with username in the realm. |

| Error code | Error message |
| --- | --- |
| `INVALID_ATTRIBUTE_CONFIGURATION` | Could not get the value for the configured user attribute. |
| `INVALID_BIOGRAPHIC_MATCHING` | Unexpected key value found in Biographic Matching. |
| `IDENTITY_NOT_FOUND` | Could not find the identity with username in the realm. |
| `IDENTITY_VERIFICATION_FAILED` | Identity verification failed. |
| `JSON_PROCESSING_ERROR` | Error processing JSON data. |
| `MISSING_ATTRIBUTE_FROM_PROFILE` | Could not get attribute from user profile. |
| `MISSING_PINGONE_USER_ID_FROM_SHARED_STATE` | Expected PingOne User ID to be set in sharedState. |
| `MISSING_PINGONE_VERIFY_TRANSACTION_ID` | Expected PingOne Verify Transaction ID to be set in sharedState. |
| `MISSING_USERNAME` | Could not get the username from the context. |
| `REDIRECT_FLOW_FAILED_CODE_MISMATCH` | Redirect flow failed. Code mismatch. |
| `REDIRECT_FLOW_FAILED_MISSING_CODE` | Redirect flow failed. Code not found in request parameters. |
| `UNEXPECTED_ERROR` | An unexpected error occurred. |
| `UNEXPECTED_VERIFY_STATUS` | Unexpected status returned from PingOne Verify Transaction. |

## Example

The following example journey leverages PingOne Verify to perform user identity verification.

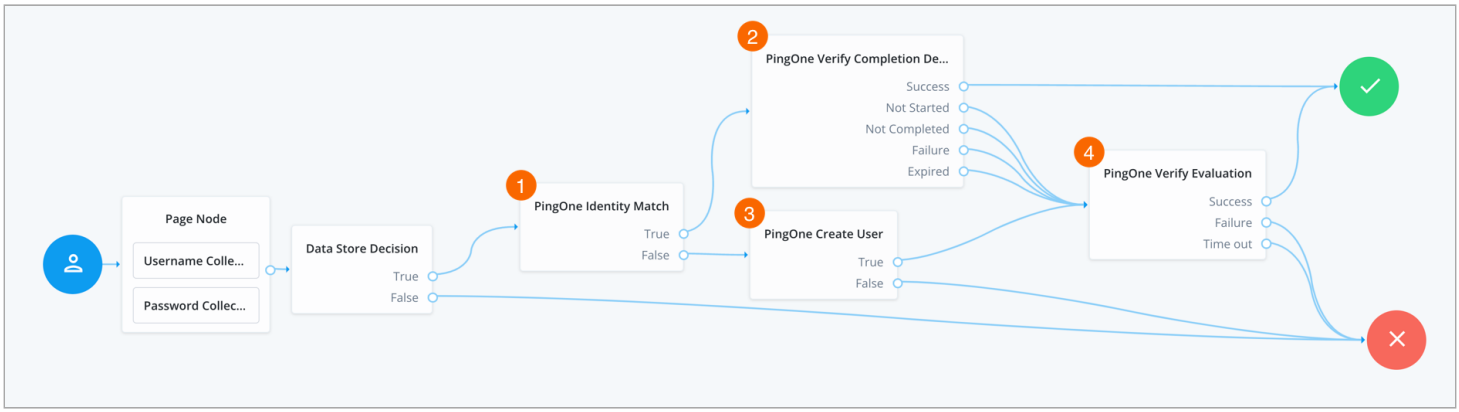



*Figure 2. Example PingOne Verify journey*

- The user enters their credentials, which the Data Store Decision node then verifies against the identity store.

- 1 The PingOne Identity Match node checks PingOne for a matching user.
- 2 If a user is found, the PingOne Verify Completion Decision node checks the user's most recent verification transaction to determine the status:

#### *Success*

The user successfully completed the most recent PingOne Verify transaction, so continue directly to the **Success** node, completing the authentication journey.

#### *Not Completed*

The user has an existing PingOne Verify transaction in progress, so continue the journey to resume the existing verification transaction.

The node adds the user's existing transaction ID to the shared node state in a variable named `pingOneVerifyTransactionId`.

#### *Not Started / Failure / Expired*

The user either does not have an existing transaction (`Not Started`), or did not successfully complete the most recent PingOne Verify transaction, or it expired, so continue the journey to start a new verification transaction.

- 3 If a user is not found, the PingOne Create User node creates a new user in PingOne.
- 4 The PingOne Verify Evaluation node starts a new PingOne Verify evaluation, or continues an existing one if `pingOneVerifyTransactionId` is present in the shared node state, and either completes or fails the journey based on the result.

## Platform Password node

The **Platform Password** node prompts the user to enter their password and stores it in a configurable property of the shared node state.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

This node uses the `_id` of the object for policy evaluation.

For existing users, the user's `_id` must be in the shared state to evaluate user-specific policies, such as password history, cannot-contain-others, and so on. No `_id` is available for new users.

## Dependencies

If this node's **Validate Password** setting is enabled, the node relies on the underlying identity service (PingIDM) for password policies.

## Configuration

| Property | Usage |
|---|---|
| **Validate Password** | When enabled, this node uses the password policies in the underlying identity service (PingIDM) to validate the user's input. It returns any policy failures as errors. For example, if you submitted an invalid password on registration, the response from this node would include a list of failed policies:<br>`{ "name": "failedPolicies", "value": [ "{ \"params\": { \"minLength\": 8 }, \"policyRequirement\": \"MIN_LENGTH\" }", "{ \"params\": { \"numCaps\": 1 }, \"policyRequirement\": \"AT_LEAST_X_CAPITAL_LETTERS\" }", "{ \"params\": { \"numNums\": 1 }, \"policyRequirement\": \"AT_LEAST_X_NUMBERS\" }" ] }`<br>Default: disabled |
| **Password Attribute** | The attribute used to store a password in the managed identity object.<br>Default: `password` |
| **Confirm Password** | Enable this option to require the user to enter the password identically in a second field.<br>**Note**<br>This property only displays when the node is placed within a Page node.<br>Default: disabled |
| **Checkmark Policy Display** | Enable this option to show a checkmark instead of faded bullet points on successful password validation.<br>**Note**<br>This property only displays when the node is placed within a Page node.<br>Default: disabled |

### Outputs

On success, this node updates the **Password Attribute** property in the shared node state with the password.

The captured password is transient, persisting only until the authentication flow reaches the next node requiring user interaction. It may be persisted to the secure state if required later in the journey.

### Outcomes

Single outcome path.

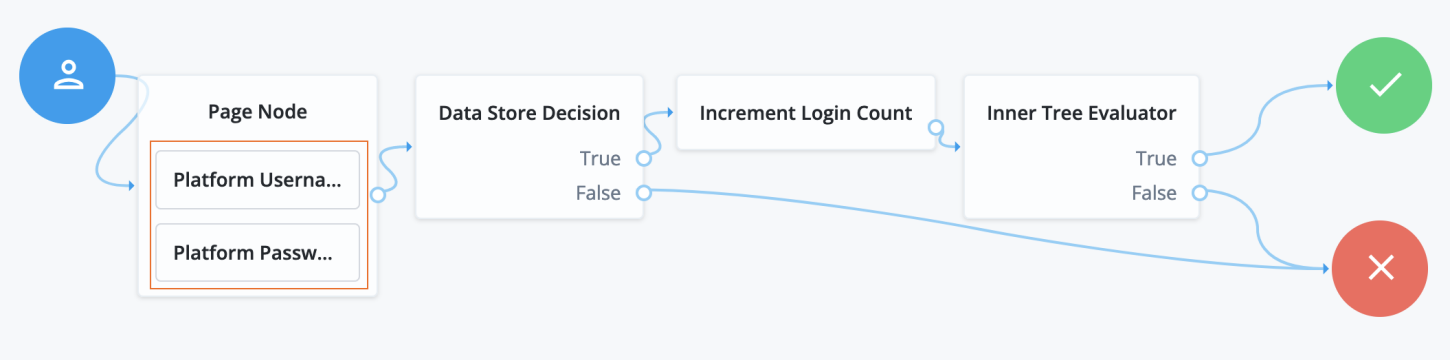
### Errors

This node does not log error or warning messages of its own.

If it fails to get the result from the underlying identity service (PingIDM) for a validation request, this node throws an exception with a `Communication failure` message.

### Example

The following journey uses a Page node containing the Platform Username node and Platform Password node to collect the username and password and set their values in the shared node state:




- The Page node presents a page with input fields to prompt for the username and password.
  - The Platform Username node collects and injects the `userName` into the shared node state.
  - The Platform Password node collects and injects the `password` into the shared node state.
- The Data Store Decision node uses the username and password to determine whether authentication is successful.
- The Increment Login Count node updates the login count on successful authentication.
- The Inner Tree Evaluator node invokes a nested journey for progressive profiling.

## Platform Username node

The **Platform Username** node prompts the user to enter their username and stores it in a configurable property of the shared node state.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

None.

## Dependencies

If this node's **Validate Username** setting is enabled, the node relies on the underlying identity service (PingIDM) for username policies.

## Configuration

| Property | Usage |
|---|---|
| **Validate Username** | When enabled, this node uses the username policies in the underlying identity service (PingIDM) to validate the user's input. It returns any policy failures as errors.<br>**Important**<br>Only enable this field if you're using this node as part of a *registration* journey. Don't enable this field in an *authentication* journey because the validation includes verifying that the provided username doesn't exist in the identity store.<br>Default: disabled |
| **Username Attribute** | The attribute used to store a username in the managed identity object.<br>Default: `userName` |

## Outputs

On success, this node updates the **Username Attribute** property in the shared node state with the username.

### Outcomes

Single outcome path.

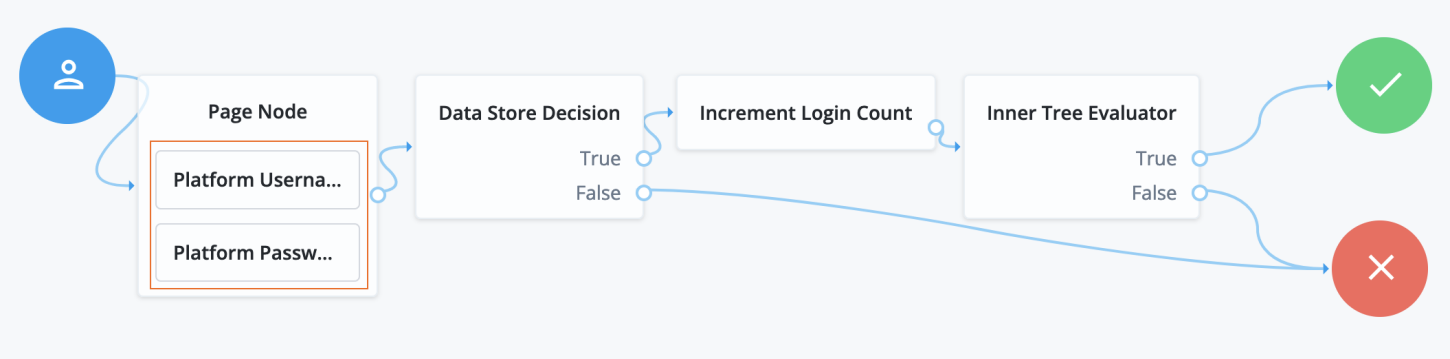
### Errors

This node does not log error or warning messages of its own.

If it fails to get the result from the underlying identity service (PingIDM) for a validation request, this node throws an exception with a `Communication failure` message.

### Example

The following journey uses a Page node containing the Platform Username node and Platform Password node to collect the username and password and set their values in the shared node state:




- The Page node presents a page with input fields to prompt for the username and password.
  - The Platform Username node collects and injects the `userName` into the shared node state.
  - The Platform Password node collects and injects the `password` into the shared node state.
- The Data Store Decision node uses the username and password to determine whether authentication is successful.
- The Increment Login Count node updates the login count on successful authentication.
- The Inner Tree Evaluator node invokes a nested journey for progressive profiling.

## Profile Completeness Decision node

Use progressive profile flows to check how much of a user's profile has been completed, where the completeness of a profile is expressed as a percentage of user-viewable, and user-editable fields that are not `null`.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Outcomes

- `True`
- `False`

### Properties

| Property | Usage |
|---|---|
| Profile Completeness Threshold | Percentage of user-viewable and user-editable fields in a profile that must be filled for the node to pass. Express this as a number between 0 and 100. |
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM). |

## Query Filter Decision node

The **Query Filter Decision** node checks if the contents of a user's profile match the specified query filter.

Use this node to check whether an attribute of the user profile matches a specific pattern. For instance, use this in progressive profile flows to check if marketing preferences are set on a user's profile.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

This node reads the **Identity Attribute** from the shared node state. If it can't read the **Identity Attribute**, it reads the `userName`.

It uses the value to look up the identity object.

## Dependencies

This node depends on the underlying identity service (PingIDM) to look up the user object.

## Configuration

| Property | Usage |
| --- | --- |
| **Query Filter** | A query filter used to check the contents of an object.<br>Learn about constructing effective query filters in the corresponding documentation for:<br>• Ping Identity Platform<br>• Advanced Identity Cloud<br>Default: none |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

## Outputs

None.

## Outcomes

### `True`

The node user profile matched the query.

### `False`

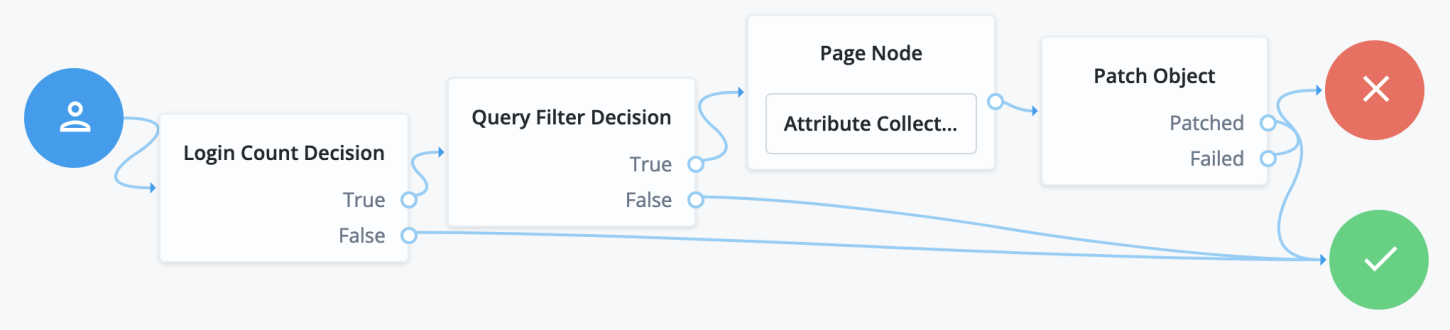
Any other case.

### Errors

This node doesn't log error or warning messages of its own.

### Example

Other journeys invoke the following progressive profile journey to capture missing profile attributes:




- The Login Count Decision node triggers the rest of the journey depending on the login count and its settings.
- The **Query Filter Decision** node determines whether managed object profile fields are missing.
- The Attribute Collector node in the Page node requests additional input for the profile.
- The Patch Object node stores the additional input in the managed object profile.

## Required Attributes Present node

Checks the underlying identity resource and determines if all attributes required to create the specified object exist within the shared node state. The default identity resource is as follows:

- PingAM `managed/user`
- Advanced Identity Cloud `managed/alpha_user`

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Outcomes

- `True`
- `False`

### Properties

| Property | Usage |
| --- | --- |
| Identity Resource | The type of managed identity resource object this node creates. It must match the identity resource type for the current flow.<br><br>**Tip**<br>To check for the available managed identity resource types, go to the IDM admin UI, and open the **Manage** drop-down list in the upper right corner of the screen. Identity managed object types are preceded by the icon. |

## Select Identity Provider node

The **Select Identity Provider** node presents an end user with a list of configured, enabled, social identity providers to use for authentication.

Use this node with the Social Provider Handler node for social authentication.

**Note**

In Advanced Identity Cloud or Ping Identity Platform deployments, you can configure this node to show only the identity providers the user has already associated with their account. This is useful, for example, in account claiming flows, where a user wants to associate a new social identity provider with an account that's being authenticated through social authentication.
In cases such as account claiming, where the user has already authenticated once and is linking a new identity provider, the node only displays a local sign-in option if it detects that the user's account has a `password` attribute.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

The node reads the `username` and `password` from shared state.

For standalone AM deployments, implement a Username Collector node and a Password Collector node earlier in the journey.

For Advanced Identity Cloud and Ping Identity Platform deployments, implement a Platform Username node and a Platform Password node earlier in the journey.

## Dependencies

The Social Identity Provider service must be configured with the details of at least one social identity provider.

Learn more in the Social Identity Provider service documentation:

- Advanced Identity Cloud
- PingAM

## Configuration

| Property | Usage |
|---|---|
| **Include local authentication** | Whether local authentication is included as a method for authenticating. |
| **Offer only existing providers** | Advanced Identity Cloud and Ping Identity Platform deployments only.<br>When enabled, the node limits the social identity provider choices to those already associated with the user object. Use this when a user is authenticating using a new social identity provider, and an account associated with that user already exists (also known as "account claiming"). |
| **Password attribute** | Advanced Identity Cloud and Ping Identity Platform deployments only.<br>The attribute in the user object that stores a user's password for use during local authentication. |
| **Identity Attribute** | Advanced Identity Cloud and Ping Identity Platform deployments only.<br>The attribute used to identify an existing user. The node uses this attribute to query the user in the backend datastore to obtain the list of existing providers. |
| **Filter Enabled Providers** | By default, the node displays all identity providers marked as **Enabled** in the Social Identity Provider Service. Enter one or more providers here to restrict the user's choice of providers to this list.<br>**Tip**<br>You can see the configured social identity providers in **Realms** > ***Realm name*** > **Services** > **Social Identity Provider Service** > **Secondary Configurations**.<br>Providers you specify here must be enabled in the **Social Identity Provider** service. If this field is empty, the node displays all enabled providers. |

### Outputs

The node writes the selected social identity provider to the shared state in the `SELECTED_IDP` key.

### Outcomes

The node has two possible outcomes:

- `Social Authentication`
- `Local Authentication`

To turn off local authentication, disable **Include local authentication** in the node configuration.

If more than one social identity provider is configured, or if a single provider is configured and **Local Authentication** is enabled, the node returns the `SelectIdPCallback`. It then requires a choice from the end user.

If no end-user choice is required, authentication proceeds to the next node in the journey.

Learn more in the documentation on the `SelectIdPCallback` callback for:

- Advanced Identity Cloud
- PingAM

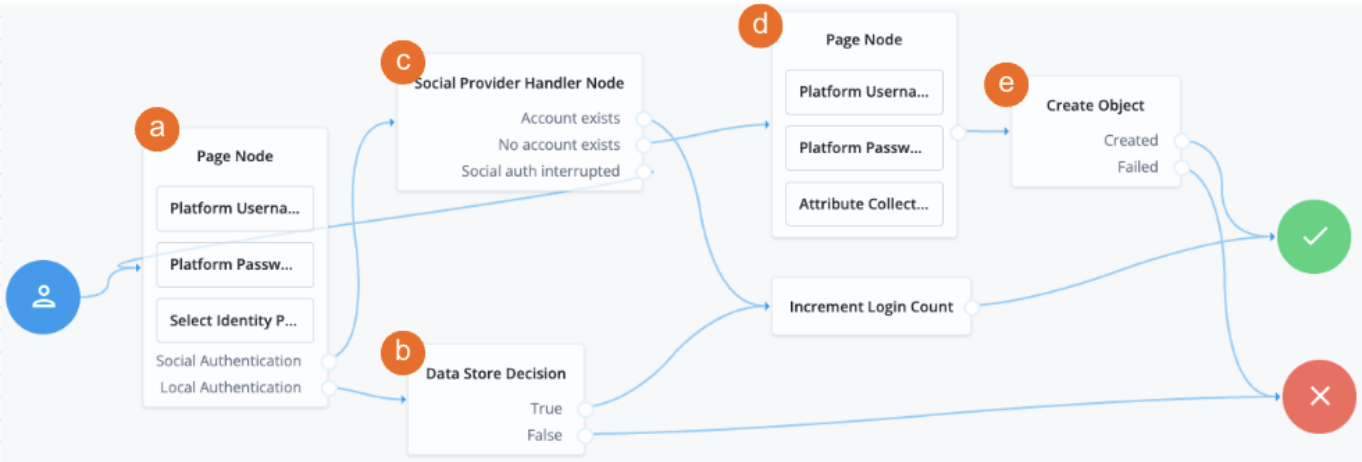
### Errors

This node logs no errors.

## Examples

Advanced Identity Cloud

**Example 1: Social authentication** This example shows the **Social Provider Handler** and **Select Identity Provider** nodes in a social authentication journey.




a The Page node containing the Select Identity Provider node prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c If the user selects social authentication, the Social Provider Handler node does the following:

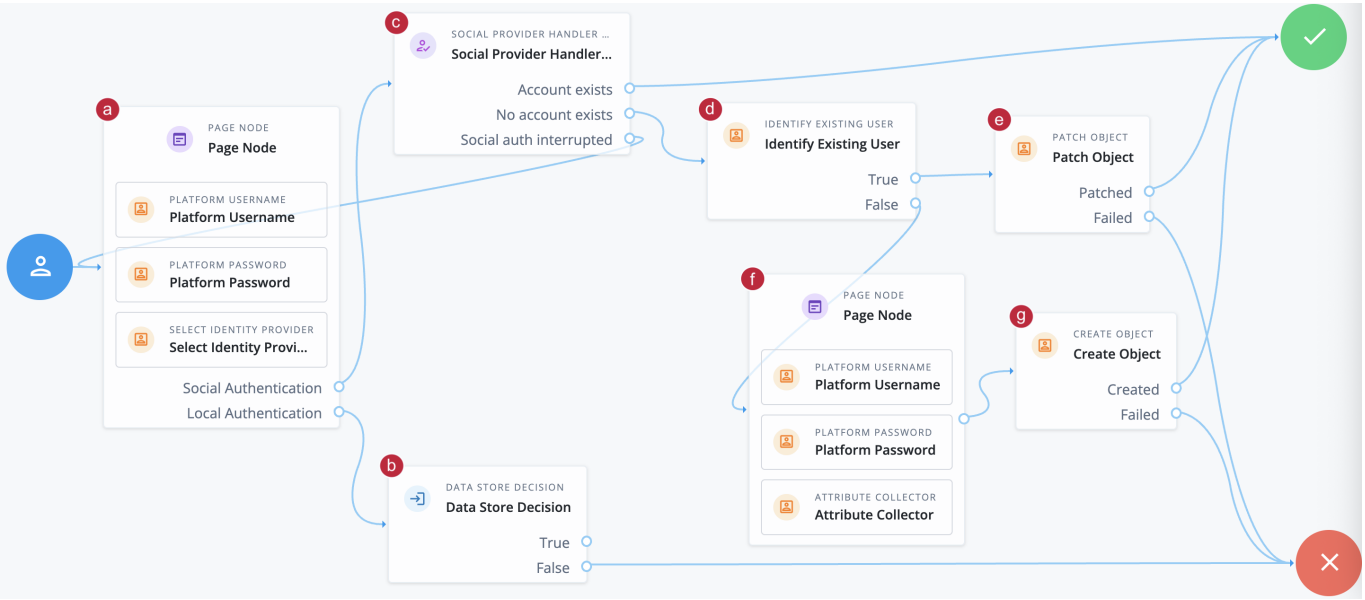
- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information, and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
    - If the user has an existing identity, authenticates that identity
    - If the user doesn't have an identity, routes the user to another page node
    - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

d The nodes on the page node request the information required to *register* a new identity.

e The Create Object node creates the new identity in AM.

**Example 2: Dynamic account linking**

This example shows a social authentication journey with dynamic account linking.




a A Page node contains the Select Identity Provider node node that prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c If the user selects social authentication, the **Social Provider Handler** node does the following:

- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information, and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
  - If the user has an existing identity, authenticates that identity
  - If the user doesn't have an identity, routes the user to the Identify Existing User node
  - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

d The Identify Existing User node checks if the user exists in AM using the **Identity Attribute** specified and does the following:

- If the user exists, routes the user to the Patch Object node
- If the user doesn't exist, routes the user to another page node

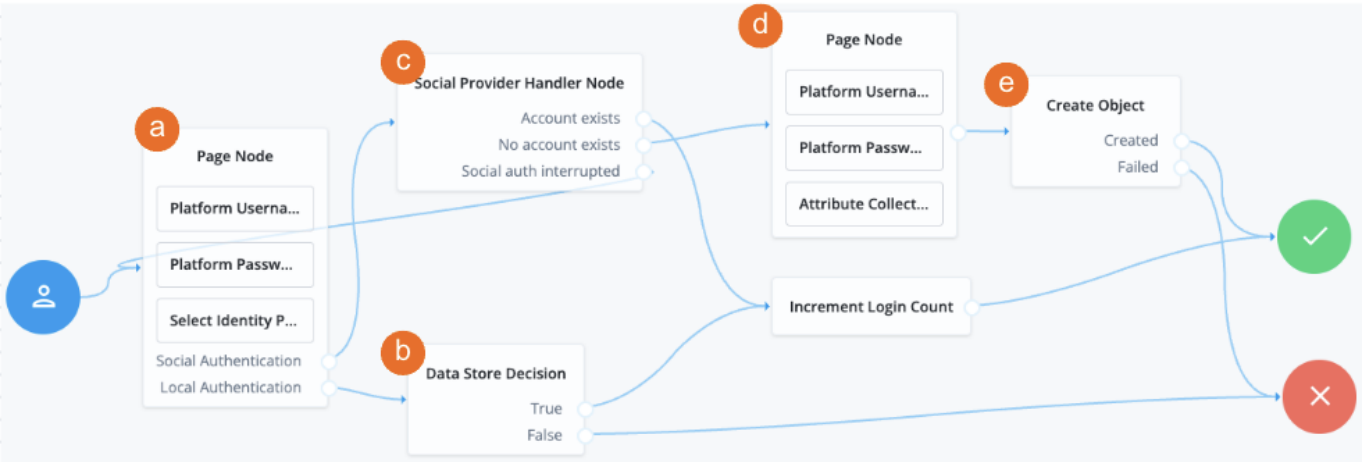
e The Patch Object node updates the existing user to create the link.

f The nodes on the page node request the information required to *register* a new identity.

g The Create Object node creates the new identity in AM.

PingAM

This example shows the **Social Provider Handler** and **Select Identity Provider** nodes in a social authentication journey.




a The Page node containing the Select Identity Provider node prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c If the user selects social authentication, the **Social Provider Handler** node does the following:

- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information, and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
  - If the user has an existing identity, authenticates that identity
  - If the user doesn't have an identity, routes the user to another page node
  - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

d The nodes on the page node request the information required to *register* a new identity.

e The Create Object node creates the new identity in AM.

## Terms and Conditions Decision node

Verifies the user has accepted the active set of terms and conditions.

You set up terms and conditions in the Ping Identity Platform admin UI. Learn more in the documentation on *Terms and conditions* for:

- PingAM
- Advanced Identity Cloud

Use this node to verify the user has accepted terms and conditions before proceeding, for example, during login or progressive profile data collection.

You can use this node with the Accept Terms and Conditions node.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Outcomes

- `True`
- `False`

### Properties

| Property | Usage |
|---|---|
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM). |

## Time Since Decision node

Checks if a specified amount of time has passed since the user was registered.

For example, to prompt users to review your terms and conditions after the account is a week old, set the **Elapsed Time** property to `1 week`. After that time has elapsed, the next time the user logs in, they are prompted to review your terms and conditions.

Use this node for progressive profile completion.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |

| Product | Available? |
| --- | --- |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Outcomes

- `True`
- `False`

## Properties

| Property | Usage |
| --- | --- |
| Elapsed Time | The amount of time since the user was created, in minutes, that needs to elapse before this node is triggered.<br>This property also supports specifying basic time units. For example, when setting the property to `10080` minutes, writing `7 days` or `1 week` also works. |
| Identity Attribute | The attribute used to identify the managed object in the underlying identity service (PingIDM). |

# Utility nodes

## Agent Data Store Decision node

The **Agent Data Store Decision** node authenticates the agent using the data store for agent profiles and sets its authentication identifier if successful.

> **Note**
>
> This node only authenticates agents, such as PingGateway and the Java and web agents.
> Use the Data Store Decision node for other identities.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the `username` and `password` properties in the incoming node state.

Obtain the agent credentials from the user or with a Zero Page Login Collector node.

### Dependencies

This node depends on the underlying data store for agent profiles.

### Configuration

This node has no configurable properties.

### Outputs

This node copies the shared and transient states into the outgoing node state.

## Outcomes

### True

The credentials match those found in the data store for agent profiles.

### False

The credentials do *not* match those found in the data store for agent profiles.

## Errors

This node can log the following warnings:

### Exception in data store decision node

The node couldn't connect to the data store, or another error occurred.

### invalid password error

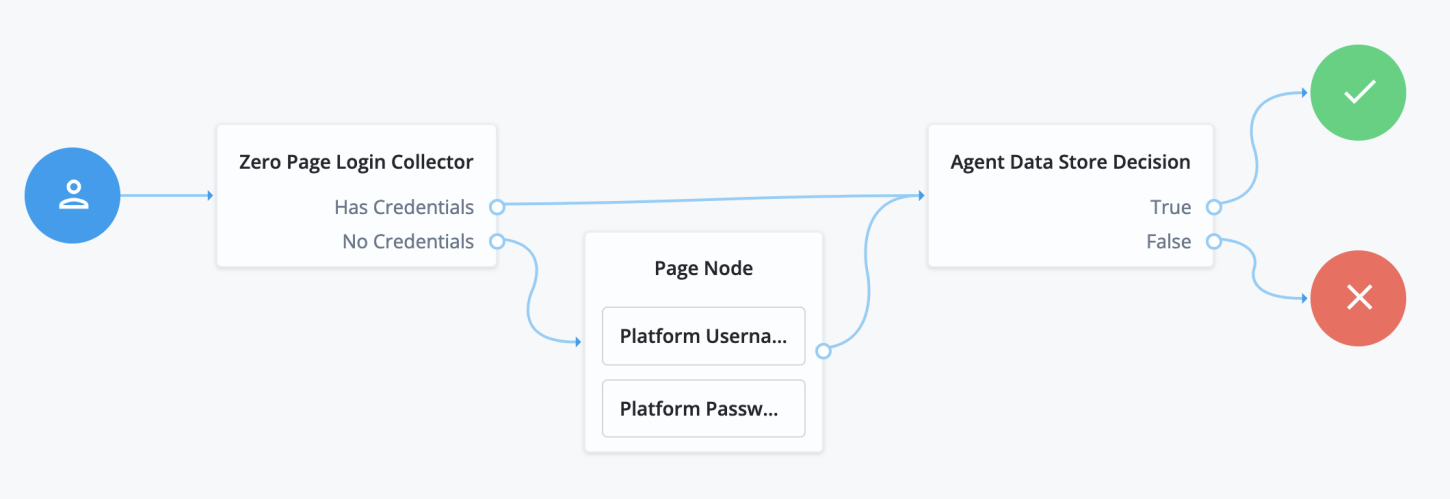
The password doesn't match.

### invalid username error

The username doesn't match any profiles found in the data store.

## Example

The following example uses this node to authenticate an agent with the credentials provided:




- The Zero Page Login Collector node collects the username and password from HTTP headers if provided.
- The Page node collects the username and password interactively from the user.
- The **Agent Data Store Decision** node verifies the agent credentials match those in the data store.

# Amster Jwt Decision node

PingAM

The **Amster Jwt Decision** node lets AM authenticate Amster connections using SSH keys.

The Amster client signs the JWT using a local private key. AM verifies the signature using the list of public keys in the `authorized_keys` file. Specify the path to the `authorized_keys` file in the node configuration.

If the entry in the authorized keys file contains a `from` parameter, only connections originating from a qualifying host are permitted.

Find more information in Private key connections in the Amster documentation.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

The node reads the `NONCE_STATE_KEY` from the Amster client.

## Dependencies

None.

## Configuration

| Property | Usage |
|---|---|
| **Authorized Keys** | Location of the `authorized_keys` file used to validate remote Amster connections. This file has the same format as an OpenSSH `authorized_keys` file. |

## Outputs

This node doesn't change the shared state.

### Outcomes

#### *True*

The journey follows this outcome if the node can validate the incoming private key against the public keys in the `authentication_keys` file. Successful authentication creates an `amAdmin` session in AM.

#### *False*

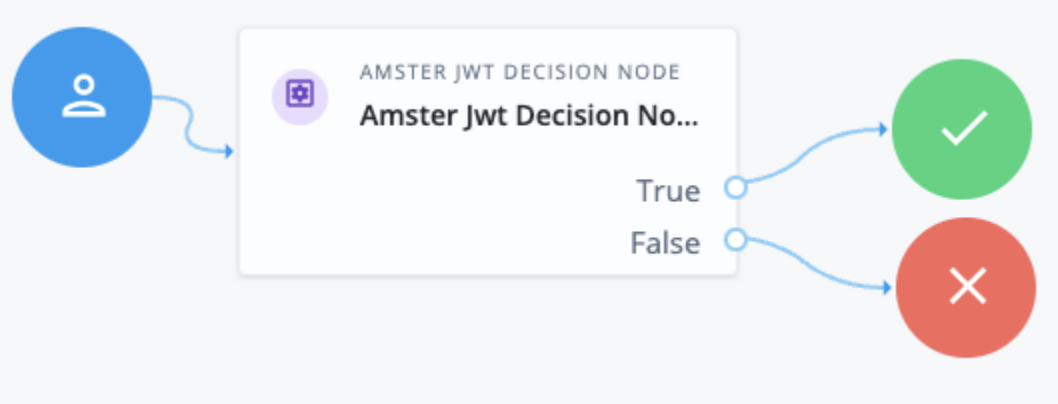
The journey follows this outcome if the node can't validate the incoming private key against the public keys in the `authentication_keys` file, either because the incoming key is invalid, or because the `authentication_keys` file is inaccessible.

### Errors

If the node can't read the `authorized_keys` file, it returns the error `AmsterJwtDecisionNode: Could not read authorized keys file filename`.

### Examples

This node is used only by the `amsterService` authentication tree:




**Caution**

Changing or removing this tree could prevent Amster from connecting to AM.

## Anonymous Session Upgrade node

Upgrades an anonymous session to a non-anonymous session.

Use this as the first node in the flow.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

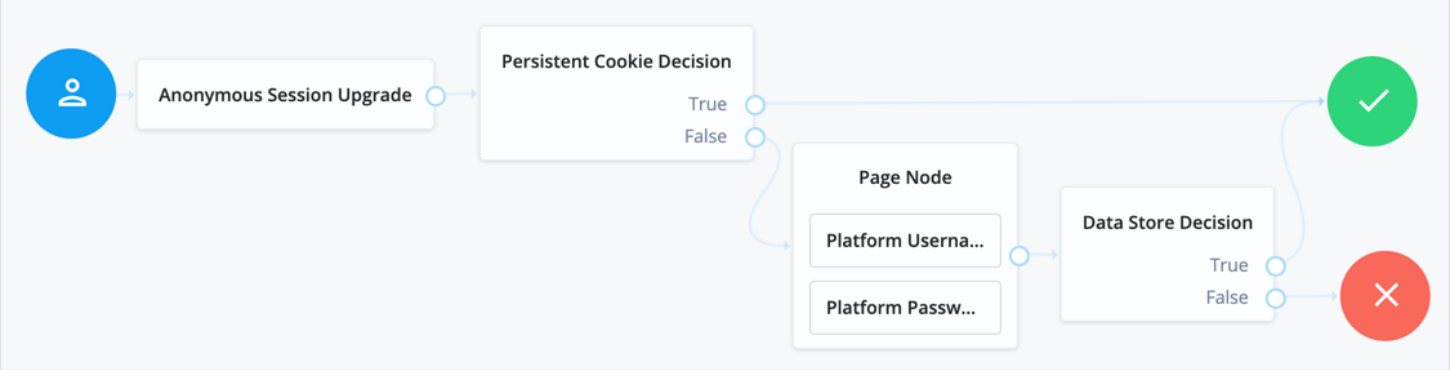
Single outcome path.

### Properties

This node has no configurable properties.

### Example

After using the Anonymous User Mapping node to access AM as an anonymous user, this node lets users upgrade their session to a non-anonymous one:




## Anonymous User Mapping node

Lets users log in to an application or website without providing credentials, by assuming the identity of a specified existing user account. The default user for this purpose is named `anonymous`.

Take care to limit access for such users. For example, grant anonymous users access to public downloads on your site.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

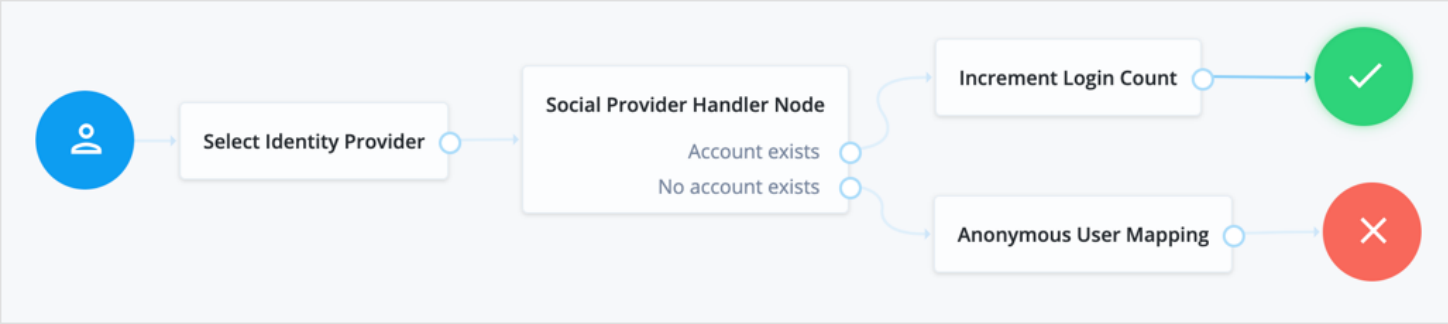
Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Anonymous User Name | Specifies the username of an account that represents anonymous users. This user must already exist in the realm, and its user status must be `active`. |

### Example

The following example uses this node to grant access as an anonymous user to users who have performed social authentication access and do not have an existing profile:




## Backchannel Initialize node

Advanced Identity Cloud

The **Backchannel Initialize** node lets you start a separate journey that runs asynchronously, possibly by a different user or agent. The node takes an incoming user ID and generates a URL to a journey where the identified user or agent authenticates.

Together with the Backchannel Status node, this node lets you implement *backchannel authentication* from within a journey. Find more information in the documentation on *Backchannel authentication* for Advanced Identity Cloud.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | No |
| Ping Identity Platform (self-managed) | No |

## Inputs

This node optionally reads the user ID of the *subject* the journey's being initialized for from the incoming node state. The user ID is stored in the `nodeState` key specified in the **Subject Name Key** property.

## Dependencies

None

## Configuration

| Property | Usage |
|---|---|
| **Journey** | The asynchronous journey to initialize.<br>Select a journey from the list of configured journeys. |
| **Subject Type** | The type of subject to initialize the journey for:<br>***User***<br>The subject is a user identity.<br>***Agent***<br>The subject is a web agent or Java agent.<br>***None***<br>Setting the **Subject Type** to `None` means any subject can log in using the initialized journey. |
| **Subject Name Key** | The `nodeState` key that contains the user ID of the subject you're initializing the journey for. This property is ignored if the **Subject Type** is `None`. |
| **Data Object Key** | The node state key that contains the data object (if present) to pass to the journey at the root level of the shared state. |

| Property | Usage |
| --- | --- |
| **Redirect URL Type** | The type of redirect URL to save to node state:<br>By default, the base URL of the redirect URI is retrieved from the incoming HTTP request.<br>***Get***<br>The node redirects the user to a URL based on the base URL service. The redirect uses a GET request to the `/XUI/Login` endpoint.<br>***Post***<br>The node redirects the user to a URL based on the base URL service. The redirect uses a POST request to the `authenticate` endpoint.<br>***Custom***<br>The node redirects the user to the URL specified in the **Custom Redirect URL** field.<br>***None***<br>If the none of the redirect URL mechanisms (`Get`, `Post`, or `Custom`) meet your business requirements, you can use the transaction ID stored in state to make your own redirect URL outside of the functionality of this node. Connect this output to a Scripted Decision node to achieve this. |
| **Custom Redirect URL** | If **Redirect URL Type** is `Custom`, set this field to the custom redirect URL for the authentication journey. |

## Outputs

The node writes the following to the shared state:

| Shared state key | Information |
| --- | --- |
| `backchannel-transaction` | The transaction ID of the backchannel authentication request. |
| `backchannel-redirectUri` | The generated redirect URI. |
| `backchannel-data` | An optional data object with additional information about the authenticating user. |

## Outcomes

### `Created`

The journey follows this outcome path if the node was able to create the backchannel authentication request.

### `Unknown Subject`

The journey follows this outcome path if the subject in the incoming node state doesn't match an identity object in the backend identity store.

### `Error`

The journey follows this outcome path if the node can't retrieve the subject from the node state.

## Errors

- If the node can't retrieve the subject from the incoming state, it logs the following warning:

```
Error retrieving subject from node state.
```

- If the node can't initialize the backchannel authentication journey, it logs the following error:

```
Error initializing back channel transaction.
```

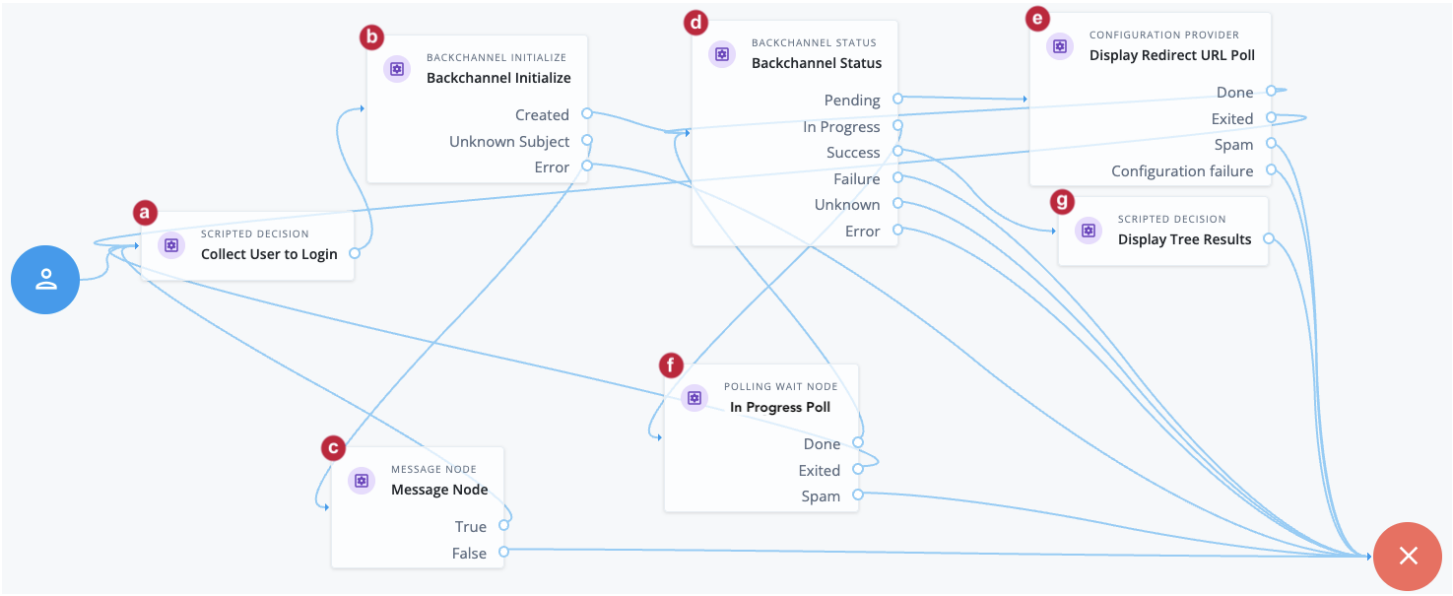
## Examples

This example uses the **Backchannel Initialize** and **Backchannel Status** nodes to implement backchannel authentication.

The example shows two journeys:

- The main journey initializes a backchannel authentication journey.
- The backchannel journey is a simple authentication journey.

### Main journey




a The **Collect User to Login** node is a Scripted Decision node. The script writes the attributes required for the backchannel authentication into the shared state.

Advanced Identity Cloud

The script queries the backend identity object to get the `userId`, then writes that and the attributes required for the backchannel authentication into the shared state.

```
if (callbacks.isEmpty()) {
    // Request callbacks
    callbacksBuilder.nameCallback("User to authenticate");
} else {
    // Callbacks returned from browser, save username and password
    var username = callbacks.getNameCallbacks().get(0);
    var queryRes = openidm.query("managed/alpha_user", {
        "_queryFilter": `/userName eq '${username}'`
    }, ["*", "_id"]);
    var userId = queryRes.result[0]._id
    var identity = idRepository.getIdentity(userId);
    nodeState.putShared("backchannel-user", identity.getName());
    nodeState.putShared("backchannel-data", {
        "username": username,
        "objectAttributes": {
            "userName": username,
            "_id": userId
        }
    });
    nodeState.putShared("_id", userId);
    outcome = "outcome";
}
```

PingAM

Not yet available in PingAM

b The **Backchannel Initialize node** reads the value of the `backchannel-user` key from the shared state. This key contains the `userName`:

- If the `userName` is available and is valid, the node generates a redirect URI to start the backchannel authentication journey. The node writes the redirect URI and the transaction ID of the backchannel transaction to the shared state, and the journey proceeds to the **Backchannel Status** node.
- If the `userName` can't be read, the journey follows the Error outcome and fails.
- If the `userName` can be read but the user or agent isn't valid, the journey proceeds to a Message node (c) and redirects the user to the start of the journey to attempt gathering data again.

d The **Backchannel Status** node reads the transaction ID and provides status on the authentication request:

- If the backchannel authentication request is `Pending`, the journey proceeds to the **Display Redirect URL Poll** node (e), which is a Configuration Provider node.

- When the backchannel authentication is `In progress`, the journey proceeds to the **In Progress Poll** node (f), which is a Polling Wait node.
- When the backchannel authentication completes successfully, the journey proceeds to the **Display Tree Results** node (g), which is a Scripted Decision node.

e The **Configuration Provider** node imitates a Polling Wait node that uses a script to display the backchannel redirect URI as long as the backchannel authentication request is in a `Pending` state.

```
var uri = nodeState.get("backchannel-redirectUri").asString();
config = {
    "spamDetectionTolerance": 3,
    "spamDetectionEnabled": true,
    "exitMessage": {},
    "waitingMessage": {
        "en": uri
    },
    "secondsToWait": 5,
    "exitable": true
};
```

- After 5 seconds, the journey returns to the **Backchannel Status** node.
- If the journey exits before it returns to the **Backchannel Status** node, the user is redirected to the start of the main journey to attempt gathering data again.
- If the **Configuration Provider** node detects spam or misconfiguration, the main journey follows the failure outcome path.

f The **In Progress Poll** node is a Polling Wait node that pauses the main journey until the Backchannel journey is complete.

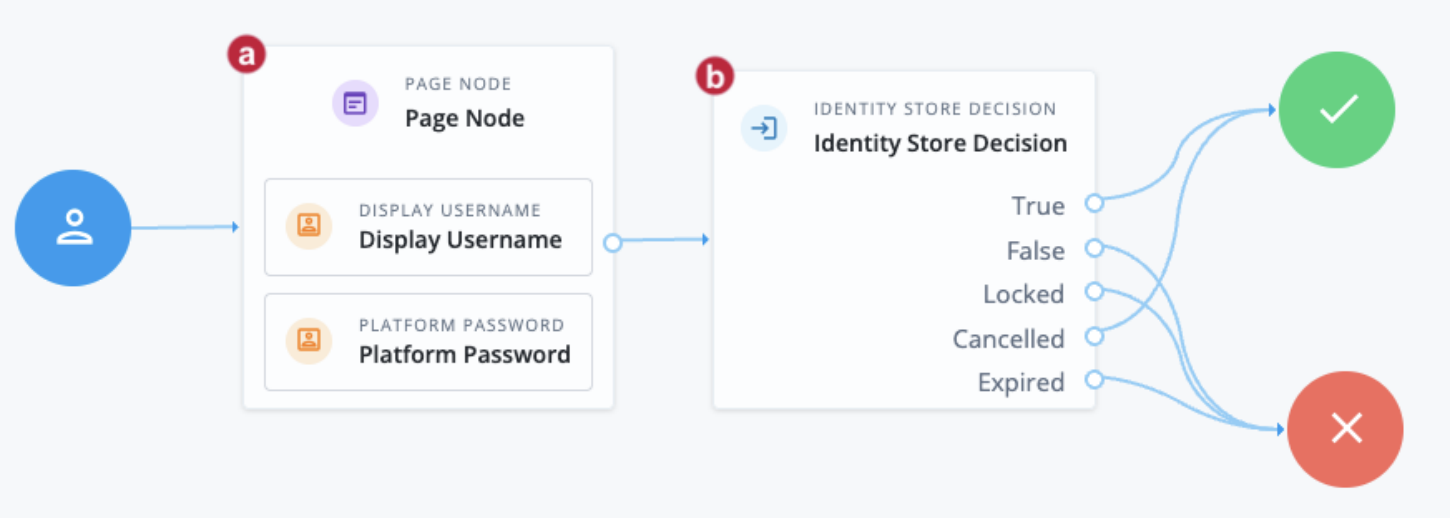
- After 8 seconds, the journey returns to the **Backchannel Status** node.
- If the journey exits before it returns to the **Backchannel Status** node, the user is redirected to the start of the main journey to attempt gathering data again.
- If the node detects spam, the main journey follows the failure outcome path.

g The **Display Tree Results** node is a Scripted Decision node that displays the outcome of the backchannel authentication journey.

```
/*
- Data made available by nodes that have already executed are available in the sharedState variable.
- The script should set outcome to either "true" or "false".
*/
if (callbacks.isEmpty()) {
    var sessionProperties = nodeState.get("backchannel-sessionProperties");
    callbacksBuilder.textOutputCallback(0, sessionProperties);
} else {
    outcome = "outcome";
}
```

> **Note**
>
> This journey always ends on the Failure node as it is not in itself an authentication journey.

### Backchannel authentication journey




This is a basic authentication journey that takes credentials and authenticates the user based on their existence in the backend identity store.

a The **Page** node includes a Display Username node and a Platform Password node. The username has been supplied in the shared state from the main journey. The user needs to enter their password.

b The **Identity Store Decision** node assesses the user credentials. Find more information on this node and its outcomes in Identity Store Decision node.

The main journey polls for completion of this subjourney. When this journey completes, the main journey continues.

## Backchannel Status node

Advanced Identity Cloud

The **Backchannel Status** node checks the status of an asynchronous (backchannel) user journey.

Together with the Backchannel Initialize node, this node lets you implement *backchannel authentication* from within a journey. Find more information in the documentation on *Backchannel authentication* for Advanced Identity Cloud.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | No |
| Ping Identity Platform (self-managed) | No |

### Inputs

This node requires the transaction ID of the backchannel authentication request from the node state. Implement a Backchannel Initialize node before this node in the journey to provide this input.

### Dependencies

None

### Configuration

| Property | Usage |
| --- | --- |
| **Record Journey Session Info** | If `true`, the node records the session information returned from the journey in the transient state when the journey completes successfully. |

### Outputs

If **Record Journey Session Info** is `true`, the node writes the journey session properties to the transient state in the `backchannel-sessionProperties` key.

### Outcomes

#### *Pending*

The journey follows this outcome if the backchannel authentication journey has not yet started.

#### *In Progress*

The journey follows this outcome if the backchannel authentication journey has been started but not yet completed.

#### *Success*

The journey follows this outcome if the backchannel authentication journey has completed successfully.

#### *Failure*

The journey follows this outcome if the backchannel authentication journey completed but failed.

#### *Unknown*

The journey follows this outcome if the node is unable to assess the status of the backchannel authentication journey, usually because it timed out.

#### *Error*

The journey follows this outcome in any other case.

### Errors

If the node is unable to assess the status of the backchannel authentication journey, it writes the following error to the log:

```
Error checking back channel transaction status.
```

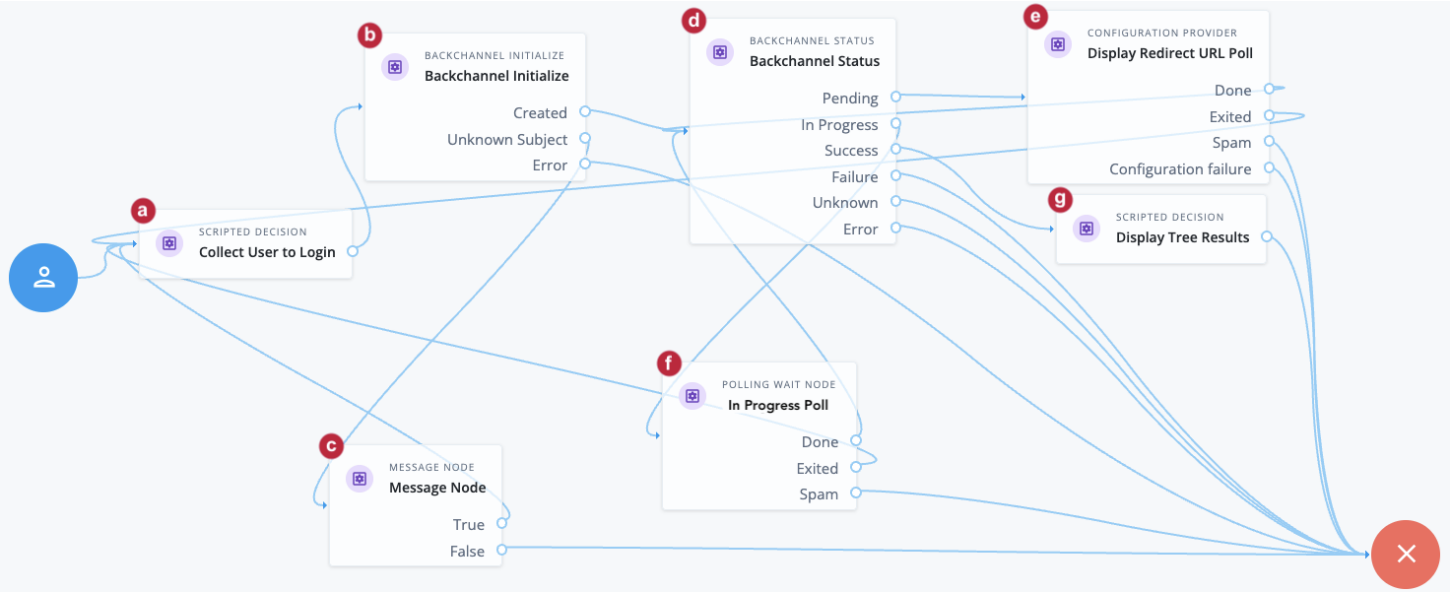
### Examples

This example uses the **Backchannel Initialize** and **Backchannel Status** nodes to implement backchannel authentication.

The example shows two journeys:

- The main journey initializes a backchannel authentication journey.
- The backchannel journey is a simple authentication journey.

#### Main journey




a The **Collect User to Login** node is a Scripted Decision node. The script writes the attributes required for the backchannel authentication into the shared state.

Advanced Identity Cloud

The script queries the backend identity object to get the `userId`, then writes that and the attributes required for the backchannel authentication into the shared state.

```
if (callbacks.isEmpty()) {
    // Request callbacks
    callbacksBuilder.nameCallback("User to authenticate");
} else {
    // Callbacks returned from browser, save username and password
    var username = callbacks.getNameCallbacks().get(0);
    var queryRes = openidm.query("managed/alpha_user", {
        "_queryFilter": `/userName eq '${username}'`
    }, ["*", "_id"]);
    var userId = queryRes.result[0]._id
    var identity = idRepository.getIdentity(userId);
    nodeState.putShared("backchannel-user", identity.getName());
    nodeState.putShared("backchannel-data", {
        "username": username,
        "objectAttributes": {
            "userName": username,
            "_id": userId
        }
    });
    nodeState.putShared("_id", userId);
    outcome = "outcome";
}
```

PingAM

Not yet available in PingAM

b The **Backchannel Initialize node** reads the value of the `backchannel-user` key from the shared state. This key contains the `userName`:

- If the `userName` is available and is valid, the node generates a redirect URI to start the backchannel authentication journey. The node writes the redirect URI and the transaction ID of the backchannel transaction to the shared state, and the journey proceeds to the **Backchannel Status** node.
- If the `userName` can't be read, the journey follows the Error outcome and fails.
- If the `userName` can be read but the user or agent isn't valid, the journey proceeds to a Message node (c) and redirects the user to the start of the journey to attempt gathering data again.

d The **Backchannel Status** node reads the transaction ID and provides status on the authentication request:

- If the backchannel authentication request is `Pending`, the journey proceeds to the **Display Redirect URL Poll** node (e), which is a Configuration Provider node.

- When the backchannel authentication is `In progress`, the journey proceeds to the **In Progress Poll** node (f), which is a Polling Wait node.
- When the backchannel authentication completes successfully, the journey proceeds to the **Display Tree Results** node (g), which is a Scripted Decision node.

e The **Configuration Provider** node imitates a Polling Wait node that uses a script to display the backchannel redirect URI as long as the backchannel authentication request is in a `Pending` state.

```
var uri = nodeState.get("backchannel-redirectUri").asString();
config = {
    "spamDetectionTolerance": 3,
    "spamDetectionEnabled": true,
    "exitMessage": {},
    "waitingMessage": {
        "en": uri
    },
    "secondsToWait": 5,
    "exitable": true
};
```

- After 5 seconds, the journey returns to the **Backchannel Status** node.
- If the journey exits before it returns to the **Backchannel Status** node, the user is redirected to the start of the main journey to attempt gathering data again.
- If the **Configuration Provider** node detects spam or misconfiguration, the main journey follows the failure outcome path.

f The **In Progress Poll** node is a Polling Wait node that pauses the main journey until the Backchannel journey is complete.

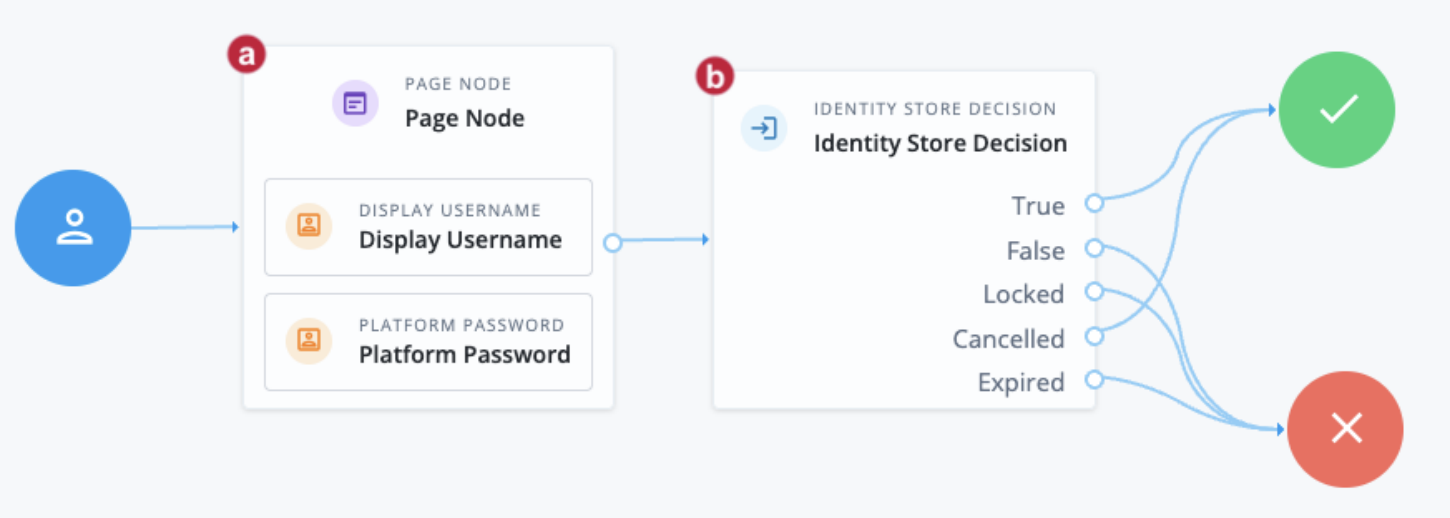
- After 8 seconds, the journey returns to the **Backchannel Status** node.
- If the journey exits before it returns to the **Backchannel Status** node, the user is redirected to the start of the main journey to attempt gathering data again.
- If the node detects spam, the main journey follows the failure outcome path.

g The **Display Tree Results** node is a Scripted Decision node that displays the outcome of the backchannel authentication journey.

```
/*
- Data made available by nodes that have already executed are available in the sharedState variable.
- The script should set outcome to either "true" or "false".
*/
if (callbacks.isEmpty()) {
    var sessionProperties = nodeState.get("backchannel-sessionProperties");
    callbacksBuilder.textOutputCallback(0, sessionProperties);
} else {
    outcome = "outcome";
}
```

> **Note**
>
> This journey always ends on the Failure node as it is not in itself an authentication journey.

### Backchannel authentication journey




This is a basic authentication journey that takes credentials and authenticates the user based on their existence in the backend identity store.

a The **Page** node includes a Display Username node and a Platform Password node. The username has been supplied in the shared state from the main journey. The user needs to enter their password.

b The **Identity Store Decision** node assesses the user credentials. Find more information on this node and its outcomes in Identity Store Decision node.

The main journey polls for completion of this subjourney. When this journey completes, the main journey continues.

## Choice Collector node

Define two or more options to present to the user when authenticating.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

- *Choice 1*

...

- *Choice n*

### Properties

| Property | Usage |
|---|---|
| Choices | Enter two or more choice strings to display to the user.<br>To remove a choice, select its **Delete** icon ✕.<br>To delete all choices, select the **Clear all** button in the **Choices** field. |
| Default Choice *(required)* | Enter the value of the choice to be selected by default.<br>**Important**<br>If you do not specify a default choice, the first choice in the list becomes the default. |
| Prompt *(required)* | Enter the prompt string to display to the user when presenting the choices. |
| Field Display Type | The format of the options presented to the user.<br>**Note**<br>• This property only displays when the node is placed within a Page node.<br>• This property is not visible in the AM admin UI. It requires the Platform UI (Advanced Identity Cloud admin console).<br>Possible values are:<br>`select`<br>Lets the user select one or more options from a selection (default).<br>`radio`<br>Lets the user select a single option from a group of radio buttons. |

## Configuration Provider node

The **Configuration Provider** node is a scripted node that dynamically imitates another node and replaces it in the journey.

The script builds a map of configuration properties matching settings for the imitated node. The **Configuration Provider** node uses the settings to imitate the other node.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

The specific shared state inputs depend on your script and the configuration it builds. The shared state data must include all required **Script Inputs** properties.

In other words, shared state data must include whatever the **Script** requires to prepare configuration data for the imitated node.

## Dependencies

To use this node, you need to prepare a script that provides values for the settings of the imitated node.

1. Choose the type of node to imitate
2. Get a template script
3. Configure the script

### Choose the type of node to imitate

The imitated node must have a *defined set of outcomes*.

This can be a variable set of outcomes from a predefined list, such as the Polling Wait node, or a fixed set of outcomes, such as the Message node.

You can't use a node type whose outcomes are defined *entirely* by configuration, such as the Scripted Decision node.

### Get a template script

Use the REST API to get the required configuration properties for the imitated node.

Advanced Identity Cloud

1. Call the following API endpoint with the `configProviderScript` action to get a template script for the type of node you want to imitate:

   ```
   /realm-config/authentication/authenticationtrees/nodes/node-type?_action=configProviderScript
   ```

   For example, the following request returns a base64-encoded template script for a Polling Wait node:

   ```
   $ curl \
   --request POST \
   --header 'Accept-API-Version: resource=1.0' \
   --header "Authorization: Bearer access-token"\
   --header "Content-Type: application/json" \
   "https://tenant-env-fqdn/am/json/realms/root/realms/alpha/realm-config/authentication/
   authenticationtrees/nodes/PollingWaitNode?_action=configProviderScript"
   {
     "script":"LyoqCiAqIFRoZSBmb2xsb3dpbmcgc2.."
   }
   ```

2. Decode the `script` value, for example, by using an online base64 decoder.

   The decoded output shows the required properties. For example, a request for the **Polling Wait node** generates the following template script:

   ```
   config = {
     "secondsToWait" : 8,
     "spamDetectionEnabled" : false,
     "spamDetectionTolerance" : 3,
     "waitingMessage" : { },
     "exitable" : false,
     "exitMessage" : { }
   };
   ```

PingAM

1. Call the following API endpoint with the `configProviderScript` action to get a template script for the type of node you want to imitate:

   ```
   /realm-config/authentication/authenticationtrees/nodes/node-type?_action=configProviderScript
   ```

   Learn more about the node endpoint in the API explorer.

2. Specify the language as `JAVASCRIPT` or `GROOVY` in the request body. If omitted, the default is JavaScript.

   For example, the following request returns a base64-encoded JavaScript template script for a Polling Wait node:

   ```
   $ curl \
   --request POST \
   --header "iPlanetDirectoryPro: admin-tokenId" \
   --header "Content-Type: application/json" \
   --data '{"language":"JAVASCRIPT"}' \
   "https://am.example.com:8443/am/json/realm-config/authentication/authenticationtrees/nodes/
   PollingWaitNode?_action=configProviderScript"
   {
     "script":"LyoqCiAqIFRoZSBmb2xsb3dpbmcgc2.."
   }
   ```

3. Decode the `script` value, for example, by using an online base64 decoder.

   The decoded output shows the required properties. For example, a request for the **Polling Wait node** generates the following template script:

   ```
   config = {
     "secondsToWait" : 8,
     "spamDetectionEnabled" : false,
     "spamDetectionTolerance" : 3,
     "waitingMessage" : { },
     "exitable" : false,
     "exitMessage" : { }
   };
   ```

### Configure the script

Create a script for the node.

Advanced Identity Cloud

1. Use the imitated node's properties to create a **Configuration Provider Node** or **Configuration Provider Node (Next-Gen)** type script.

   Find more information in Auth scripting.

2. Base your script on the `config-provider-node.js` sample.

   Your script must have a `config` object. This object must define the configuration properties that match the settings of the imitated node.

   For example, a script for the Email Suspend node has the following structure:

   ```
   config = {
     "emailTemplateName" : "registration",
     "emailAttribute" : "mail",
     "emailSuspendMessage" : {
       "en" : "An email has been sent to the address you entered. Click the link in that email to
   proceed."
     },
     "objectLookup" : false,
     "identityAttribute" : "userName"
   };
   ```

   A Configuration Provider node script can use either the next-generation or legacy script engine. It has access to all the common bindings available for its script type. You can use these to help you set the values of the configuration properties.

   Refer to the example for a script that uses shared state data to set the node configuration.

3. Save your script, and select it from the list when you configure the node's **Script** property.

PingAM

1. Use the imitated node's properties to create a **Config Provider** or **Config Provider Node (Next-Gen)** type script.

   Find more information in Manage scripts (UI).

2. Base your script on the `config-provider-node.js` sample.

   Your script must have a `config` object. This object must define the configuration properties that match the settings of the imitated node.

   For example, a script for the Email Suspend node has the following structure:

   ```
   config = {
     "emailTemplateName" : "registration",
     "emailAttribute" : "mail",
     "emailSuspendMessage" : {
       "en" : "An email has been sent to the address you entered. Click the link in that email to
   proceed."
     },
     "objectLookup" : false,
     "identityAttribute" : "userName"
   };
   ```

   A Configuration Provider node script can use either the next-generation or legacy script engine. It has access to all the common bindings available for its script type. You can use these to help you set the values of the configuration properties.

   Refer to the example for a script that uses shared state data to set the node configuration.

3. Save your script, and select it from the list when you configure the node's **Script** property.

## Configuration

| Property | Usage |
|---|---|
| **Script** | Select the script you created for this node. The list displays legacy and next-generation Configuration Provider scripts.<br>Advanced Identity Cloud only: Click + to open a script editor and create a new script. |
| **Node Type** | Select the type of node to imitate. The list is restricted to node types with outcomes that are fixed or variable based on predefined values. |
| **Script Inputs** | Optionally, limit the shared state data properties in the shared state input to the selected **Script**.<br>Default: `*` (Any available shared state property) |

## Outputs

The outputs match those of the imitated node.

## Outcomes

The **Configuration Provider** node inherits the outcomes of its configured **Node Type**. Connect these as you would the outcomes of the imitated node.

Nodes with a variable set of outcomes inherit all the possible outcomes even if runtime configuration makes them unreachable. Connect these outcomes to the Failure node.

This node also has a **Configuration failure** outcome. The **Configuration failure** outcome arises when:

- The **Configuration Provider** node failed to build the configuration map.
- The configuration map is missing required values.
- The configuration map is invalid.

## Errors

In addition to the messages from the imitated node, this node can log the following:

### *Warnings*

- `Failed to collect inputs of contained node: node-type`

  A required input property was missing.

- `Failed to get outcome provider for node type.`

  The **Node Type** outcomes were missing.

### *Errors*

- `Failed to configure node: node-type`

  This corresponds to the **Configuration failure** outcome.

To troubleshoot HTTP errors this node causes, refer to the *Errors* section of the imitated node.

## Examples

### Example 1: Imitate the Message node

In the following example, the **Configuration Provider** node imitates a Message node.

The **Configuration Provider** settings are the following:

### *Script*

A script to configure a Message node dynamically.

The script assigns values to the required properties, accessing the `username` from shared state data to set the message:

```
config = {
    "message": {"en-GB": `Hi ${nodeState.get("username")}. Please confirm you are over 18.`},
    "messageYes": {"en-GB": "Confirm"},
    "messageNo": {"en-GB": "Deny"},
    "stateField" : null
}
```

### *Node Type*

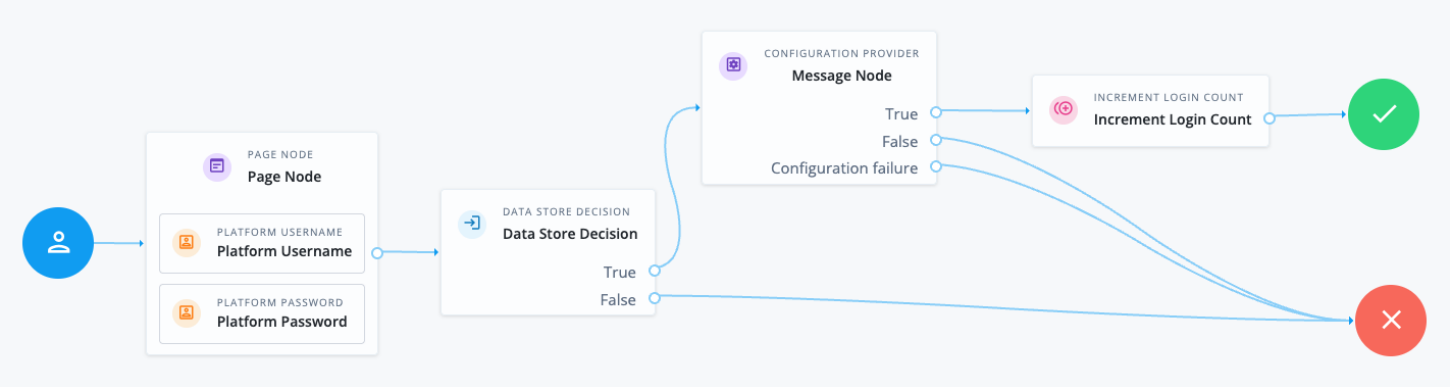
`Message Node`

### *Script Inputs*

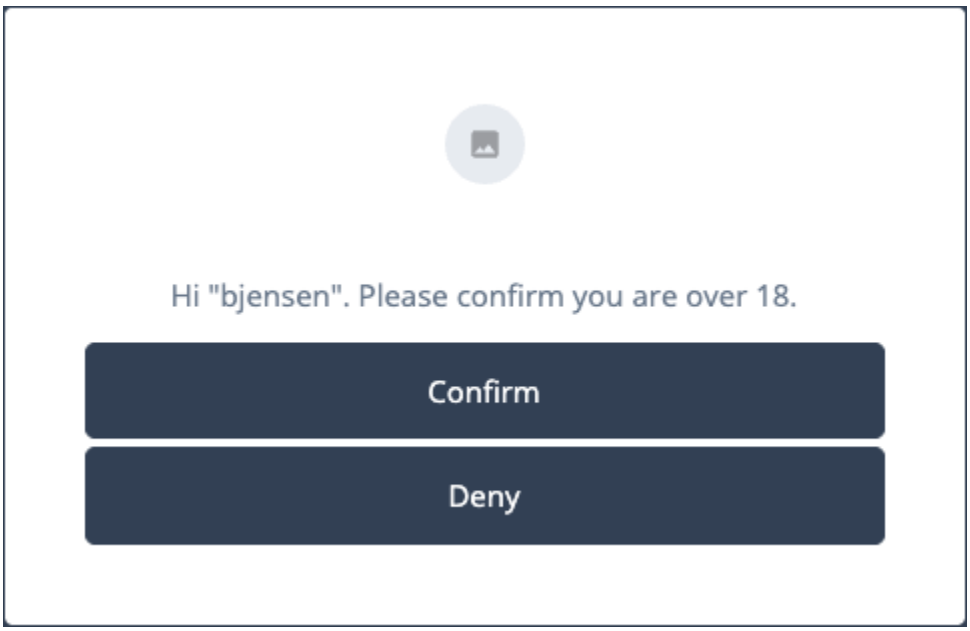
`username`

The default, `*`, also works because `username` is one of the available shared state properties.

The **Configuration Provider** node is part of a journey where the user enters their username and password before getting the message screen, so their username is in the shared state data. Notice the outcomes of the node include those of the Message node (**True**, **False**):




When the journey reaches the **Configuration Provider** node, the script for the node retrieves the `username` and dynamically configures the node. The **Configuration Provider** node, imitating a Message node, prompts the user with the message:

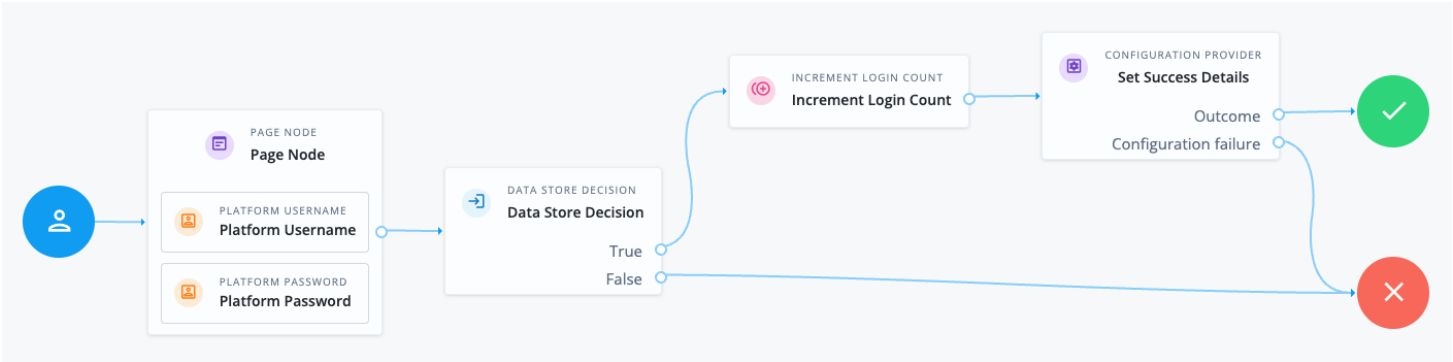



- When the user clicks **Confirm**, the journey continues to the Increment Login Count node.
- When the user clicks **Deny**, the journey continues to the Failure node.
- If the configuration process fails, the node triggers the **Configuration failure** outcome and the journey continues to the Failure node. In this case, you can find the reason for the failure in the logs.

### Example 2: Imitate the Set Success Details node

This example uses the **Configuration Provider** node to imitate a Set Success Details node and add the following additional details to the JSON response:

- A dynamic `apiResponse` field, which returns the response from the `monitoring/health` endpoint.
- A static `authMethod:password` field.
- A `universalId` session property, which returns the corresponding value from the session when the user authenticates.




- The Page node containing the Platform Username node and Platform Password node prompts for credentials.
- The Data Store Decision node validates the username-password credentials.
- The Increment Login Count node updates the number of successful authentications in the user profile.

- The **Configuration Provider** node, imitating a Set Success Details node, adds the additional configured details to the JSON response upon successful authentication. This example uses the following configuration:

  ### *Script*

  The following next-generation script calls the `monitoring/health` endpoint and dynamically configures the node by setting the required properties. It includes the API response in the JSON response if authentication is successful along with the static field and session property:

  ```
  var requestAPI = "https://example.com/monitoring/health";

  var response = httpClient.send(requestAPI).get();

  var apiResponse = response.json();

  config = {
      "successDetails": {
          "authMethod": "password",
          "apiResponse": apiResponse
      },
      "sessionProperties": {
          "universalId": "sun.am.UniversalIdentifier"
      }
  };
  ```

  ### *Node Type*

  `Set Success Details`

  ### *Script Inputs*

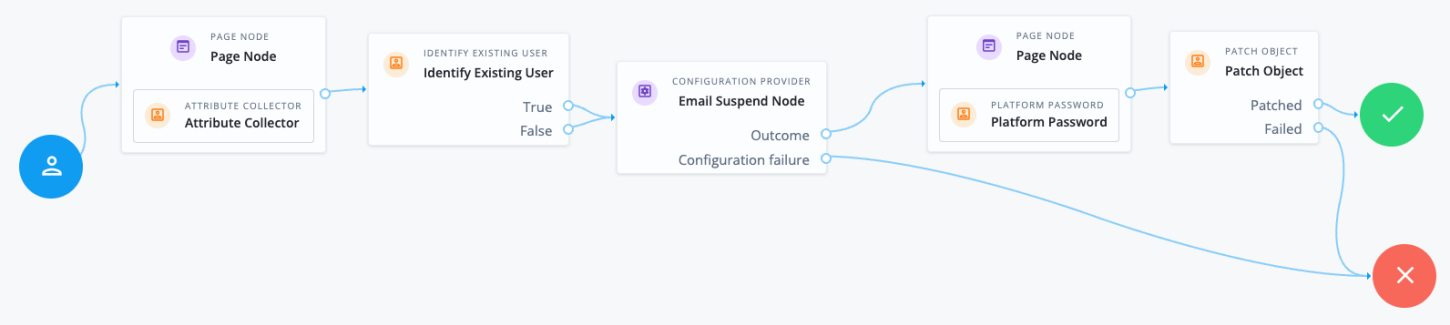
  `*`

When the user authenticates successfully using this journey, the JSON response includes the additional details you configured. For example:

```
{
    "tokenId": "AQIC5wM…TU3OQ*",
    "successUrl": "/enduser/?realm=/alpha",
    "realm": "/alpha",
    "universalId": "id=bjensen,ou=user,o=alpha,ou=services,ou=am-config",
    "apiResponse": {
       "status": "OK"
     },
    "authMethod": "password"
}
```

**Example 3: Imitate the Email Suspend node**

This example extends the default `resetPassword` journey to include the user's email address in the email suspend message. This is achieved by using the **Configuration Provider** node to imitate the Email Suspend node.




The **Configuration Provider** node in this example uses the following configuration:

### *Script*

The following next-generation script dynamically configures the node by setting the required properties. It retrieves the user's email address from the shared state and includes it in the email suspend message:

```
var attributes = nodeState.getObject("objectAttributes");

config = {
  "emailTemplateName" : "resetPassword",
  "emailAttribute" : "mail",
  "emailSuspendMessage" : {
    "en" : `An email has been sent to ${attributes.get("mail")}. Click the link in that email to reset your
password.`
  },
  "objectLookup" : true,
  "identityAttribute" : "mail"
};
```

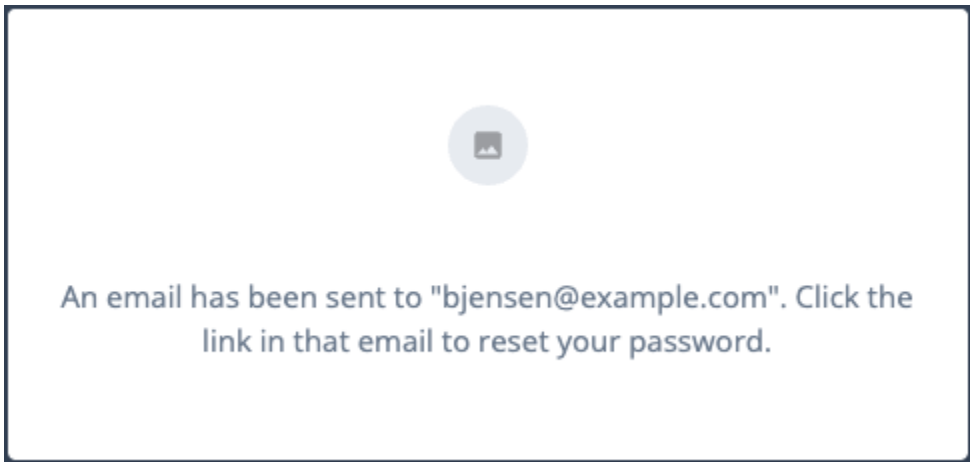
### *Node Type*

`Email Suspend Node`

### *Script Inputs*

`*`

When the journey is suspended, the updated suspend message is shown. For example:



# Email Suspend node

The **Email Suspend** node generates and sends an email, such as an address verification email based on an email template. This node relies on an email provider to send the email.

For self-managed deployments, the email service is configured in IDM.

This node generates a unique link and passes it as the `resumeURI` property for the template.

The **External Login Page URL** is used as the base of the `resumeURI` if it's been configured. Otherwise, the **Base URL Source** service is used to construct the base of the `resumeURI`.

Find more information in the corresponding documentation:

- For PingAM: Core authentication attributes > General and Configure the Base URL source service.
- For Advanced Identity Cloud: Core authentication attributes > General and Base URL Source.

The journey is suspended until the end user clicks the link in the email to resume it. Make sure the journey session is long enough for the end user to complete the journey so that it doesn't time out.

Find more information in the documentation on suspended authentication for:

- PingAM
- Advanced Identity Cloud

If you don't need to suspend a journey and wait for a reply, use the Email Template node instead.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

The **Email Suspend** node either uses the identity profile in the shared state data or looks up the user profile. In either case, the node uses any applicable profile properties to populate the email template, omitting missing values from the populated template.

If **Object Lookup** is *not* enabled for the node (default), the shared state data must hold the **Email Attribute** with the recipient's email address and any properties the email template uses.

If **Object Lookup** is enabled for the node, the shared state data must hold the profile value to match the configured **Identity Attribute**. The **Email Suspend** node uses the **Identity Attribute** to look up the profile, and its **Email Attribute** to get the recipient's email address from the profile.

## Dependencies

Before you use the **Email Suspend** node:

### Advanced Identity Cloud

- (Optional) Configure an email provider.

  By default, tenants use the built-in SMTP server.

- Prepare an email template.

  The template expressions refer to user profile properties.

  Record the email template name for use when configuring the **Email Suspend** node.

### PingAM

- Configure IDM integration in AM.
- Configure outbound email in IDM.
- Prepare an email template in IDM.

  Record the email template name for use when configuring the **Email Suspend** node.

**Tip**

You can find the email template name in the required format in the URL when you're configuring the template. For example, the **Forgotten Username** email template's name is `forgottenUsername`:

```
https://tenant-env-fqdn/?realm=alpha#/email/templates/edit/forgottenUsername
```

## Configuration

| Property | Usage |
| --- | --- |
| **Email Template Name** | The name of the email template prepared as a dependency.<br>Default: `registration` |
| **Email Attribute** | The shared state data property or profile attribute for the recipient's email address.<br>Default: `mail` |

| Property | Usage |
| --- | --- |
| **Email Suspend Message** | The localized message to display when the node suspends the journey.<br>According to OWASP authentication recommendations, the message should be the same regardless of the validity of the recipient's email address.<br>You can use plain text or HTML code in this message.<br>Default: `An email has been sent to the address you entered. Click the link in that email to proceed.` |
| **Object Lookup** | Whether to look up the managed identity profile.<br>Default: disabled |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>The node uses this when **Object Lookup** is enabled.<br>Default: `userName` |
| **Suspend Duration** [1] | (Optional) The length of time a journey session can be suspended in minutes. The time allowed for suspending the journey must be the same as or less than the maximum duration of the journey session.<br>If set, this overrides the suspend duration set in the core authentication settings.<br>Find more information in the documentation on suspended authentication for:<br>• PingAM<br>• Advanced Identity Cloud<br>Values from `1` to `2147483647` are allowed. |

[1] Currently available only in the rapid release channel.

## Outputs

This node doesn't add to the shared state data.

## Outcomes

The **Email Suspend** node has a single outcome path.

Evaluation continues when the end user clicks the link in the email to resume the flow.

## Errors

This node doesn't log any error or warning messages of its own.

## Examples

The following default journeys use the **Email Suspend** node:

- `ForgottenUsername`

- `ResetPassword`
- `UpdatePassword`

### Example 1: Forgotten username

In the default journey for recovering a forgotten username, the end user enters their email address to recover their username.

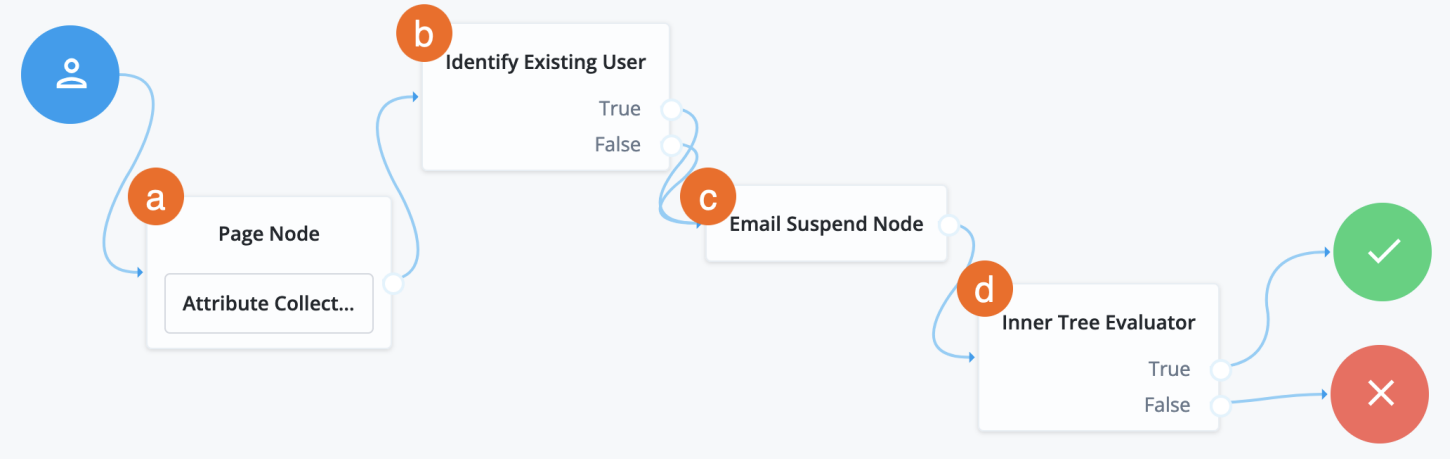
### Before you start

- Configure the email service.

  For Advanced Identity Cloud, use the built-in SMTP server.

- Optionally use the email template editor to modify the `forgottenUsername` template.

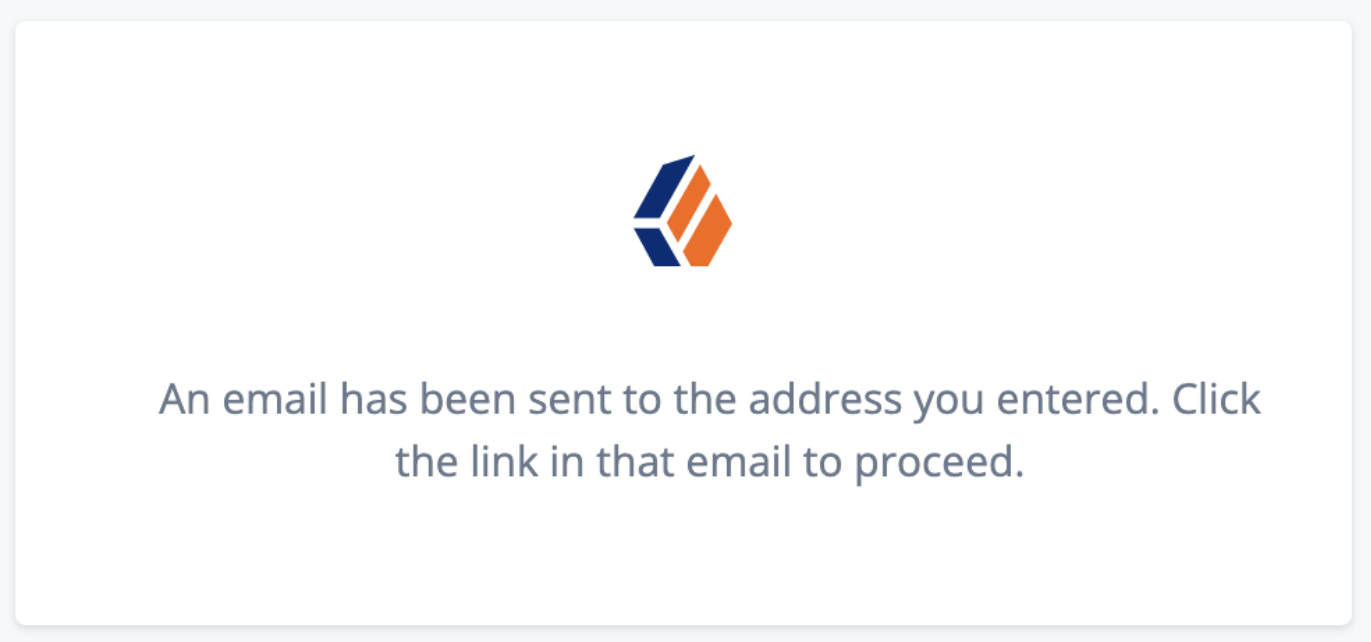
### The journey




1. The Page node with an Attribute Collector node prompts for the end user's email address.
2. The Identify Existing User node attempts to look up the username by matching the email address to the email address in an identity profile.

   The lookup fails if more than one user profile uses the same email address.

3. The **Email Suspend** node reads the user profile, generates a unique `resumeURI` link to resume the journey, and populates the `forgottenUsername` email template. On success, the node makes a request to the email service to send the email. In any case, it displays the suspend message:



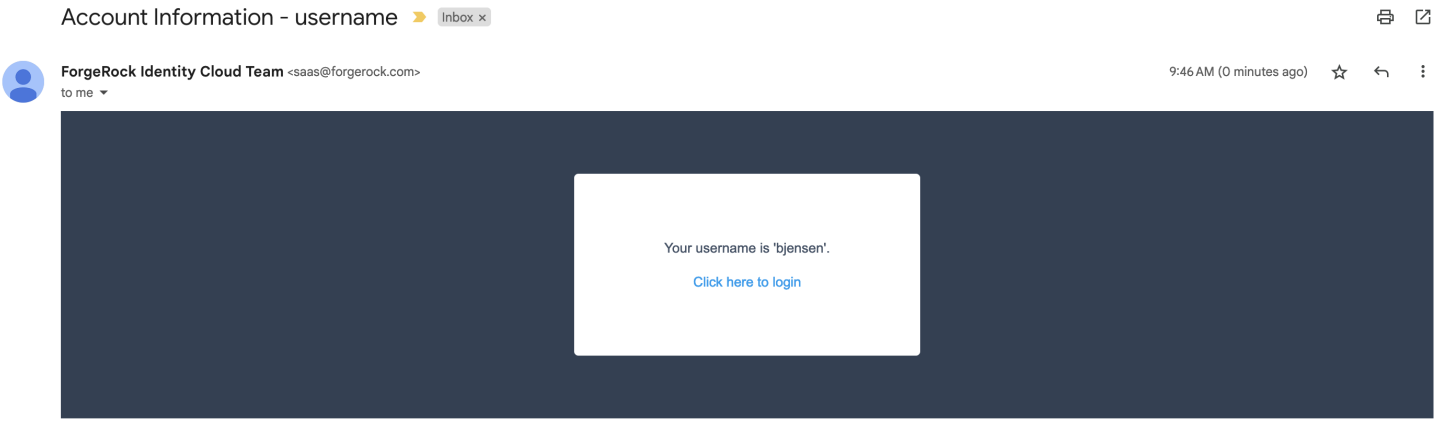

The node's settings are:

| | |
|---|---|
| **Email Template Name** | `forgottenUsername` |
| **Email Attribute** | `mail` (default) |
| **Email Suspend Message** | `An email has been sent to the address you entered. Click the link in that email to proceed.` (default) |
| **Object Lookup** | Enabled |
| **Identity Attribute** | `mail` |

4. When the end user clicks the link to resume the journey, the Inner Tree Evaluator node starts the `Login` journey.

### Try the journey

Use the journey to recover the username for an account whose email you have access to. For example, if Babs Jensen's account has your email address, the **Email Suspend** node sends you a message such as the following:



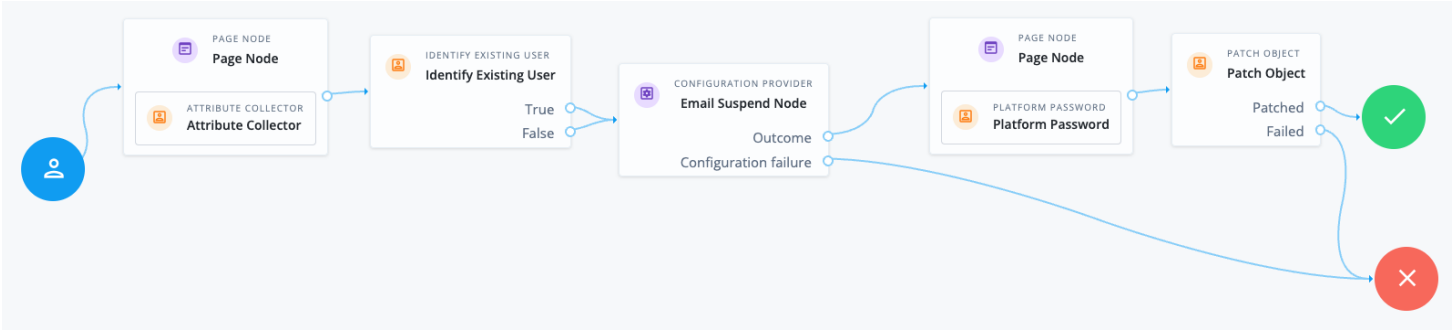

Follow the link to continue the journey and log in as Babs Jensen.

**Example 2: Registration**

For an example registration journey showing how to use the **Email Suspend** node and the Email Template node, read the **Email Template** node examples.

**Example 3: Add dynamic user data to the email suspend message**

This example uses the Configuration Provider node to imitate the **Email Suspend** node. Using the **Configuration Provider** node lets you include dynamic user data, such as their email address, in the email suspend message.




Find more information in Example 3: Imitate the Email Suspend node.

## Email Template node

The **Email Template** node generates and sends an email, such as a welcome email based on an email template. This node relies on an email provider to send the email.

For self-managed deployments, the email service is configured in IDM.

This node doesn't wait for a reply. If authentication should pause and wait for a reply to the email, use the Email Suspend node instead.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

## Inputs

The **Email Template** node uses the identity in the shared state data to get the profile, meaning the journey must have successfully authenticated or at least identified the recipient. When the journey reaches this node, the shared state must hold the profile value to match the configured **Identity Attribute**. The value can be in the `username` property or in a property having the same name as the **Identity Attribute**.

The **Email Template** node uses its **Identity Attribute** to look up the profile, and its **Email Attribute** to get the recipient's email address from the profile. In other words, the node finds the recipient's address and other properties in the *profile*, not the shared state data.

For example, if the node uses default configuration settings and Babs Jensen authenticated to the journey, the shared state includes `"username": "bjensen"`. The node looks for a profile with `"userName": "bjensen"`. It gets the recipient address from the profile's `mail` attribute, such as `"mail": "bjensen@example.com"`. The node uses any applicable profile attributes to populate the email template, omitting missing values from the populated template.

## Dependencies

Before you use the **Email Template** node, follow these steps:

### Advanced Identity Cloud

- (Optional) Configure an email provider.

  By default, tenants use the built-in SMTP server.

- Prepare an email template.

  The template expressions refer to user profile properties.

  Record the email template name for use when configuring the **Email Template** node.

PingAM

- Configure IDM integration in AM.
- Configure outbound email in IDM.
- Prepare an email template in IDM.

  Record the email template name for use when configuring the **Email Template** node. \

**Tip**

You can find the email template name in the required format in the URL when you're configuring the template. For example, the **Forgotten Username** email template's name is `forgottenUsername`:

```
https://tenant-env-fqdn/?realm=alpha#/email/templates/edit/forgottenUsername
```

## Configuration

| Property | Usage |
|---|---|
| **Email Template Name** | The name of the email template prepared as a dependency.<br>Default: `welcome` |
| **Email Attribute** | The profile attribute for the recipient's email address.<br>Default: `mail` |
| **Identity Attribute** | The attribute used to identify the managed object in the underlying identity service (PingIDM).<br>Default: `userName` |

## Outputs

This node doesn't add to the shared state data.

If the outcome is `Email Sent`, this node has sent the templated message to the recipient through the email service.

## Outcomes

### `Email Sent`

The node completed a request to send the message to the recipient.

If the message doesn't reach its destination, the problem is with the delivery, not with the node.

### `Email Not Sent`

The node failed to complete a request to send the message.

This outcome arises, for example, when one of the following happens:

- The node can't get the user profile.
- The template doesn't match the user profile.
- The specified **Email Attribute** doesn't contain an address.

According to OWASP authentication recommendations, any messages displayed in the journey should be the same in both cases.

### Errors

This node doesn't log any error or warning messages of its own.

### Examples

Use the **Email Template** node to send an email message when the journey doesn't depend on a reply. For example, send a welcome message when a user completes registration.

This example augments the default **Registration** journey and sends a welcome email.

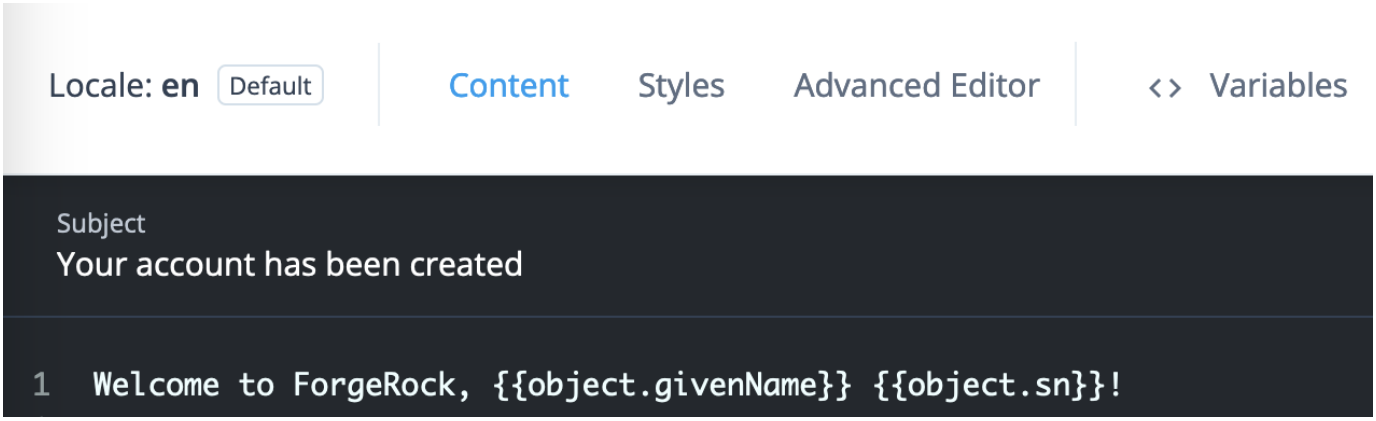
Before setting up the journey:

- Configure the email service.

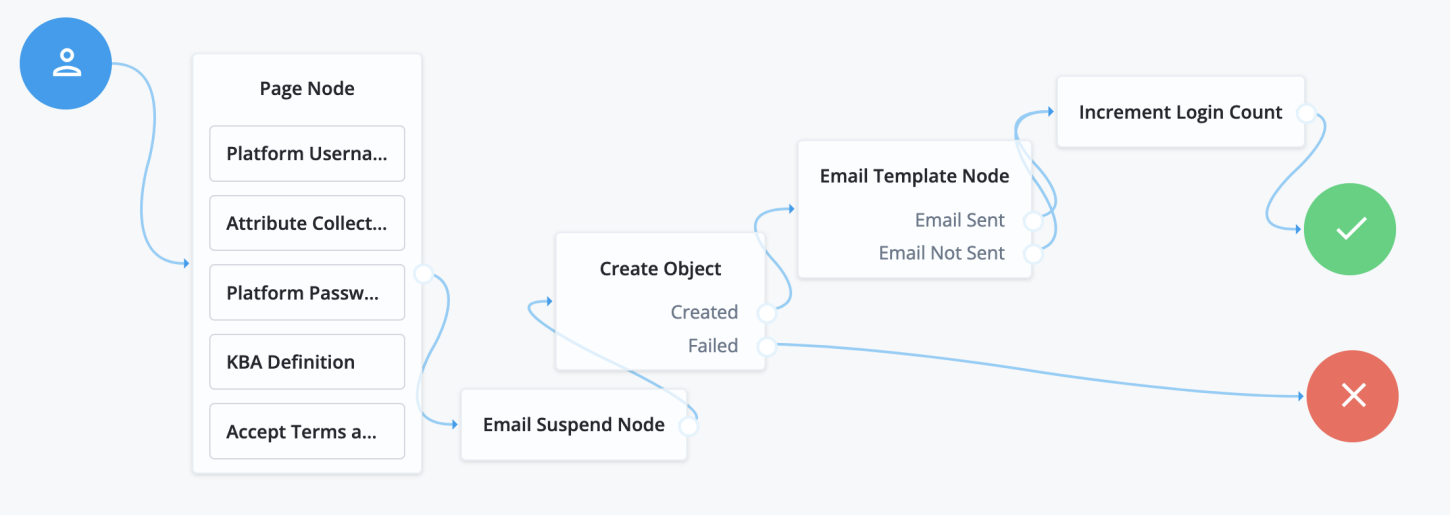
  In Advanced Identity Cloud you can use the built-in SMTP server.

- Create an email template for the **Email Template** node.

  Use the platform email template editor to duplicate the `Welcome` template and customize your copy:



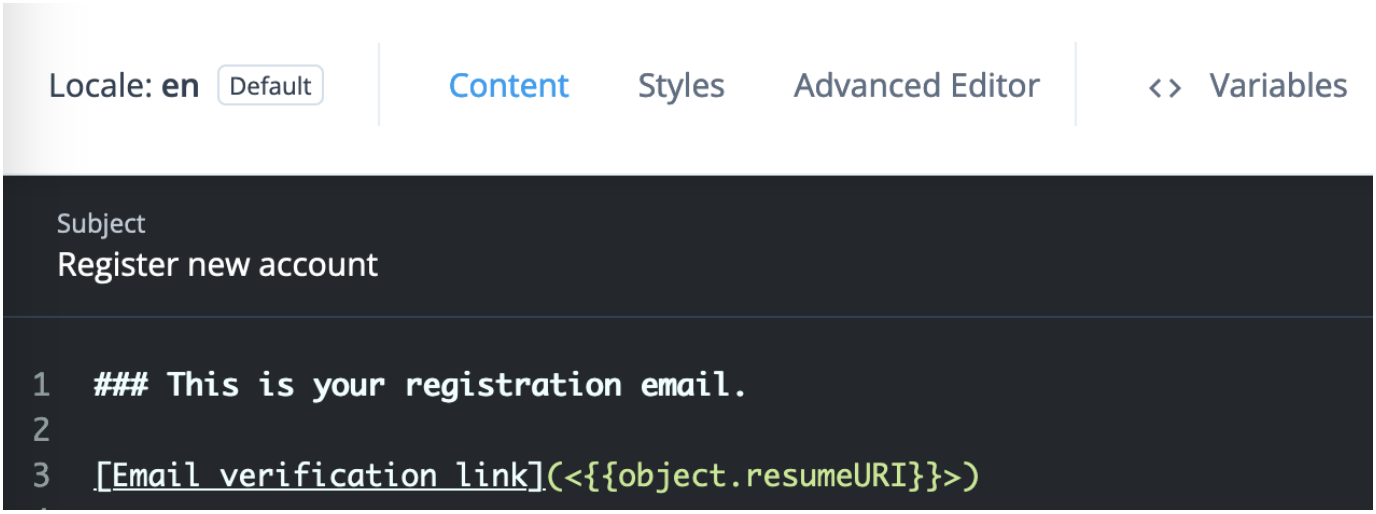

The journey is as follows:




1. The Page node prompts for the same information as the default **Registration** journey.
2. The Email Suspend node sends a message to the registered email address with a link for the user to click.

   The journey proceeds when the user clicks the link, confirming their email address.

   It has default settings and uses the default `Registration` email template:



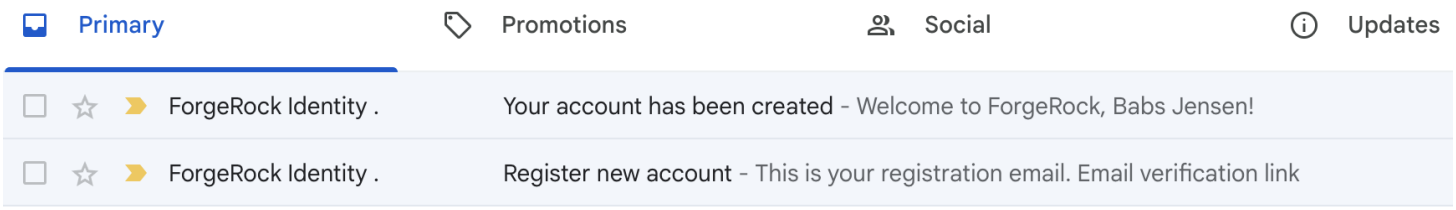

3. The Create Object node stores the newly registered user's profile.
4. The **Email Template** node reads the user profile and populates the template from profile attribute values. It makes a request to the email service to send the message.

   Its settings are:

| | |
|---|---|
| **Email Template Name** | The name of your welcome email template |
| **Email Attribute** | `mail` (default) |
| **Identity Attribute** | `userName` (default) |

5. The Increment Login Count node updates the count on successful authentication.

Use the journey to register an account for Babs Jensen with your email as the address. You receive two messages:




- The `Register new account` message has a link to click to continue the journey, confirming you can access the registered email account.
- The `Your account has been created` welcomes you on successful registration.

This demonstrates you have successfully used the **Email Template** node in a journey.

## Failure URL node

Sets the redirect URL when authentication fails.

> **Note**
>
> Specifying a failure URL overrides any `gotoOnFail` query string parameters.

Find more information on configuring the Validation Service to trust redirection URLs and on how redirection URLs are determined in the corresponding documentation for:

- PingAM
- Advanced Identity Cloud

> **Tip**
>
> The URL is also saved in the shared `nodeState` object on the `failureUrl` key.
> PingAM only: Find more information in Customize authentication trees.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
| --- | --- |
| Failure URL *(required)* | Specify the full URL to redirect to when authentication fails. |

## Flow Control node

The **Flow Control** node lets you control the authentication flow by randomly sending traffic down different paths of a journey. This means you can use the node to evaluate changes before rolling out changes to a production environment. For example, configure the node to direct a percentage of requests to a new authentication journey to observe the user experience and check for potential failures.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

None. This node doesn't read shared state data.

### Dependencies

This node has no specific dependencies.

### Configuration

| Property | Usage |
| --- | --- |
| **Path A Percentage** | The percentage of requests to send down path A. The remaining percentage follows path B. |
| **Path A Name** | The display name for the outcome of path A. |

| Property | Usage |
| --- | --- |
| **Path B Name** | The display name for the outcome of path B. |

### Outputs

None

### Outcomes

- `Path A Name` *(configured display name)*
- `Path B Name` *(configured display name)*

### Errors

This node doesn't log any errors or warnings.

### Example

The following sample journey lets an administrator phase in a new authentication path by initially routing a small number of incoming requests to the new login journey for monitoring.

The **Flow Control** node is configured with the following values:

#### *Path A Percentage*

`90`

#### *Path A Name*

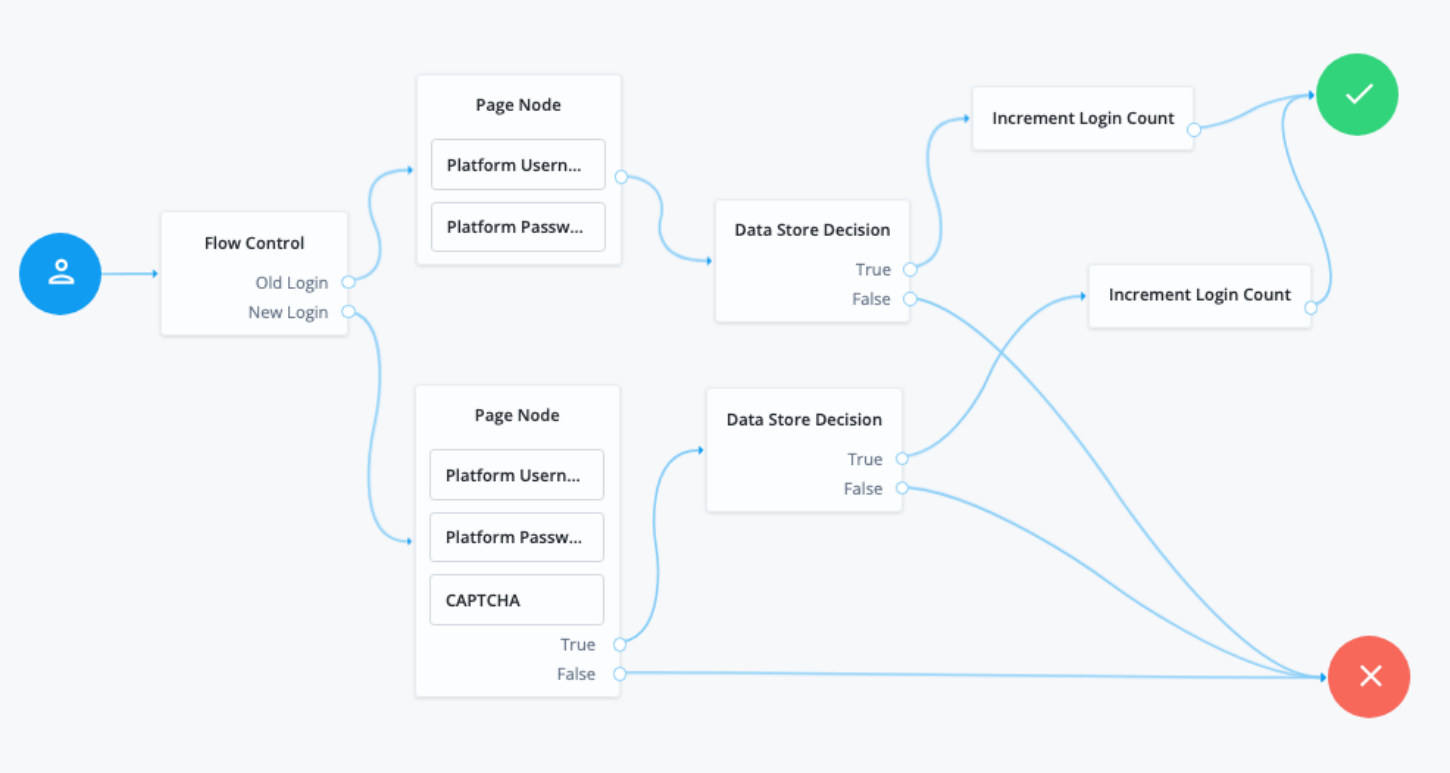
`Old Login`

#### *Path B Name*

`New Login`

The **Flow Control** node randomly assigns 90% of authentication requests to path A ( `Old Login` ) and the remaining 10% to path B ( `New Login` ).

The user follows the allocated path to complete authentication.




## Get Session Data node

The **Get Session Data** node retrieves the value of a specified key from a user's session data, and stores it in the specified key of the shared state (in scripts, the `nodeState` object).

Use this node only during session upgrade—when the user has already successfully authenticated previously and is now upgrading their session for additional access.

Find more information in the session upgrade documentation for:

- Advanced Identity Cloud
- PingAM

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads values from the user's session data.

## Dependencies

This node can complete its function only when the user has an existing session. Precede this node in the flow with a Scripted Decision node using a script to determine whether an existing session is present:

```
if (typeof existingSession !== 'undefined') {
  outcome = "hasSession";
} else {
  outcome = "noSession";
}
```

## Configuration

All the configuration properties are required:

| Property | Usage |
| --- | --- |
| **Session Data Key** | Specify the session data key whose value the node reads.<br>Default: none |
| **Shared State Key** | Specify the name of the shared node state field to hold the session data.<br>Default: none |

## Outputs

This node writes the **Session Data Key** value in the **Shared State Key** field of the shared node state.

It also writes the field and its value to the `objectAttributes` object in the shared node state.

## Outcomes

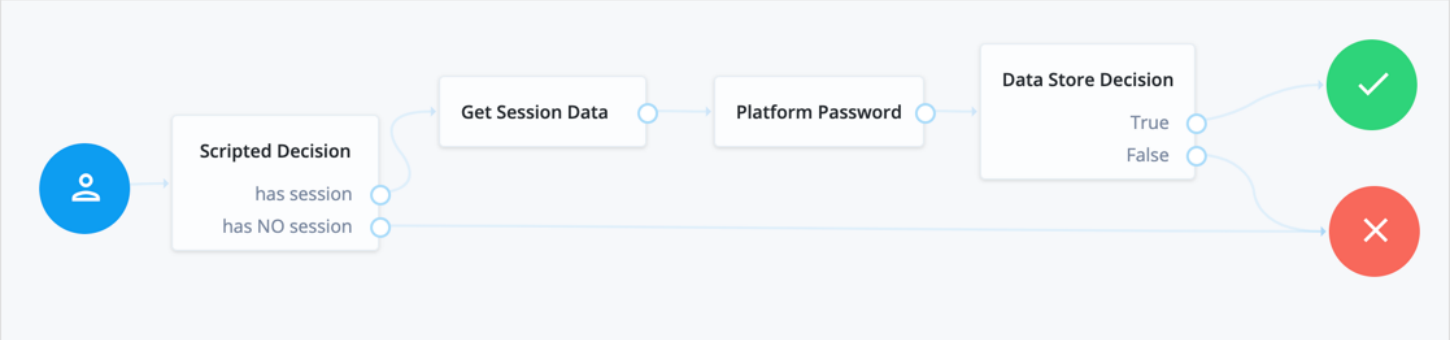
Single outcome path; on success, the **Shared State Key** in the shared node state holds the session data.

## Errors

If it cannot read the **Session Data Key** value, this node logs an `Exception occurred trying to get data (<session-data-key>) from existing session` error message.

## Example

When the user has an active session, the following example gets the username from the session, collects the password, and confirms the username-password credentials:




The following table includes example keys from an existing session with their corresponding sample values:

| Key | Sample value |
|---|---|
| `AMCtxId` | `e370cca2-02d6-41f9-a244-2b107206bd2a-122934` |
| `amlbcookie` | `01` |
| `authInstant` | `2023-04-04T09:19:05Z` |
| `AuthLevel` | `0` |
| `CharSet` | `UTF-8` |
| `clientType` | `genericHTML` |
| `FullLoginURL` | `/am/XUI/?realm=alpha#login/` |
| `Host` | `34.117.172.39` |
| `HostName` | `am.forgeblocks.com` |
| `Locale` | `en_US` |
| `Organization` | `dc=openam,dc=forgerock,dc=org` |
| `Principal` | `uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org` |
| `Principals` | `amAdmin` |
| `Service` | `ldapService` |
| `successURL` | `/openam/console` |
| `sun.am.UniversalIdentifier` | `uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org` |
| `UserId` | `amAdmin` |
| `UserProfile` | `Required` |

| Key | Sample value |
| --- | --- |
| `UserToken` | `amAdmin` |
| `webhooks` | `myWebHook` |

## Inner Tree Evaluator node

The **Inner Tree Evaluator** node lets you nest authentication journeys as children within a parent. There is no limit to the depth of nesting.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

Any information collected or set by the parent journey, such as a username or the authentication level, is available in the child journeys.

Learn more in the documentation on shared state data for:

- Advanced Identity Cloud
- PingAM

### Dependencies

None.

### Configuration

| Property | Usage |
| --- | --- |
| **Tree Name** | Select or enter the name of the authentication journey to evaluate.<br>You must set this value; there's no default. |

## Outputs

Shared node state data collected by child journeys is available to the parent when evaluation of the child is complete, but data stored in transient and secure state is not. For instance, if a child journey collects and stores the user's password in transient state, it cannot be retrieved by a node in the parent journey when evaluation continues.

Learn more in the documentation on shared state data for:

- Advanced Identity Cloud
- PingAM

## Outcomes

### `True`

Successfully reached the `Success` node of the child.

### `False`

Any other case.

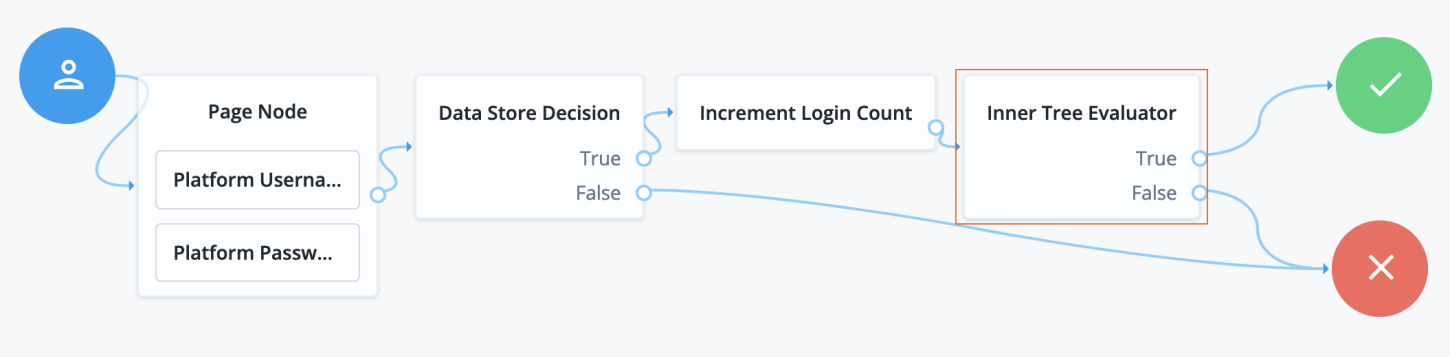
## Errors

If it cannot get the shared node state from the child, this node logs an `Exception when gathering inner tree inputs` message.

If the **Tree Name** doesn't match an existing journey, this node throws an exception with a `Configured tree does not exist: <tree-name>` message.

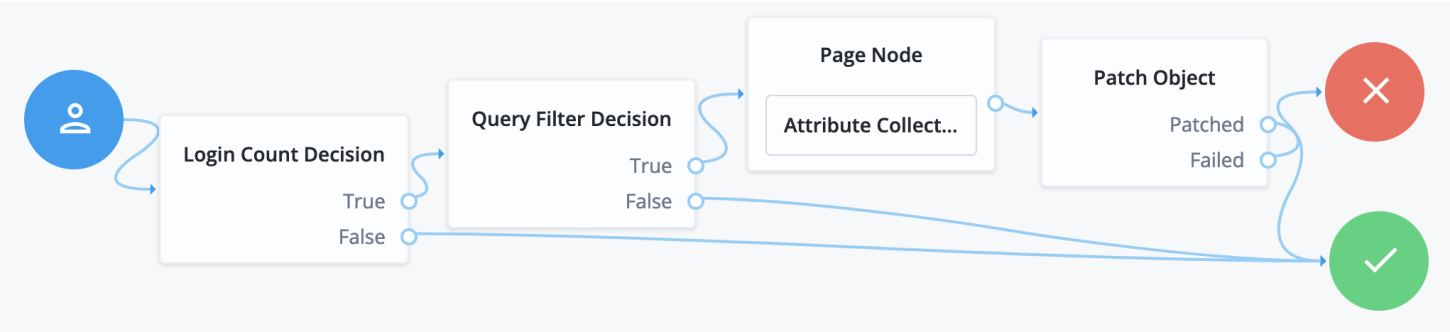
## Example

The following journey uses an Inner Tree Evaluator node for progressive profiling:




- The Page node presents a page with input fields to prompt for the username and password.
  - The Platform Username node collects and injects the `userName` into the shared node state.
  - The Platform Password node collects and injects the `password` into the shared node state.
- The Data Store Decision node uses the username and password to determine whether authentication is successful.

- The Increment Login Count node updates the login count on successful authentication.
- The Inner Tree Evaluator node (outlined) invokes a nested journey:




- The Login Count Decision node triggers the rest of the journey depending on the login count and its settings.
- The Query Filter Decision node determines whether managed object profile fields are still missing.
- The Attribute Collector node in the Page node requests additional input for the profile.
- The Patch Object node stores the additional input in the managed object profile.

## Message node

The **Message** node presents a custom, localized message to the user with customizable, localized positive and negative answer buttons the user must click to proceed.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads preferred locales from the incoming request context.

It doesn't read from the shared node state.

This node has no required predecessor nodes.

### Dependencies

None.

## Configuration

| Property | Usage |
| --- | --- |
| **Message** | Add a custom, localized message per locale:<br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[1]<br>The incoming HTTP request can include an `Accept-Language` header indicating the user's preferred locales. If the incoming HTTP request doesn't include the header or the preferred locales don't match any configured locales, the node uses default settings. It uses the **Realms > *Realm Name* > Authentication > Settings > General > Default Authentication Locale** setting from the AM admin UI.<br>In PingAM and Ping Identity Platform deployments, if there's no default authentication locale, the node uses **Deployment > Servers > *Server Name* > General > System > Default Locale**.<br>3. In the **Value** field, enter the message to display to the user.<br>If you leave this blank, the message node displays a localized version of `Default message` to the user.<br><br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |
| **Positive answer** | Add the text per locale for the positive answer button that triggers a **True** outcome:<br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[1]<br>If the incoming HTTP request doesn't include the header or the preferred locales don't match any configured locales, the node uses the first text in the list.<br>3. In the **Value** field, enter the text to display to the user.<br>If you leave this blank, the button displays a localized version of `Yes`.<br><br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |
| **Negative answer** | Add the text per locale for the negative answer button that triggers a **False** outcome:<br>1. Click **Add**.<br>2. In the **Key** field, enter the locale.[1]<br>If the incoming HTTP request doesn't include the header or the preferred locales don't match any configured locales, the node uses the first text in the list.<br>3. In the **Value** field, enter the text to display to the user.<br>If you leave this blank, the button displays a localized version of `No`.<br><br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |

| Property | Usage |
| --- | --- |
| **Shared State Property Name** | The name of the node state property.<br>If set, the node adds the property to shared node state, setting its value to the numeric value of the outcome:<br>***0***<br>The user clicked the positive answer button.<br>***1***<br>The user clicked the negative answer button.<br>For example, if you set this to `messageNodeOutcome` and the user clicks the positive answer button, the node adds `"messageNodeOutcome": 0` as a shared node state property. |
| **Only Positive Answer** | When enabled, the node displays only the positive answer button.<br>**Note**<br>This property is not visible in the AM admin UI. It requires the Ping Identity Platform admin UI.<br>**Note**<br>This property only displays when the node is placed within a Page node. |
| **Show buttons as links** | When enabled, the node shows the buttons as links instead.<br>**Note**<br>This property is not visible in the AM admin UI. It requires the Ping Identity Platform admin UI.<br>**Note**<br>This property only displays when the node is placed within a Page node. |

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

## Outputs

When the **Shared State Property Name** setting has a value, the node adds the property to the shared node state. The property's value is the numeric value of the outcome:

**`0`**

The user clicked the positive answer button.

**`1`**

The user clicked the negative answer button.

## Outcomes

Returns a boolean outcome:

### *True*

The user clicked the positive answer button.

### *False*

The user clicked the negative answer button.

## Errors

This node doesn't cause authentication to fail unless you connect one of the outcomes to a Failure node.

If the message or answer button settings specify a locale Java doesn't support, the node throws a configuration exception with an `Invalid locale provided` message. If this happens, fix the locale setting.

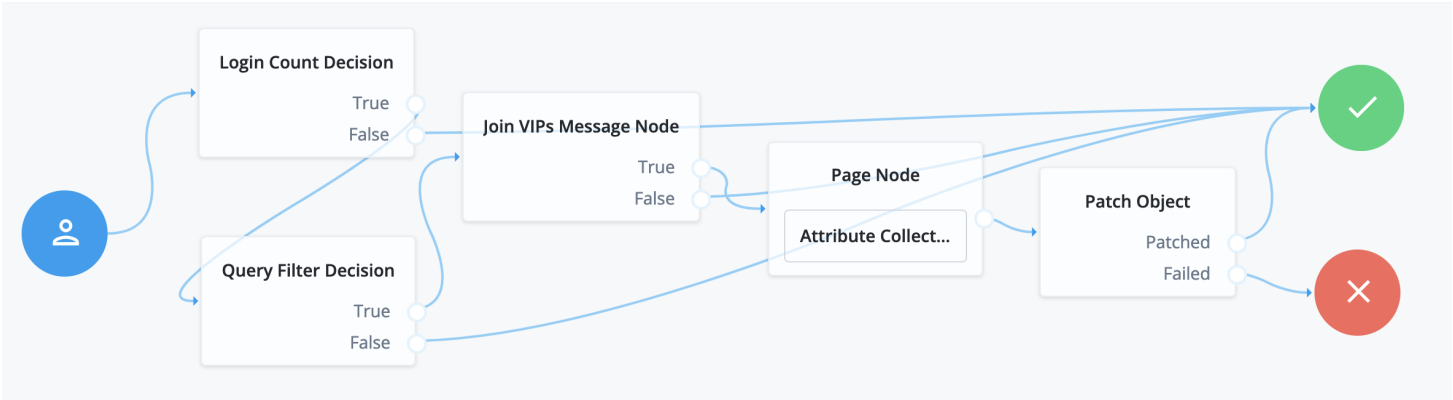
If this node encounters an internal configuration issue, it logs a warning message `Error attempting to retrieve the realm/global default locale`. If the warning persists, contact Ping Identity Support.

## Examples

Use a **Message** node to:

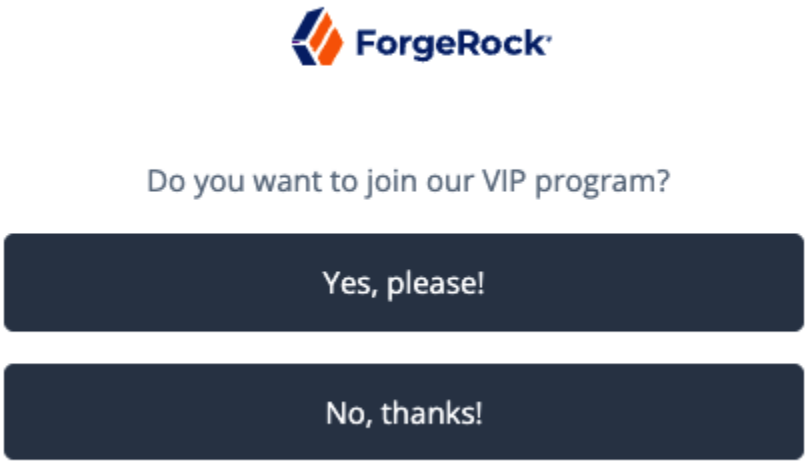
- Communicate an important message for the user to acknowledge.
- Ask a question with a yes/no answer.

The following journey uses the `Join VIPs Message Node` to prompt the user to join the VIP program:




- The Login Count Decision node triggers the Query Filter Decision node after every tenth authentication.
- The Query Filter Decision node queries the user profile to determine whether they have signed up for the VIP program.

  If the user hasn't signed up yet, the **True** outcome triggers the `Join VIPs Message Node`, which prompts the user to join the program:




Node property settings:

### *Message*

`en-gb` ; `Do you want to join our VIP program?`

### *Positive answer*

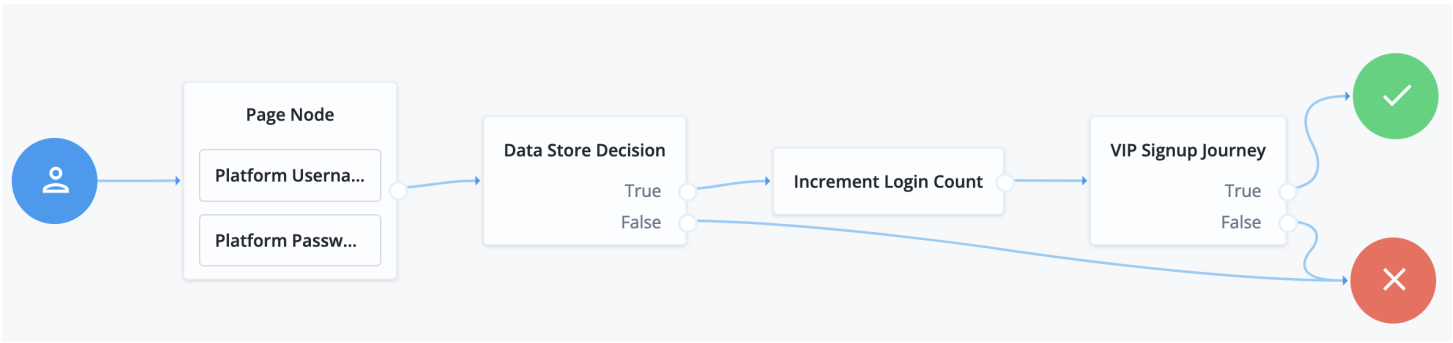
`en-gb` ; `Yes, please!`

### *Negative answer*

`en-gb` ; `No, thanks!`

- If the user clicks **Yes, please!** the Page node with an embedded Attribute Collector node collects opt-in choices to store in user profile attributes.
- The Patch Object node updates the user profile with the attributes collected.

Call the journey using an Inner Tree Evaluator node from another authentication journey directly after an Increment Login Count node note:




The `VIP Signup Journey` uses the login count from the Increment Login Count node in the Login Count Decision node to decide whether to prompt the user to join the VIP program.

# Meter node

Increments a specified metric key each time evaluation passes through the node.

PingAM only: Find information on the `Meter` metric type in Monitoring metric types and Monitor AM instances.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

Single outcome path.

## Properties

| Property | Usage |
|---|---|
| Metric Key *(required)* | Specify the name of a metric to increment when evaluation passes through the node.<br>PingAM only: Example: `authentication.success`. Find a list of available metrics, refer to Monitoring metrics. |

# Page node

The **Page** node lets you combine multiple nodes that request input onto a single page for display to the user.

Drag and drop nodes onto the Page node to combine them. Only add nodes that use callbacks to request input. Don't add other nodes, such as the Data Store Decision node and the Push Sender node to this node.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |

| Product | Available? |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

The inputs are determined by the collective inputs of the contained nodes.

## Dependencies

None.

## Configuration

| Property | Usage |
| --- | --- |
| **Page Header** | Optional. A localized title for the Page node and the nodes contained within it. Use this when components of an authentication journey need a title. For example, dividing a registration flow into labeled sections. |
| **Page Description** | Optional. A localized description for the Page node and the nodes contained within it. Use this when you need additional descriptive text in an authentication journey. Advanced Identity Cloud and Ping Identity Platform deployments only: You can use HTML code to format the description. |
| **Stage** | Optional. A stage name to pass to the client to aid in rendering. NOTE: You can't configure a **Stage** for this node if you configure a **Page Footer**, **Theme** or **Submit Button Text**. These properties overwrite the value of the **Stage** property. |
| **Submit Button Text** | Optional. Use the **Key** and **Value** fields to set the text of the **Submit** button.<br>**Note**<br>This property is not visible in the AM admin UI. It requires the Ping Identity Platform admin UI. |
| **Page Footer** | Optional. A localized footer for the page node and the nodes contained within it. Use this when you need additional descriptive text in an authentication journey. Advanced Identity Cloud and Ping Identity Platform deployments only: You can use HTML code to format the description.<br>**Note**<br>This property is not visible in the AM admin UI. It requires the Ping Identity Platform admin UI. |

| Property | Usage |
| --- | --- |
| **Theme** | Optional. If using hosted pages, specify a theme to override this journey's UI theme.<br><br>**Note**<br>This property is not visible in the AM admin UI. It requires the Ping Identity Platform admin UI. |

**Note**

This node's optional properties are passed in the response, but a self-hosted or custom UI must support these properties to make them visible to the end user.

### Outputs

The outputs are determined by the collective outputs of the contained nodes.

### Outcomes

The outcomes are determined by the last node in the Page node. Only the last node in the page can have more than one outcome path.

### Errors

This node can log the following error messages:

| Message | Notes |
| --- | --- |
| `Failed to collect inputs of contained node: <node-name>` | The <node-name> could not retrieve required properties from the shared node state |
| `Failed to collect outputs of contained node: <node-name>` | The <node-name> could not retrieve required properties to include in the shared node state |
| `Could not find the identity based on the information available on context` | Failed to find the account profile with this `username` in this `realm` |
| `An error occurred when trying to lock out the user account` | Failed to update the account status; applies when locking and unlocking the account |

This node can throw exceptions with the following messages during operation:

| Message | Notes |
| --- | --- |
| `This page has no nodes in it, so cannot proceed` | A Page node must contain at least one other node |

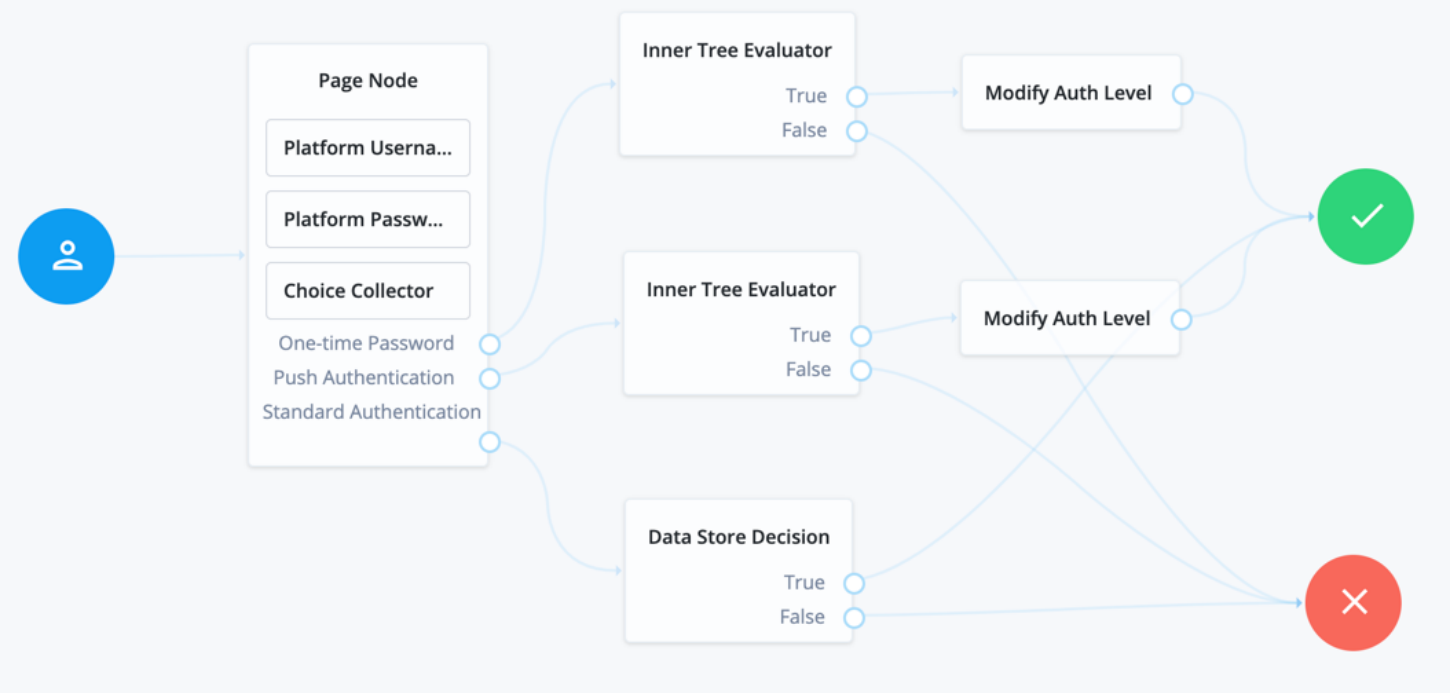
| Message | Notes |
| --- | --- |
| `No outcome and only metadata callbacks found` | Failed to get to an outcome while processing the contained nodes |
| `Node properties cannot be fetched` | Failed to access the properties of a contained node |

This node can throw exceptions with the following messages when saving the journey:

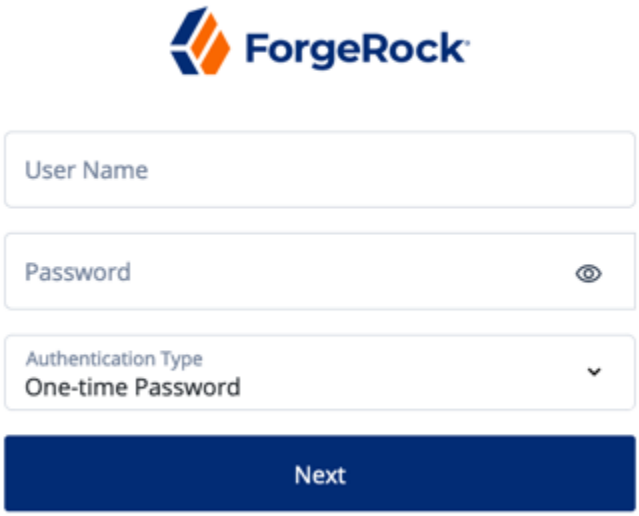
| Message | Notes |
| --- | --- |
| `Illegal child node type: <node-type>` | A Page node can't contain a <node-type> |
| `Node does not have any outcomes: <node-type>` | The contained nodes must have at least a single outcome path |
| `Only the last node in a page can have more than one outcome` | Consider rearranging the contained nodes |
| `Node does not exist: <node-id>` | Use the journey editor to fix the problem |
| `Could not load child node: <node>` | Use the journey editor to fix the problem |
| `Could not obtain outcomes for node: <node>` | Use the journey editor to fix the problem |

## Example

The following example uses a Page node containing a Platform Username node, Platform Password node, and Choice Collector node:




The flow prompts the user for all input on a single page:




## Polling Wait node

Pauses authentication progress for a specified number of seconds, for example, to wait for a response to a one-time passcode email or push notification.

Requests made during the wait period are sent a `PollingWaitCallback` callback and an authentication ID. For example, the following callback indicates a wait time of 10 seconds:

```
{
    "authId": "eyJ0eXAiOiJK...u4WvZmiI",
    "callbacks": [
        {
            "type": "PollingWaitCallback",
            "output": [
                {
                    "name": "waitTime",
                    "value": "10000"
                },
                {
                    "name": "message",
                    "value": "Waiting for response..."
                }
            ]
        }
    ]
}
```

The client must wait 10 seconds before returning the callback data, including the `authId`:

```
$ curl \
--request POST \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "Content-Type: application/json" \
--data '{
  "authId": "eyJ0eXAiOiJK…u4WvZmiI",
  "callbacks": [
      {
          "type": "PollingWaitCallback",
          "output": [
              {
                  "name": "waitTime",
                  "value": "10000"
              },
              {
                  "name": "message",
                  "value": "Waiting for response…"
              }
          ]
      }
  ]
}' \
'https://am.example.com:8443/am/json/realms/root/realms/alpha/authenticate?
authIndexType=service&authIndexValue=Example'
```

The end user UI automatically waits for the required amount of time and resubmits the page to continue evaluation. The message displayed during the wait is configurable with the **Waiting Message** property.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

- `Done`
- `Exited` (configurable)
- `Spam` (configurable)

Evaluation continues along the `Done` outcome path when the next request is received after the wait time has passed.

Enabling Spam detection adds a `Spam` outcome path to the node. Evaluation continues along the `Spam` outcome path if more than the specified number of requests are received during the wait time.

Enabling the user to exit without waiting adds an `Exited` outcome path to the node. Evaluation continues along the `Exited` outcome path if the user clicks the button that appears when the option is enabled. The message displayed on the exit button is configurable by using the **Exit Message** property.

## Properties

| Property | Usage |
|---|---|
| Seconds To Wait | Specify the number of seconds to pause authentication.<br>Default: `8` |
| Enable Spam Detection | Specify whether to track the number of responses received during the wait time, and continue evaluation along the `Spam` outcome path if the number specified in the **Spam Tolerance** property is exceeded.<br>Default: Disabled |
| Spam Tolerance | Specify the number of responses to allow during the wait time before continuing evaluation along the `Spam` outcome path. This property only applies if spam detection is enabled.<br>Default: `3` |

| Property | Usage |
| --- | --- |
| Waiting Message | The optional message to display to the user.<br>Provide the message in multiple languages by specifying the locale in the **KEY** field. For example, `en-gb`.[(1)]<br>The locale selected for display is based on the user's locale settings in their browser.<br>Leave blank to use the default message.[(2)] |
| Exitable | Whether the user can exit the node during the wait period.<br>Enabling this option adds a button with a configurable message to the page. Clicking the button causes evaluation to continue along the `Exited` outcome path.<br>Default: Disabled |
| Exit Message | The optional message to display to the user on the button used to exit the node before the wait period has elapsed. For example, `Cancel` or `Lost phone? Use Recovery Code`. This property only applies if the **Exitable** property is enabled.<br>Provide the message in multiple languages by specifying the locale in the **KEY** field. For example, `en-gb`.[(1)]<br>The locale selected for display is based on the user's locale settings in their browser.<br>Leave blank to use the default message.[(2)] |

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

(2) PingAM only: Learn more about customizing and translating default messages in Internationalize nodes.

## Query Parameter node

The **Query Parameter** node lets you insert query parameter values from a journey URL into configurable node state properties. This lets you customize journeys based on the query parameter values.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node takes its inputs from the journey URL and maps each query parameter to a property in the node state. For multi-valued parameters, the node delimits the parameters by a comma ( `,` ) symbol. In this case, the value assigned to the node state property is a comma-delimited list of items.

The node doesn't read input from the state.

The node has no required predecessor nodes.

## Dependencies

None.

## Configuration

| Property | Usage |
| --- | --- |
| **Allowed query parameters** | List the query parameters the node can obtain from the URL and map them to a property in the node state:<br>1. Click **Add**.<br>2. In the **Key** field, enter the query parameter name.<br>3. In the **Value** field, enter the node state property that will hold the value of this query parameter.<br>This field sets an allow list of parameters the node can pull into the node state. Exercise caution when you create this list to avoid injecting harmful data into the node state.<br>If a query parameter in the URL isn't in this list, the node ignores it.<br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |
| **Allowed query parameters to be delimited** | Specify the allowed query parameters that can take multiple values and whose values you want to store in the node state in a comma-delimited list. For example, `["yellow", "green", "red"]`.<br>Enter the query parameter name in the **Add value** field and click **Add**.<br>If you don't delimit the values of a multi-valued query parameter, the node stores the values in the node state as a single string value. For example, `["yellow, green, red"]`.<br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |

## Outputs

- If the **Allowed query parameters** setting has one or more values, the node adds the values of the listed URL parameters to the corresponding properties in the node state.
- If the **Allowed query parameters** setting has a value but that query parameter isn't present in the URL, the node sets an empty list ( `[]` ) in the corresponding node state property.

- If an allowed query parameter is also listed in the **Allowed query parameters to be delimited**, the node sets the values of the listed URL parameter in the node state as a comma-delimited list.

**Important**

- The node URL-decodes query parameters before putting them into the node state.
  If a query parameter includes `%2C`, the node puts this value into the node state as a comma ( `,` ). If the query parameter is included in the **Allowed query parameters to be delimited** the node interprets `%2C` as a comma delimiter.
  The node decodes the plus symbol ( `+` ) as a *space*. If you need a `+` in a query parameter value, encode it as `%2B` in the URL.
- Values stored in the node state can override values in the authentication journey.
  Take special care when you configure this node so that you don't unintentionally override parameters such as usernames and passwords.
  The output of this node isn't under the control of the node itself. Encode sensitive values appropriately, either at node output or before the values are used later in the journey.

## Outcomes

Single outcome that passes an updated node state to the next node in the journey.

## Errors

- `No parameters configured - this node is redundant`

  The node logs this error if you include it in a journey but don't configure any **Allowed query parameters**.

- `Cannot delimit parameter if it is not configured as a parameter to be stored in node state`

  The node logs this error if you add a parameter to the list of **Allowed query parameters to be delimited** but not to the list of **Allowed query parameters**.

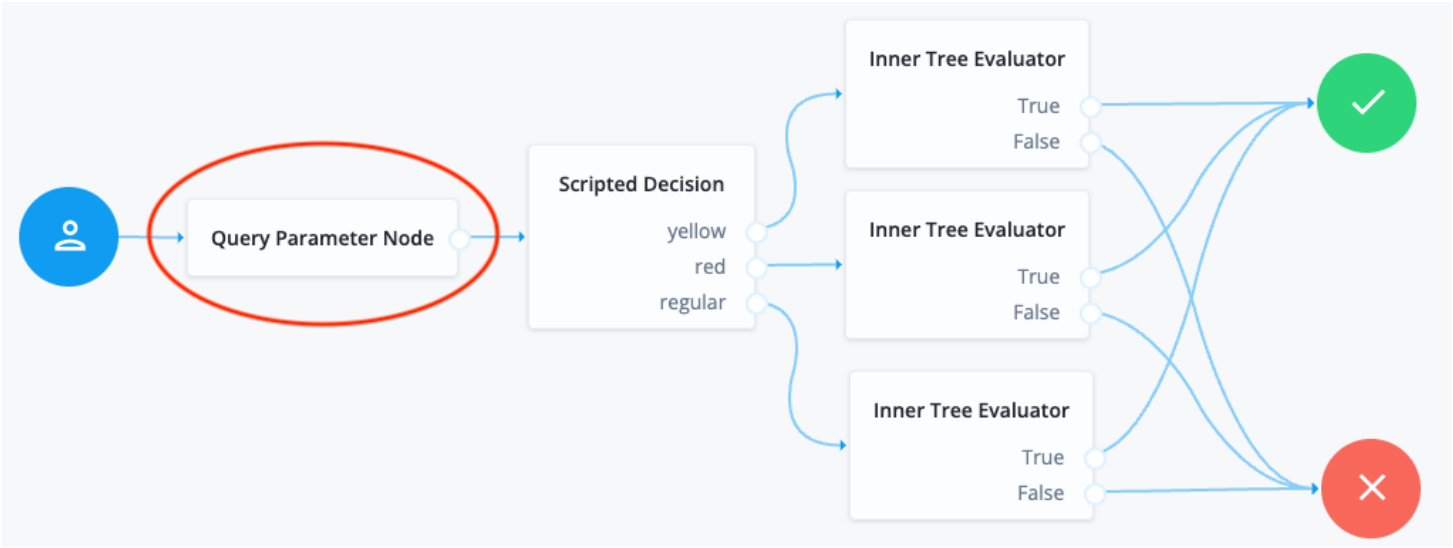
## Examples

Use the Query Parameter node to customize a journey based on query parameters in the URL. The following scenarios illustrate how this node could be used:

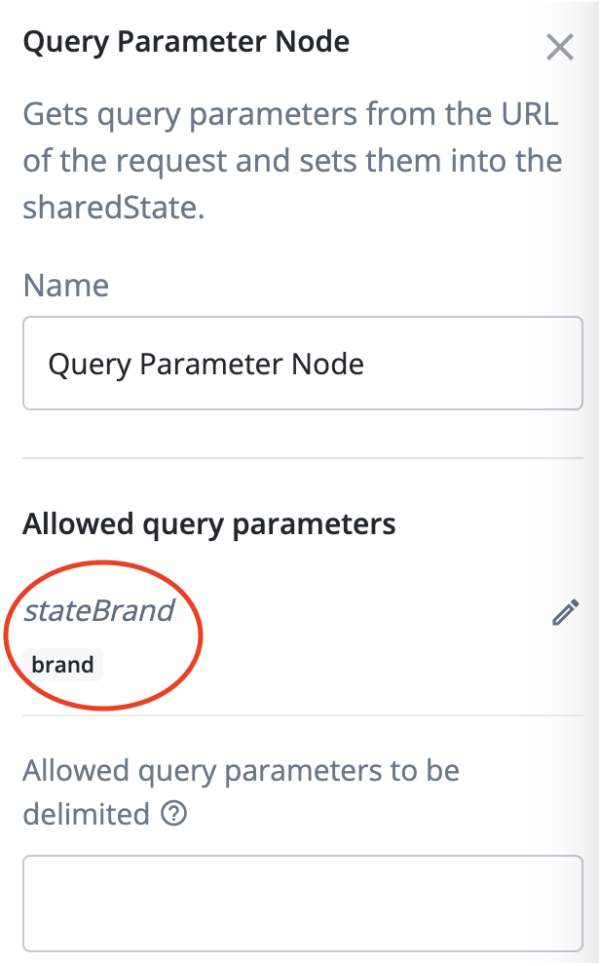
### Customized branding

An organization has several brands that use the same journey. Use this node to customize the brand the user sees, based on the query parameters.

Consider the following authentication journey:




1. The configuration of the Query Parameter node maps the `brand` query parameter to a property in the node state named `stateBrand`



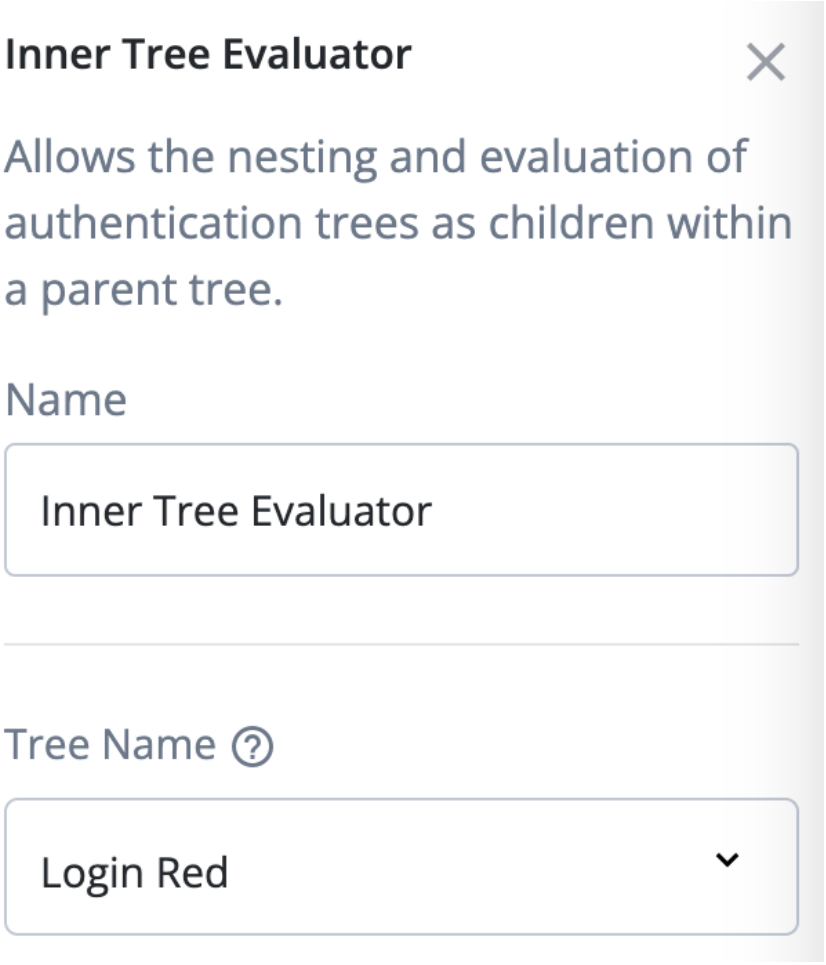

2. The user accesses the journey at a URL that can be one of the following:
   - `https://<tenant-env-fqdn>/am/XUI/?realm=alpha&authIndexType=service&authIndexValue=Login`

- `https://<tenant-env-fqdn>/am/XUI/?realm=alpha&authIndexType=service&authIndexValue=Login&brand=yellow`
- `https://<tenant-env-fqdn>/am/XUI/?realm=alpha&authIndexType=service&authIndexValue=Login&brand=red`

3. The Query Parameter node obtains the value of `brand` from the URL and sets that value in the `stateBrand` property in the node state. For example, `stateBrand=yellow`
4. The journey progresses to the scripted decision node that includes the following script:

```
var brand = JSON.parse(nodeState.get('stateBrand'));

if (brand.indexOf("yellow") >= 0) {
  outcome = "yellow";
} else if (brand.indexOf("red") >= 0) {
  outcome = "red";
} else {
  outcome = "regular";
}
```

5. The script routes the journey to one of three outcomes; `yellow`, `red`, or `regular`, depending on the value of the `stateBrand` property.
6. The outcomes direct the user to a custom branded Login journey configured in an Inner Tree Evaluator node. For example:



7. Each Inner Tree Evaluator node routes the end user to a login journey that uses a custom brand.

**Redirection from an external system**

An external system redirects a user to this authentication journey. The external system must share information about the user with the journey. Use this node to obtain the relevant query parameters and inform the journey of their values.

## Register Logout Webhook node

Registers the specified webhook to trigger when a user's session ends. The webhook triggers when a user explicitly logs out or the maximum idle time or expiry time of the session is reached.

The webhook is only registered if evaluation passes through this node. You can register multiple webhooks during the authentication process, but they must be unique.

For more information in the documentation on webhooks for:

- PingAM
- Advanced Identity Cloud

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Webhook name | Specify the name of the webhook to register. |

## Remove Session Properties node

Removes properties from the session. The session properties may have been set by a Set Session Properties node elsewhere in the flow.

If a specified key is not found in the list of session properties it is added to the session upon successful authentication, no error is thrown, and evaluation continues along the single outcome path.

If a specified key is found, the evaluation continues along the single outcome path after setting the value of the property to `null`.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Property Names *(required)* | Enter one or more key names of properties to remove from the session. |

## Request Header node

The **Request Header** node lets you insert values from request headers into configurable node state properties. This lets you customize journeys based on the request header values.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node takes its inputs from one or more request headers and maps these values to properties in the node state. For multi-valued headers, the node delimits headers with a comma ( `,` ) and assigns the value to the node state property as a comma-delimited list.

The node doesn't read input from the state.

The node has no required predecessor nodes.

## Dependencies

None.

## Configuration

| Property | Usage |
| --- | --- |
| **Allowed headers** | List the header names the node can obtain from the URL and map them to a property in the node state:<br>1. Click **Add**.<br>2. In the **Key** field, enter the query parameter name.<br>3. In the **Value** field, enter the node state property that will hold the value of this request header.<br>This field sets an allow list of headers the node can pull into the node state. Exercise caution when you create this list to avoid injecting harmful data into the node state.<br>If a provided request header isn't in this list, the node ignores it.<br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |
| **Allowed headers to be delimited** | The allowed headers that can take multiple values and whose values you want to store in the node state in a comma-delimited list. For example, `["yellow", "green", "red"]`.<br>Enter the header name in the **Add value** field and click **Add**.<br>If you don't delimit the values of a multi-valued header, the node stores the values in the node state as a single string value. For example, `["yellow, green, red"]` .<br>To edit an entry, click the **Pencil** icon (✎).<br>To remove an entry, click the **Delete** icon (🗑). |

## Outputs

- If the **Allowed headers** setting has one or more values, the node adds the values of the listed request headers to the corresponding properties in the node state.
- If the **Allowed headers to be delimited** setting has a value but that header isn't provided in the request, the node sets an empty list ( `[]` ) in the corresponding node state property.

- If an allowed header is also listed in the **Allowed headers to be delimited**, the node sets the values of that header in the node state as a comma-delimited list.
- The node performs no decoding or sanitization on the header value. It simply passes the value into the node state as a string.

> **Important**
>
> Values stored in the node state can override values in the authentication journey.
> Take special care when you configure this node so that you don't unintentionally override parameters such as usernames and passwords.
> The output of this node isn't under the control of the node itself. Encode sensitive values appropriately, either at node output or before the values are used later in the journey.

### Outcomes

Single outcome that passes an updated node state to the next node in the journey.

### Errors

- `No headers configured — this node is redundant`

  The node logs this error if you include it in a journey but don't configure any **Allowed headers**.

- `Cannot delimit header if it is not configured as a header to be stored in node state`

  The node logs this error if you add a header to the list of **Allowed headers to be delimited** but not to the list of **Allowed headers**.

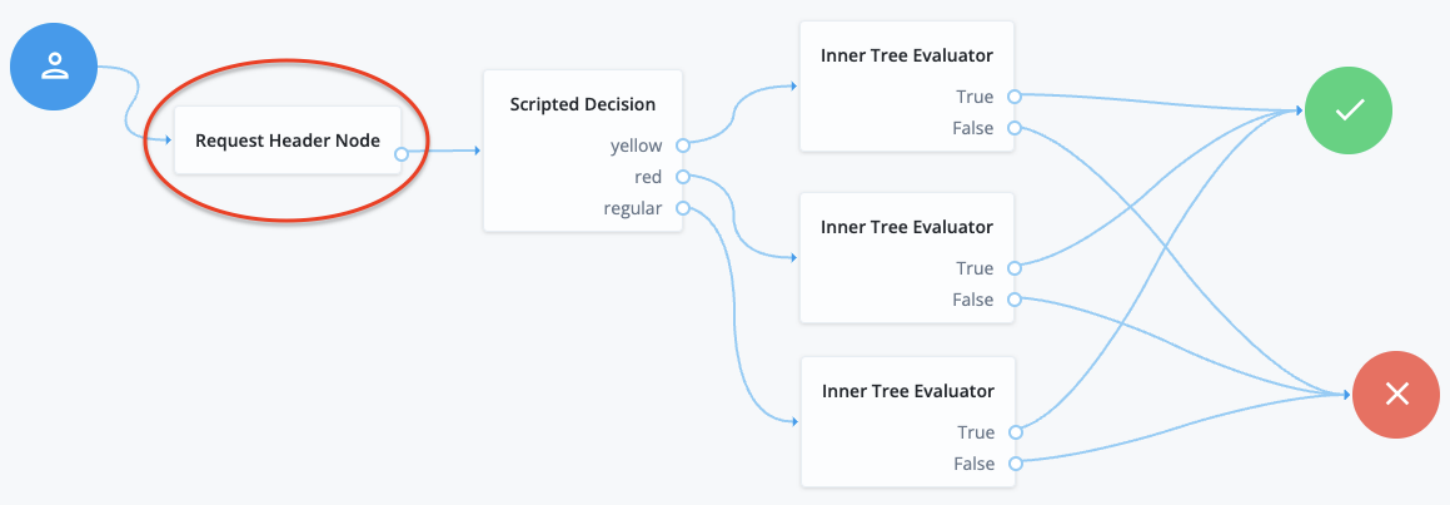
### Examples

Use the Request Header node to customize a journey based on the values of specific request headers. The following scenarios illustrate how this node could be used:

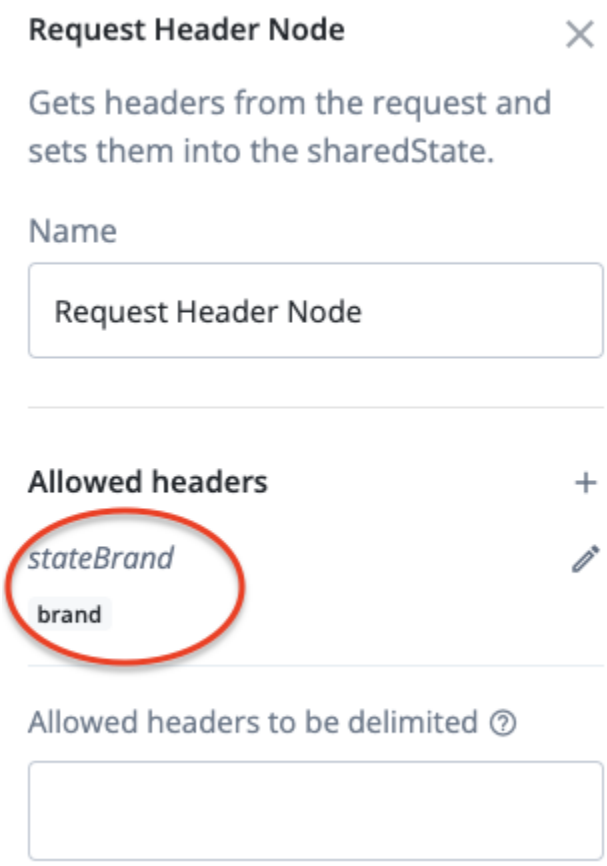
#### Customized branding

An organization has several brands that use the same journey. Use this node to customize the brand the user sees, based on the request header.

Consider the following authentication journey:




1. The configuration of the Request Header node maps the `brand` header to a property in the node state named `stateBrand`

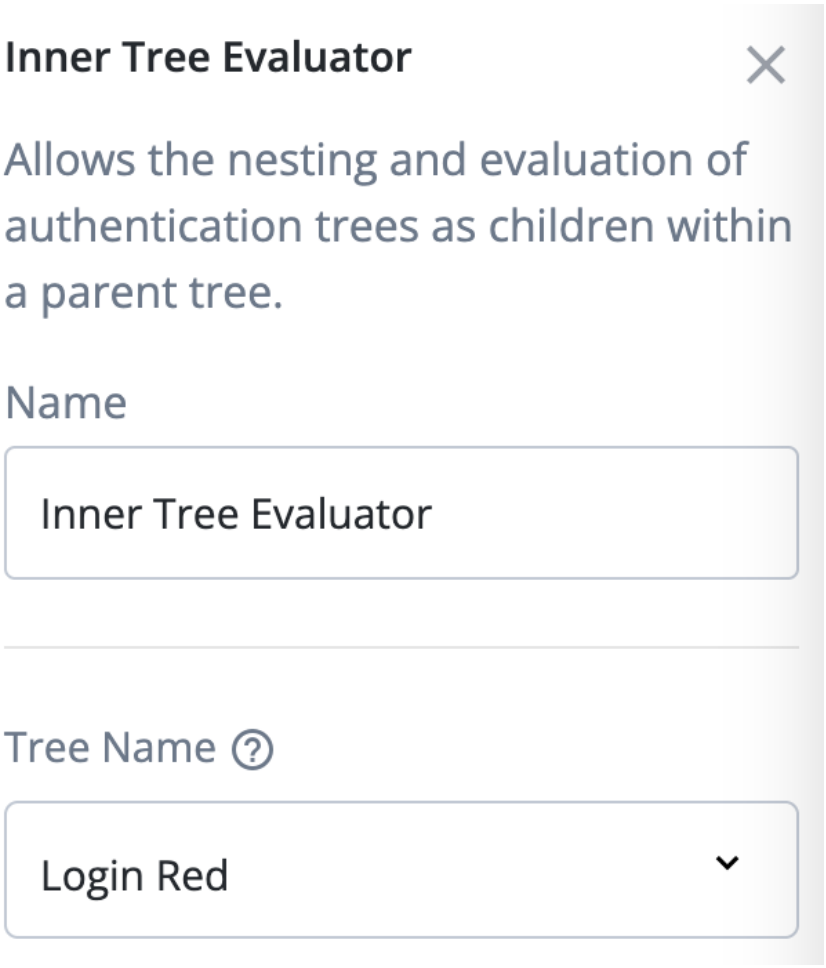



2. The REST request to access the journey includes one of the following headers:
   - `--header "brand: yellow"`
   - `--header "brand: red"`
   - `--header "brand: regular"`

3. The Request Header node obtains the value of the `brand` header and sets that value in the `stateBrand` property in the node state. For example, `stateBrand=yellow`
4. The journey progresses to the scripted decision node that includes the following script:

```
var brand = JSON.parse(nodeState.get('stateBrand'));

if (brand.indexOf("yellow") >= 0) {
  outcome = "yellow";
} else if (brand.indexOf("red") >= 0) {
  outcome = "red";
} else {
  outcome = "regular";
}
```

5. The script routes the journey to one of three outcomes; `yellow`, `red`, or `regular`, depending on the value of the `stateBrand` property.
6. The outcomes direct the user to a custom branded Login journey configured in an Inner Tree Evaluator node. For example:




7. Each Inner Tree Evaluator node routes the end user to a login journey that uses a custom brand.

### Redirection from an external system

An external system redirects a user to this authentication journey. The external system must share information about the user with the journey. Use the Request Header node to obtain the relevant request headers and inform the journey of their values.

## Retry Limit Decision node

The **Retry Limit Decision** node tracks failed authentications. If the number of failed authentications is below a specified **Retry Limit**, the user can attempt authentication again. Otherwise, the node continues evaluation along the `Reject` outcome path.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the `realm` and `username` properties in the incoming node state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

### Dependencies

This node has no dependencies.

### Configuration

| Property | Usage |
|---|---|
| **Retry limit** | Specify the number of retries to allow.<br>Default: `3` |

| Property | Usage |
| --- | --- |
| **Save Retry Limit to User** | Specify whether the number of failed login attempts persists across multiple journeys until authentication is successful. Possible values are:<br>***Enabled***<br>The node saves the number of failed login attempts to the `retryLimitNodeCount` attribute in the user's profile. New flows using this node start with the stored value and continue to the retry limit.<br>AM resets the count after the user authenticates successfully with an authentication journey that contains this node.<br>If AM can't find the user's profile, authentication ends with an error.<br>***Disabled***<br>The node saves the number of failed login attempts in a shared state property named `nodeId.retryCount` and discards the value when the authentication journey ends.<br>For security reasons, you should enable this setting.<br>Default: Enabled. |

## Outputs

If **Save Retry Limit to User** is enabled, the node increments the retry count and saves the number of failed attempts to the `retryLimitNodeCount` attribute in the user's profile. If the user can't be identified during the journey, the journey ends with an error.

If **Save Retry Limit to User** is disabled, the node increments the retry count and saves the number of failed attempts to a shared state property named `nodeId.retryCount`. The count is lost if the journey is restarted.

## Outcomes

### `Retry`

The user hasn't exceeded the number of allowed retries and can attempt authentication again.

### `Reject`

The user has exceeded the number of allowed retries.

## Errors

This node can log the following:

### *Warnings*

- `Error clearing attribute`

  The node can't reset the `retryLimitNodeCount` user attribute after the user has successfully authenticated.

### *Errors*

- `Error getting current retry count`

  The node can't retrieve the current retry count.

- `Failed to save retryLimitNodeCount to user: Identity Repo has not been upgraded.`

  The node can't save retry count details to the `retryLimitNodeCount` user attribute during the authentication flow.

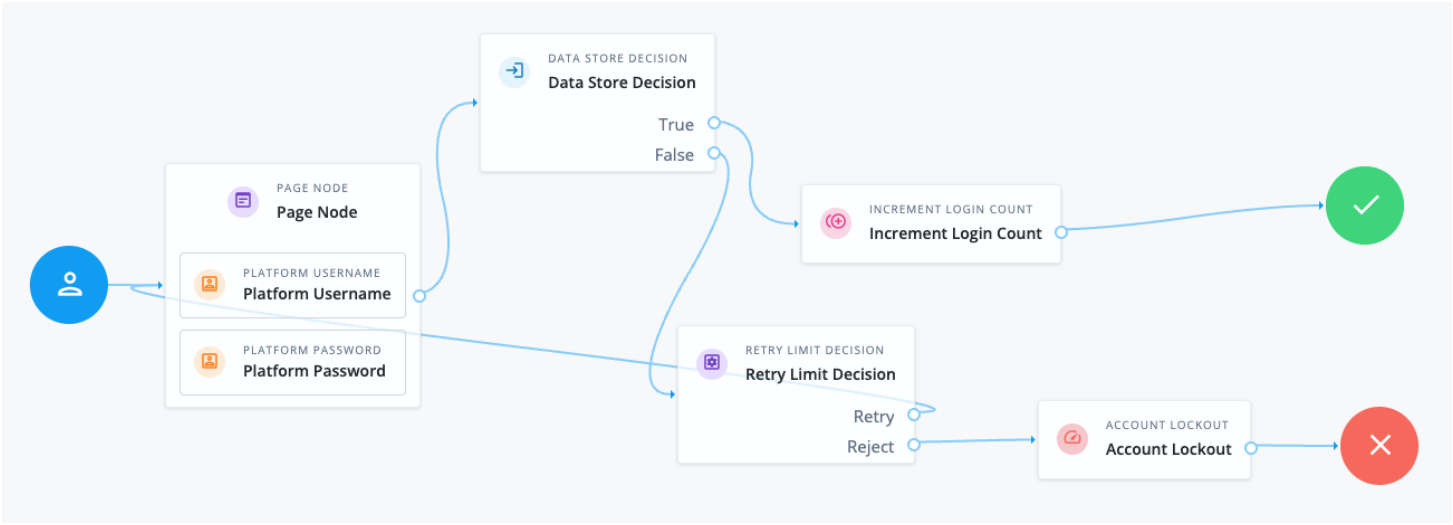
- `Error setting retry count on user attribute`

  The node can't increment the retry count on the `retryLimitNodeCount` user attribute during the authentication flow.

These warnings and errors typically occur if the identity store is unreachable or the user no longer exists.

### Examples

This example uses the **Retry Limit Decision** node to allow a user three attempts to authenticate. Otherwise, their account is locked.




- The Page node containing the Platform Username node and Platform Password node prompts for credentials.
- The Data Store Decision node validates the username-password credentials.
- The Increment Login Count node updates the number of successful authentications in the user profile.
- The **Retry Limit Decision** node is configured to allow three login attempts and either retries the login attempt or rejects it depending on the number of failed attempts.
- The Account Lockout node locks the user's account on their fourth failed attempt.

## Scripted Decision node

The **Scripted Decision** node lets you run a server-side script in an authentication journey. It exists to let you connect the script to other nodes with the journey editor.

The script makes a decision to set the outcome for the node.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

**Scripted Decision** node inputs depend entirely on the node's server-side script.

The script has access to the authentication context including:

- Headers from the request
- Query string parameters from the request
- Secrets configured for the realm
- Shared state data
- User profile data

The script can use callbacks to prompt the user for information.

Find information on the inputs available to the script in the *scripting* documentation for:

- PingAM
- Advanced Identity Cloud

You can restrict available inputs using the **Script Inputs** field when configuring the node.

## Dependencies

Before you configure this node, you must create a script of a specific type:

- For PingAM and Ping Identity Platform deployments, the script type is **Decision node for authentication trees**.
- For Advanced Identity Cloud the script type is **Journey Decision Node**.

## Configuration

| Property | Usage |
|---|---|
| **Script** | Select the script to run from the drop-down list. |

| Property | Usage |
| --- | --- |
| **Outcomes** | Enter one string for each `outcome` the script can set.<br>The node shows only the outcomes you configure. If you omit an `outcome` string, you can't connect it in the journey editor.<br>When the script sets an `outcome` you omitted in the configuration, it logs a warning. This can prevent the journey from completing successfully. |
| **Script Inputs** | Optionally, list the shared state data properties required by the script.<br>If you change the setting, you must declare each property or `null` for no properties.<br>Default: `*`. The script has access to all shared and transient state data.<br><br>**Important**<br>Sensitive data in transient state upgrades to *secure* state if:<br>• The node sends a callback to the user.<br>• The node detects a downstream node requesting the transient state data as input.<br><br>Unless the downstream node explicitly requests the secure state data by name, the authentication journey removes it from the node state after processing the next callback.<br>For example, a node in a registration journey stores a user's password in transient state. The node sends a callback to the user before an inner tree node, downstream in the journey, consumes that password. As part of the callback, the journey assesses what to add to the secure state. It does this by checking the state inputs that downstream nodes in the journey require. Nodes that *only* request `*` are ignored, as this would result in putting everything in transient state into secure state, and retaining sensitive information longer than necessary.<br>If a downstream node requires the password, it must explicitly request it as state input, even if it lists the `*` wildcard as input. |
| **Script Outputs** | Optionally, list the shared state data properties the node expects the script to set.<br>If you change the setting, you must declare each property or `null` for no properties.<br>Default: `*`. The node doesn't validate the script outputs at all. |

## Outputs

**Scripted Decision** node outputs, such as updates to shared state data, depend entirely on the node's server-side script.

You can restrict available outputs using the **Script Outputs** field when configuring the node.

## Outcomes

The script defines the outcomes by setting its `outcome` variable to an outcome string before returning.

You include all possible `outcome` strings from the script in the **Outcome** field when configuring the node.

The authentication journey continues along the outcome path from the script.

### Errors

The server-side script can log messages.

The node logs the following warning messages:

Warnings:

- `Found an action result from scripted node, but it was not an Action object`: An action in a legacy script didn't return an object with type `Action`.
- `Found an action result from scripted node, but it was not an ActionWrapper object`: An action in a next generation script didn't return an object with type `ActionWrapper`.
- `invalid script outcome <outcome>`: The <outcome> is missing in the **Outcome** field of the node configuration.
- `invalid script outcome <action-outcome> in action`: The <action-outcome> is missing in the **Outcome** field of the node configuration.
- `script outcome error`: The script set an outcome not found in the **Outcome** field of the node configuration.

### Examples

You use a **Scripted Decision** node when no other available node does what you need.

In this example, the node depends on the **Decision node for authentication trees**] script (PingAM deployments) or the **Journey Decision Node**] script Advanced Identity Cloud. The script gets the user's names from their profile and stores a message in a shared state property:

Next-generation

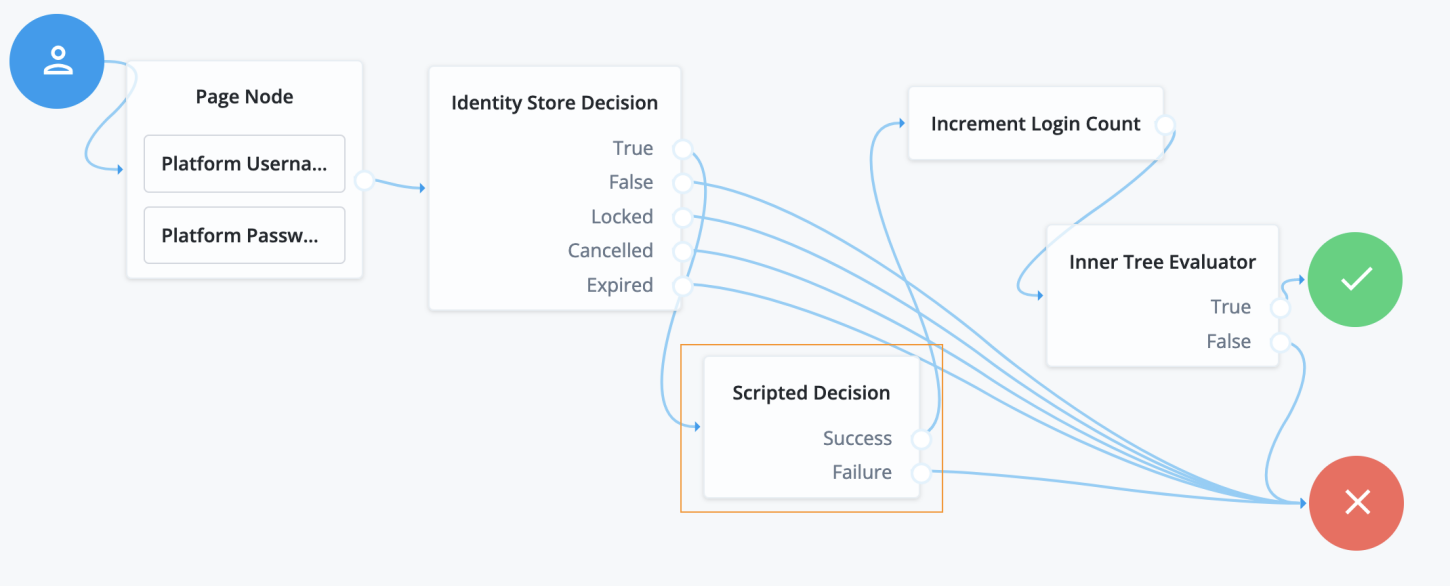
```
// Get the username from shared state data:
var username = nodeState.get('username')

// Get the given name(s) and surname(s) from the user profile:
var profile = idRepository.getIdentity(username)
var givenname = profile.getAttributeValues('givenName')
var surname = profile.getAttributeValues('sn')
if (!(givenname && surname)) {
  var error = `Failed to get names for ${username}: ${givenname} ${surname}`
  action.goTo('Failure').withErrorMessage(error);
} else {
  // Record who authenticated in the shared state data:
  var firstGivenName = givenname[0]
  var firstSurname = surname[0]
  var now = new Date().toLocaleString()
  var message = `${firstGivenName} ${firstSurname} logged in at ${now}.`
  nodeState.putShared('message', message)
  action.goTo('Success');
}
```

Legacy

```
var goTo = org.forgerock.openam.auth.node.api.Action.goTo

// Get the username from shared state data:
var username = nodeState.get('username').asString()

// Get the given name(s) and surname(s) from the user profile:
var profile = idRepository.getIdentity(username)
var givenname = profile.getAttributeValues('givenName')
var surname = profile.getAttributeValues('sn')
if (!(givenname && surname)) {
  var error = `Failed to get names for ${username}: ${givenname} ${surname}`
  action = goTo('Failure').withErrorMessage(error).build()
} else {
  // Record who authenticated in the shared state data:
  var firstGivenName = givenname[0]
  var firstSurname = surname[0]
  var now = new Date().toLocaleString()
  var message = `${firstGivenName} ${firstSurname} logged in at ${now}.`
  nodeState.putShared('message', message)
  action = goTo('Success').build()
}
```

Notice the script sets the outcomes using the `Action.goTo(outcome)` function.

The journey is as follows:




1. The Page node prompts the user for their username and password.
2. The Identity Store Decision node checks the username and password with those in the identity store.

   The node configuration enables the optional **Username as Universal Identifier** setting. As a result, the `nodeState.get('username').asString()` function in the script returns the user's `_id`. The `username` in `idRepository.getIdentity(username)` must be the `_id`, not the `userName`, to get the attribute from the user's profile.

> **Note**
>
> In a PingAM deployment, replace the Identity Store Decision node with a Data Store Decision node to check the username and password.

3. The **Scripted Decision** node runs the script and has the following settings:

| | |
|---|---|
| **Script** | The name of the script |
| **Outcomes** | `Success`, `Failure` |
| **Script Inputs** | `username` |
| **Script Outputs** | `*` |

   Notice the **Outcomes** setting lists all outcome strings from the script.

4. The Increment Login Count node updates the count on successful authentication.

5. The Inner Tree Evaluator node refers to another journey to perform more steps.

   This node is optional.

If you activate debug mode for the journey and select **Enable Debug Popup**, you find the message in the debug popup window when authenticating:

```
{
  "universalId": "id=<_id>,ou=user,o=alpha,ou=services,ou=am-config",
  "transactionId": "<transaction-id>",
  "password": "<password>",
  "pageNodeCallbacks": {
    "0": 0,
    "1": 1
  },
  "realm": "/alpha",
  "message": "Babs Jensen logged in at August 16, 2023 9:55:33 AM UTC.",
  "authLevel": 0,
  "objectAttributes": {
    "password": "<password>"
  },
  "username": "id=<_id>"
}
```

## Set Error Details node

The **Set Error Details** node adds details to the JSON response when a journey ends in an error. You can configure the node properties to return an error message and extra information in the form of static `key:value` fields.

Place the **Set Error Details** node before the node that errors in the journey.

## Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes [1] |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

[1] Currently available only in the rapid release channel.

## Inputs

None. This node doesn't read shared state data.

## Dependencies

None.

## Configuration

| Property | Usage |
| --- | --- |
| **Error Message** | The message to add to the JSON response when a journey ends in an error. Add a custom, localized message per locale:<br>1. Click **+**.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message. For example, `Invalid outcome from script`.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done. |

<table>
<tr><th>Property</th><th>Usage</th></tr>
<tr><td>Error Details</td><td>The details to add to the JSON response when a journey ends in an error:<br><br>1. Click +.<br>2. In the Key field, enter a name to identify the details. For example, <code>redirect_url</code>.<br>3. In the Value field, enter the details to return. For example, <code>https://example.com</code>.<br>The value can be a simple text string, a boolean value, or a JSON formatted value. The value is formatted appropriately when output in the JSON response.<br>For example:<br><br><table><tr><th>Key</th><th>Value</th><th>Output</th></tr><tr><td><code>example</code></td><td><code>this is a test value</code></td><td><pre>"example": "this is a test value"</pre></td></tr><tr><td><code>boolean</code></td><td><code>true</code></td><td><pre>"boolean": true</pre></td></tr><tr><td><code>field</code></td><td><code>{ "nested": "nested value" }</code></td><td><pre>"field": {<br>  "nested": "nested value"<br>}</pre></td></tr></table><br>4. Click Done.<br>5. Click + Add to repeat and add more messages.<br>6. Save your changes.</td></tr>
</table>

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

## Outputs

This node doesn't change the shared state.

## Outcomes

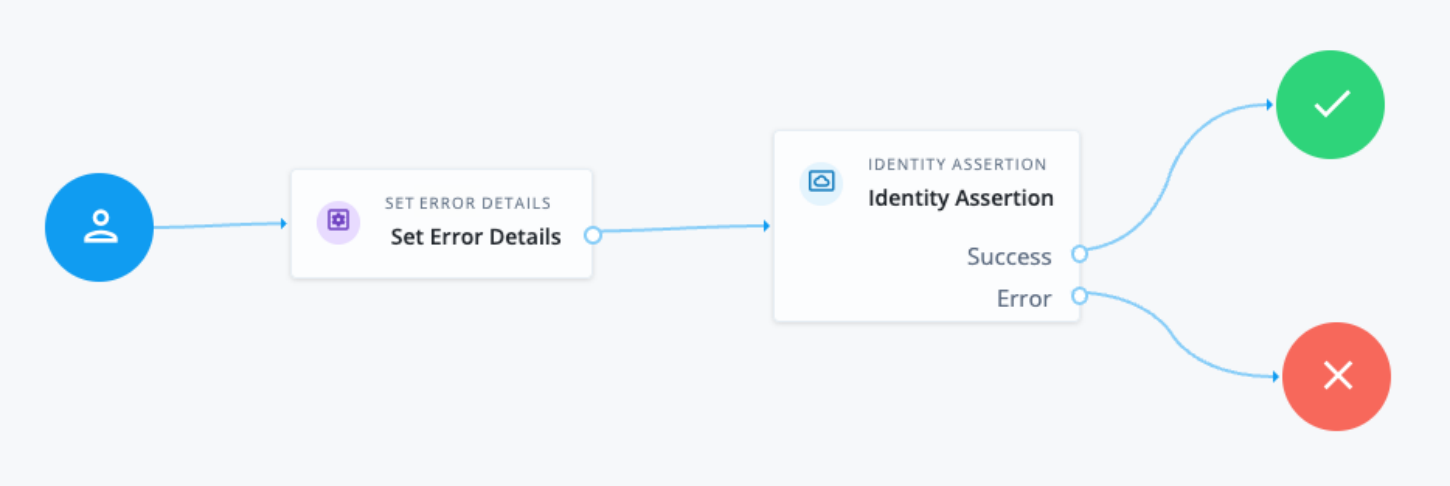
Single outcome path: when the journey ends in an error, this node adds the configured details to the JSON response.

## Errors

This node doesn't log messages of its own.

## Examples

This example uses the **Set Error Details** node to handle errors from the Identity Assertion node when the journey ends in an error.




- The **Set Error Details** node adds details to the JSON response when the journey ends in an error. This example uses the following configuration:

  ### *Error Message*

  - **Key**: `en-gb`
  - **Value**: `Identity assertion failure`

  ### *Error Details*

  - **Key**: `errorUrl`
  - **Value**: `https://example.com/error`

- The **Identity Assertion** node is configured as described in the documentation.

  If an error is encountered, the **Set Error Details** node displays the configured message to the user and adds both the message and the details to the JSON response.

  For example:

```
{
    "code": 401,
    "reason": "Unauthorized",
    "message": "Identity assertion failure",
    "detail": {
      "errorUrl": "https://example.com/error"
    }
}
```

## Set Failure Details node

The **Set Failure Details** node adds details to the JSON response when a journey ends in failure. You can configure the node properties to return a failure message and extra information in the form of static `key:value` fields.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

None. This node doesn't read shared state data.

### Dependencies

None.

## Configuration

| Property | Usage |
| --- | --- |
| **Failure Message** | The message to add to the JSON response when authentication fails:<br>Add a custom, localized message per locale:<br>1. Click +.<br>2. In the **Key** field, enter the locale. For example, `en-gb`.[1]<br>3. In the **Value** field, enter the message. For example, `Your account has been locked`.<br>4. Click **Done**.<br>5. Repeat to add more messages and save your changes when you're done. |
| **Failure Details** | The details to add to the JSON response on journey failure:<br>1. Click +.<br>2. In the **Key** field, enter a name to identify the details. For example, `Reason`.<br>3. In the **Value** field, enter the details to return. For example, `Exceeded max retries`.<br>The value can be a simple text string, a boolean value, or a JSON formatted value. The value is formatted appropriately when output in the JSON response.<br>For example:<br>4. Click **Done**.<br>5. Click + **Add** to repeat and add more messages.<br>6. Save your changes. |

| Key | Value | Output |
| --- | --- | --- |
| `example` | `this is a test value` | `"example": "this is a test value"` |
| `boolean` | `true` | `"boolean": true` |
| `field` | `{ "nested": "nested value" }` | `"field": { "nested": "nested value" }` |

(1) Specify a locale that Java supports, such as `en-gb`. Otherwise, the node throws a configuration exception with an `Invalid locale provided` message.

### Outputs

This node doesn't change the shared state.

### Outcomes

Single outcome path: when the journey ends in failure, this node adds the configured details to the JSON response.

### Errors

This node doesn't log messages of its own.

### Examples

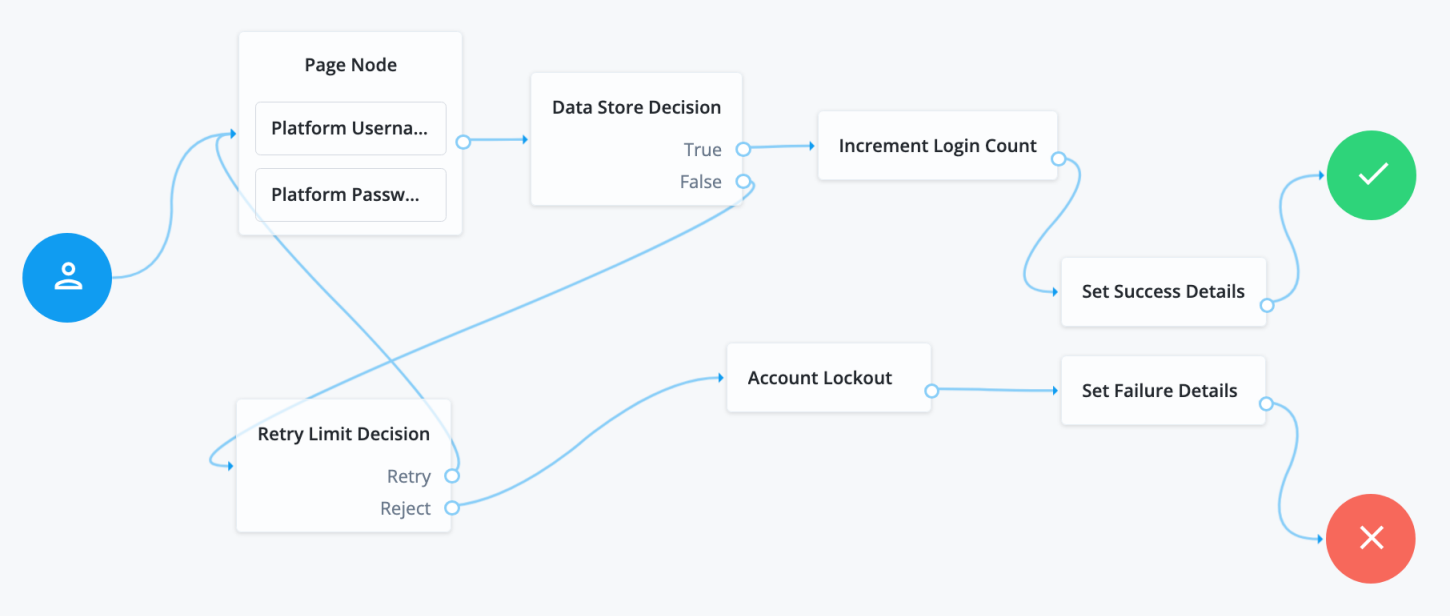
This example uses the **Set Failure Details** node and assumes the following configuration:

#### *Failure Message*

- **Key**: `en-gb`
- **Value**: `Your account is locked`

#### *Failure Details*

- **Key**: `Reason`
- **Value**: `Exceeded max retries`




- The Page node containing the Platform Username node and Platform Password node prompts for credentials.

- The Data Store Decision node validates the username-password credentials.
- If authentication is successful:
  - The Increment Login Count node updates the number of successful authentications in the user profile.
  - The Set Success Details node adds any configured details to the JSON response.
- If authentication fails:
  - The Retry Limit Decision node checks the number of failed authentications against the configured limit. If the retry limit is reached, the journey continues on the `Reject` outcome path.
  - The Account Lockout node locks the account.
  - The **Set Failure Details** node displays the configured message to the user and adds both the message and the details to the JSON response.

    For example:

```
{
  "code":401,
  "reason":"Unauthorized",
  "message":"Your account is locked",
  "detail":{
    "Reason":"Exceeded max retries"
  }
}
```

## Set Session Properties node

The **Set Session Properties** node lets you change details in the resulting authenticated session. You can do one or both of the following:

### *Set session properties*

Use this node to add `key:value` properties to the user's authenticated session on successful authentication.

> **Tip**
>
> You can access session properties using a variable in a webhook.
> Learn more in the documentation on configuring authentication webhooks for:
> - Advanced Identity Cloud
> - PingAM

### *Update the authenticated session timeouts*

Use this node to update the authenticated session timeout settings. For example, you could have different session settings for different branches within the journey based on factors such as which region the user was authenticating from or the authentication methods being used.

Consider the following when updating authenticated session timeout settings:

- If a journey has multiple nodes that set session timeouts, AM uses the settings associated with the last executed node to determine the timeouts for the resulting authenticated session.
- If an inner journey includes nodes that set session timeouts, AM uses the updated timeouts in the parent journey.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads the authenticated session timeout settings from the user, journey or the Session service.

Find information in the *session termination* documentation for:

- PingAM
- Advanced Identity Cloud

## Dependencies

Make sure the user can successfully authenticate and get a session.

This node is ignored unless the user gets an authenticated session.

## Configuration

| Property | Usage |
|---|---|
| **Properties** | The session properties to set.<br>• To add a session property:<br>1. Click +, then + **Add** in the **Properties** modal.<br>2. Enter the session property name in the **Key** field and the value to set in the **Value** field.<br>3. Click **Done**.<br>• To edit a property:<br>1. Click the **Pencil** icon (✎).<br>2. Update the **Key** and **Value** as when adding properties.<br>• To remove a property, click the **Delete** icon (🗑).<br>When finished, click **Save** to keep your settings. |
| **Maximum Session Time** [1] | The maximum authenticated session time in minutes.<br>If set, this overrides the maximum authenticated session time set in the journey or the Session service.<br>Find information in the *session termination* documentation for:<br>• PingAM<br>• Advanced Identity Cloud<br>**Note**<br>If a user has session timeouts set, the user-specific settings are always used. |
| **Maximum Idle Time** [1] | The maximum idle time for the authenticated session in minutes.<br>If set, this overrides the maximum idle time set in the journey or the Session service.<br>Find information in the *session termination* documentation for:<br>• PingAM<br>• Advanced Identity Cloud<br>**Note**<br>If a user has session timeouts set, the user-specific settings are always used. |

[1] Currently available only in the rapid release channel.

## Outputs

This node sets *session* properties. It doesn't change the shared state data.

This node can't override system session properties, such as the principal or the authentication level. Use a different journey to re-authenticate the user rather than trying to change such properties with this node.

Additionally, this node updates the authenticated session timeouts if new values are provided.

## Outcomes

Single outcome path.

## Errors

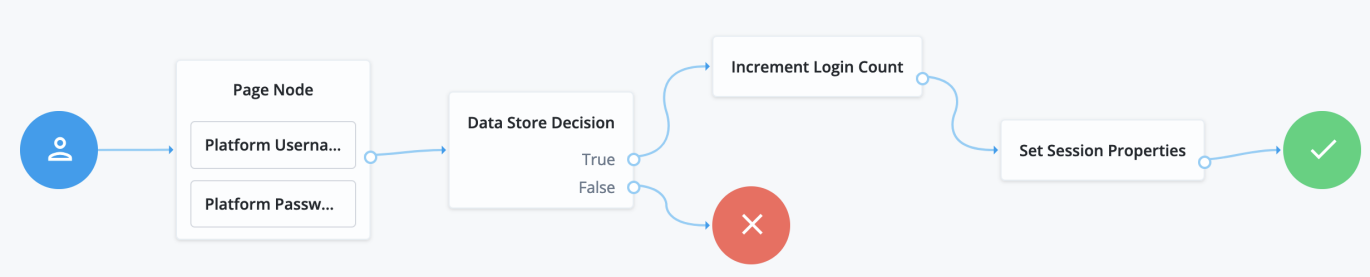
This node doesn't log messages of its own.

## Examples

### Example 1: Update the `successURL` session property

This example uses the **Set Session Properties** node to update the `successURL` session property.

- A first platform journey updates the session property on successful authentication:

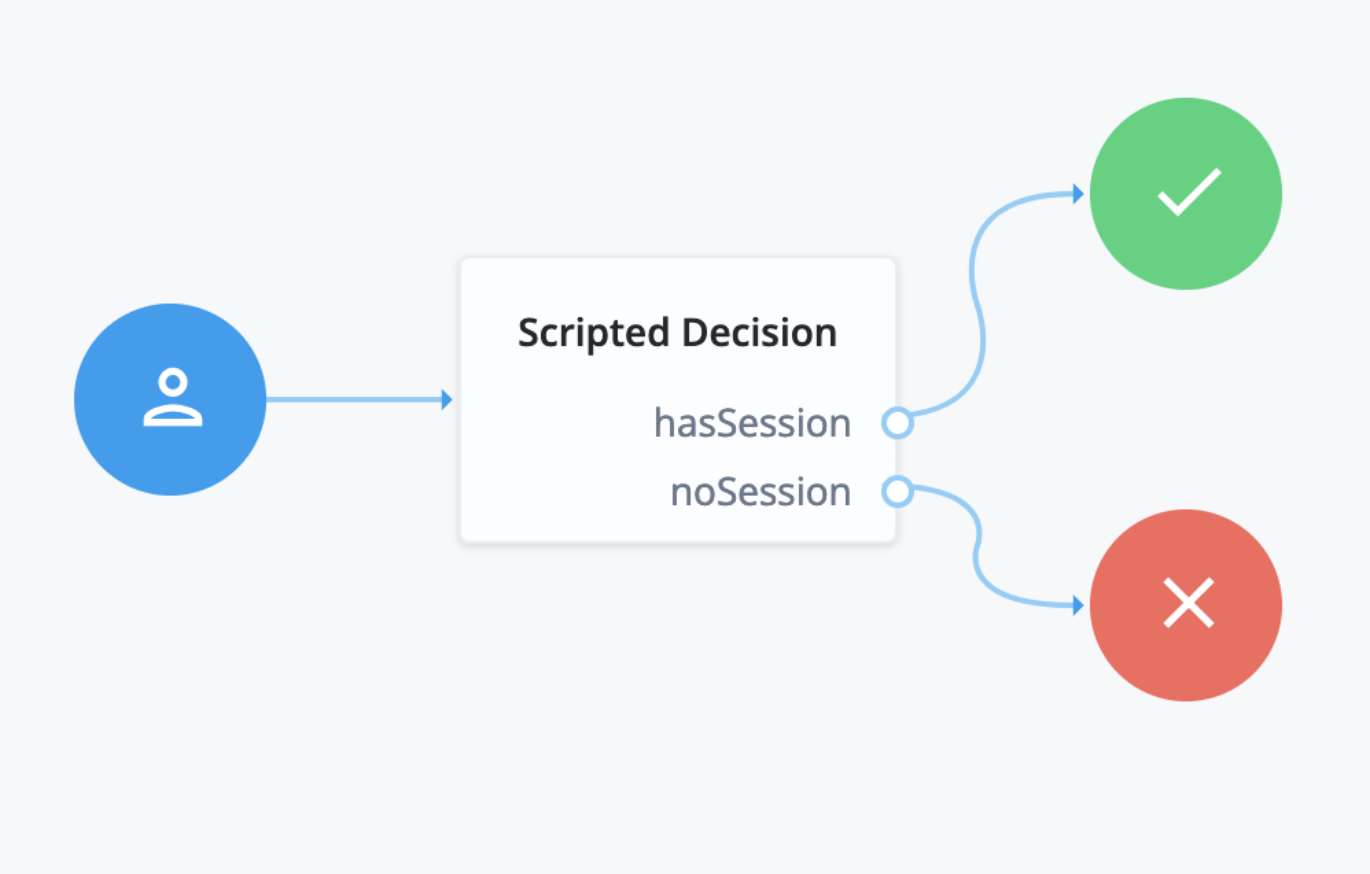



  - The Page node containing the Platform Username node and Platform Password node prompts for credentials.
  - The Data Store Decision node validates the username-password credentials.
  - The Increment Login Count node updates the number of successful authentications in the user profile.
  - The **Set Session Properties** node, sets the `successURL` session property.

    Configure the **Properties** to add a `successURL` property with the URL of your choice.

  When the journey completes successfully, AM updates the `successURL` in the user's session data.

- A second journey uses a script to display the session properties after the user authenticates:




The Scripted Decision node calls the following script to inject the session properties into the shared state data so the journey can display them though a debug popup:

```
if (typeof existingSession !== 'undefined') {
  nodeState.putShared('session', existingSession)
  action.goTo('hasSession')
} else {
  nodeState.putShared('session', null)
  action.goTo('noSession')
}
```

The second journey has **Debug mode** and **Enable Debug Popup** selected.

Follow these steps to try the example:

1. Create both journeys using the journey editor.
2. Sign in through the first journey with a test user account.

   The browser shows the user profile page.
3. In the same browser window, browse to the URL for the second journey.

   The debug popup window displays the shared state data including session properties:

```
{
    "transactionId": "...",
    "session": {
        "successURL": "<your-success-url>",
        "...": "..."
    },
    "realm": "/alpha",
    "authLevel": 0,
    "username": "test"
}
```

The `successURL` property is set to <your-success-url>, the one you configured as the value in **Properties** of the **Set Session Properties** node.

4. Sign out as the test user.
5. Sign in through the *default* journey as the test user.

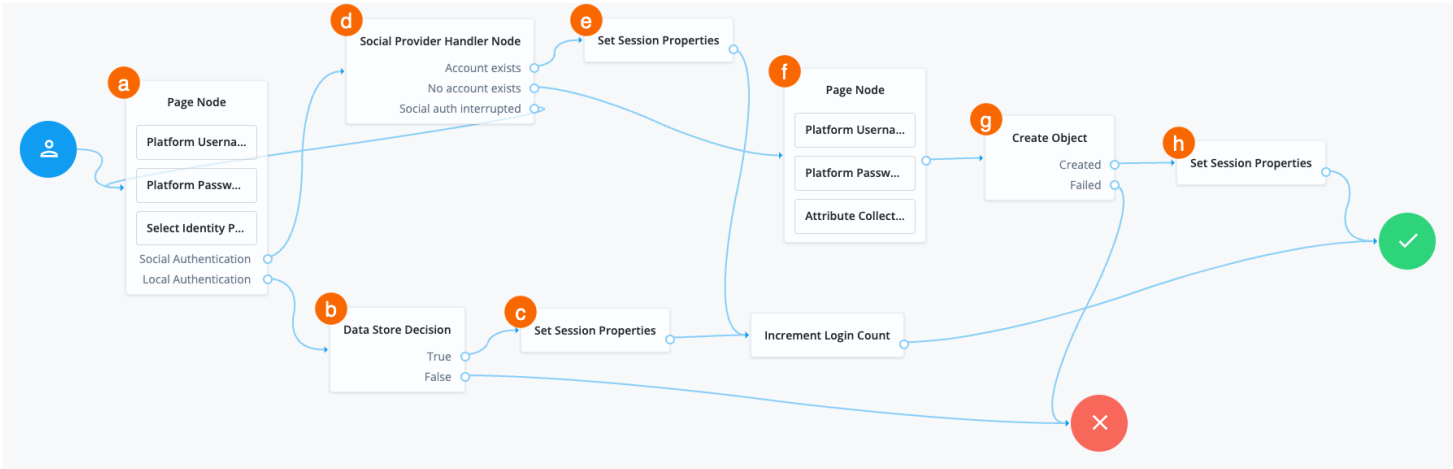
   The default journey doesn't use the **Set Session Properties** node with your configuration, so it uses the default value for the `successURL` session property.
6. In the same browser window, browse to the URL for the second journey again.

   The debug popup window displays the shared state data, including session properties with the default `successURL` value.

### Example 2: Update the authenticated session timeouts

This example uses the **Set Session Properties** node to set different session timeouts depending on the authentication method used. If the user chooses to authenticate with a username and password, the resulting authenticated session will have a shorter lifetime than if they use a social provider to authenticate. For both authentication methods, the idle timeout will be lower than the default.

> **Note**
>
> Instead of setting the same maximum session idle timeout in each node, you could set it at the journey level and leave the node setting blank. They have been set in the nodes in this example for demonstration purposes.



a The Page node containing the Select Identity Provider node prompts the user to select a social identity provider or to authenticate with a username and password.

b If the user selects local authentication, the Data Store Decision node takes care of the authentication.

c The **Set Session Properties** node for local authentication updates the session timeout values with the following settings:

- **Maximum Session Time**: 60
- **Maximum Idle Time**: 20

d If the user selects social authentication, the Social Provider Handler node does the following:

- Routes the user to the selected social provider to authenticate there
- Retrieves the user's profile information and transforms it into a format that AM can use
- Assesses whether the user has an existing identity in AM
  - If the user has an existing identity, authenticates that identity
  - If the user doesn't have an identity, routes the user to another page node
  - If the user interrupts the social authentication, routes the user back to the Select Identity Provider node

e The **Set Session Properties** node for social authentication updates the session timeout values with the following settings:

- **Maximum Session Time**: 180
- **Maximum Idle Time**: 20

f The nodes on the page node request the information required to *register* a new identity.

g The Create Object node creates the new identity in AM.

h The **Set Session Properties** node for social authentication updates the session timeout values with the following settings:

- **Maximum Session Time**: 180
- **Maximum Idle Time**: 20

## Set State node

The **Set State** node lets you set the values of any configured attributes in the shared state. The node removes any previous values of the specified attributes from all states (transient, secure, and shared) and resets them to the configured values in the shared state.

> **Caution**
>
> Don't use this node to add sensitive data, such as passwords, to the shared state.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node reads any attributes stored in the shared state by previous nodes in the journey.

## Dependencies

This node has no specific dependencies.

## Configuration

| Property | Usage |
|---|---|
| **Attributes** | Add the attributes you want to set in the shared state. For each attribute:<br>1. Click +.<br>2. In the **Key** field, enter the attribute name.<br>3. In the **Value** field, enter the value of the attribute you want to set.<br>4. Click **Save** when you have finished. |

## Outputs

This node writes any configured attributes and their values to the shared state.

## Outcomes

None

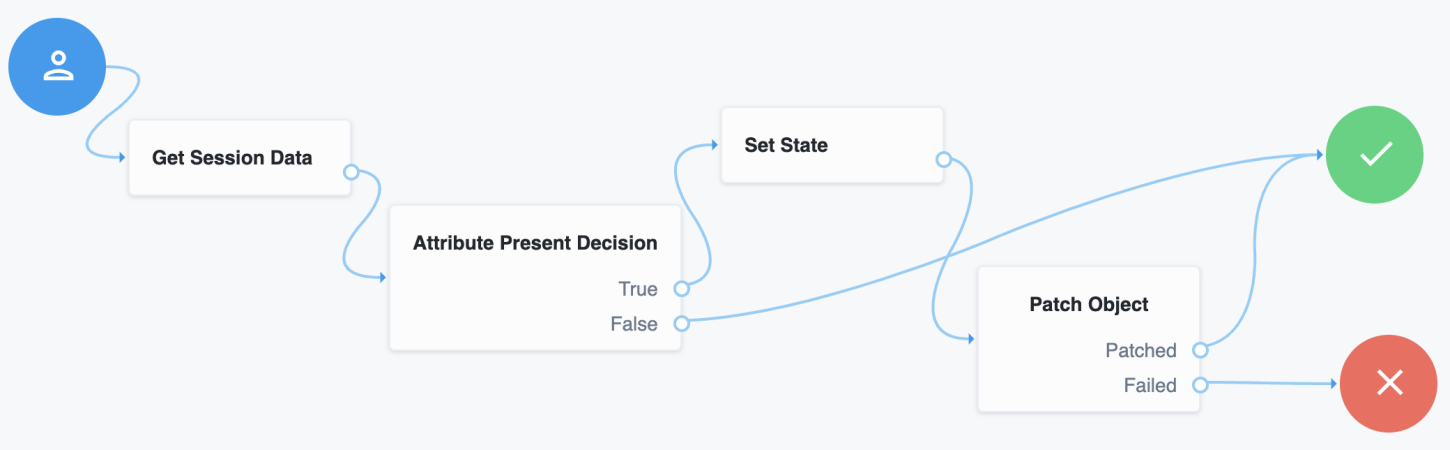
## Errors

This node doesn't log any errors or warnings.

### Example

The following sample journey assumes that an organization wants to overwrite the `costCenter` of specific employees when they authenticate.

- If an employee doesn't have a `costCenter` attribute in their profile, the journey proceeds without changing the user object.
- If the user does have a `costCenter` attribute, the journey uses the **Set State** node to overwrite the value of that attribute.




The user has already authenticated before beginning this journey:

- The Get Session Data node gets the `userName` from the session.
- The Attribute Present Decision node checks the user object to determine whether the user has a `costCentre` attribute.
  - If the user doesn't have a `costCentre` attribute, the journey proceeds to the success outcome.
  - If the user does have a `costCentre` attribute, the Attribute Present Decision node stores its value in the shared state.
- The **Set State** node overwrites the value of the `costCentre` attribute with the value set in the node configuration.
- The Patch Object node node updates the user object with the new `costCentre`.
- If the patch of the user object is successful, the journey proceeds to the success outcome; otherwise, the journey follows the failure outcome.

## Set Success Details node

The **Set Success Details** node adds additional details to the JSON response on successful authentication. You can add either or both of the following:

- **Success details**: Lets you add static `key:value` fields to the JSON response.
- **Session properties**: Lets you add `key:value` fields to the JSON response, where `value` corresponds to the value of the specified session property.

  This lets you retrieve session properties in the journey without needing additional post-authentication processing.

> **Note**
>
> You can't override the `tokenId`, `successUrl`, and `realm` fields returned in the JSON response by default.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

None. This node doesn't read shared state data.

## Dependencies

Make sure the user can successfully authenticate and get a session.

If this node is added to a no session journey or the `noSession` query parameter is used during authentication, this node has no effect.

## Configuration

<table>
<tr><th>Property</th><th>Usage</th></tr>
<tr><td>Success Details</td><td>The success details to add:<br>• To add a static field:<br>1. Click + Add in the Success Details modal.<br>2. Enter the field name to display in the Key field and the corresponding value to display in the Value field.<br>The value can be a simple text string, a boolean value, or a JSON formatted value. The value is formatted appropriately when output in the JSON response.<br>For example:<br><table><tr><th>Key</th><th>Value</th><th>Output</th></tr><tr><td><code>example</code></td><td><code>this is a test value</code></td><td><pre>"example": "this is a test value"</pre></td></tr><tr><td><code>boolean</code></td><td><code>true</code></td><td><pre>"boolean": true</pre></td></tr><tr><td><code>field</code></td><td><code>{ "nested": "nested value" }</code></td><td><pre>"field": {<br>  "nested": "nested value"<br>}</pre></td></tr></table>3. Click Done.<br>• To edit a field:<br>1. Click the Pencil icon (✎).<br>2. Update the Key and Value as when adding fields.<br>• To remove a field, click the Delete icon (🗑).<br><br>When finished, click Save to keep your settings.</td></tr>
</table>

| Property | Usage |
| --- | --- |
| **Session Properties** | The session properties to add.<br><br>**Note**<br>Session property details are only added to the JSON response if the session property exists in AM and it's available in the authenticated session.<br><br>You can find a list of the default session properties in the documentation for:<br><br>• PingAM<br>• Advanced Identity Cloud<br>• To add a session property:<br>1. Click + **Add** in the **Session Properties** modal.<br>2. Enter the field name to display in the **Key** field and the session property name in the **Value** field.<br>3. Click **Done**.<br>• To edit a property:<br>1. Click the **Pencil** icon (✎).<br>2. Update the **Key** and **Value** as when adding properties.<br>• To remove a property, click the **Delete** icon (🗑).<br><br>When finished, click **Save** to keep your settings. |

## Outputs

This node doesn't change the shared state.

## Outcomes

Single outcome path: when the journey completes successfully, this node adds the additional configured details to the JSON response.

## Errors

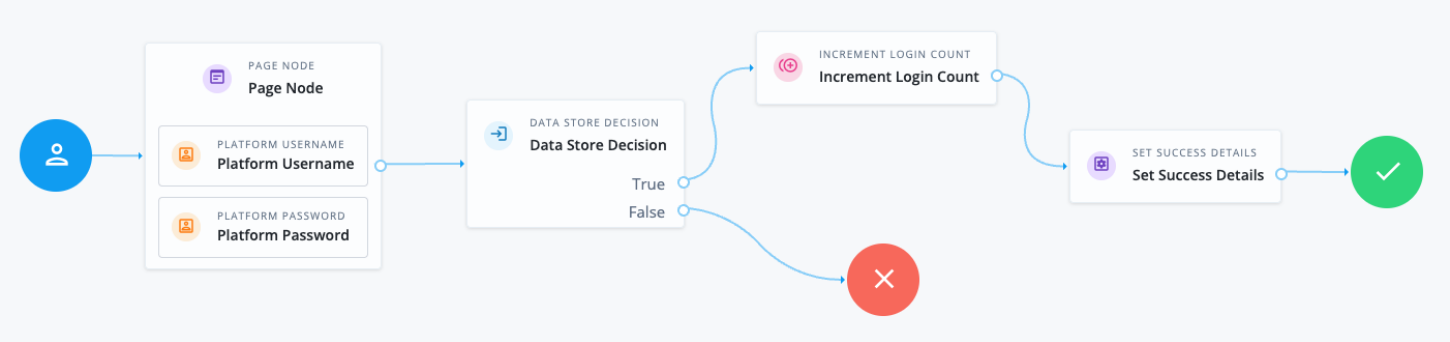
This node doesn't log messages of its own.

## Examples

### Example 1: Add static content and session properties

This example uses the **Set Success Details** node to add the following additional details to the JSON response:

- A static `authMethod:password` field.
- A `universalId` session property, which returns the corresponding value from the session when the user authenticates.




- The Page node containing the Platform Username node and Platform Password node prompts for credentials.
- The Data Store Decision node validates the username-password credentials.
- The Increment Login Count node updates the number of successful authentications in the user profile.
- The **Set Success Details** node adds the additional configured details to the JSON response upon successful authentication. This example uses the following configuration:

  ### *Success Details*

  **Key**: `authMethod`

  **Value**: `password`

  ### *Session Properties*

  **Key**: `universalId`

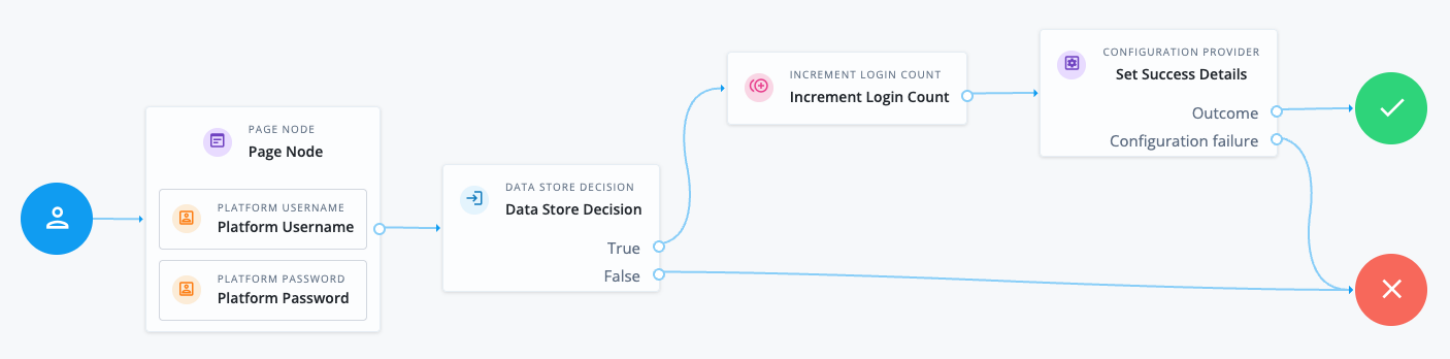
  **Value**: `sun.am.UniversalIdentifier`

When the user authenticates successfully using this journey, the JSON response includes the additional details you configured. For example:

```
{
    "tokenId": "AQIC5wM…TU3OQ*",
    "successUrl": "/enduser/?realm=/alpha",
    "realm": "/alpha",
    "universalId": "id=bjensen,ou=user,o=alpha,ou=services,ou=am-config",
    "authMethod": "password"
}
```

**Example 2: Add dynamic content**

This example uses the Configuration Provider node to imitate the **Set Success Details** node. Using the **Configuration Provider** node lets you add dynamic content to the JSON response, such as an API response, in addition to static content and session properties.




Find more information in Example 2: Imitate the Set Success Details node.

## State Metadata node

The **State Metadata** node returns selected attributes from the shared node state as metadata.

This node sends a `MetadataCallback` to retrieve shared state values, which it adds to the JSON response from the `/authenticate` endpoint. This example shows how a shared state attribute, `mail`, is returned:

```
{
    "callbacks": [
        {
            "type": "MetadataCallback",
            "output": [
                {
                    "name": "data",
                    "value": {
                        "mail": "bjensen@example.com"
                    }
                }
            ]
        }
    ]
}
```

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads its configured **Attributes** from the shared node state.

### Dependencies

None.

### Configuration

| Property | Usage |
| --- | --- |
| **Attributes** | Specify one or more shared state attribute names for return.<br>Default: none |

### Outputs

This node only sends the callback. It does not modify the shared node state.

### Outcomes

Single outcome path.

Evaluation continues after the callback.

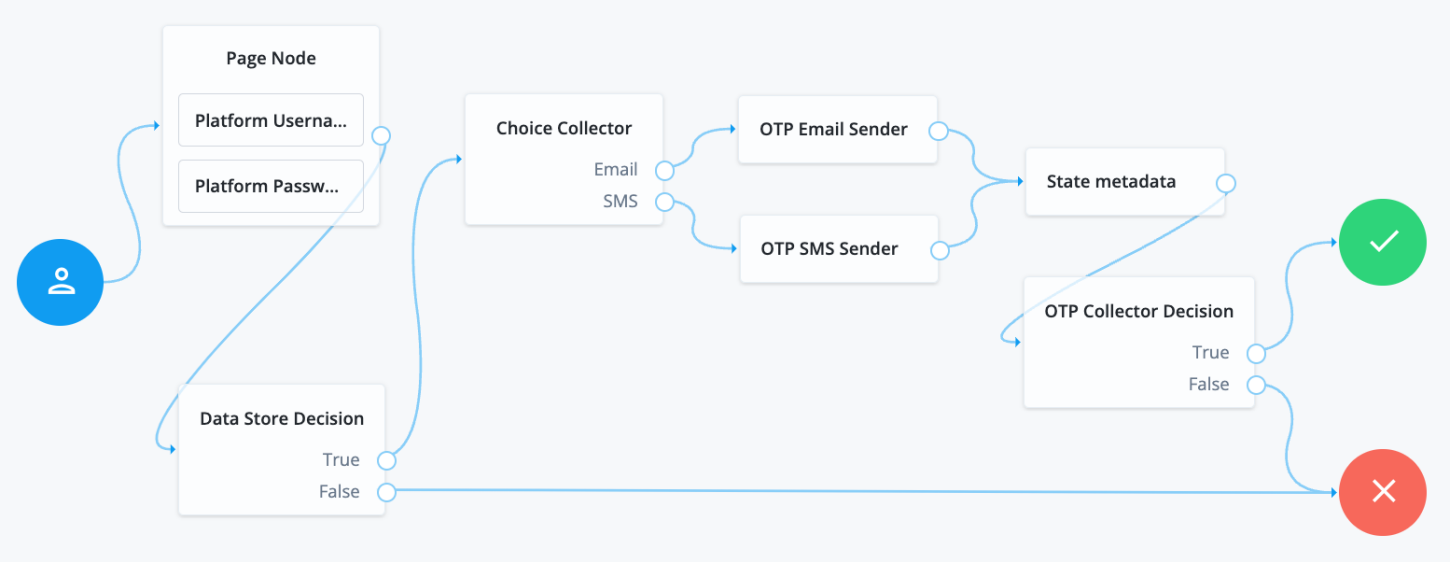
### Errors

This node does not log error or warning messages of its own.

### Example

Use this node to display custom information that includes user attributes without having to alter the existing flow.

For example, for OTP authentication with a choice of email or SMS, use this node to return the user's email address or phone number. You can use the attributes with an OTP Collector Decision node, and optionally, a Scripted Decision node, to customize the data for display later.




- The Page node with a Platform Username node and Attribute Collector node prompts for the credentials.
- The Data Store Decision node confirms the user's credentials.
- The Choice Collector node lets the user opt for notification through email or a text message.
- The OTP Email Sender node sends the one-time passcode (OTP) as email.
- The OTP SMS Sender node sends the OTP as a text message.
- The **State Metadata** node injects attributes for additional information.
- The OTP Collector Decision node displays the additional information when collecting the OTP to verify.

## Success URL node

Sets the redirect URL when authentication succeeds.

> **Note**
>
> Specifying a success URL overrides any `goto` query string parameters.

Find more information on configuring the Validation Service to trust redirection URLs and on how redirection URLs are determined in the corresponding documentation for:

- PingAM
- Advanced Identity Cloud

> **Tip**
>
> The URL is also saved in the `nodeState` object on the `successUrl` key.
> PingAM only: Find more information in Customize authentication trees.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Properties

| Property | Usage |
|---|---|
| Success URL *(required)* | Specify the full URL to redirect to when the authentication succeeds. |

## Timer Start node

Starts a named timer metric, which you can stop with a Timer Stop node.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Start Time Property | Specify a property name into which to store the current time.<br>Specify the same value in any instances of the Timer Stop node that measure the time elapsed since evaluation passed through this node. |

# Timer Stop node

Records the time elapsed since evaluation passed through the Timer Start node in the specified metric name.

Note that this node does not reset the time stored in the specified **Start Time Property** property. Other Timer Stop nodes can also calculate the time elapsed since evaluation passed through the same Timer Start node.

For PingAM deployments, you can find more information on the `Timer` metric type in Monitor AM instances and Monitoring metric types.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

Single outcome path.

## Properties

| Property | Usage |
|---|---|
| Start Time Property | Specify the property name containing the time from which to calculate the elapsed time. |

| Property | Usage |
| --- | --- |
| Metric Key *(required)* | Enter the name for a new metric that stores the calculated elapsed time.<br>PingAM deployments only:<br>The name you select is used to identify the metric that exposes the data collected by this node. For example, if you enter `calculated.time`, AM exposes a new metric with this name to the Common REST, JMX, or Graphite interfaces. If you use Prometheus, the name is prefixed with `am_` and appended with `_seconds` to become `am_calculated_time_seconds`.]<br>**Tip**<br>Metrics collate data from multiple invocations of a journey. To record the time it takes for a particular journey to complete, use a Scripted Decision node to store the start time in shared state. Use a script at the end of the journey to capture the end time and output the calculated journey time to the authentication audit logs.<br>Find more information in the *Audit* documentation for:<br>• PingAM<br>• Advanced Identity Cloud |

## Update Journey Timeout node

The **Update Journey Timeout** node updates the maximum duration of the journey session. You can either set a new timeout value to override the maximum duration or specify a timeout adjustment to modify the maximum duration.

You can use multiple instances of this node in a journey if required. For example, you could increase the maximum duration to allow the end user sufficient time to fetch a document, such as their passport. You could then increase it again to allow them to fetch a second document after they complete that step. Or you could have different timeouts for separate branches within the journey to accommodate different authentication requirements.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes [1] |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

[1] Currently available only in the rapid release channel.

## Inputs

This node reads the maximum duration of the journey session from the journey or the core authentication settings. Learn more in the documentation on *Maximum duration* for:

- PingAM
- Advanced Identity Cloud

## Dependencies

This node has no dependencies.

## Configuration

| Property | Usage |
| --- | --- |
| **Operation** | The timeout update operation.<br>Possible values are:<br>• `Set` : This operation overrides the maximum duration of the journey session with the new value specified.<br>• `Modify` : This operation modifies the maximum duration of the journey session by adding or subtracting the value specified.<br>**Warning**<br>Make sure you consider the impact of modifying the journey session duration in all possible scenarios. For example, if a journey loops back through this node multiple times, it could be possible to end up with an infinite length session. |
| **Value** | Enter the required value in minutes as follows:<br>• If you're using the `Set` operation, enter a positive integer to set the new maximum journey session duration.<br>• If you're using the `Modify` operation, enter a positive integer to increase the maximum journey session duration or a negative integer to decrease the maximum session duration by the specified value.<br>Values up to `2147483647` are allowed. |

## Outputs

This node updates the maximum duration of the journey session and stores it in the shared state.

## Outcomes

Single outcome path.

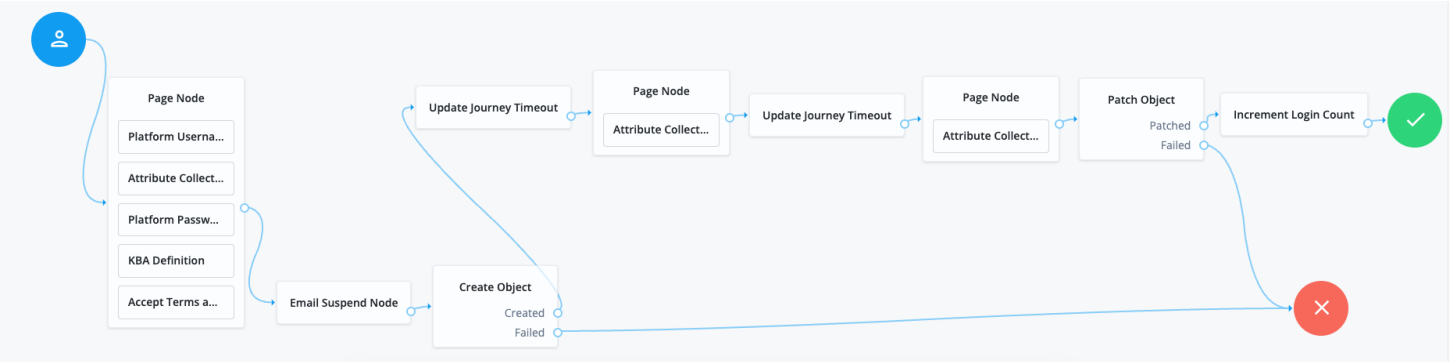
## Errors

If the node can't validate the configuration successfully, it logs `An error occurred validating the update journey timeout node configuration`.

## Examples

This example uses the **Update Journey Timeout** node to increase the maximum journey session duration to give the end user time to fetch their passport. It then increases it further to allow them time to fetch another document after they've entered their passport details.

This example extends the default **Registration** journey to verify the end user's email address, collect their passport details and collect details from an additional supporting document.




- The Page node prompts for the same information as the default **Registration** journey.
- The Email Suspend node sends an email to the user to verify their email address and suspends the journey.

  The journey proceeds when the user clicks the link, confirming their email address.
- The Create Object node stores the newly registered user's profile.
- The first **Update Journey Timeout** node increases the maximum session duration to `10` minutes with the following settings (this assumes the default of `5` minutes is unchanged):
  - **Operation**: `Modify`
  - **Value**: `5`
- The first Page node containing the Attribute Collector node prompts the user to input their passport details.
- The second **Update Journey Timeout** node increases the maximum session duration to `14` minutes with the following settings:
  - **Operation**: `Modify`
  - **Value**: `4`
- The second Page node containing the Attribute Collector node prompts the user to input details from their supporting document.
- The Patch Object node updates the user object with the collected details.

- The Increment Login Count node updates the count on successful authentication.

# Uncategorized nodes

## Debug node

Displays debug information about the current authentication tree.

This node collects information, such as the shared node state, the identity object's `universalId`, and the transaction ID, which are useful for reference in log messages.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | No |

### Outcomes

Single outcome path.

### Properties

| Property | Usage |
|---|---|
| Enable Debug Popup | If enabled, a popup window displays debug logs as you step through the flow in a browser. |

## Identity Assertion node

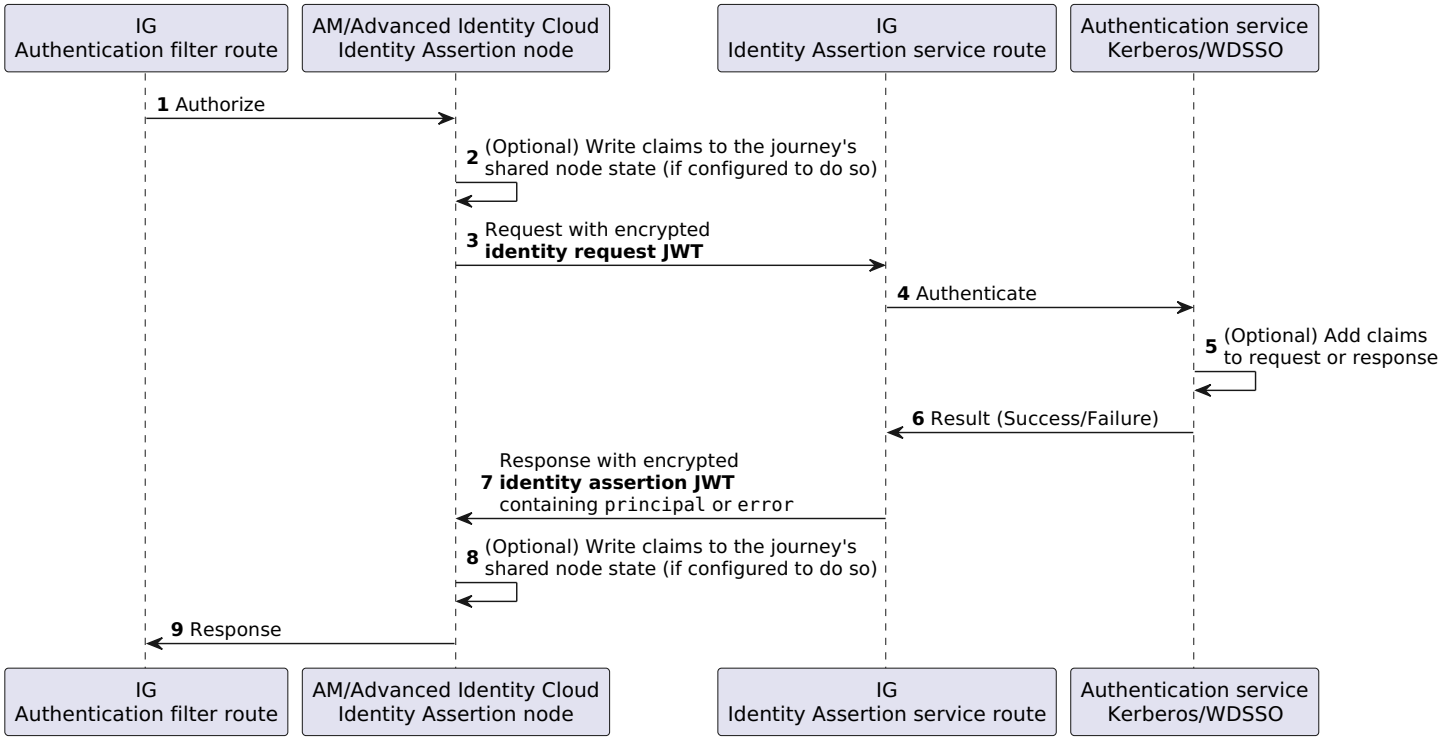
The **Identity Assertion** node provides a secure communication channel for authentication journeys to communicate directly with PingGateway.

You can find more information in the PingGateway documentation for:

- Advanced Identity Cloud
- PingAM

The node adds PingGateway's routing capabilities and supporting identity assertion with third-party authentication services. Authentication services include Windows Desktop SSO and Kerberos.

The following image shows the flow of an authentication request:




PingAM/Advanced Identity Cloud and PingGateway share a symmetric key for encryption and decryption at both ends of the flow.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

All shared node state properties listed in `Mapping to server claims` are valid optional inputs to this node.

To allow the node to validate that an Identity Assertion JWT is the result of an identity request, the nonce must be present in the shared node state as `identityAssertionNonce`. This isn't required for the initiating authentication request.

## Dependencies

The Identity Assertion node relies on the following prerequisites:

- An Identity Assertion service must be configured in the same realm, with at least one server configuration that can be selected for use with the Identity Assertion node.

  For PingAM deployments, the service can be configured globally.

- The Identity Assertion service server must have a valid shared secret encryption key configured in a secret store.

  In Advanced Identity Cloud the encryption key must be configured as an ESV.

- The Identity Assertion server must be deployed, running, and accessible to the Identity Assertion node.

  It must also be configured with the shared secret encryption key.

  PingGateway can fulfil the role of the Identity Assertion server.

To use the Identity Assertion node in your environment, you must complete the following steps, as described in detail in the worked Example:

- Create and import a secret encryption key
- Configure the Identity Assertion service
- Map the secret label to the encryption key
- Configure PingGateway as an Identity Assertion Server

## Configuration

The configurable properties for this node are:

| Property | Usage |
| --- | --- |
| **Node name** | The name given to this node in the Journey.<br>Default: `Identity Assertion`. |
| **Identity Assertion server ID** | The ID of the Identity Assertion server that handles assertion requests. The ID is composed of the server's ID and realm (if realm-scoped). |
| **Mapping to server claims (optional)** | Mapping of:<br>• Key: Shared node state key<br>• Value: Identity request JWT claim<br><br>Required only if the server requires additional data.<br>When a shared node state attribute has a value for a mapped key, the value is added to the identity request JWT claims according to the corresponding claim. |

| Property | Usage |
| --- | --- |
| **Mapping from server result (optional)** | Mapping of:<br>• Key: Identity Assertion JWT claim<br>• Value: Shared node state key<br><br>Required only if the server requires additional data.<br>Default: the JWT `principal` claim is mapped to the shared node state `username` attribute.<br>When an Identity Assertion JWT claim has a value for a mapped claim, the value is added to the shared node state according to the corresponding shared node state key. |

## Outputs

Any data mapped from the claims returned by the Identity Assertion server stored in the shared node state of the journey.

### *Successful Identity Assertion*

The configuration `Mapping from server result (optional)` determines the shared node state property to set for the mandatory claim `principal`. The value of the shared node state property is set with the value of the `principal` claim.

For example, if `principal` is mapped to `usernameReceived`, the attribute `usernameReceived` is set in the shared node state. By default, `principal` is mapped to `username`.

Other values mapped in `Mapping from server result (optional)` are set in the shared node state only if the claim exists in the resulting Identity Assertion JWT.

### *Failed Identity Assertion*

The shared node state property `error` is set with the value of the `error` claim in the resulting Identity Assertion JWT.

## Outcomes

### Success

The Identity Assertion server indicates that authentication was successful. It provides the authenticated `principal`.

### Error

The Identity Assertion server indicates that authentication failed. It provides information about the error.

## Troubleshooting

If the node logs an error, review the log to find the reason for the error.

## Example

The following worked example describes how to use the Identity Assertion node to authenticate internal access.

### Create and import a secret encryption key

Advanced Identity Cloud

Identity Assertion in AM uses a single secret for all encryption and decryption:

- Advanced Identity Cloud uses the key to encrypt the identity request JWT; PingGateway uses it to decrypt the identity request JWT.
- PingGateway uses the key to encrypt the resulting Identity Assertion JWT; Advanced Identity Cloud uses it to decrypt the Identity Assertion JWT.

Provide the encryption key as a PEM-encoded AES secret key or `base64aes`-encoded secret key ESV.

The example route in Configure PingGateway as an Identity Assertion Server uses PEM for PingGateway and imports the secret key value as an ESV into Advanced Identity Cloud or AM.

Configure a key for that route as described in this section. Learn more in Map ESV secrets to secret labels.

Create a PEM key and import it into the tenant ESV

1. Generate a random 32-byte AES secret-key:

```
$ head -c32 /dev/urandom | base64
Rf8...Ig=
```

2. Put the key string into a .pem file called `idassert.pem`, for use on the PingGateway side.

```
-----BEGIN AES SECRET KEY-----
Rf8...Ig=
-----END AES SECRET KEY-----
```

   In the following steps, you import the raw key as an ESV and configure it in an ESV secret store.

3. Using information in Authenticate to Identity Cloud REST API with access token, get an access token for your tenant.

4. Using the access token, import the new ESV into your tenant. The following example imports an ESV named `esv-idassert`:

```
$ export VALUE=$(echo -n Rf8…Ig= | base64)

$ curl -X PUT "<tenant-env-fqdn>/environment/secrets/esv-idassert" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -H "Accept-API-Version: protocol=1.0;resource=1.0" \
  --data-raw "{\"encoding\":\"base64aes\",\"useInPlaceholders\":false,\"valueBase64\":\"$VALUE\"}"
```

   Consider the following points for ESVs:

   - Prefix the name with `esv-`.
   - Don't suffix the name with a number.

- Use `base64aes` encoding. PEM encoding is also available.

5. Using information in Tenant settings, check that the ESV is displayed in the **Environment Secrets and Variables** panel.

PingAM

Identity Assertion in AM uses a single secret for all encryption and decryption:

- PingAM uses the key to encrypt the identity request JWT; PingGateway uses it to decrypt the identity request JWT.
- PingGateway uses the key to encrypt the resulting Identity Assertion JWT; PingAM uses it to decrypt the Identity Assertion JWT.

Provide the encryption key in PEM format, as a JWK, or in a keystore. For example, create and import an AES PEM key into a secret store.

**Configure the Identity Assertion service**

**Enable the service**

Advanced Identity Cloud

1. In the Advanced Identity Cloud admin UI, go to **Native Consoles** > **Access Management** > **Realms** > ***Realm name*** > **Services**.
2. Click **+Add a Service** and select **Identity Assertion Service** to create.
3. In the Identity Assertion Service page, ensure **Enable** is selected.

PingAM

1. In the AM admin UI, go to **Configure** > **Global Services** > **Identity Assertion Service**.
2. Alternatively, to add the service for a realm, go to **Realms** > ***Realm name*** > **Services**, click **+Add a Service** and select **Identity Assertion Service** to create.
3. In the Identity Assertion Service page, ensure **Enable** is selected.

### Configure a server

1. In the `Secondary Configurations` tab, click **+Add a Secondary Configuration** and enter the following information:
   - **Name**: A unique name for the Identity Assertion server. For example, use `IG01`.
   - **Identity Assertion server URL**: The Identity Assertion server URL. For example, enter `https://ig.ext.com:8443`.
   - **Shared Encryption Secret**: Advanced Identity Cloud and AM use this identifier to create a secret label for encrypting the identity request JWT and resulting Identity Assertion JWT.

     The secret label takes the form `am.services.identityassertion.service.identifier.shared.secret` where identifier is the value of **Shared Encryption Secret**. For example, use identifier `idassert` to create a label called `am.services.identityassertion.service.idassert.shared.secret`.
2. Click **Create**.
3. Keep the default values for **JWT TTL (seconds)** and **Skew Allowance (seconds)** and save your changes.

Learn more in the Identity Assertion service documentation for:

- PingAM
- Advanced Identity Cloud

### Map the secret label to the encryption key

Advanced Identity Cloud

The example route in Configure PingGateway as an Identity Assertion Server uses PEM for PingGateway and imports the secret key value as an ESV into Advanced Identity Cloud or AM .

1. Log in to your Advanced Identity Cloud admin UI, and go to **Native Consoles > Access Management**.
2. In the Realm Overview page, click **Secret Stores**.
3. Map the secret label used by the Identity Assertion service to the ESV secret store.

   In the `Mappings` tab, enter the following information:
   - **Secret Label**: Enter the value for the **Shared Encryption Secret** you created in the Identity Assertion Service. For example, enter `idassert`.

     You can configure the secret only after you have named it in the Identity Assertion service secondary configuration.

     The full-length secret label is automatically constructed from the value. In this example, the full-length secret label is `am.services.identityassertion.service.idassert.shared.secret`.
   - **Aliases**: Enter the alias to the secret you created earlier. For example, enter `esv-idassert`.

PingAM

To map the encryption key in the secret store, follow the steps in Map and rotate secrets using these values:

- **Secret Label**: Find the secret label to map by entering the value of the **Shared Encryption Secret** you used in the service configuration.

  For example, enter `idassert` to find `am.services.identityassertion.service.idassert.shared.secret`.

  You can find and configure the secret only after you have entered it in the **Shared Encryption Secret**.

- **Aliases**: Enter the alias to the encryption key secret you created earlier.

### Configure PingGateway as an Identity Assertion Server

Configure PingGateway to:

- Validate the identity request JWT.
- Create an encrypted Identity Assertion JWT to send back to Advanced Identity Cloud or AM.

The PingGateway configuration includes two routes:

#### *Authentication filter route*

Directs unauthenticated requests to an authentication journey.

For testing purposes, configure Advanced Identity Cloud or AM and PingGateway for cross-domain single sign-on:

Advanced Identity Cloud

1. Follow the steps in Cross-domain single sign-on.

   The setup configures a demo user and validation service required for the example.

2. In `cdsso-idc.json`, the CrossDomainSingleSignOnFilter uses Advanced Identity Cloud's default authentication service. Add the property `authenticationService` to the CrossDomainSingleSignOnFilter to direct requests to the journey. ---

PingAM

1. Follow the steps in Cross-domain single sign-on (CDSSO).

   The setup configures a demo user and validation service required for the example.

2. In `cdsso.json`, the CrossDomainSingleSignOnFilter uses AM's default authentication service. Add the property `authenticationService` to the CrossDomainSingleSignOnFilter to direct requests to the journey.
   ---

The following example redirects unauthenticated requests to a journey called `IgCallout`.

```
{
  "name": "CrossDomainSingleSignOnFilter-1",
  "type": "CrossDomainSingleSignOnFilter",
  "config": {
    ...
    "authenticationService" : "IgCallout",
    ...
  }
```

### *Identity Assertion service route*

Directs unauthenticated requests to a local authentication service such as Kerberos or Windows Desktop SSO.

Consider the example in PingGateway's Example Identity Assertion service route for Identity Assertion node. The route contains an `IdentityAssertionHandler` that calls a `ScriptableIdentityAssertionPlugin` to manage local authentication.

The route requires the following:

- The key and PingAM / Advanced Identity Cloud setup described in this worked example.
- That the `IdentityAssertionHandler`'s `peerIdentifier` property refers to the host:port part of the deployment URL (PingAM deployments) or the host part of the tenant URL (Advanced Identity Cloud tenants).
- That the `IdentityAssertionHandler`'s `condition` refers to the same path as the `Route` configured in the node. In this example, it refers to `/idassert`.

### Configure the example authentication journey

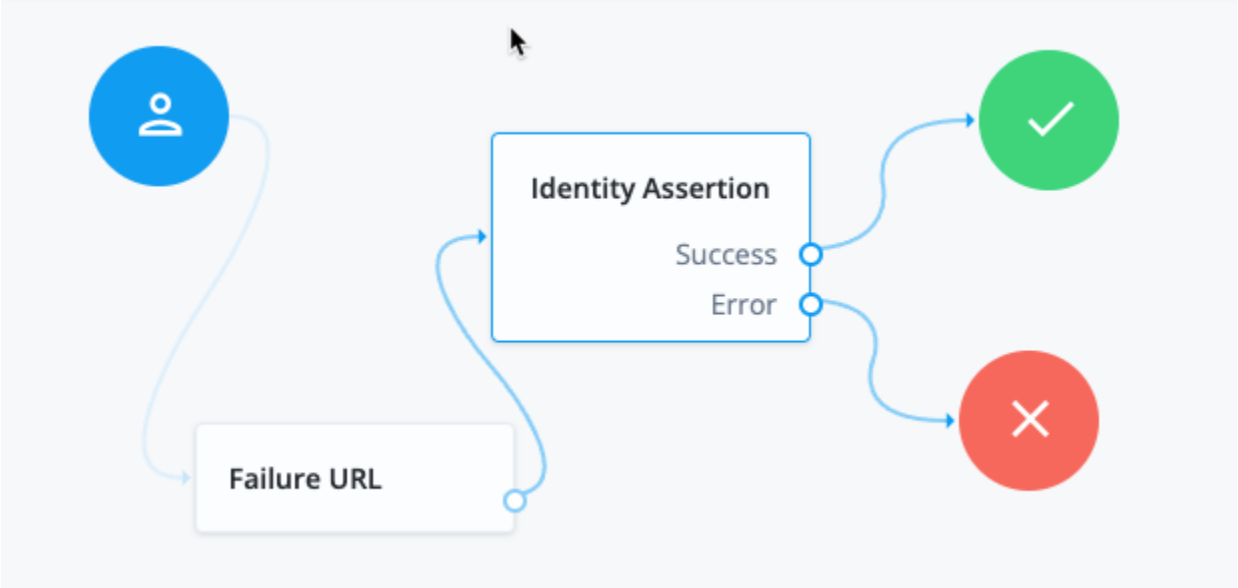



*Figure 1. Advanced Identity Cloud*

**Note**

Add a Failure URL node to manage the journey flow if assertion fails. Configure the node with a URL to use for failed requests. For example, the following URL returns PingGateway to the CDSSO redirect endpoint:

```
https://ig.ext.com:8443/home/cdsso/callback?
error=Login%20failed&error_description=Identity%20Assertion%20Failure
```

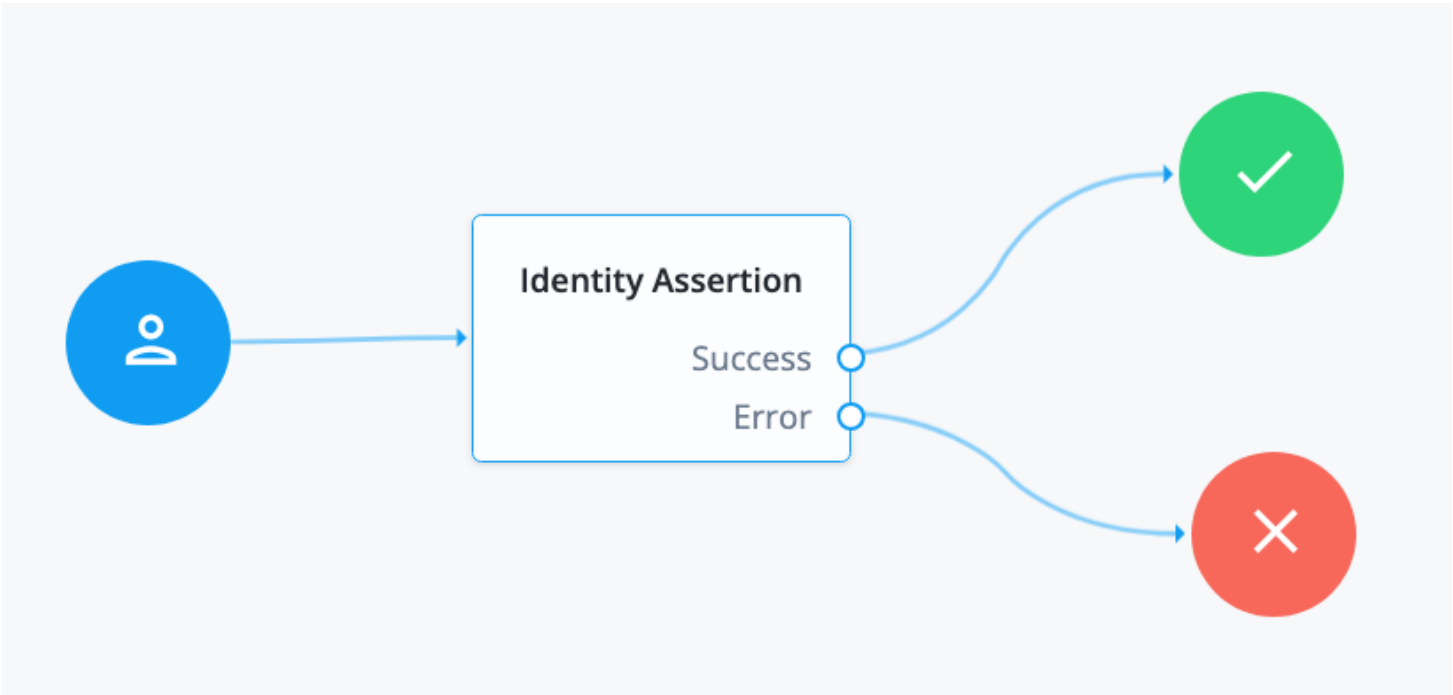



*Figure 2. PingAM*

Configure the Identity Assertion node as follows:

- **Identity Assertion server ID**: Select the ID and realm configured for the PingGateway server that supports Identity Assertion. For example, enter `IG01 [/alpha]`, where `IG01` is the name of the server created in the Configure the Identity Assertion service.
- **Route**: Enter the value of the `condition` property in the PingGateway route that will handle Identity Assertion requests. For example, enter `/idassert`, as used for the example route in Configure PingGateway as an Identity Assertion Server.

  When a request matches the path `/idassert`, the journey accesses the PingGateway route in PingGateway's Example Identity Assertion service route for IdentityAssertionNode.

## SpyCloud Auth Node

Use the SpyCloud Auth node to integrate the SpyCloud service with Advanced Identity Cloud and assess if a user's password is compromised. If a user's password is compromised, the administrator can take remedial action, such as denying access and asking the user to reset their password, or forcing a password reset.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

> **Note**
>
> AM customers can get the SpyCloud node here.

### Inputs

This node reads a customizable user identifier from shared state. The identifier is set in the `identifierSharedStateKey` property in the node configuration.

### Dependencies

To use this node, you should have already set up integration with the SpyCloud service in PingOne Advanced Identity Cloud. Refer to the SpyCloud documentation to set up the SpyCloud service in your environment.

### Configuration

| Property | Usage |
| --- | --- |
| **API URL** | The SpyCloud API URL. |
| **API Key** | The SpyCloud API key. |
| **Severity** | Allows you to filter based on the numeric severity code. You can find more information about severity codes in SpyCloud documentation. |
| **identifierSharedStateKey** | The shared state key to find the identifier value. |

### Outputs

Only when an error occurs does this node store the error in shared state.

### Outcomes

**`Compromised`**

Compromised password detected.

**`Not compromised`**

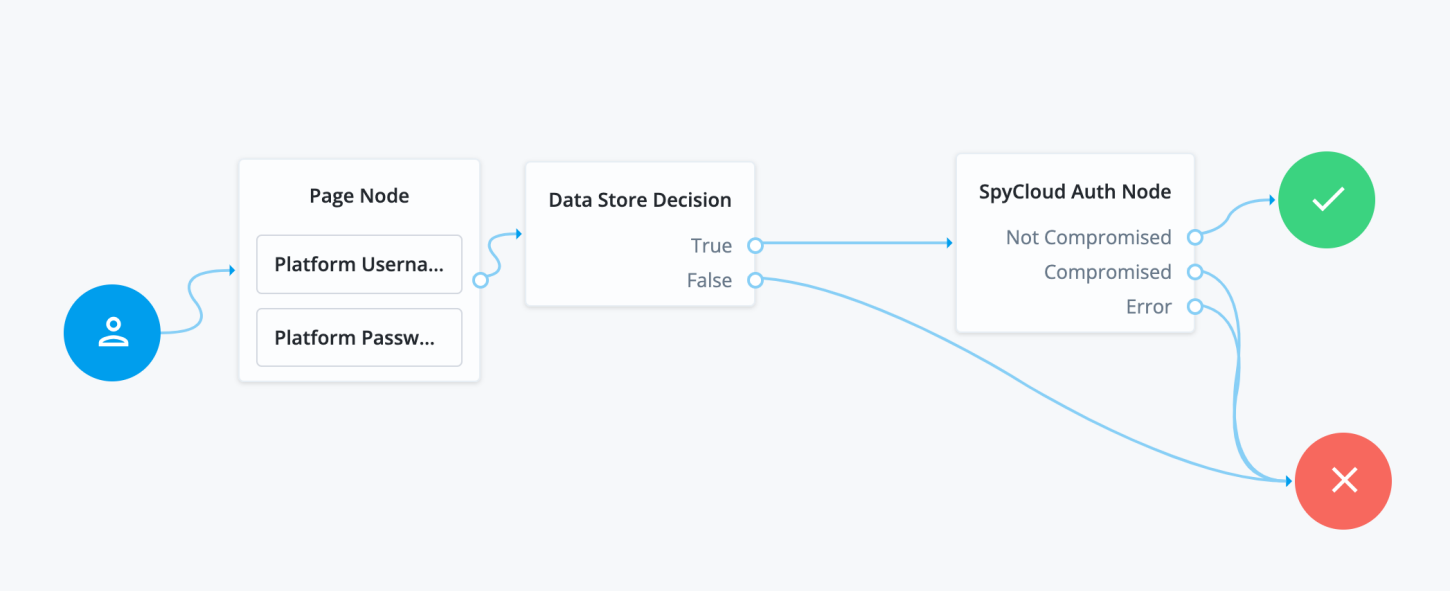
Password is not compromised.

**`Error`**

There was an error while assessing if the password was compromised.

### Example

The following sample journey illustrates the use of this node to assess if a user's password is compromised:




## Troubleshooting

If this node logs an error, then an error message is sent to shared state. For example:

- `"[SpyCloud] StackTrace", new Date() + ": " + stackTrace`
- `"[SpyCloud] Exception", new Date() + ": " + ex.getMessage()`

Review the log messages to find the reason for the error and address the issue appropriately.

# Thing nodes

## Authenticate Thing node

This node authenticates a *thing*. A thing represents an IoT device, service, or the IoT Gateway.

### Dependencies

Before you configure this node, you must configure the IoT Service for the realm.

> **Important**
>
> Support for this node is provided by the IoT SDK.

The node supports two methods of authentication:

1. Proof of Possession JWT

   The node collects a proof-of-possession JWT from the request and does the following:

   - Checks that the claims are valid.
   - Checks that an identity with the same ID as the name of the JWT subject exists.
   - Checks that the identity contains a confirmation key that matches the JWT `kid`.
   - Validates the JWT signature, using the confirmation key stored in the identity.

2. Client Assertion

   The node collects a JWT Bearer token from the request for authentication and validates the request according to the JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

- `Success`
- `Failure`
- `Requires Registration`

If all checks are successful, evaluation continues through the `Success` path, and adds the username and the verified claims to the shared node state.

If the identity does not exist, or AM cannot match the identity with the confirmation key, evaluation continues through the `Requires Registration` outcome.

If any other check fails, evaluation continues through the `Failure` outcome.

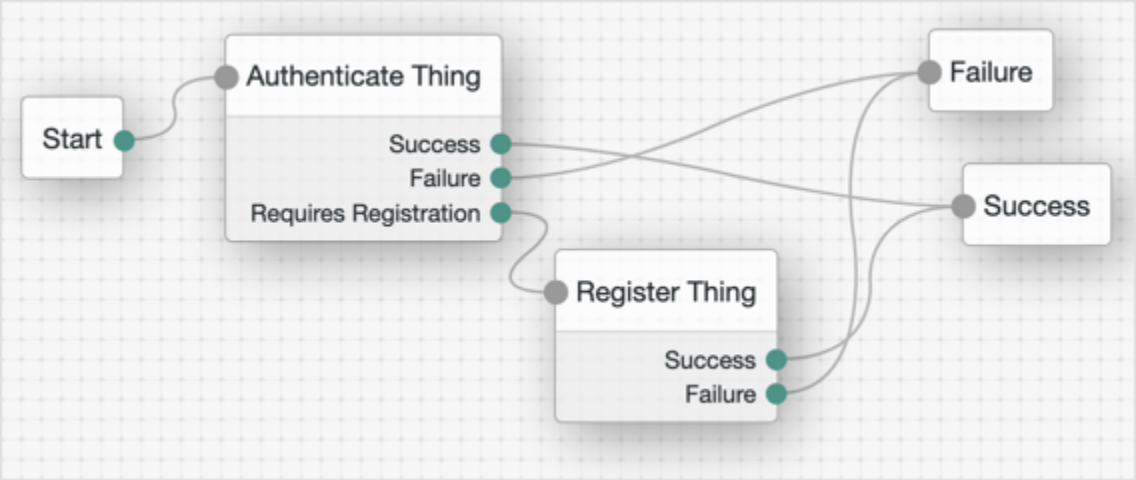
## Configuration

| Property | Usage |
|---|---|
| **JWT Authentication Method** | Choose the required JWT authentication method:<br>***Proof of Possession***<br>Prove that the signer of the JWT is the owner of the key by including a challenge nonce in the JWT. Validation is according to the JWT Proof of Possession specification.<br>***Client Assertion***<br>Present a JWT Bearer token for authentication and validate the request according to the JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants. |
| **Issue Restricted Token** | If this setting is enabled, the node adds a Proof of Possession restriction to the session token issued on successful authentication.<br>Any requests accompanied by the token must be signed with the key that was used to sign the authentication JWT. |
| **Additional Audience Values** | Any additional audience values that will be permitted when verifying JWTs.<br>These audience values are in addition to the AM base, issuer and token endpoint URIs for the Client Assertion authentication method or the realm path for Proof of Possession. |

## Examples

The following example shows how to authenticate a thing when the identity already exists in the identity store and when its profile contains a confirmation key:


The following example shows how to authenticate a thing when the identity does not exist, or when it needs to refresh its confirmation key:




## Register Thing node

This node registers a *thing*. A thing represents an IoT device, service, or the IoT Gateway.

### Dependencies

Before you configure this node, you must configure the IoT Service for the realm.

> **Important**
>
> Support for this node is provided by the IoT SDK.

The node collects a JWT from the request and validates the JWT according to the configured JWT registration method.

If the JWT is valid, the node uses the claims in the JWT to create an identity for the thing and register (or rotate) a confirmation key for it. Then, evaluation continues through the `Success` outcome.

If the node cannot validate the JWT, evaluation continues through the `Failure` outcome.

For an example on how to use this node, refer to Authenticate Thing node.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | No |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Outcomes

- `Success`
- `Failure`

## Configuration

| Property | Usage |
|---|---|
| **JWT Registration Method** | Choose the method to validate the JWT:<br>***Proof of Possession & Certificate***<br>Register using a Proof of Possession JWT that includes an X.509 certificate for providing trust. A challenge nonce is presented in the callback and must be included in the signed JWT.<br>***Proof of Possession & Software Statement***<br>Register using a Proof of Possession JWT and a Software Statement for providing trust. A challenge nonce is presented in the callback and must be included in the signed Proof of Possession JWT. The claims in the Software Statement take precedence over the claims in the Proof of Possession JWT.<br>***Proof of Possession***<br>Register using a Proof of Possession JWT without using a trusted third party. A challenge nonce is presented in the callback and must be included in the signed JWT.<br>***Software Statement***<br>Register using a Software Statement, without doing proof of possession. If you select this registration method, the resultant session token will not include a proof of possession restriction.<br>Default: Proof of Possession & Certificate |
| **Verify Certificate Subject** | If the configured JWT registration method is `Proof of Possession & Certificate`, this option verifies that the subject provided in the JWT is the same as the X.509 certificate subject CN or UID.<br>Default: Enabled |

| Property | Usage |
| --- | --- |
| **Create Identity** | Whether AM creates an ID for the thing if one doesn't exist.<br>Default: Disabled |
| **Rotate Confirmation Key** | Whether multiple confirmation keys can be registered for a thing. Disable this setting to allow only one key per thing.<br>Default: Disabled |
| **Default Attribute Values** | The default values for the thing's attributes, where **KEY** is the name of the attribute in the data store, and **VALUE** is the default value of the attribute. |
| **Claim to Attribute Mapping** | If **Create Identity** is enabled, this property lets you map verified claims in the JWT to attributes in the thing identity. **KEY** is the claim name and **VALUE** is the name of the attribute in the data store. |
| **Overwrite Attributes** | Whether the node overwrites the value for an existing profile attribute when a claim with a different value is provided in the JWT.<br>Default: Disabled |

# Marketplace nodes

## PingOne integration

### PingOne Service

The PingOne Service lets you set up the PingOne service in your Advanced Identity Cloud tenant so that you can add PingOne nodes to your authentication journeys. You must set up this service in your Advanced Identity Cloud tenant before using the PingOne nodes in an authentication journey.

#### Configure the PingOne service

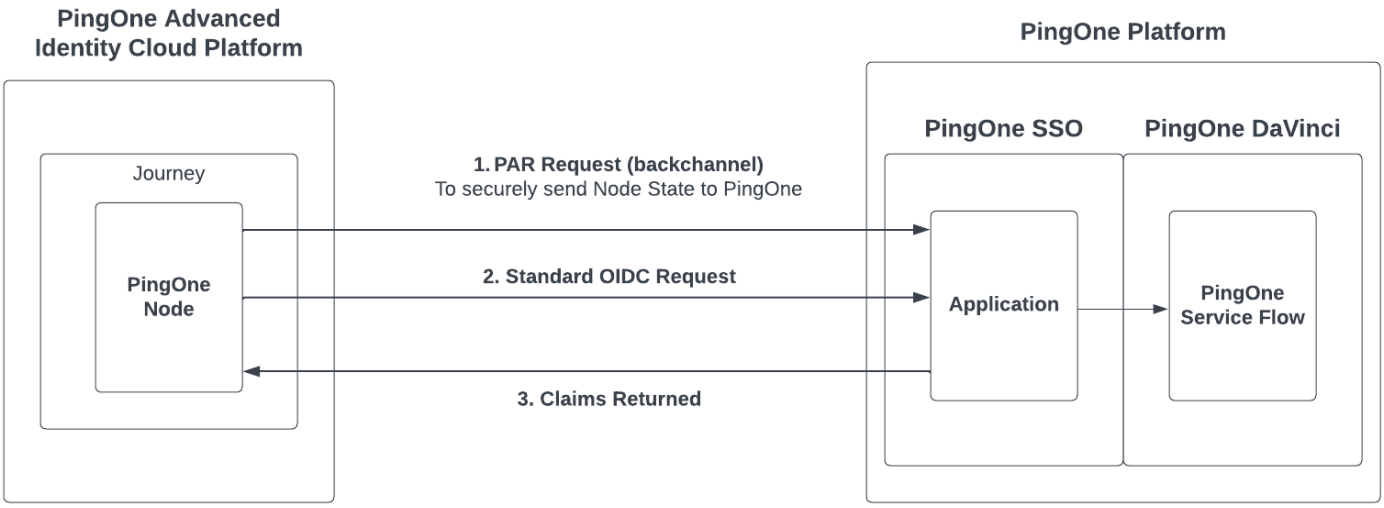
1. Sign on to your Advanced Identity Cloud admin UI, and go to **Native Consoles > Access Management**.
2. In the Realm Overview page, click **Service Management**.
3. Click **+ Add a Service**.
4. Select **Ping One Service** from the **Choose a service type** menu, and click **Create**.
5. In the Ping One Service page, ensure that **Enable** is selected.
6. Click **Save Changes**.
7. Go to the **Secondary Configuration** tab, in the **New PingOne Service configuration** page, and configure these parameters:
    1. **Environment ID**: Your Advanced Identity Cloud environment ID
    2. **Environment Region**: The region in which your Advanced Identity Cloud environment is located
    3. **PingOne Client ID**: Your PingOne node client ID
    4. **PingOne Node Client Secret**: Your PingOne node client secret
    5. **PingOne Node Redirect URL**: Your PingOne node redirection URL
    6. **PingOne DaVinci API Key**: Your PingOne DaVinci API key
    7. **Worker Application Client ID**: The client ID of your worker application
    8. **Worker Application Client Secret**: The client secret of your worker application
8. Click **Create**.

### PingOne node

The **PingOne** node establishes trust between PingOne and PingOne Advanced Identity Cloud by using a federated connection.

This node performs an OIDC request to PingOne to delegate the user flow from Advanced Identity Cloud to PingOne using a standard OIDC redirect.

> **Note**
>
> Use this node only if you need to configure PingOne as an external identity provider for Advanced Identity Cloud or to execute a PingOne DaVinci flow containing UI screens. In all other cases, use the PingOne DaVinci API node instead.




### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

Any data in the node state that needs to be sent to PingOne.

### Dependencies

Before using the PingOne node, you must do the following:

- Configure a PingOne OIDC application to connect to Advanced Identity Cloud
- Configure the PingOne Service
- (Optional) If you plan to trigger a DaVinci flow from the PingOne node, configure the PingOne application to the DaVinci Flow policy.

### Configure a PingOne OIDC application to connect to Advanced Identity Cloud

Use the **Applications** page in the PingOne interface to add an application to connect to Advanced Identity Cloud.

1. Go to **Applications > Applications**.
2. Click **+**.
3. Create an application profile with these parameters:
    1. **Application name**: Advanced Identity Cloud Federation.
    2. **Description** (optional): Enables Advanced Identity Cloud federation with PingOne.
    3. Select `OIDC Web App` as the **Application Type**.
4. Click **Save**.
5. After the application profile is created, go to the **Configuration** tab and click the pencil icon to edit the application.
    1. In the **PKCE Enforcement** the drop-down, select `S256_REQUIRED`.
    2. In the **Token Endpoint Authentication Method** drop-down, select `Client Secret Basic`.
    3. Select **Require Pushed Authorization Request**.
    4. Enter the **Redirect URIs** of your Advanced Identity Cloud AM instance.
6. Click **Save**, and then select **Enable**.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Service** | The ID of the PingOne Worker service for connecting to PingOne. |
| **ACR Values**(optional) | For triggering a specific PingOne application policy. |
| **Username** | The attribute that contains the name of the user for the object. |
| **State Inputs** | A multi-value field to select specific attributes from node state to include in the federation request to PingOne. By default, the wildcard (*) value includes the entire journey node state in the federation request to PingOne. |

### Outputs

Any claims returned by PingOne during federation will be stored in the node state.

### Outcomes

#### `Account exists`

If the account returned by PingOne during federation matches an existing account, and it is linked to the account in Advanced Identity Cloud.

**`Account exists, no link`**

If the account returned by PingOne during federation exists in Advanced Identity Cloud, but it is not yet linked to the existing account in Advanced Identity Cloud.

**`No account exists`**

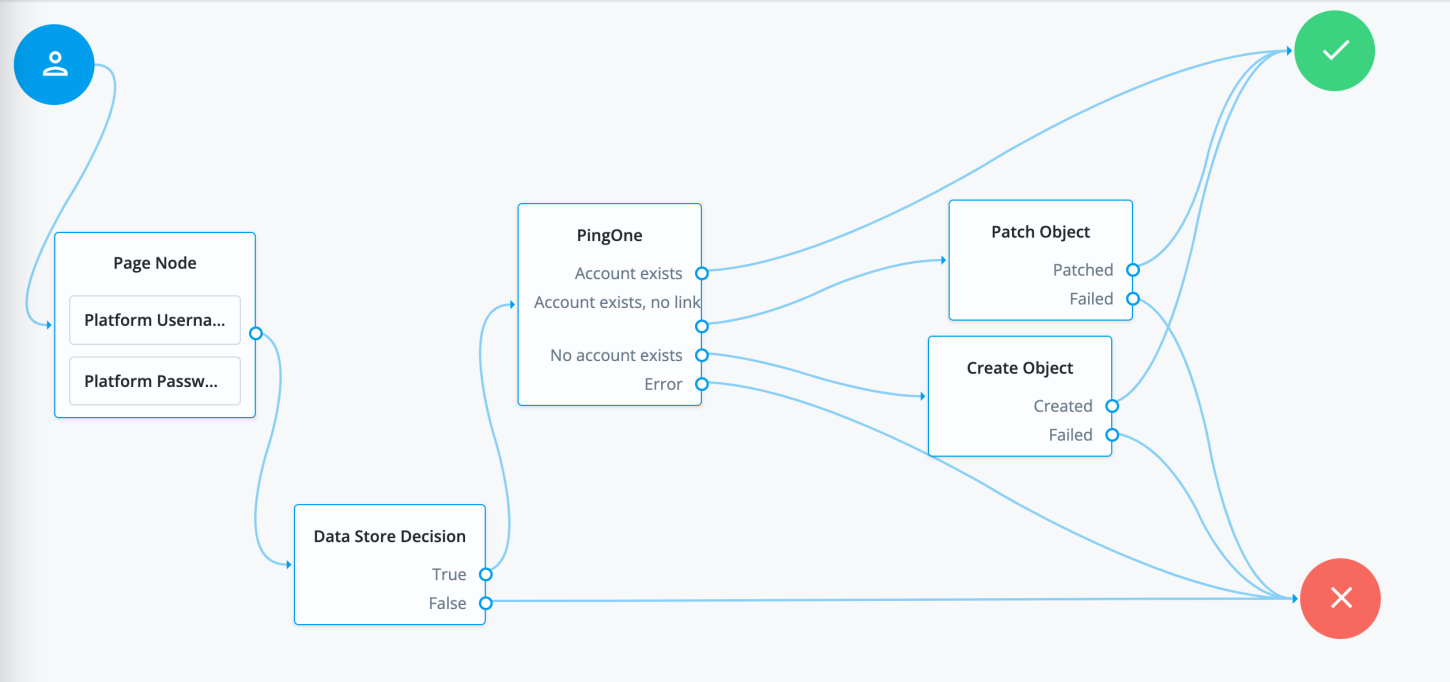
If the account returned by PingOne during federation does not exist in Advanced Identity Cloud.

**`Error`**

An error occurred causing the request to fail. Check the response code, response body, or logs to see more details of the error.

### Examples

This example journey highlights the use of the PingOne node:




## PingOne Authorize node

The **PingOne Authorize** node sends a decision request to a specified decision endpoint in your PingOne Authorize environment. These authorizations include:

- Policy decision authorization
- Individual requests

To use the PingOne Authorize node, you must first set up the PingOne Service.

### Availability

| Product | Available |
|---|---|
| Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes (download from Marketplace) |
| Ping Identity Platform (self-managed) | Yes (download from Marketplace) |

### Inputs

This node retrieves the attribute map from the shared state.

Additionally, the node first attempts to locate in shared state the `PingOne Authorize Policy Attribute(s)` defined in the policy that corresponds to the active decision endpoint.

### Dependencies

You must set up the following before using the PingOne Authorize node:

- Create an authorization policy
- Create a worker application
  - Requires Identity Data Admin role
- PingOne Worker service

### Configuration

| Property | Usage |
|---|---|
| **PingOne Worker Service** | Service for specific PingOne Worker application. |
| **Decision Endpoint ID** | The Decision Endpoint ID from the PingOne Authorize service. |
| **Attribute Map** | The attribute map is to overcome the name differences between shared state attributes in PingOne Advanced Identity Cloud and the request parameters in PingOne. For example, if the shared store `firstName` refers to `givenName` in PingOne, then the `Attribute Map` entry would be: `firstName ⇒ givenName`. |
| **Statement Codes** | Set the node outcomes based on the statements from the PingOne Authorize decision. |
| **Continue** | Use the `Continue` toggle for a single outcome case. |

### Outputs

This node produces no outputs.

### Outcomes

#### `Permit`

Satisfied the active policy's permit condition and authorized the user.

#### `Deny`

Satisfied the active policy's deny condition and did not authorize the user.

#### `Indeterminate`

Does not satisfy the active policy's permit or deny conditions.

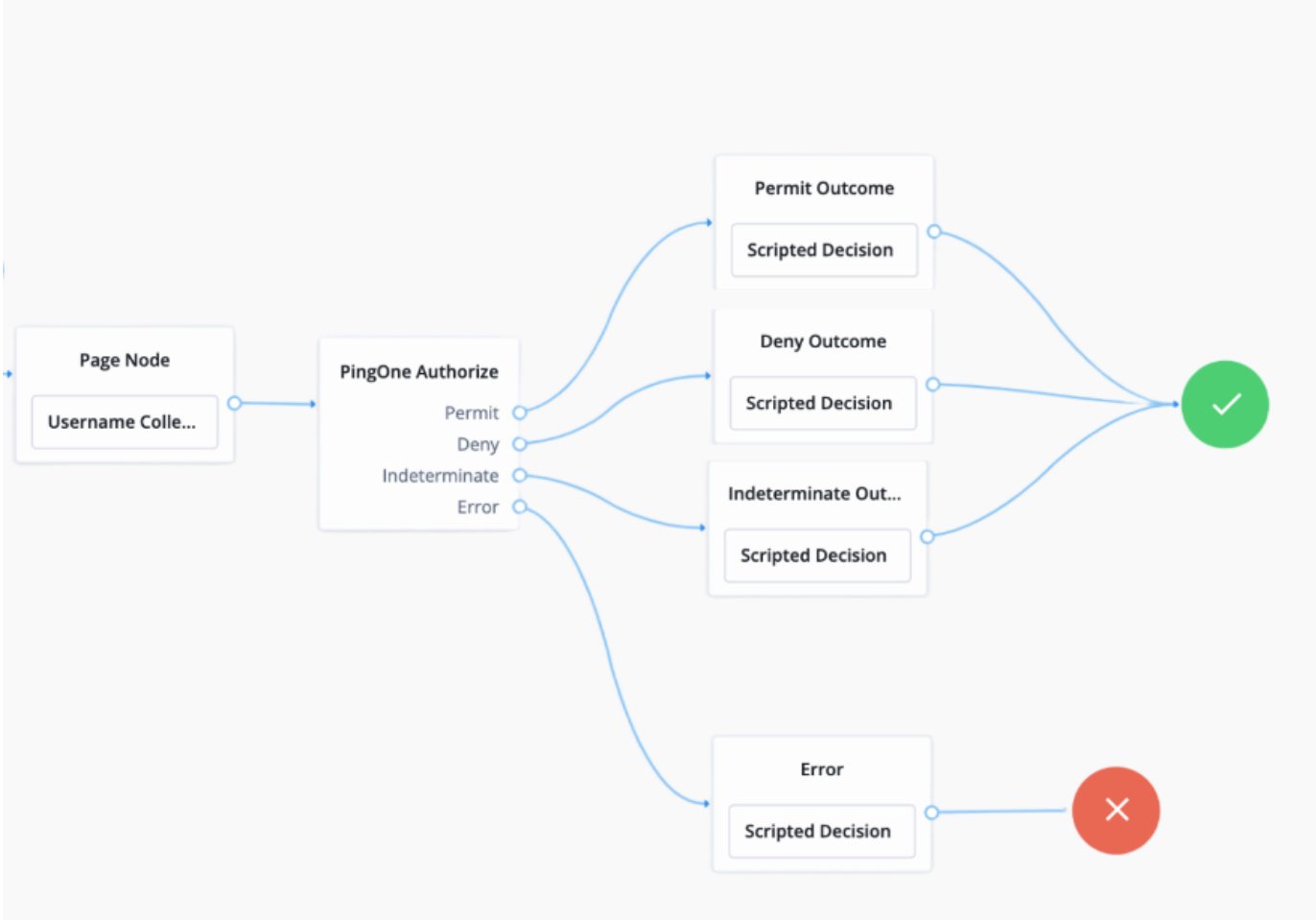
#### `Error`

There was an error during the authorization process.

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately.

### Example

The following example journey illustrates how to use the PingOne Authorize node:



The PingOne Authorize node gets the username from Username Collector node and evaluates the level authorization for the user. Based on the authorization level, further action is taken.

## PingOne Credentials nodes

The PingOne Credentials nodes use the PingOne Credentials service to implement digital wallet pairing, credential management, and credential verification. You can find detailed information about PingOne Credentials service in Introduction to PingOne Credentials

The PingOne Credentials scenarios provide high-level examples of how you can use the service.

To use the PingOne Credentials nodes in your authentication journeys, PingOne Advanced Identity Cloud provides the following artifacts :

- PingOne Worker service
- PingOne Credentials Find Wallets node
- PingOne Credentials Pair Wallet node
- PingOne Credentials Delete Wallet node
- PingOne Credentials Issue node
- PingOne Credentials Update node
- PingOne Credentials Revoke node
- PingOne Credentials Verification node

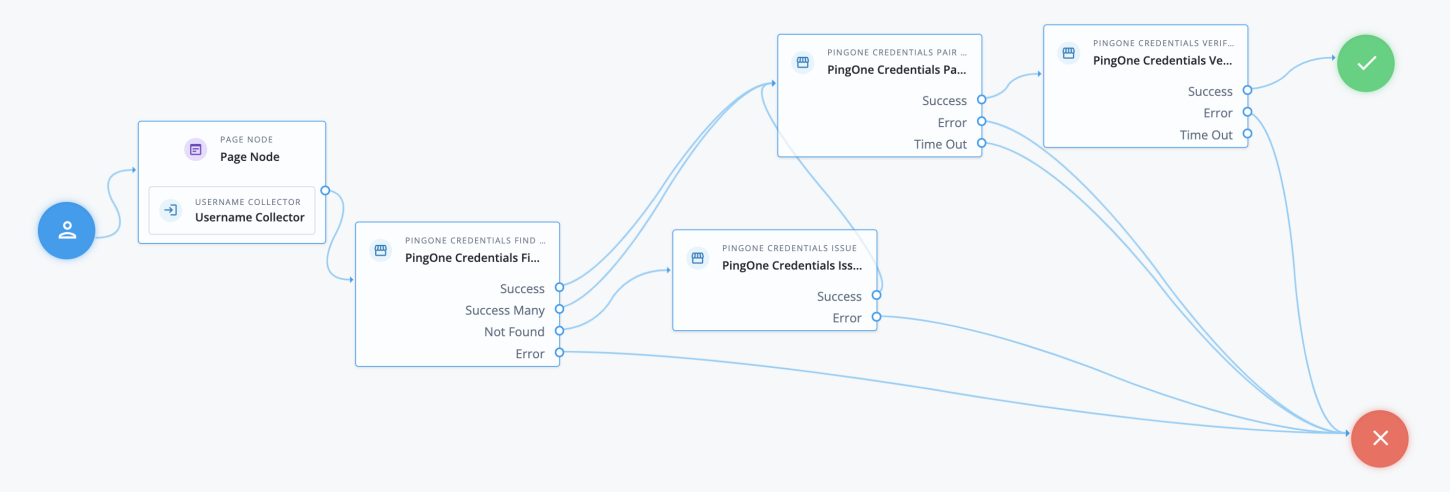
### Dependencies

Before using the PingOne Credentials nodes, complete the following tasks:

- Creating a credential
- Adding a worker application
    - Requires setting up `Identity Data Admin` role
- Configure the PingOne service

### Example

The following example journey shows how to use the PingOne Credentials nodes in a journey:




- The PingOne Credentials Find Wallets node searches for actively paired wallets for an end-user.
- The PingOne Credentials Pair Wallet node initiates a request for pairing a digital wallet through an SMS or email to the user, or a QR code that the user must scan with their digital wallet application.
- The PingOne Credentials Issue node creates and issues a new credential to the paired wallet using the attributes defined in the shared state.
- The PingOne Credentials Verification node initiates a verification request through a QR code the user must scan or a push notification to their digital wallet application.

## PingOne Credentials Delete Wallet node

The **PingOne Credentials Delete Wallet** node lets you remove a paired digital wallet from a PingOne user.

### Inputs

This node retrieves the `pingOneUserId` and `pingOneWalletId` from the journey state.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and remove a paired digital wallet from the PingOne user.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Worker service ID** | The ID of the PingOne Worker service for connecting to PingOne. |
| **PingOne UserID Attribute** | Local attribute name from which to retrieve `pingOneUserId`. The journey state is first looked up, then the local datastore. |

| Property | Usage |
| --- | --- |
| **Digital Wallet ID Attribute** | Local attribute name to retrieve the digital wallet ID from the journey state. |

### Outputs

None

### Outcomes

#### Success

The wallet is successfully removed.

#### `Not Found`

No digital wallet was found to remove.

#### `Error`

There was an error during the wallet removal process.

### Errors

If the API call to PingOne Credentials fails, the following error is logged:

- `Error: PingOne Credentials Delete a Digital Wallet - Status Code - Response Body`

### Example

Learn more in an example journey using Pingone Credentials nodes.

## PingOne Credentials Find Wallets node

The **PingOne Credentials Find Wallets** node lets you list all paired digital wallets from the PingOne user.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

To perform the wallet look up, this node requires the PingOne User ID (UUID). The node first attempts to get the UUID from the `pingOneUserId` attribute. If `pingOneUserId` attribute is not set, then it gets the UUID from the `pingOneUserId` as a child attribute in `objectAttributes` in the shared state.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and list the paired wallets of a user.

### Configuration

| Property | Usage |
| --- | --- |
| **PingOne Service** | The ID of the PingOne Worker service for connecting to PingOne. |
| **PingOne UserID Attribute** | Local attribute name from which to retrieve `pingOneUserId`. The journey state is first looked up, then the local datastore. |

### Outputs

- `pingOneWalletId`: The PingOne digital wallet ID.
- `pingOneApplicationInstanceId`: The Application Instance ID of the digital wallet where the credential was stored.
- `pingOneActiveWallets`: The PingOne User's active digital wallets.

### Outcomes

#### `Success`

One digital wallet was found and returned.

#### `Success Many`

All digital wallets were found and listed.

#### `Not Found`

No digital wallet was found.

#### `Error`

There was an error during the process of finding the wallet.

### Errors

If the API call to PingOne Credentials fails, the following error is logged:

- Error: `PingOne Credentials Find a Wallet- Status Code - Response Body`

### Example

Learn more in an example journey using PingOne Credentials nodes.

## PingOne Credentials Issue node

The **PingOne Credentials Issue** node lets you create a PingOne credential in a journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node retrieves from the shared state the `pingOneUserId` and shared state attributes defined in the attribute map.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and create a PingOne credential.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Worker service ID** | The ID of the PingOne Worker service for connecting to PingOne. |
| **PingOne UserID Attribute** | Local attribute name from which to retrieve `pingOneUserId`. The journey state is first looked up, then the local datastore. |
| **Credential Type ID** | The requested credential name |
| **Attribute map** | The `Key` - `Value` mapping used for associating journey state attributes with credentials. The `Key` is the P1 Credential attribute, and the `Value` is the shared state attribute. |

### Outputs

`pingOneCredentialId` - The ID of the created credential.

### Outcomes

#### Success

The required credential was created and issued successfully.

#### Error

There was an error during the process of creating credentials.

### Errors

If the API call to PingOne Credentials fails, the following error is logged:

- Error: PingOne Credentials Create a User Credential

### Example

Learn more in an example journey using PingOne Credentials nodes.

## PingOne Credentials Pair Wallet node

The **PingOne Credentials Pair Wallet** node lets you pair PingOne digital wallet credentials with a PingOne user ID.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node retrieves `pingOneUserId` from the journey state, or from `objectAtrributes` in the journey state.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and pair digital wallet credentials with a Ping user ID.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Worker service ID** | The ID of the PingOne Worker service for connecting to PingOne. |

| Property | Usage |
| --- | --- |
| **PingOne UserID Attribute** | The local attribute name from which to retrieve the PingOne userID. The journey state is first searched for the PingOne User ID. If the ID is not found, the localdatastore is looked up. |
| **PingOne Wallet Application ID** | Digital Wallet Application ID from PingOne Credentials admin UI. |
| **Digital Wallet Pairing URL delivery method** | The method—QR Code, email, or SMS used to deliver the digital wallet. |
| **Allow user to choose the URL delivery method** | If enabled, prompt the user to select the URL delivery method. |
| **Delivery method message** | The message to display to the user, so that they can select the method—QRCODE, SMS, or EMAIL, used to receive the pairing URL. |
| **QR code message** | The message with instructions to scan the QR code to begin the digital wallet pairing process. |
| **Submission timeout** | Timeout value, in seconds, for pairing the digital wallet. |
| **Waiting Message** | Localization overrides for the waiting message. This is a mapping of a locale to a message. |
| **Store Wallet Response** | Store the list of verified data submitted by the user in the shared state under a key named `pingOneWallet`.<br>**Note**<br>The key is empty if the node is unable to retrieve the wallet pairing data from the PingOne service. |

### Outputs

**pingOneWallet** - The new PingOne User's digital wallet.

### Outcomes

#### `Success`

All configured checks passed.

#### `Error`

There was an error during the pairing process

#### `Time Out`

The pairing process reached the configured timeout value.

### Errors

If the API call to PingOne Credentials fails, the following error is logged:

- `Error: PingOne Credentials Create a Digital Wallet`

### Example

Learn more in an example journey using Pingone Credentials nodes.

## PingOne Credentials Revoke node

The **PingOne Credentials Revoke** node lets you revoke existing PingOne credentials in a journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node retrieves the `PingOne User ID` and `Credential Type ID` from the shared state.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and revoke PingOne credentials.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Service** | The ID of the PingOne Worker service for connecting to PingOne. |
| **PingOne UserID Attribute** | Local attribute name from which to retrieve `pingOneUserId`. The journey state is first looked up, then the local datastore. |
| **Credential ID Attribute** | The local attribute name to retrieve the credential ID from the shared state. |

### Outputs

None

### Outcomes

#### Success

The credential is successfully revoked.

#### Not Found

No credential was found.

#### Error

There was an error while revoking a credential.

### Errors

If the API call to PingOne Credentials fails, the following error is logged:

- `Error: PingOne Credentials Revoke a User's Credential- Status Code - Response Body`

## PingOne Credentials Update node

The **PingOne Credentials Update** node lets you update a PingOne credential in a journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node retrieves `pingOneUserId` and `pingOneCredentialId` from the journey state.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and update PingOne credentials.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Worker service ID** | The ID of the PingOne Worker service for connecting to PingOne. |

| Property | Usage |
| --- | --- |
| **PingOne UserID Attribute** | Local attribute name from which to retrieve `pingOneUserId`. The journey state is first looked up, then the local datastore. |
| **Credential Type ID** | The requested credential name. |
| **Credential Id Attribute** | The local attribute name to retrieve the credential ID attribute from the shared state. |
| **Attribute map** | The `Key` - `Value` mapping used for associating shared state attributes with credentials. The `Key` is the shared state attribute, and the `Value` is the corresponding credential attribute. |

### Outputs

- `pingOneCredentialUpdate` — The response from the PingOne Credentials Update operation.

### Outcomes

#### Success

The credential is updated.

#### `Error`

There was an error when updating the credential.

### Errors

If the API call to PingOne Credentials fails, the following error is logged:

- `Error: PingOne Credentials Update a User Credential- Status Code - Response Body`

## PingOne Credentials Verification node

The **PingOne Credentials Verification** node initiates verification of PingOne credentials. The actual task of verification is performed by the PingOne Credentials service and not by this node.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

If the `Push` delivery method is selected, this node retrieves `pingOneApplicationInstanceId` from the journey state. If `Custom Requested Credentials` is selected, this node retrieves `requestedCredentials` from journey state.

### Dependencies

This node requires that the PingOne Worker Service is configured so that it can connect to your PingOne instance and initiate PingOne credentials verification.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Worker service ID** | The ID of the PingOne Worker service for connecting to PingOne. |
| **Credential Type** | Type of credential to verify. Must be the name of a PingOne credential type issued by the credential issuer. |
| **Disclosure Attribute Keys** | Attribute key names for selective disclosure to return from the credential. |
| **Digital Wallet Application ID** | Digital Wallet Application ID from PingOne Credentials required for the Push delivery method. |
| **Verification URL Delivery Method** | The delivery method (QR code or Push) for delivering the verification URL. |
| **Allow user to choose the URL delivery method** | To prompt the user to select the URL delivery method. |
| **Delivery method message** | The message to display to the user allowing them to select the delivery method (QR Code or `Push`) to receive the credential verification URL. |
| **QR code message** | The message with instructions to scan the QR code and begin credential verification. |
| **Waiting Message** | Localization overrides for the waiting message. This is a map of locale to message. |
| **Push Message** | A custom message for the user when requesting the credential. |
| **Store Credential Verification Response** | Store the list of verified data submitted by the user in the shared state under the `pingOneCredentialVerification` key.<br>**Note**<br>The key is empty if the node is unable to retrieve the wallet pairing data from PingOne. |
| **Verification Timeout** | The period of time (in seconds) to wait for a response to the verification request. If no response is received within this time, the node times out and the verification process fails. |

| Property | Usage |
| --- | --- |
| **Custom Requested Credentials** | If selected, a custom requested credentials payload is retrieved from the `requestedCredentials` attribute in shared state. |

### Outputs

- `pingOneCredentialVerification`: The new PingOne Credential Verification request status and full response.
- `pingOneApplicationInstanceId`: The identifier of the application running the Wallet SDK on the user's device and registered with the service.

### Outcomes

#### `Success`

The credential verification was successful.

#### `Error`

There was an error during the verification process.

#### `Time Out`

The verification process reached the configured timeout value.

### Errors

If any of the API calls to PingOne Credentials fail, the following errors may be logged:

- `Error: PingOne Credentials Create Verification session- Status Code - Response Body`.
- `Error: PingOne Credentials Create Push Verification session- Status Code - Response Body`.
- `Error: PingOne Credentials Read a Verification session- Status Code - Response Body`.

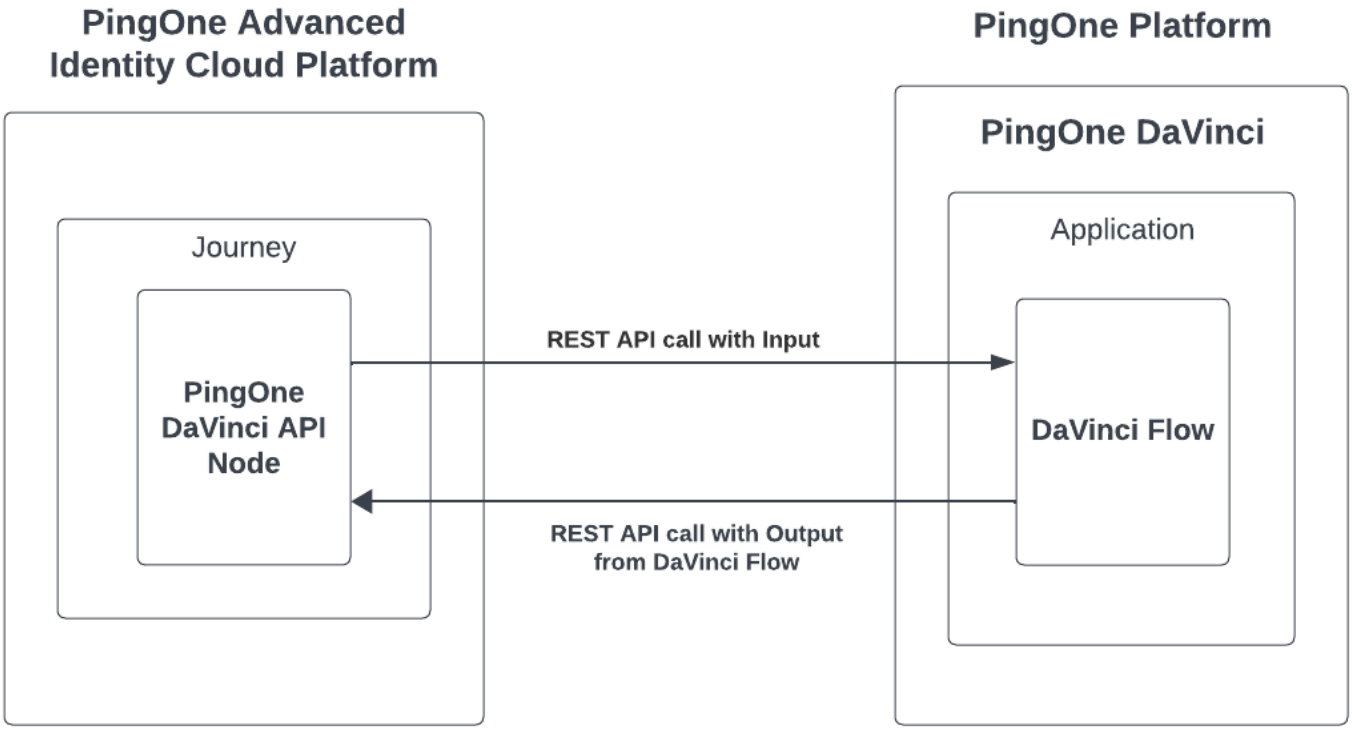
### Example

Learn more in an example journey using PingOne Credentials nodes.

## PingOne DaVinci API node

The **PingOne DaVinci API** node executes an API call to PingOne DaVinci to launch a specific DaVinci flow. This node lets an Advanced Identity Cloud journey trigger a PingOne DaVinci flow through the API integration method and does not render any front-end UI pages.

> **Note**
>
> This node is only effective for DaVinci flows without a UI component, as it is using the DaVinci flow API integration.




### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |
| Ping Identity Platform (self-managed) | Yes |

[1] This functionality requires that you configure AM as part of a Ping Identity Platform deployment.

### Inputs

This node does not have any required input fields.

### Dependencies

Before using this node, you must:

- Prepare the PingOne DaVinci flow
- Configure the input schema
- Create a PingOne DaVinci application
- Set up the PingOne service

This procedure only covers the steps and nodes required to prepare a PingOne DaVinci flow for invoking the API. It assumes you've already created the PingOne DaVinci flow.

### Prepare the PingOne DaVinci flow

1. In PingOne DaVinci, go to the **Flows** tab.
2. Find the flow you need, and click **Edit**.
    1. At the end of the **Success Path**, add an HTTP node to send a JSON success response.
    2. At the end of the **Failure Path**, add an HTTP node to send a JSON error response.
3. Click **Save**.
4. Click **Deploy**.

Refer to the PingOne DaVinci: Configuring the Flow page for further details.

### Configure the input schema

The PingOne DaVinci API node sends the journey node state to the PingOne DaVinci flow. You must configure the node state as an input parameter for the PingOne DaVinci flow.

1. Click **Input Schema** on the DaVinci flow canvas.
2. Click **Add** to add an input parameter.
    1. In the **Parameter Name** field, enter nodeState.
    2. In the **Data Type** field, select Object.
3. Click **Save**.

### Create a PingOne DaVinci application

1. In PingOne DaVinci, go to the **Applications** tab.
2. Click **Add Application**. The Add Application modal opens.
3. In the **Name** field, enter a name for the application, and click [.label]#Create.
4. Find the application you need and click **Edit**.
5. On the General tab, note down the values for **Company ID** and **API Key**.

    You'll need these to configure the PingOne DaVinci API node parameters.
6. Go to the **Flow Policy** tab.
7. Click **+ Add Flow Policy**.
    1. In the **Name** field, enter a name for the flow policy.
    2. In the **Flow List**, select your flow.
    3. In the **Version List**, select the desired flow version.

8. Click **Create Flow Policy**. The **Edit Your Weight Distribution** modal opens.
9. Click **Save Flow Policy**.
10. Note down the Policy ID of your flow policy. You'll need it to configure the PingOne DaVinci API node parameters.

Refer to the PingOne DaVinci: Create Application page for further details.

### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| **PingOne Service** | The ID of the PingOne Worker service for connecting to PingOne. |
| **Flow Policy ID** | The PingOne DaVinci Flow Policy configured for the specific flow. |
| **State Inputs** | A list of state inputs that will be passed to PingOne DaVinci, the DaVinci Flow input schema includes the node state parameter. This is a multi-value field to select specific node state attributes which are to be included in the API request to PingOne DaVinci.<br>By default, the wildcard `(*)` value will include the entire journey node state in the API request to PingOne DaVinci. |

### Outputs

Any data configured to be returned to the PingOne DaVinci flow is put into the node state.

### Outcomes

#### `True`

The PingOne DaVinci flow executed and returned a Success response.

#### `False`

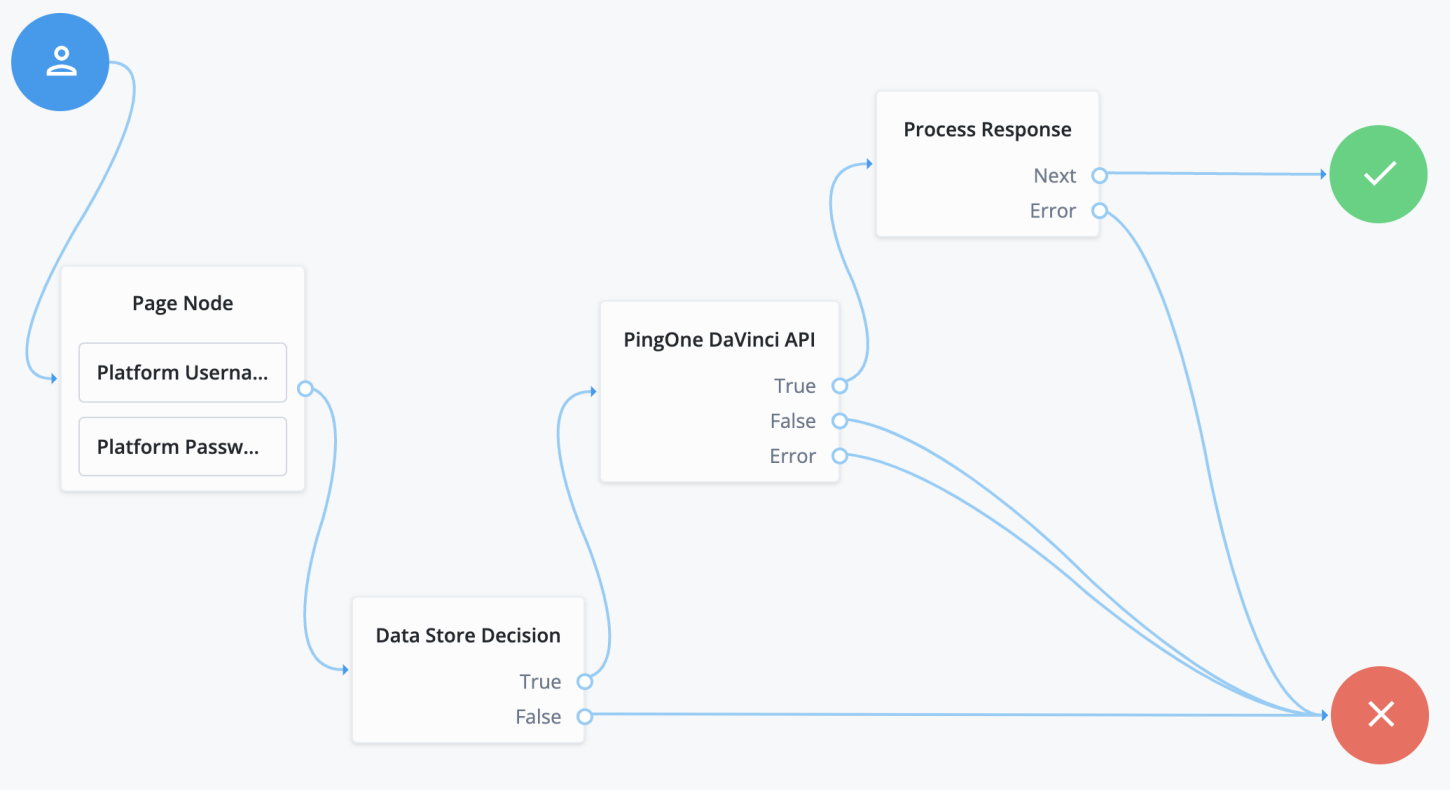
The PingOne DaVinci flow executed and returned an Error response.

#### `Error`

An error occurred causing the request to fail. Check the response code, response body, or logs to see more details of the error.

### Example

This example highlights the use of the PingOne DaVinci node in a user registration journey:




## PingOne Protect Marketplace nodes

**Important**

The old PingOne Protect nodes in Advanced Identity Cloud are now renamed as PingOne Protect Marketplace nodes and are deprecated. Instead, use the new PingOne Protect nodes that are now available:

- PingOne Protect Initialization node
- PingOne Protect Evaluation node
- PingOne Protect Result node

Before using any of the PingOne Protect nodes, you must:

1. Create a worker application in PingOne.
2. Configure the PingOne Worker service in your Advanced Identity Cloud environment

## PingOne Verify Authentication node

The **PingOne Verify Authentication** node lets you integrate PingOne Verify biometric authentication functionality in your journey. The biometric authentication is achieved by comparing a stored picture to a live selfie.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads these inputs from shared state:

The node reads the `username` from shared state.

To provide the `username` in shared state earlier in the journey, configure a node such as the Platform Username node.

Additionally, the node first looks in the shared state for the **attribute containing the PingOne UserID** and the **reference picture attribute**, which contains a Base64-encoded reference self-image in JPEG format. If these two attributes are not found in the shared state, the node looks up the user in the local datastore to retrieve the PingOne UserID and the reference picture.

If the PingOne UserID is not found in the local datastore or the shared store, a new user is created in PingOne to perform facial-biometric authentication.

### Dependencies

You must configure PingOne Verify service before using this node.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Service** | The ID of the PingOne Worker service for connecting to PingOne. |
| **PingOne Verify Policy ID** | The policy ID PingOne Verify node to use. The policy is expected to have the following details set:<br>• ID Verification is set to `DISABLED`.<br>• Facial Comparison is set to `REQUIRED`.<br>• Liveness is set to `REQUIRED`. |
| **Verify URL delivery mode** | QR code to display or E-mail/SMS for direct delivery. |
| **Let user choose the delivery method** | If selected, the user is prompted for a delivery method. |
| **Delivery message choice** | The message to display and allow user to select the delivery route (QR, SMS, eMail). The verify code displays along with the message. |

| Property | Usage |
| --- | --- |
| **Reference Picture Attribute** | The attribute key for retrieving the local reference picture. The node first looks in the shared state for the attribute containing the PingOne UserID and the reference picture attribute, which contains a Base64-encoded reference selfie in JPEG format. If these two attributes are not found in the shared state, the node looks up the user in the local datastore to retrieve the PingOne UserID and the reference picture.<br>If `Let user choose the delivery method` is enabled or `Verify URL delivery mode` is set to use QR code, then you should store the reference picture in the shared state.<br>If the reference picture is in the shared state, `Let user choose the delivery method` is not enabled, and `Verify URL delivery mode` is not set to use QR code, then you should store the reference picture in the transient state. |
| **Attribute containing the PingOne UserID** | Local attribute name that contains the PingOne UserID. |
| **Submission timeout** | Verification submission timeout value in seconds. The value must be within the authentication session validity time. |
| **Waiting message** | The message to display while waiting for the user to complete the authentication with PingOne Verify. |
| **Save verification metadata from PingOne Verify to Transient State** | Save verification explanation data from PingOne Verify to Transient State with a key of `VerifyMetadataResult`. |
| **Leave access token in transientState** | If seleted, the PingOne access token is preserved in the transient state. |
| **Leave PingOne Verify transaction id in transientState** | If selected, the PingOne access token is preserved in the transient state, with a key of `VerifyAT`. |
| **Save verification metadata from PingOne Verify to Transient State** | Save verification explanation data from PingOne Verify to Transient State with a key of `VerifyMetadataResult`. |
| **Leave access token in transientState** | If selected, the PingOne access token is preserved in the transient state, with a key of `VerifyAT`. |
| **Leave PingOne Verify transaction id in transientState** | If checked, PingOne transaction ID is preserved in transient state with a key of `VerifyTransactionID`. |
| **Demo mode** | When checked, the node always returns `SUCCESS` outcome. |

### Outputs

If the outcome is `Success (Patch ID)` or `Fail (Patch ID)`, the `Attribute containing the PingOne UserID` key is placed in shared state and in the `objectAttribute` object so the local user can be patched with the new user GUID that was created in PingOne for the verification. Save the returned GUID to the local user so the node doesn't need to create a new PingOne user on the next use.

### Outcomes

#### Success

Successfully authenticated the user's stored selfie and live selfie.

#### Success (Patch ID)

Successfully authenticated the user's stored picture and live selfie. Additionally, if the stored GUID on the local user was invalid or did not exist, the node created a new PingOne user to perform the verification. The node stored the new user's PingOne GUID in the shared state and in the objectAttribute, so the GUID can be used for future verification.

#### Fail

Failed to authenticate the user's stored picture and live selfie.

#### Fail (Patch ID)

Failed to authenticate the user's stored picture and live selfie.

#### Error

There was an error during the authentication process.

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately.

## PingOne Verify Proofing node

The **PingOne Verify Proofing** node lets administrators integrate PingOne Verify verification functionality using `Government ID`, `Facial Comparison`, and `Liveness` in a journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

The node reads the `username` from shared state.

To provide the `username` in shared state earlier in the journey, configure a node such as the Platform Username node.

Additionally, the node first looks for the attribute containing the PingOne UserID in the shared state. If that information is not found in the journey state, the node looks up the user in the local datastore to retrieve the PingOne UserID.

If the PingOne UserID does not exist in the local datastore, or does not exist in the PingOne datastore, a new user is created in PingOne to perform the verification.

### Dependencies

You must configure PingOne Verify service before using this node.

### Configuration

| Property | Usage |
|---|---|
| **PingOne Service** | The ID of the PingOne Worker service for connecting to PingOne. |
| **PingOne Verify Policy ID** | PingOne Verify Policy ID to use. The policy is expected to have the following details set:<br>• ID Verification is set REQUIRED.<br>• Facial Comparison is set REQUIRED.<br>• Liveness is set REQUIRED. |
| **Verify URL delivery mode** | Select from these options:<br>• To display the verify URL to the end user as a QR code, select **QR**.<br>• To email the end user with the verify URL, select **EMAIL**.<br>• To send an SMS to the end user with the verify URL, select **SMS**.<br>• To redirect the end user to the PingOne Verify service, select **REDIRECT**. |
| **Let user choose the delivery method** | If selected, the user is prompted for the delivery method. |
| **Delivery message choice** | The message to display and allow user to select the delivery route (QR, SMS, or EMAIL). If QR delivery route is selected, the verify code displays along with the message. |
| **Document type required** | For any valid government ID leave ANY, otherwise, specify the document type to enforce. |
| **PingOne UserID Attribute** | Local attribute name to retrieve the PingOne UserID from. Will look in journey state first, then the local datastore. |
| **Age threshold** | If specified (years), the node extracts the date of birth from the claims and validates if age is equal or greater than the specified threshold. (Set 0 to disable age check). |
| **Attribute map** | Map PingOne Verify Claims to `objectAttributes` on shared state. The KEY is `objectAttribute` and the VALUE is the `Verify Claim Key`. Use IDM keys for claim mapping. Refer to user mapping details for claim mapping keys. |

| Property | Usage |
| --- | --- |
| **Attribute match confidence map** | Optionally, send the attributes entered by the user during registration to verify with imprecise matching in PingOne Verify. Value represents minimum confidence level to mark verification successful (LOW/MEDIUM/HIGH/EXACT). |
| **Fail expired documents** | For documents that contain expiration date, fail if out of date. |
| **Submission timeout** | Verification submission timeout in seconds. Value must be under authentication session validity time. |
| **Waiting message** | The message to display while waiting for the user to complete the authentication with PingOne Verify. |
| **Save verified claims from PingOne Verify to Transient State** | To save verified claims from PingOne Verify API response to transient state with a key of `VerifyClaimResult`. |
| **Save verification metadata from PingOne Verify to Transient State** | Save verification explanation data from PingOne Verify to transient state with a key of `VerifyMetadataResult`. |
| **Leave access token in transientState** | If checked, PingOne access token is preserved in transient state, with a key of `VerifyAT`. |
| **Leave PingOne Verify transaction id in transientState** | If checked, PingOne transaction ID is preserved in transient state with a key of `VerifyTransactionID`. |
| **Demo mode** | When selected, the journey continues along the `Success` outcome path. |

### Outputs

- `VerifyNeedPatch`: The new PingOne User's GUID
- `VerifyClaimResult`: Verified claims
- `VerifyMetadataResult`: The verification metadata
- `VerifyAT`: The access token used to perform the PingOne Verify
- `VerifyTransactionID`: The PingOne Verify transaction ID
- `VerifyNeedPatch`: The new PingOne GUID if a new PingOne user was created
- `VerifiedFailedReason`: If a failure occurs, summary detail of reason

#### Outcomes

##### Success

All configured checks passed.

##### `Success (Patch ID)`

All configured checks passed. Additionally, the node needed to create a new PingOne user in PingOne to perform the Verification. This is because the stored GUID on the local user was invalid or didn't exist. The node stored the new users PingOne GUID in the shared state on the `PingOne UserID Attribute` key and on the objectAttribute, so the GUID can be saved to the local users account and used for future verifications.

##### `Fail`

One of the configured checks failed.

##### `Fail (Patch ID)`

One of the configured checks failed. Additionally, the node needed to create a new PingOne user in PingOne to perform the Verification. This is because the stored GUID on the local user was invalid or did exist. The node stored the new users PingOne GUID in the shared state on the `PingOne UserID Attribute` key and on the objectAttribute, so the GUID can be saved to the local users account and used for future verifications.

##### `Error`

There was an error during the authentication process.

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately.

### PingOne Verify service

The PingOne Verify service lets you configure and use PingOne Verify nodes (PingOne Verify Authentication node and PingOne Verify Proofing node) to provide four types of secure user verification Government ID, Facial Comparison Government ID,nFacial Comparison Reference Selfie, and Liveness.

At this time, these nodes don't support other PingOne user verification methods.

You must set up the following before using the PingOne Verify nodes:

- Create a verification policy
- Create a worker application
    - Requires Identity Data Admin role
- Set up PingOne Service

## BioCatch Session node

Manages the interaction with the BioCatch scoring API.

This node initializes a session with the BioCatch scoring API, associates the session with the user who authenticates, and links the session ID in the user-agent with the BioCatch server.

## Prerequisites

Before you start, create at least one scripted policy to determine access based on BioCatch scores.

### BioCatch script

The policy relies on a policy condition script to grant or deny access depending on the BioCatch score.

1. In the Advanced Identity Cloud admin UI, go to **Scripts > Auth Scripts**, click **+ New Script**, and select **Policy Condition**.
2. Name your script, replace the default JavaScript with the following sample, update the default variables at the top of the script with values that suit your deployment, and save your work:

```
try {
//=================== Update these variables for the deployment ======================
  var biocatchEndpoint = "https://api-customer-id.eu.v2.customers.biocatch.com/api/v6/score";
  var customerId = "customer-id";
  var minScore = 0;
  var maxScore = 500;
  var advices = ["Fraud Alert"]; // Advices to return for a fraudulent request
//====================================================================================

  var customerSessionID = null;

  /**
   * Sends a request to Biocatch to get the score of a customer session.
   *
   * @returns {*} The score of a customer session.
   */
  function getScore() {

    var loginDoRequest = new org.forgerock.http.protocol.Request();

    //Set the method type.
    loginDoRequest.setMethod("POST");

    //set the POST URL
    loginDoRequest.setUri(biocatchEndpoint);

    //set some header values
    loginDoRequest.getHeaders().add('Content-Type', 'application/json; charset=UTF-8');

    var user = String(session.getProperty("UserToken"));

    //set some body values
    var theBody = JSON.stringify({
      "action": "getScore",
      "customerSessionID": customerSessionID,
      "uuid": user,
      "solution": "ATO",
      "activityType": "LOGIN",
      "customerID": customerId
    });

    loginDoRequest.getEntity().setString(theBody);
    var response = httpClient.send(loginDoRequest).get();

    var resultJSON = JSON.parse(response.getEntity().getString());
    return parseInt(resultJSON.score);
  }

  /**
   * Retrieve and validate the variables required to make the external HTTP calls.
   *
   * @returns {boolean} Will be true if validation was successful.
   */
  function validateAndInitializeParameters() {

    if (username == null || biocatchEndpoint == null || maxScore == null || customerId == null || advice ==
null || minScore == null)
      return false;

    if (!environment) {
```

```
      logger.warning("No environment parameters specified in the evaluation request.");
      return false;
    }

    if (environment.get("customerSessionID") != null &&
environment.get("customerSessionID").iterator().hasNext()) {
      customerSessionID = environment.get("customerSessionID").iterator().next();
    } else {
      logger.warning("No customerSessionId specified in the evaluation request environment parameters.");
      return false;
    }

    return true;
  }


  if (validateAndInitializeParameters()) {

    var scoreFromBiocatch = getScore();

    if (scoreFromBiocatch >= minScore && scoreFromBiocatch <= maxScore) {
      logger.message("Authorization Succeeded");
      authorized = true;
    } else {
      logger.message("Authorization Failed");
      advice.put("advice", advices);
      authorized = false;
    }

  } else {
    logger.message("Required parameters not found. Authorization Failed.");
    advice.put("advice", ["Required parameters not found"]);
    authorized = false;
  }
} catch (error) {
  logger.error(error);
  advice.put("advice", ["Error occurred"]);
  authorized = false;
}
```

3. Adapt the sample script for the deployment as necessary.

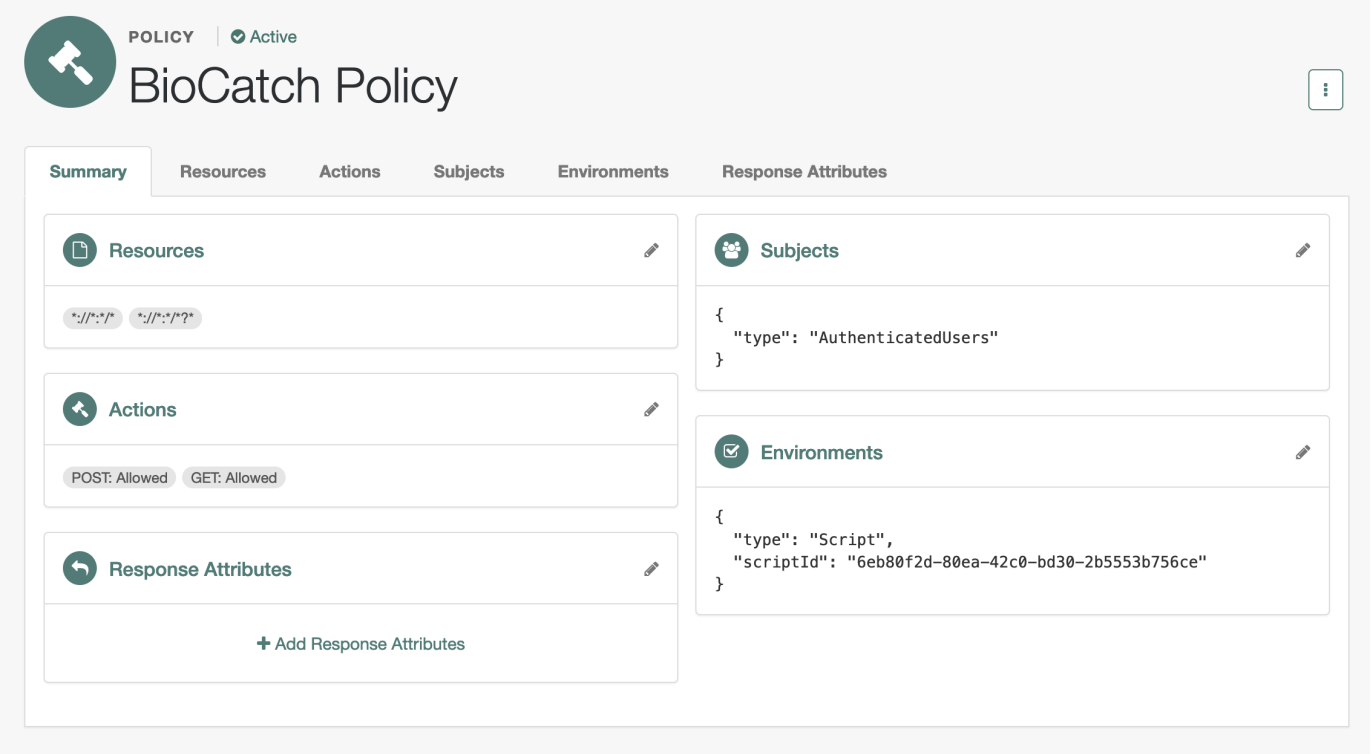
### BioCatch policy

1. Create a policy set for BioCatch policies.

   Find more information in Policy sets.

2. Create a policy with the BioCatch policy decision script as an environment condition.

   The following policy grants authenticated users with an appropriate score HTTP GET and POST access to URLs:




Find more information in Policy sets in the UI.

## Outcomes

### `True`

Initialization succeeded.

### `False`

Initialization failed.

### `Error`

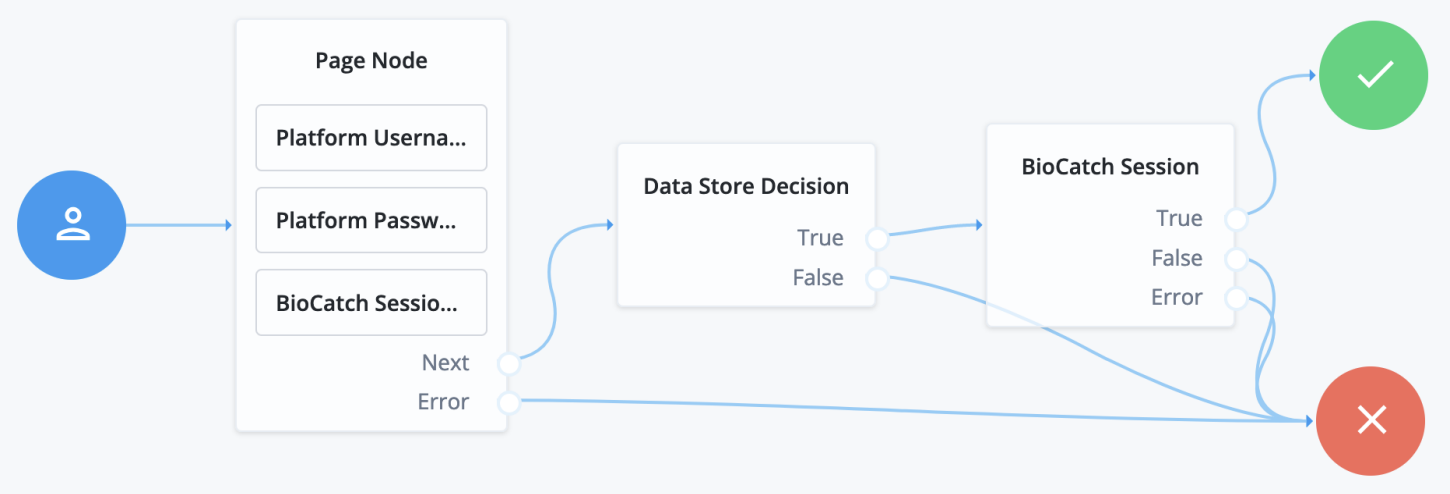
An error occurred.

## Properties

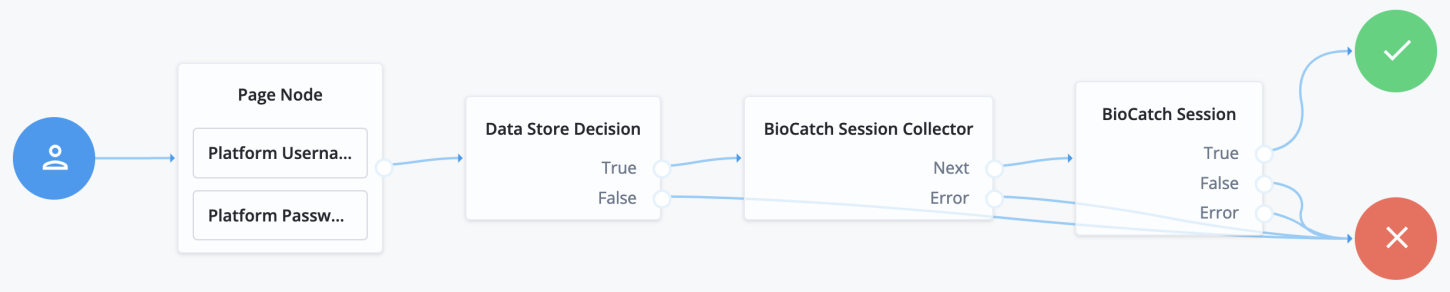
| Property | Usage |
| --- | --- |
| BioCatch End Point | URL for the BioCatch initialization API |
| Customer Id | The customer or project identifier from BioCatch |

### Examples

The following example injects a unique session identifier ( `customerSessionID` ) in the page for collecting credentials with the BioCatch Session Profiler node and initiates scoring:




The following example relies on the customer web application having the JavaScript to inject the user's unique session identifier ( `customerSessionID` ). It collects the identifier with the BioCatch Session Collector node and initiates scoring:




In both cases, the policy configured as a prerequisite determines access based on the score from BioCatch.

## BioCatch Session Collector node

Collects the user's BioCatch unique session identifier ( `customerSessionID` ).

### Outcomes

**`Next`**

Successfully collected the identifier.

**`Error`**

An error occurred.

### Properties

This node has no configurable properties.

### Example

For an example in context, refer to the BioCatch examples.

## BioCatch Session Profiler node

Injects BioCatch JavaScript into the user-agent to set the unique session identifier ( `customerSessionID` ).

### Outcomes

**`Next`**

Successfully injected the JavaScript.

**`Error`**

An error occurred.

### Properties

| Property | Usage |
| --- | --- |
| BioCatch JavaScript URL | URL for the BioCatch JavaScript |

### Example

For an example in context, refer to the BioCatch examples.

## Duo node (Deprecated)

> **Important**
>
> Ping Identity has deprecated the Duo node because Duo has deprecated Traditional Duo Prompt that is used by the Duo node. You should use the Duo Universal Prompt node in place of the Duo node.

Integrates Duo for an additional authentication factor.

### Outcomes

- `True` (success)

- `False` (failure)

### Configuration

1. Create a Duo account at https://signup.duo.com/.
2. In the Duo admin console under **Applications**, click **Protect an Application**.
3. Search for and create an application profile for `Web SDK`.
4. Record the following for use when configuring the node:
    - **Integration key**
    - **Secret key**
    - **API hostname**
5. Generate an application key for the node, which is a random string at least 40 characters long.

    How you generate the random string for the application key is up to you. For example, use a random string generator service or a Python script to generate an application key:

    ```
    python3
    >>> import os, hashlib
    >>> hashlib.sha1(os.urandom(32)).hexdigest()
    '<your-application-key>'
    >>> exit()
    ```

### Properties

| Property | Usage |
|---|---|
| Integration Key | The integration key from Duo for the Web SDK application. |
| Secret Key | The secret key from Duo for the Web SDK application. |
| API Host Name | The API hostname from Duo for the Web SDK application. |
| Application Key | The application key you generated. |
| Duo Javascript URL | The link to Duo's JavaScript for your Web SDK application.<br>Default: `https://api.duosecurity.com/frame/hosted/Duo-Web-v2.min.js` |

## Duo Universal Prompt node

The **Duo Universal Prompt** node integrates with the Duo service to provide two-factor authentication using Duo's Universal Prompt authentication interface. You can integrate Universal Prompt with your web applications using the Duo Web v4 SDK.

The Duo Universal Prompt node redirects the user to the Duo service, where they complete their registration or authentication; and return to the journey.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
|---|---|---|
| **Username** | username | Shared state |

## Dependencies

To use this node, you should have already set up PingOne Advanced Identity Cloud integration with the Duo authentication service. Refer to Duo node (Deprecated).

## Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| **Client ID** | The client's ID on Duo. |
| **Client Secret** | The client's secret on Duo. |
| **API Hostname** | The Duo API host name. |
| **Failure Mode When Duo is Down** | When Duo health check fails, two-factor authentication is not performed and this property is used to determine the outcome from the node:<br>• If set to `OPEN`, the node proceeds to the **True** outcome.<br>• If set to `CLOSED`, the node proceeds to the **False** outcome. |
| **OpenAM Callback URL** | The PingOne Advanced Identity Cloud callback URL to return and resume the journey. |

### Outputs

In case of an `Error` outcome, the error message is output to the shared state.

### Outcomes

#### `True`

Duo authentication is successful.

#### `False`

Duo authentication failed.

#### `Error`

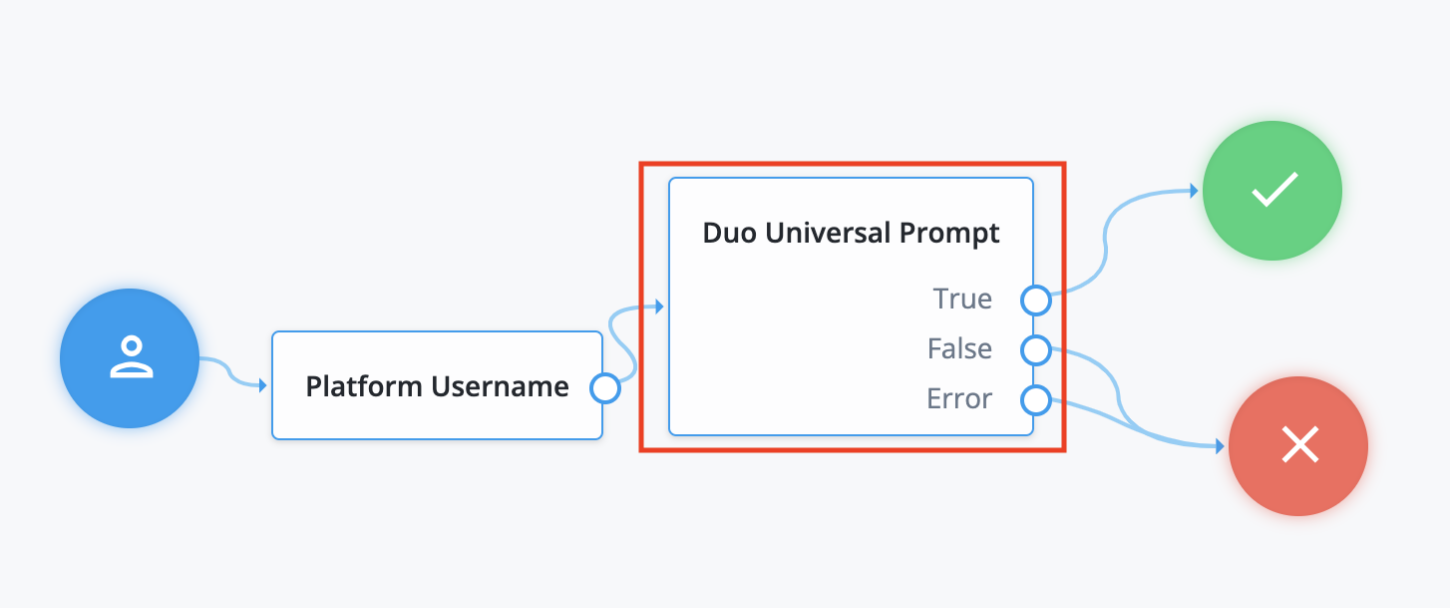
Any error that occurred while authenticating. An error message is output to the shared state.

### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately.

### Example

This example journey highlights the use of the Duo Universal Prompt node. In this case, the Duo Universal Prompt node redirects the user to the Duo service to complete their Duo registration or authentication. Upon completion of registration or authentication, the user returns to the node and continues the journey.



# Fingerprint nodes

The Fingerprint Profiler and Fingerprint Response nodes let you integrate your Advanced Identity Cloud environment with the Fingerprint platform to help reduce fraud and improve customer experience. The integration with Fingerprint provides browser fingerprinting directly from an authentication journey with high confidence, at an average score of 99.5%.

When you identify browsers or devices with Fingerprint, you get back the `visitorId` value. You can use this value in your business logic to find suspicious activity or for marketing analytics. In some cases you do not want the client devices receive `visitorID` value from Fingerprint. Instead, you can receive a random `requestID` that can be used in business logic. This mode of not sending back `visitorID` is called Zero Trust Mode (ZTM).

You must Setup the prerequisites on the Fingerprint site before you can use Fingerprint nodes.

**Tip**

To deploy the Fingerprint nodes in a self-managed AM environment:

1. Download the fingerprint-1.0.13.jar file.
2. Copy the `fingerprint-1.0.13.jar` file to the `web-container/webapps/openam/WEB-INF/lib` directory where PingAM is deployed.
3. Restart the web container to pick up the new node.

PingOne Advanced Identity Cloud provides two authentication nodes for Fingerprint authentication journeys:

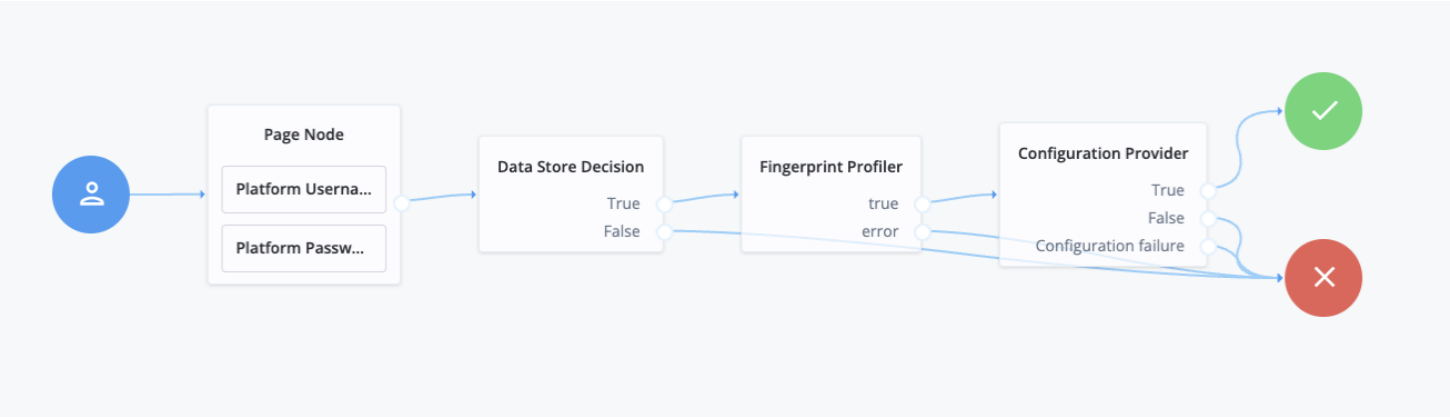
- Fingerprint Profiler node
- Fingerprint Response node

## Sample journeys using Fingerprint nodes

To understand and address the most common use cases, download the sample journeys that use the Fingerprint nodes. Before testing or using the sample Fingerprint journeys, you must Setup the API key and secret keys.

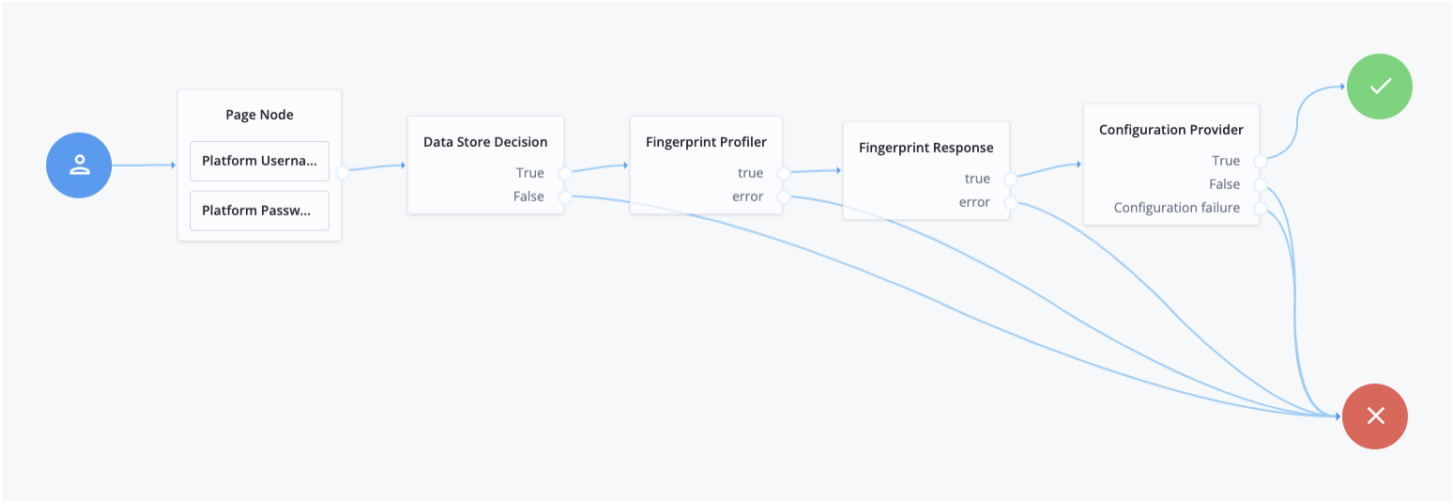
These samples are provided only for development and testing; don't use them in production environments.

### Fingerprint client-side journey



The Fingerprint client-side journey delivers the fingerprint and confidence score using the Fingerprint Profiler node only. Download the JSON file for this sample journey here.

### Fingerprint ZTM journey




The example Fingerprint ZTM journey delivers the fingerprint and confidence score using the Fingerprint Profiler and Fingerprint Response nodes. Download the JSON file for this sample journey here.

## Setup

Before you can use Fingerprint nodes on your PingOne Advanced Identity Cloud environment, perform these steps on the Fingerprint site:

1. Create an account on the Fingerprint site.
2. Click **New application** to set up a new application.
3. Go to **Application Settings > API Keys** to create an API public key.
4. [Optional] If you're planning to use the zero-trust mode (ZTM):
    1. Ensure that ZTM is enabled on your chosen application.
    2. Create an API secret key.
5. Configure a custom domain to authenticate correctly when using Fingerprint.

## Fingerprint Profiler node

The Fingerprint Profiler node injects the client-side Javascript code required for the fingerprinting process.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

[1]

### Inputs

This node does not require any input data.

### Dependencies

This node uses the externally hosted JavaScript code defined in the **ScriptURL Pattern** configuration property.

### Configuration

| Property | Usage |
|---|---|
| **Public API Key** | Public API key for the Fingerprint application. |
| **Script URL Pattern** | URL path to a hosted JavaScript when using a custom domain. The default URL pattern is https://fpjscdn.net/v3. |
| **Endpoint** (optional) | URL path to the endpoint address when using a custom domain. This is an optional property. |
| **Fingerprint Region** | The Fingerprint region in which the Fingerprint application is set up. To use the US or the default region, specify `GLOBAL`. |
| **Shared State VisitorID** | Name of the shared state variable to store the device fingerprint. When ZTM is enabled, set the same value in the Fingerprint Response node. |
| **Zero Trust Mode** | You can enable ZTM if your application is set up in ZTM, and the Fingerprint Response node is configured to deliver the fingerprint. |

### Outputs

| Variable | Description |
|---|---|
| **deviceRequestId** | Transaction ID for the unique fingerprint request from Fingerprint server. If ZTM is enabled, this attribute is used by the Fingerprint Response node to retrieve device fingerprint. |
| **deviceFingerPrint** | Unique figerprint of the browser. This property name is configurable in the **Shared State VisitorID** setting. |
| **deviceConfidenceScore** | Confidence level for the retrieved fingerprint. |

> **Note**
>
> If your application is using ZTM, only **deviceRequestId** is returned to shared state.

If an error occurs, error messages with the keys [Marketplace]Exception and [Marketplace]StackTrace are output to the shared state.

### Outcomes

#### Success

The profiler node successfully completed.

#### Error

There was a problem with the node. Check all the settings and the error messages in shared state.

### Troubleshooting

Review the log messages and address the issue appropriately.

### Examples

The following examples show the use of Fingerprint Profiler and Fingerprint Response nodes in authentication journeys:

- Fingerprint client-side journey
- Fingerprint ZTM journey

1. **To deploy the Fingerprint nodes in a self-managed AM environment**, download the fingerprint-1.0.13.jar file locally, copy the `fingerprint-1.0.13.jar` file to the `web-container/webapps/openam/WEB-INF/lib` directory where AM is deployed, and then restart the web container to pick up the new nodes.

## Fingerprint Response node

The **Fingerprint Response** node is used in the zero-trust model to fetch the fingerprint and server-side confidence score.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

[1]

### Inputs

| Attribute name | Description | Source |
|---|---|---|
| **deviceRequestId** | Mandatory variable to identify the authentication request. | Value is stored in the shared state by the **Fingerprint Profiler** node. |

### Dependencies

This node uses an external API from the Fingerprint site to deliver the fingerprint.

### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| **Secret API Key** | Secret API key configured on the Fingerprint site. |
| **Events API URL** | Path to an events API. The default is https://eu.api.fpjs.io/events/. Other options include:<br>• https://api.fpjs.io/events/<br>• https://ap.api.fpjs.io/events/ |
| **Shared State VisitorID** | Name of the shared state variable to store the fingerprint. This should be the same as in the Fingerprint Profile node. |
| **Get full response payload** | Disabled by default. Enable if you want to store the full fingerprint API response in shared state. |
| **Shared State Response** | Name of the variable to store the fingerprint API response. Used only when `Get full response payload` is enabled. |

### Outputs

This node adds the following in the shared store:

| Attribute | Description |
|---|---|
| **deviceFingerPrint** | Unique signature of the browser (fingerprint). The name of this variable is configured using the **Shared State VisitorID** property. |
| **deviceConfidenceScore** | Confidence level for the verified fingerprint. |
| **payload** (optional) | This optional attribute contains the full API response from the Fingerprint site. The name of this variable is configured using the **Share State Response** configuration property. |

If an error occurs, error messages with the keys `[Marketplace]Exception` and `[Marketplace]StackTrace` are output to the shared state.

### Outcomes

#### Success

The response node completed successfully.

#### Error

There was a problem with the node. Check all the settings and the error messages in shared state.

### Troubleshooting

Review the log messages and address the issue appropriately.

### Examples

Fingerprint ZTM journey

1. **To deploy the Fingerprint nodes in a self-managed AM environment**, download the fingerprint-1.0.13.jar file locally, copy the `fingerprint-1.0.13.jar` file to the `web-container/webapps/openam/WEB-INF/lib` directory where AM is deployed, and then restart the web container to pick up the new nodes.

## Gateway Communication overview

The **Gateway Communication** node provides a secure communication channel for PingOne Advanced Identity Cloud authentication journeys to communicate directly with the PingGateway.

This secure communication channel extends the Advanced Identity Cloud capabilities with PingGateway features, such as validating a Kerberos ticket and performing other certificate handshakes.

Advanced Identity Cloud provides these artifacts to enable this secure communication:

- Gateway Communication service
- Gateway Communication node

You must set up the following before using the Gateway Communication node:

- Two key pairs
- Identity Gateway JWT validation
- Gateway Communication service

For more information on this node, refer to Gateway Communication node

## Gateway Communication setup

You must set up the following before using the Gateway Communication node:

- Two key pairs
- PingGateway JWT validation
- Gateway Communication service

### Two key pairs

You must set up two sets of public and private key pairs; one set each for PingOne Advanced Identity Cloud and PingGateway. You can use `mkjwk simple JSON Web Key generator` to generate sample keys in the following format:

- Type: RSA
- Key size: 2048
- Algorithm: RSA1_5: RSAES-PKCS1-v1_5
- Key ID: Timestamp

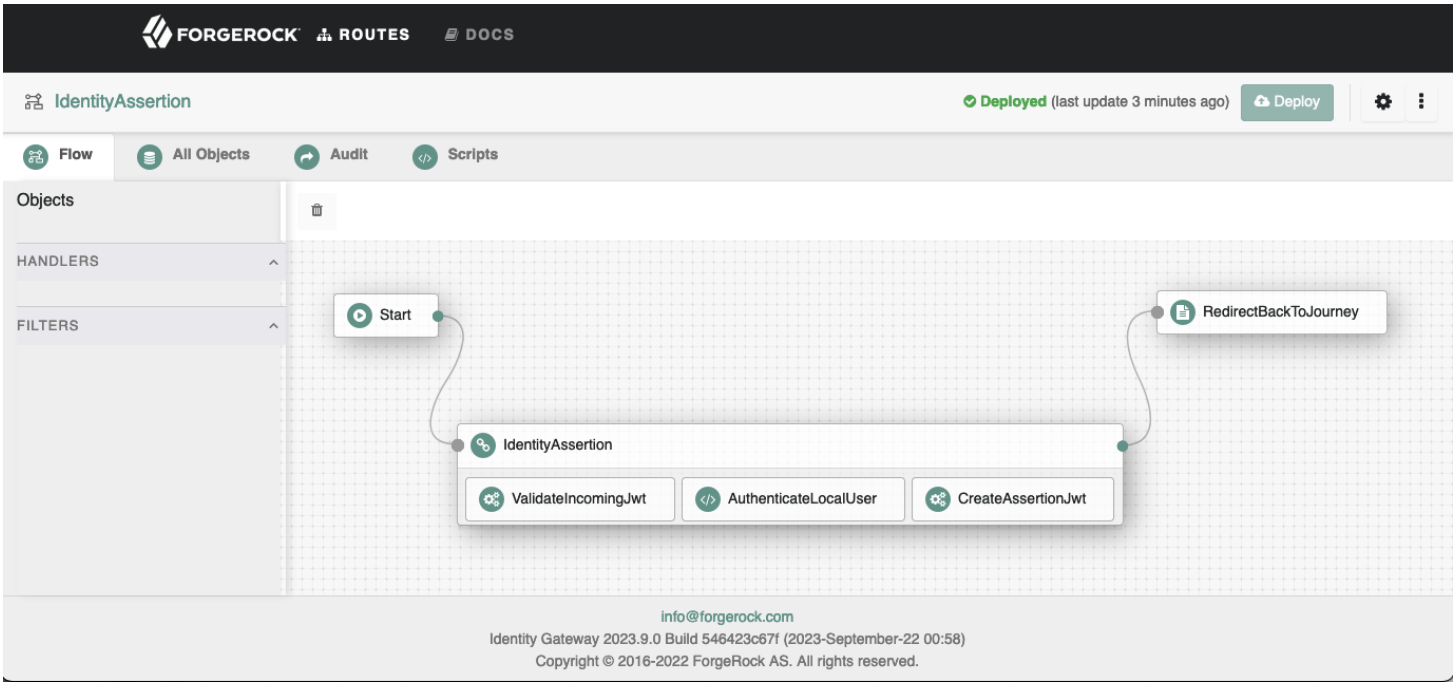
Store one set of public and private keys in PingOne Advanced Identity Cloud along with the public key from the second set.

Similarly, store the second set of public and private keys in PingGateway, along with the public key from the first set.

### PingGateway JWT validation

PingGateway must be configured to validate the JWT sent by PingOne Advanced Identity Cloud, and create a signed and encrypted JWT to be sent back to PingOne Advanced Identity Cloud.

The example shown here assumes that the public and private keys between PingOne Advanced Identity Cloud and PingGateway are in PEM format.




You can download the sample **IdentityAssertion** route from here.

The sample route assumes that the keys in PEM format have been added to the config.json heap. The keys are stored in a top-level (at the same level as config/logs/scripts) directory called **secrets** etc). The key files are named using the convention of secretId.pem.

### Gateway Communication service

You must set up the **Gateway Communication** service in your Advanced Identity Cloud tenant before using the Gateway Communication node in an authentication journey.

To configure the Gateway Communication service:

1. Log in to your Advanced Identity Cloud admin UI, and navigate to **Native Consoles > Access Management**.
2. In the Realm Overview page, click **Service Management**.
3. Click **+ Add a Service**.
4. Select **Gateway Communication Service** from the **Choose a service type** menu, and click **Create**.
5. In the Gateway Communication Service page, ensure that **Enable** is selected.
6. In the `Secondary Configurations` tab, click **Add a Secondary Configuration**, and provide the following details:
    - **Name**: A unique name for the PingGateway server. ***Identity Gateway Public Key**: Enter the public key portion of the PingGateway in JWK format. It is recommended to use an ESV for these keys.

      Here is an example of a public key:

```
{
    "kty": "RSA",
    "e": "AQAB",
    "use": "enc",
    "kid": "enc-1697673430",
    "alg": "RSA1_5",
    "n": "lGR1KKw...QOHSRTfQ"
}
```

- **Identity Cloud Public and Private Key**: Enter the public and private keypair for PingOne Advanced Identity Cloud in JWK format. We recommend that you use an ESV for these keys.

  Here is an example of a public and private keypair:

```
{
    "p": "45DaO0K...cruR85AWc",
    "kty": "RSA",
    "q": "pu8V15...H75-wXs",
    "d": "LSYzj2...bT628Q",
    "e": "AQAB",
    "use": "enc",
    "kid": "enc-1697673430",
    "qi": "YO-Ox3d...MaHSU2M",
    "dp": "n-Icwbf1...g0khhw8",
    "alg": "RSA1_5",
    "dq": "T_VO8f0W...8X7WGonTsc",
    "n": "lGR1K...1QOHSRTfQ"
}
```

- **JWT Parameter Name**: Enter the name of the parameter that contains the JWT. This name is used for sending the JWT to PingGateway and is the expected name of the JWT returned by PingGateway.

7. In the **COMMCONFIGS CONFIGURATION**, enter the additional details:

   - **Identity Gateway URL**: Full URL of your PingGateway. This should not include a specific route, because the route will be configured in the node.
   - **Send to Gateway Security**: Security type used for transporting the JWT to PingGateway. The options are:
     - **Signed**: The JWT sent to PingGateway is signed by the Advanced Identity Cloud private key.
     - **SignAndEncrypt** - The JWT sent to PingGateway is signed by the Advanced Identity Cloud public and private keys and then encrypted by the PingGateway public key.
     - **Milliseconds JWT TTL** - Time in milliseconds representing how long the JWT sent to PingGateway has for Time To Live.

8. Click **Save Changes**.

## Gateway Communication node

The **Gateway Communication** node provides a secure communication channel with PingGateway directly from within a journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | No |
| Ping Identity Platform (self-managed) | No |

### Inputs

Any data in the shared state that must be sent to PingGateway.

### Dependencies

To use this node, you must configure the Gateway Communication service.

### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| **Identity Gateway Service** | PingGateway service set up used for this communication. |
| **IG Route** | The name of the route to take when PingGateway is reached. |
| **Mapping to Gateway** | Map data sent in a JWT to PingGateway from the shared state in the journey. This is an optional property that is used only when PingGateway needs data.<br>• The key is the `Shared State Key` containing the data to map.<br>• The value is the `Claim Key Name` in the shared state sent in the claim. |
| **Mapping from Gateway** | Map data returned from PingGateway to the shared state in the journey.<br>• The key is the PingGateway key in the claim.<br>• The value is returned by PingGateway and saved to the shared state. |

### Outputs

Any data mapped from the claims returned by PingGateway that is stored in the shared state of the journey.

### Outcomes

#### Success

The gateway successfully returned a valid signed and encrypted JWT.

#### Error

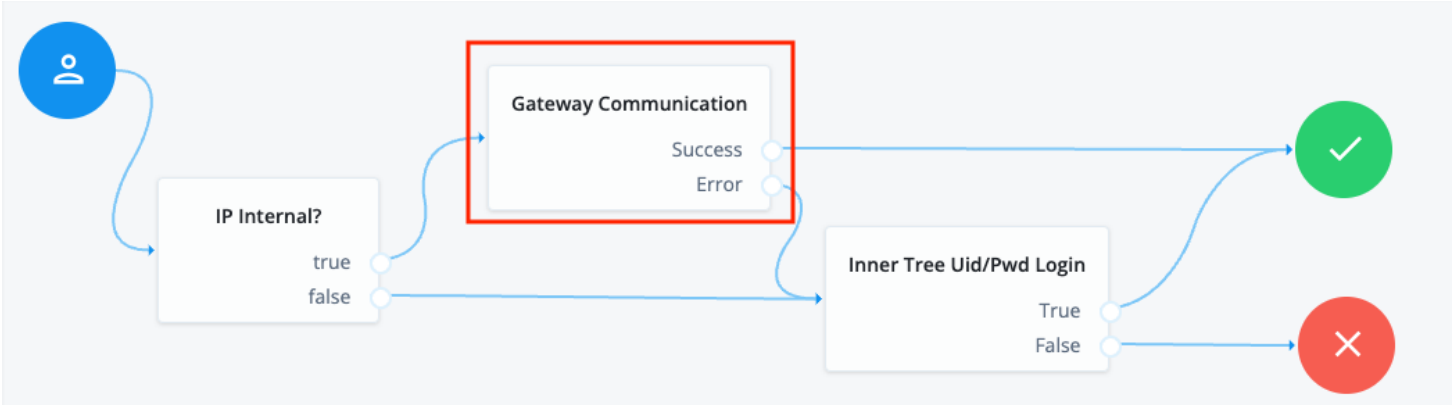
Any error that occurred during an attempt to communicate with the gateway.

### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately.

### Examples

This example journey highlights the use of the **Gateway Communication** node to authenticate internal accesses.




PingOne Advanced Identity Cloud provides sample journeys you can download to understand and address the most common Gateway Communication use cases.

## HTTP Client node

The **HTTP Client** node lets you make direct HTTP(S) requests to external APIs and services from within an authentication journey. This simplifies integration with a range of external services to invoke actions or retrieve data for use in a journey.

The HTTP Client node can make GET, POST, PUT, DELETE, PATCH, and HEAD requests. Requests can include headers, request parameters, payloads, and certificates for endpoints secured using mTLS. The node can also save the response body to shared state, and you can configure the handling of response codes.

### Compatibility

| Product | Compatible |
|---|---|
| Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes [1] |

| Product | Compatible |
| --- | --- |
| Ping Identity Platform (self-managed) | Yes [1] |

[1] For self-managed products, download the node from AdvanceMarketplace.

## Inputs

Any data in shared state that you need to include in the headers, parameters, or payload/body of the request.

## Dependencies

This node has no specific dependencies.

If you use this node to make calls to endpoints secured with Mutual Transport Layer Security (mTLS), you must provide a valid X.509 digital certificate.

Configure the node with both the public certificate and private key in PEM format.

Advanced Identity Cloud only: You can add the certificates and key values directly to the node configuration, or store the secrets as ESVs.

## Configuration

| Property | Usage |
| --- | --- |
| **Rest URL** | The endpoint URL, including scheme, port (optional), and path (optional). For example: `https://postman-echo.com/post`.<br>You can use the values of shared state variables by including the variable name in handlebars ( `${variable name}` ). For example:<br>`` `\https://postman-echo.com/postman/${username}/account` ``<br>where `username` is set in the shared state.<br>You can also use JSONpath expressions to select specific values or branches from JSON objects stored in shared state variables. Learn more in Variable substitution. |
| **Request Method** | The request type: GET, POST, PUT, DELETE, PATCH, or HEAD. |

| Property | Usage |
| --- | --- |
| **Query Parameters** | An optional set of key-value pairs added as query parameters.<br>You can use the values of shared state variables by including the variable name in handlebars ( `${variable name}` ). For example, to include a query parameter named `user_id` map this parameter to the `username` property in shared state as follows:<br>`Key: user_id`<br>`Value: ${username}`<br>You can also use JSONpath expressions to select specific values or branches from JSON objects stored in shared state variables. Learn more in Variable substitution. |
| **Headers** | An optional set of key-value pairs to be added as request headers.<br>You can use the values of shared state variables by including the variable name in handlebars ( `${variable name}` ). You can also use JSONpath expressions to select specific values or branches from JSON objects stored in shared state variables. Learn more in Variable substitution. |
| **Use HTTP "Basic" Authentication** | Includes an HTTP Basic Authentication header. |
| **HTTP "Basic" Username** | The username for HTTP Basic Authentication. |
| **HTTP "Basic" Password** | The password for HTTP Basic Authentication. |

| Property | Usage |
| --- | --- |
| **Body Content** | An optional value to include in the request payload or body. The payload format depends on the REST endpoint and can include JSON, XML, plain text, or other data.<br>You can use the values of shared state variables by including the variable name in handlebars ( `${variable name}` ).<br>• This example includes a header named `X-Transaction-ID` with the value of `username` in the shared state:<br>`Key: X-Transaction-ID`<br>`Value: ${username}`<br>• This example includes a JSON payload with values automatically substituted from shared state variables:<br>`{ "uid": "${username}", "first_name":"${givenName}", "last_name":"${sn}", "active": true }`<br>• This example includes a form payload (application/x-www-form-urlencoded):<br>`uid=${username}&first_name=${givenName}&last_name=${sn}&active=true`<br>**Note**<br>If a requested shared state variable is not set, the node substitutes a null value.<br>You can also use JSONpath expressions to select specific values or branches from JSON objects stored in shared state variables. Learn more in Variable substitution. |
| **Body Content Encoding** | The request body type: `XWWWFORMURLENCODED` , `JSON` , `XML` , `PLAIN` , or `CUSTOM` . Use the `CUSTOM` type to configure a header named `content-type` in the `Headers` configuration. |
| **Use mTLS** | Enable this option for endpoints that require mTLS authentication. If enabled, also configure the **mTLS Public Certificate** and **mTLS Private Key**. |
| **mTLS Public Certificate** | The client X.509 public certificate in PEM format.<br>Advanced Identity Cloud only: For more flexibility, store the certificate as an ESV. |

| Property | Usage |
| --- | --- |
| **mTLS Private Key** | The client private key for the X.509 certificate in PEM format. Private keys should be in PKCS8 format, typically with the PEM header value of `-----BEGIN PRIVATE KEY-----`. Private key should not be in PKCS1 format, typically with a PEM header value of `-----BEGIN RSA PRIVATE KEY-----`.<br><br>**Note**<br>If the private key is a PKCS1 key, you must convert the key to PKCS8. To make the conversion:<br>1. Execute the following openssl command:<br>`openssl pkcs8 -topk8 -inform PEM -outform PEM \ -nocrypt -in <PKCS1_PRIVATE_KEY.pem> -out <PKCS8_PRIVATE_KEY.pem>`<br>2. Use the contents of `<PKCS8_PRIVATE_KEY.pem>` for the value of `mTLS Private Key`.<br><br>Advanced Identity Cloud only: For more flexibility, store the certificate as an ESV. |
| **Disable Certificate Checks** | If you select this option, AM ignores certificate errors, such as expired certificates, associated with the external endpoint.<br><br>**Note**<br>This option applies only to server certificate validation checks, and not to client certificates. Trusting expired and invalid certificates can have serious security implications. Don't disable certificate checks in a production environment. |
| **Timeout** (seconds) | The timeout value (in seconds) for the request. The same timeout value is used for establishing the connection to the endpoint and for awaiting a response. |
| **Response Codes** | By default, this node logs `Default Response` and `Error` outcomes. Optionally, specify other response codes to add new outcomes. For example, you can add 200 and 401 codes to add two additional outcomes for handling `OK` and `Unauthorized` outcomes.<br>Use response code classes to handle a range of response codes. For example, to capture all client error responses in the range 400-499 and server error responses in the range 500-599, add the values `4xx` and `5xx`. This results in two additional outcomes that can be handled appropriately in the journey.<br><br>**Note**<br>Any matched `JSON Response Outcome` takes precedence over the defined response code outcome. |
| **Status Code Shared State Variable** | (Optional) The shared state variable in which to store the response code. |
| **Returned Body Shared State Variable** | (Optional) The shared state variable in which to store the response body. |

| Property | Usage |
| --- | --- |
| **JSON Response Handler** | For REST calls that return a JSON formatted response, specific returned values or JSON paths of the response are mapped to shared state variables. Define the key-value pairs where `key` is the shared state variable name and `value` is the JSONpath expression of the data to be saved. Learn more about JSONpath expressions in Variable substitution. |
| **JSON Response Outcome Handler** | Returned values from JSON formatted response can be evaluated and used to trigger appropriate outcomes from the node. Subsequent journey steps can handle the evaluated outcomes appropriately.<br>Outcomes are defined by key-value pairs, where `key` is the name of the outcome, and `value` is the JSONpath expression. The value is evaluated on the returned JSON data. JSONpath expressions act as filters on JSON arrays of data. An expression that results in a non-empty array is considered a successful match and should trigger the associated outcome. Learn more about JSONpath expressions in Variable substitution.<br>**Note**<br>An outcome matched by the **JSON Response Outcome Handler** overrides any outcome defined by a response code. For example, if a response code outcome is defined for code "200", but the JSON Response Outcome Handler is also matched, then the outcome of the JSON Response Outcome Handler will be followed. |

## Variable substitution

A number of this node's configuration properties let you retrieve values from or save values to shared state variables. For these properties, you can use variable substitution in the format `${variable}`. For example, to include the `username` in a URL path, you can use an expression similar to:

```
`\https://postman-echo.com/postman/${username}/account`
```

where `username` is the name of a variable currently set in shared state.

If the value in shared state is a JSON object, you can use JSONpath notation to select specific values, array items, or complete branches of the JSON object.

To use JSONpath expressions for variable substitution, use an expression of the form `${shared state variable.$.path}`. Assume, for example, the shared state variable `objectAttributes` containing the following JSON data:

```
{
"username": "bob",
"firstName": "Bob",
"lastname": "Fleming",
"telephoneNumber": "+1(555)1231234",
"bookingIDs": [ 29872, 23884, 48382 ],
"membershipTier": "platinum"
}
```

The `firstname` and `lastname` attributes can be selected as `${objectAtttributes.$.firstname}` and `${objectAttributes.$.lastname}`, respectively. The last `bookingID` can be selected as `${objectAttributes.$.bookingIDs[2]}` or `${objectAttributes.$.bookingIDs[-1]}`.

You can use similar notation with the **JSON Response Handler** property to save returned JSON response values to shared state.

For example, if the JSON object is the response to a REST API call, the following **JSON Response Handler** configuration selects the telephone number from the response and saves it to the `phone` shared state variable:

```
Key: phone
Value: $.telephoneNumber
```

The JSON Response Outcome Handler can be filtered with a suitable JSONpath expression to look for matching responses. A matching outcome is triggered when the JSONpath expression, when applied to an array containing the JSON response, finds at least one match. For example, if the previous JSON object is the response to a REST API call, the following **JSON Response Outcome Handler** configuration triggers the `Priority` outcome:

```
Key: Priority
$.[?(@.membershipTier == 'platinum')]
```

Learn more about JSONpath expressions in the README.

## Outputs

Response code values and response body can be saved to shared state for use by subsequent nodes in the journey. Learn more in Configuration.

## Outcomes

### *Matching JSONpath Outcomes*

You can add outcomes for specific JSON response data. Learn more in the JSON Response Outcome Handler property.

### *Response Codes*

You can configure outcomes dynamically for specific response codes, such as 401, and response code classes, such as 2xx. Learn more in the Response Code property.

### *Default Response*

The request is completed, but no other JSON response or response code outcomes were matched.

> **Note**
>
> This outcome indicates that a request and response were successfully processed, including responses indicating errors such as 4xx, (client error). If you want to handle error responses separately, consider adding response code outcomes.

*Error*

An error occurred causing the request to fail. Check the response code, response body, or logs for details of the error.

> **Note**
>
> In cases where multiple outcomes could apply, JSON response outcomes take precedence. For example, a REST call could result in both a matching JSON response outcome and a 200 response code outcome. The matched JSON response outcome is triggered in this case.

### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately. You can use many publicly accessible test endpoints, such as `https://httpstat.us` and `https://postman-echo.com`, to test and troubleshoot this node.

## IdentityX Auth Request Decision node

Checks the status of a Daon IdentityX authentication request for an out-of-band authentication flow over a separate, secure channel.

> **Important**
>
> Include an IdentityX Check Enrollment Status node in the journey prior to this node.

### Outcomes

**`Pending`**

The request is still in progress.

**`Success`**

The request completed successfully.

**`Failed`**

The request failed.

**`Expired`**

The request timed out.

**`Error`**

An error occurred.

### Properties

This node has no configurable properties.

### Examples

For examples in context, refer to the Daon IdentityX examples.

## IdentityX Auth Request Initiator node

Generates a Daon IdentityX authentication request for an out-of-band authentication flow over a separate, secure channel.

> **Important**
>
> Include an IdentityX Check Enrollment Status node in the journey prior to this node.

### Outcomes

**`Request Sent`**

Successfully generated and sent the authentication request.

**`Error`**

An error occurred.

### Properties

| Property | Usage |
|---|---|
| Policy Name | Name of the authentication policy to use. |
| Application ID | Name of the IdentityX application to use. |
| Use FIDO | If enabled, use the FIDO client SDK instead of the legacy device SDK.<br>Default: enabled |
| Send Push Notifications | If enabled, send a push notification to the user's device.<br>Default: enabled |

### Examples

For examples in context, refer to the Daon IdentityX examples.

## IdentityX Check Enrollment Status node

Verifies that a user is enrolled with the Daon IdentityX platform.

> **Important**
>
> This node configures integration with the IdentityX platform.
> Journeys that integrate with the IdentityX platform must include this node.

## Prerequisites

Before you start, configure the IdentityX platform and a service application client.

### Daon IdentityX configuration

The nodes require a connection to an IdentityX server. Contact your Daon representative for connection details.

Follow these high-level steps:

1. In the Daon Admin Console, go to **Administration** > **System Configuration** > **REST Authentication**.

   The URL to the **REST Authentication** screen has the form `https://api.identityx-cloud.com/your-Daon-instance/AdminConsole/#configurations/restauthentication`.

2. Update **JWT Signature Validation Keys URLs** to use your JWK URI.

   Make sure the algorithm is `RS256`.

   ```
   {
     "endpoints": [{
       "url": "https://<tenant-env-fqdn>:443/am/oauth2/alpha/connect/jwk_uri",
       "alg": "RS256"
     }]
   }
   ```

3. For **REST Authentication Mode**, enable **JSON Web Token (JWT)**.

4. Set a name for the **JWT Roles Claim Name**.

   Record the name for use when setting up the service application client.

5. Set the **JWT Read Timeout** to `500` (milliseconds).

6. Create a new role.

   Go to **Administration** > **Roles** and click **Create Role**.

   Use the following settings and save the new role:

   #### *Role Name*

   Anything (example: `forgerockjwt`)

   #### *Description*

   Anything (example: `forgerockjwt`)

#### *External ID*

Anything (example: `forgerockjwt`)

#### *Entity*

`All (*)`

#### *Permission Selector*

Select your Daon tenant.

#### *Enable these flags*

`CREATE`
`READ`
`UPDATE`
`DELETE`
`BLOCK`
`UNBLOCK`
`ALL(*)`

Record your choice for external ID for use when setting up the service application client.

### Ping Identity Platform configuration

1. Create an **OAuth2 Access Token Modification** script to use the IdentityX role you configured.

   In the Advanced Identity Cloud admin UI, go to **Scripts** > **Auth Scripts**, click **+ New Script**, and create an **OAuth2 Access Token Modification** script.

   Save a new script such as the following, where the field value is the Daon role ID:

   ```
   (function () {
     // Always includes this field in the token.
     accessToken.setField('roles', 'forgerockjwt');
   }());
   ```

   Record the name of your script for use when setting up the ForgeRock service application client.

2. Create a service application to access the IdentityX platform.

   Go to **Applications** and click **+ Add Application**.

   Select **Service** as the application type.

   Create a client ID and secret for your application.

   Record the client ID and secret for use when setting up journeys that use the IdentityX platform.

3. Use the following settings for your new service application:

#### *Grant Types*

`Client Credentials`

#### *Scopes*

`fr:idm:*`

4. Use the following advanced settings for your new service application:

#### *Default Scopes*

`fr:idm:*`

#### *Response Types*

`Token`

5. Configure signing and override OAuth 2.0 provider settings for your application.

In the AM admin UI, go to **Realms** > ***Realm Name*** > **Applications** > **OAuth 2.0** > **Clients** > ***client-ID***.

Switch to the **Signing and Encryption** tab, verify the following settings and save your changes:

#### *Token Endpoint Authentication Signing Algorithm*

`RS256`

#### *ID Token Signing Algorithm*

`RS256`

#### *Authorization Response JWT Signing Algorithm*

`RS256`

#### *Token introspection response signing algorithm*

`RS256`

Switch to the **OAuth2 Provider Overrides** tab, update the following settings and save your changes:

#### *Enable OAuth2 Provider Overrides*

Enabled

#### *Access Token Modification Plugin Type*

`SCRIPTED`

#### *Access Token Modification Script*

Your **OAuth2 Access Token Modification** script

6. Update the OAuth2 token signing algorithm in the OAuth 2.0 provider service for the realm.

In the AM admin UI, go to **Realms** > ***Realm Name*** > **Services** > **OAuth2 Provider**.

Switch to the **Advanced** tab, update the following setting and save your changes:

### *OAuth2 Token Signing Algorithm*

`RS256`

This setting must match the configuration completed on the IdentityX administrative console.

## Outcomes

### `User Enrolled`

Successfully verified enrollment.

### `User Not Enrolled`

Failed to verify enrollment.

### `Error`

An error occurred.

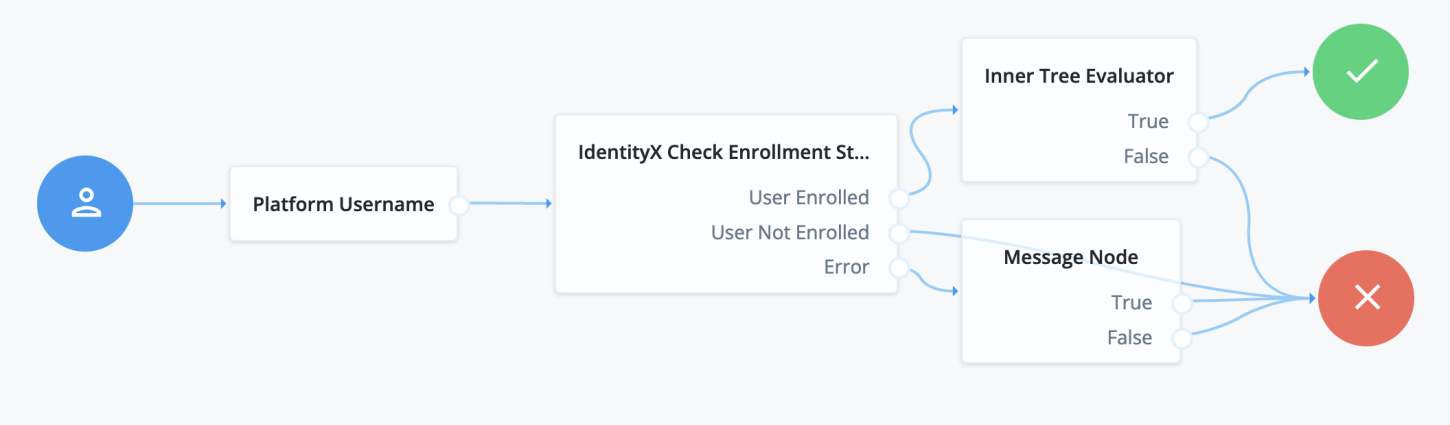
## Properties

| Property | Usage |
|---|---|
| ForgeRock Client ID | The client ID of your ForgeRock service application for communications with the IdentityX platform. |
| ForgeRock Client Secret | The client secret of your ForgeRock service application for communications with the IdentityX platform. |
| IdentityX Base URL | The IdentityX URL has the following form `https://yourHostName/yourTenantName/IdentityXServices/rest/v1`. |
| User Id Attribute | The shared state attribute that holds the Daon identifier for the end user. Leave this blank to collect the Daon User ID with a Platform Username node instead. |

## Daon IdentityX examples

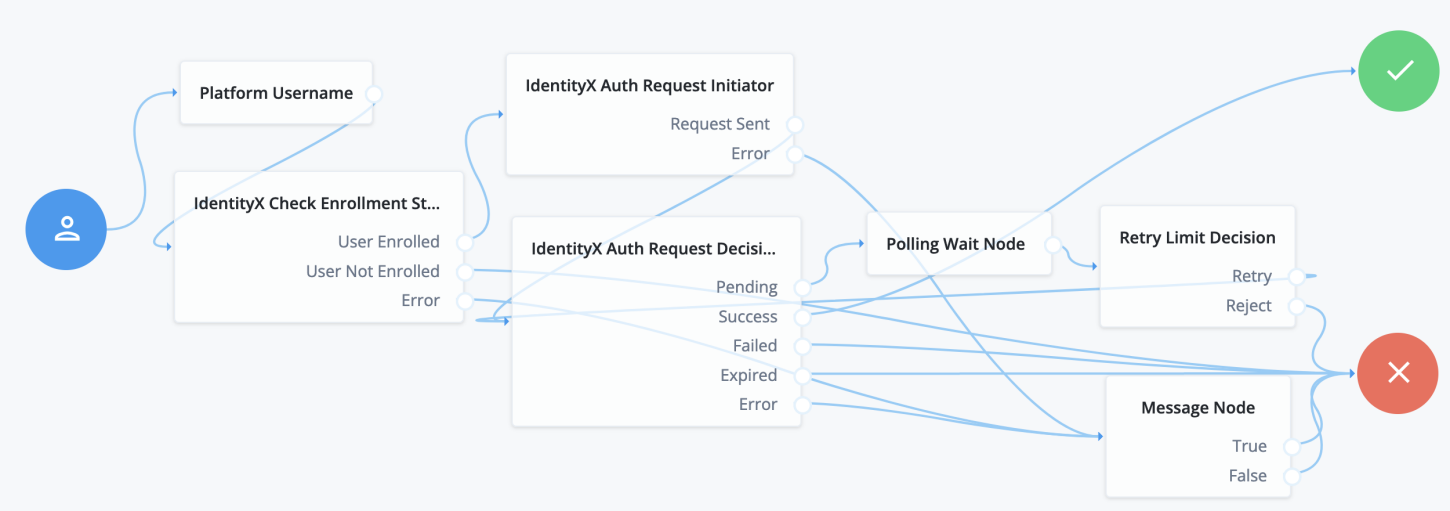
### Check enrollment before continuing

The following example demonstrates the use of this node before an inner tree with additional IdentityX nodes:



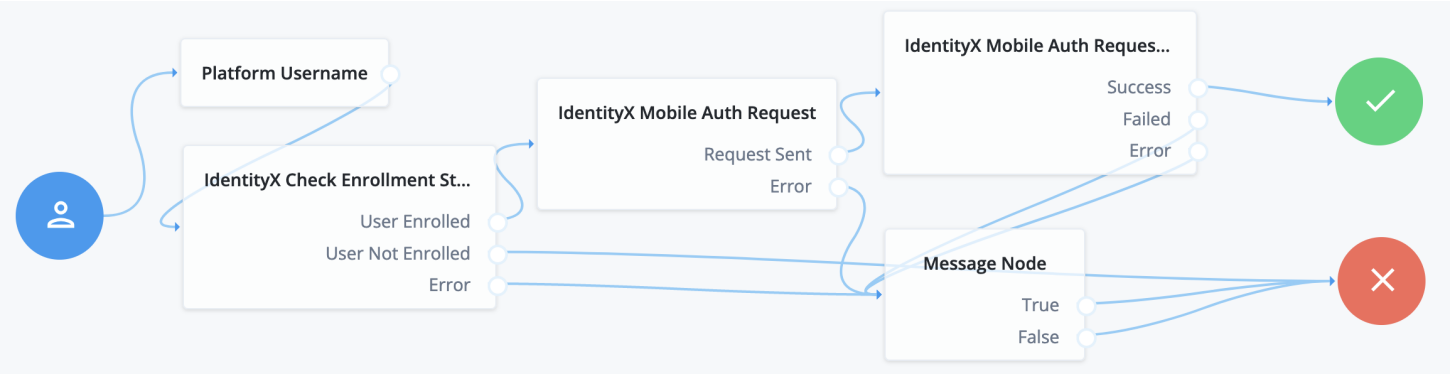

### Out-of-band authentication

The following example uses the IdentityX platform in an out-of-band flow over a separate, secure channel:



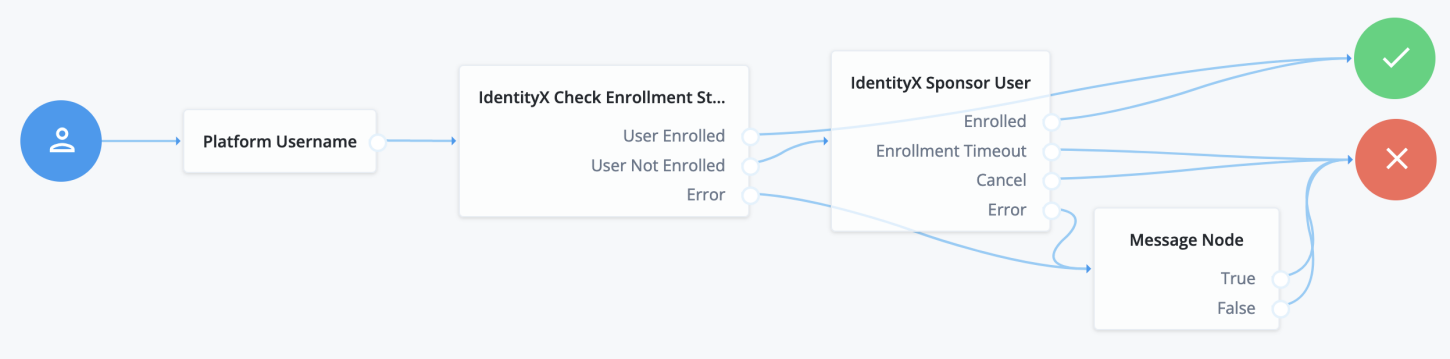

### Mobile authentication

The following example uses the IdentityX platform in a mobile authentication flow:




### Sponsor user (enrollment)

The following example enrolls the user, if necessary:




# IdentityX Mobile Auth Request node

Generates a Daon IdentityX authentication request for an end user authenticating on a mobile device.

> **Important**
>
> Include an IdentityX Check Enrollment Status node in the journey prior to this node.

## Outcomes

### `Request Sent`

Successfully generated and sent the authentication request.

### `Error`

An error occurred.

## Properties

| Property | Usage |
|---|---|
| Policy Name | Name of the authentication policy to use. |
| Application ID | Name of the IdentityX application to use. |
| Transaction Description | Description of the IdentityX authentication request for the end user.<br>Default: `OpenAM has Requested an Authentication` |

## Examples

For examples in context, refer to the Daon IdentityX examples.

# IdentityX Mobile Auth Request Validate node

Accepts a signed Daon IdentityX authentication request from a mobile device and validates the signature.

> **Important**
>
> Include an IdentityX Check Enrollment Status node in the journey prior to this node.

## Outcomes

### Success

Successfully validated the signature of the request.

### Failed

Failed to validate the signature.

### Error

An error occurred.

## Properties

| Property | Usage |
| --- | --- |
| Expected AuthRequest Status | The IdentityX authentication request status to return.<br>Choose either `COMPLETED_SUCCESSFUL` or `PENDING` from the list of options:<br>• `COMPLETED_FAILURE`<br>• `COMPLETED_SUCCESSFUL`<br>• `DECLINED`<br>• `EXPIRED`<br>• `EXPIRED_FAILURE`<br>• `EXPIRED_PENDING`<br>• `FRAUD`<br>• `PENDING` |

## Examples

For examples in context, refer to the Daon IdentityX examples.

# IdentityX Sponsor User node

Enables Daon IdentityX sponsorship (enrollment) of an end user.

> **Important**
>
> Include an IdentityX Check Enrollment Status node in the journey prior to this node.

### Outcomes

**`Enrolled`**

Successfully enrolled the end user.

**`Enrollment Timeout`**

Enrollment timed out.

**`Cancel`**

The end user canceled enrollment.

**`Error`**

An error occurred.

### Properties

| Property | Usage |
| --- | --- |
| Registration Policy | IdentityX registration policy to use for enrollment. |
| Application ID | Name of the IdentityX application to use. |
| Polling Wait Interval | Number of seconds to wait between polling the IdentityX platform. |
| Number Of Times to Poll | Number of times to poll the status of the enrollment request. |
| Message to Display | Message or instructions to be displayed on the screen below the QR code.<br>Default: `Scan the QR code with your mobile app.` |

### Examples

For examples in context, refer to the Daon IdentityX examples.

## iProov authentication

The **iProov Authentication** node integrates PingOne Advanced Identity Cloud authentication journeys with the iProov Genuine Presence Assurance and Liveness Assurance products from iProov.

iProov is a trusted provider of biometric face verification and authentication solutions that are fully optimized for usability, security, and privacy. Organizations rely on iProov's defenses against evolving biometric threats while delivering an intuitive user experience.

Advanced Identity Cloud provides the iProov Authentication node for integration with iProov.

You must configure the iProov Tenant before using the iProov Authentication node.

## iProov Authentication node

The **iProov Authentication** node integrates the iProov Genuine Presence Assurance® (GPA) and Liveness Assurance™ (LA) directly from within your authentication journey on Advanced Identity Cloud.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

A unique username is required in the shared state before the iProov Authentication node executes.

### Dependencies

To use this node, you must configure your iProov tenant. Refer to Setting up the iProov tenant.

### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| **iProov Tenant** | The hostname of your iProov tenant, either `us.rp.secure.iproov.me` or `eu.rp.secure.iproov.me`. |
| **iProov Base URL** | The iProov URL context that contains the version of the REST API, which is `/api/v2`. |
| **iProov API Key** | The API key you obtained from iProov. |
| **iProov API Secret** | The API secret from iProov. |
| **iProov OAuth Username** | The username of the OAuth user on iProov. |

| Property | Usage |
| --- | --- |
| **iProov OAuth Password** | The password of the user on iProov. |
| **iProov Assurance Type** | The type of API assurance on iProov:<br>• `GPA` : Generic Presence Assurance<br>• `LA` : Liveness Assurance<br>Default: `GPA` . |
| **iProov Authentication Type** | The type of authentication. It can be one of:<br>• `Enrol` - for enrolling the user into iProov.<br>• `Verify` - for verifying the user's liveness.<br>• `Combined` - for enrollment if the user is not enrolled, otherwise verify the user's liveness.<br>Default: `Enrol` . |
| **User Unique ID Attribute** | The unique ID of the user enrolled with iProov. This attribute must exist in the user's AM profile in the identity repository. |
| **User Search Attributes** | An alternative attribute that contains the username value, and is used to search a user in the underlying identity store. |
| **ForgeRock UI** | A boolean attribute for determining how the iProovWeb SDK is rendered to the user.<br>• When set to `true` , you can view the iProovWebSDK on the Advanced Identity Cloud admin UI.<br>• When set to `false` , you can view the iProovWebSDK by going to **Native Consoles > Access Management**.<br>Default: `true` . |
| **iProov Version** | The version of the iProov web SDK to use. Now 5.0.0 and 5.0.1 are supported. Default: 5.0.0. |
| **Title Text Color** | Adjusts the color of the title text above the central oval where the image is captured. By default, no title is used. Refer to the Custom Title attribute for more information. |
| **Surround Color** | Adjusts the color surrounding the central oval. It also affects the color of the mask in Liveness Assurance with a `clear` or `blur` filter. |
| **Prompt Text Color** | Adjusts the color of the text visible in the central prompt of the screen. |
| **Prompt Background Color** | Adjusts the color of the background in the central prompt of the screen. |
| **Header Background Color** | Adjusts the color of the background in the top bar of the application, transparent by default. |

| Property | Usage |
|---|---|
| **Custom Title** | The title of the camera view that appears above the image area when the camera is capturing the image. Specify a custom title to be shown. Default: An empty string (""). |
| **Assets URL** | Critical dependencies are loaded from the content delivery network (CDN) at `cdn.iproov.app`. In a production environment, set this property to your CDN, for example: https://cdn.iproov.app/myassets. |
| **Logo** | A relative link, absolute path or the data URI to your custom logo. The logo can be in any web format, though it is recommended to use the SVG format. If you don't specify a logo, the iProov logo is displayed. Set to `null` if you don't want a logo to be displayed. |
| **Network Timeout** | Time in seconds for the backend to acknowledge a message. If the timeout is exceeded, Advanced Identity Cloud returns an error with the feedback code `error_network`.<br>Default: 20 (seconds). |
| **iProov Camera Filter** | Controls the filter for the camera preview. The value can be classic, shaded, or vibrant. For Liveness Assurance, two additional filters, clear and blur, are provided. The blur filter is removed when the claim progresses.<br>+ Default: shaded. |
| **Prompt Rounded Corners** | The floating prompt has rounded corners by default. To disable rounded corners, set this attribute to `false`. |
| **Debug** | By default, log messages at level `info` or lower are hidden. They can be displayed on the console by setting **Debug** to `true`. Log messages at the `warning` and `error` levels are always displayed on the console. |
| **Slots** | Customize the markup styling and automatically inherit your application's styles by using the **Slots** attribute. |
| **Aria Live** | Control the priority of messages being read out by the screen reader. Refer to ARIA live regions in Mozilla documentation for more information on ARIA live. By default, this is set to `assertive` to indicate time-sensitive or critical notifications that require the user's immediate attention. This can be disabled by setting it to `off` or `polite`. |

### Outputs

The following outputs are stored in the shared node state:

| Output Variable | Variable Description |
|---|---|
| **iProovValidateResponse** | The complete validation response from iProov API in JSON format. |
| **iProoveValidatePhoto** | Photo from the validated API endpoint response. |

### Outcomes

#### Success

The iProov verification process is completed successfully.

#### Failure

The iProov verification process returned a failure because a user connection or device failed during the verification process.

#### Retry

The iProov verification process is incomplete due to a failure or user error and can be retried.

#### Error

A fatal exception occurred due to misconfiguration or an error with the user account. Exceptions are logged at the Error level, and put in the SharedState.

#### Cancel

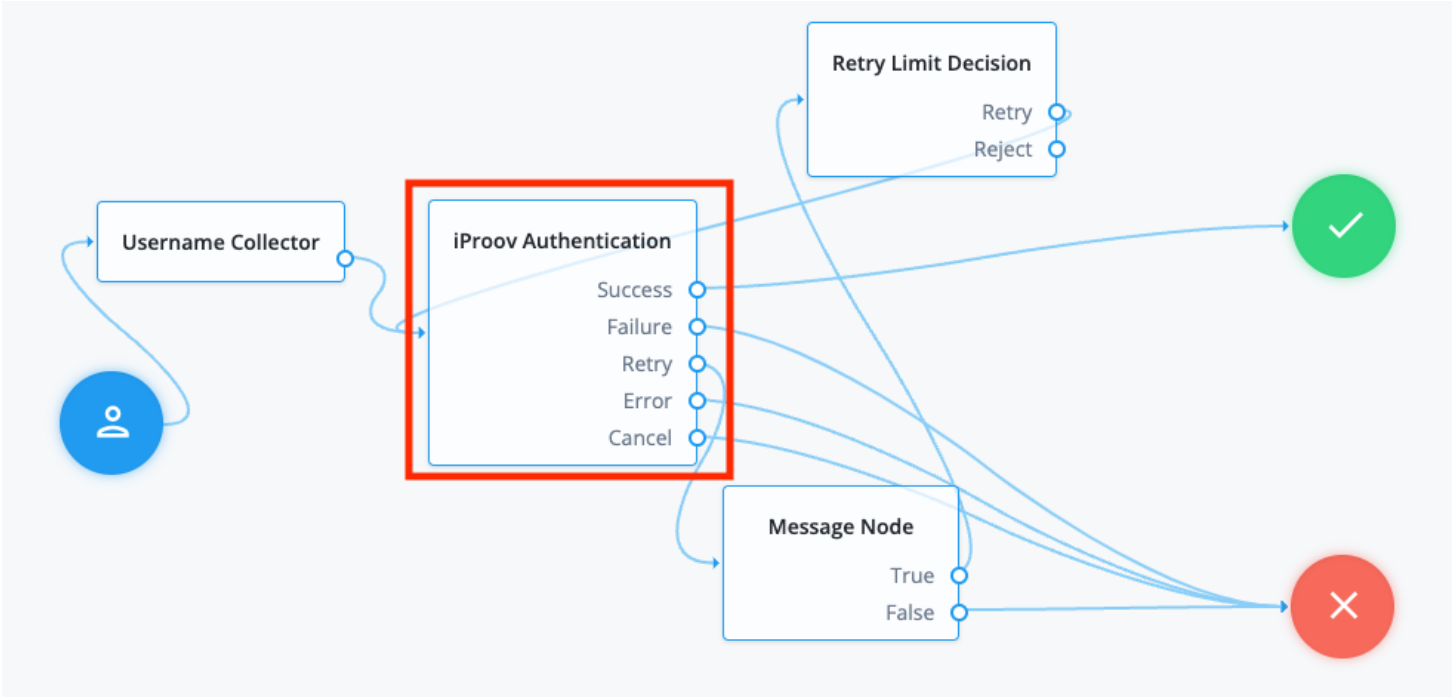
The user has opted to cancel the iProov verification.

### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error and address the issue appropriately.

### Examples

This example journey highlights the use of the **iProov Authentication** node to authenticate by using facial biometrics.



PingOne Advanced Identity Cloud provides sample journeys you can download to understand and address the most common iProov authentication use cases.

### Setting up the iProov tenant

To use your iProov verification in your Advanced Identity Cloud authentication journey, you need an active iProov tenant instance. You must create a service provider in your iProov tenant instance. Contact your iProov sales representative for more information.

To create a service provider in your iProov tenant instance:

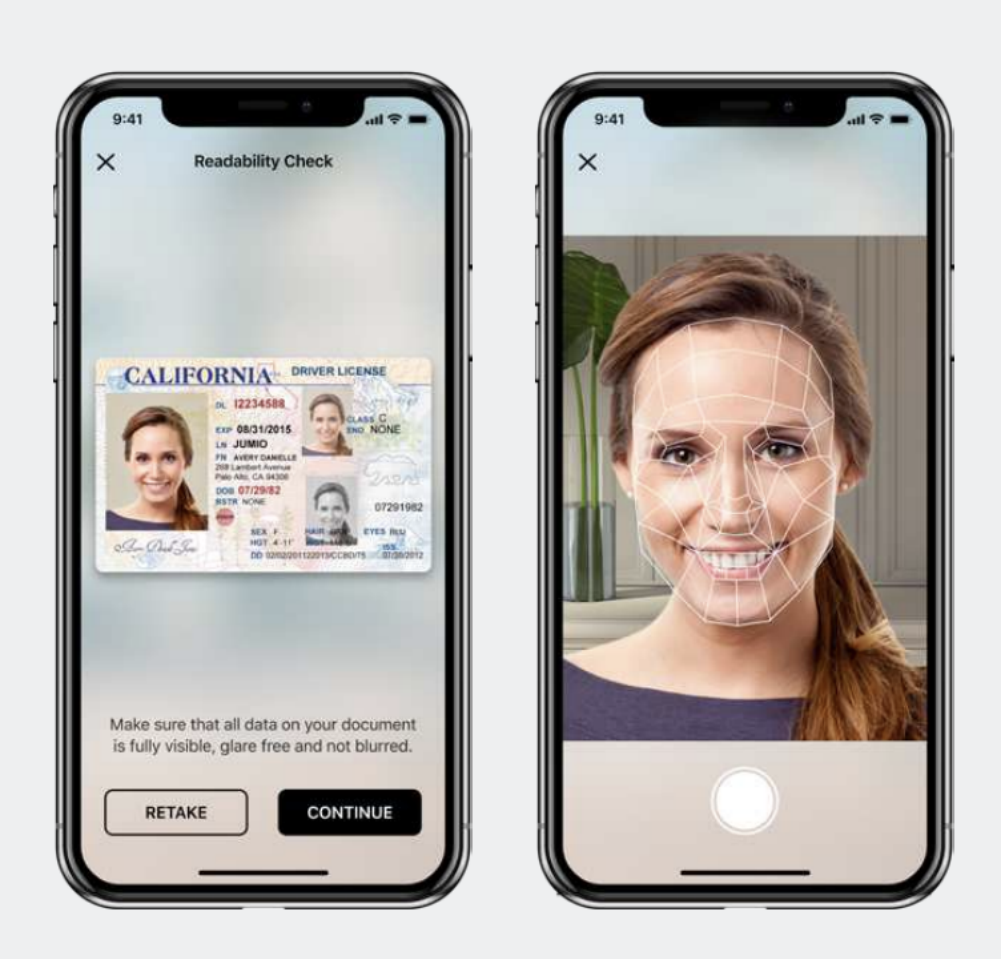
1. Log into your iProov tenant instance.
2. Go to **Service provider > Create new service provider**.
    1. Enter a name and select a suitable **Service Location**.
    2. Select the `Production`, `Development`, or `Testing` environment in which you want to use the service provider.
    3. Then click **Create**.
3. Note of the following provider details:
    1. **Service Location**
    2. **API Key**.
    3. **Primary API Secret**.
    4. **OAuth Username**.
    5. **Primary OAuth Password**.
4. Contact your iProv representative to ensure that **Liveness**, **GPA**, and **On Validate Return Frame** are enabled.

> **Note**
>
> - The **Validate Return Frame** enables the REST API to retrun the photo. The returned photo is an output to the shared node state by the iProov Authentication node.
> - **Liveness** and **GPA** need to be enabled to choose the **Assurance Type**.
> - Contact your iProov Representative to enable these parameters.

## Jumio identity verification

### Overview

PingOne Advanced Identity Cloud integrates with Jumio identity verification to let you capture and submit your government-issued ID documents easily and securely. Jumio helps you deter fraud and meet regulatory requirements by verifying ID verification in real time using machine learning. Jumio validates the user's identity, corroborates it with the user's selfie, and uses advanced liveness detection to ensure the person is actually present.




You implement Jumio integration in your PingOne Advanced Identity Cloud journeys using two authentication nodes:

- **Jumio Initiate**. This node triggers the Jumio's NetVerify service. This service scans a user's ID and their face to verify their identity. For more information refer to Jumio initiate node.
- **Jumio Decision**. This node decides if the user can successfully log in, and upon successful login, maps the Jumio properties to the PingOne Advanced Identity Cloud PingIDM attributes. For more information refer to Jumio decision node.

The user may not be authenticated due to the following reasons:

- If the ID scan is unreadable or the ID type is unsupported, the journey returns to the Jumio Initiate node, where the user can complete the NetVerify scan process.
- If the process fails or fraud is detected, authentication fails.

## Setting up the Jumio service in Advanced Identity Cloud

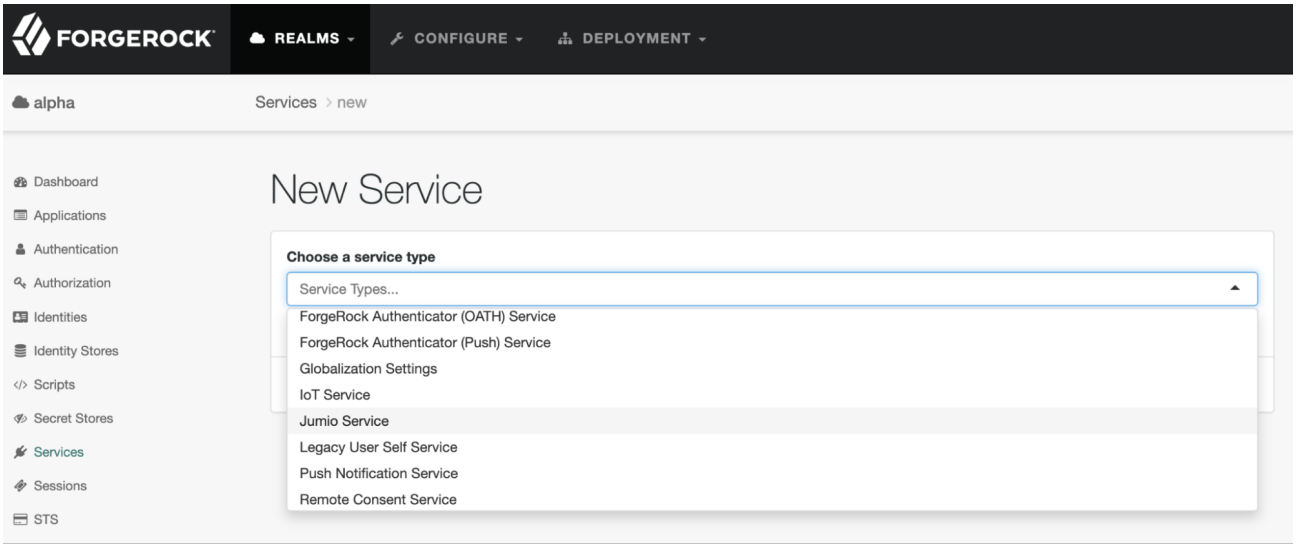
Follow these steps to set up the Jumio service:

1. Generate Jumio API token and secret:
    1. Sign in to the Jumio Customer Portal.
    2. From the toggle menu on the left, go to **Settings > Managed Services > Identity Verification**.

3. In the **IDENTITY VERIFICATION** menu, go to **Api Credentials > OAuth2 Clients**, and click **Generate a new API token**.
4. In the **Generate a new API token** window, select **Initialize** and **Retrieve & Delete** permissions, and click **Confirm**.
5. Enter your Jumio user password to create a new token.
6. Note the latest system generated token and secret. You will need these to configure the Jumio service in Advanced Identity Cloud admin UI.

2. Configure the Jumio service in Advanced Identity Cloud admin UI:

1. Using another browser window, log in to your tenant Advanced Identity Cloud admin UI.
2. From the left navigation, go to **Native Consoles > Access Management**.
3. In the AM admin UI, select **Services**. Then click **+ Add a Service**.
4. In the **Choose a service type** drop-down, select **Jumio Service**.




5. In the **New Service** window configure fields for authenticating and reporting to Jumio service.

> **Important**
>
> Do not put any personally identifiable information (PII) in these fields.
> You should have generated and noted the **Token** and **Secret** values as specified in the Note token step.
> You should have already configured **Merchant Reporting Criteria** and **Customer Internal Reference** when you set up your identity verification on the Jumio Customer Portal.
> The **Merchant Reporting Criteria** and **Customer Internal Reference** fields are used for reporting in the Jumio Customer Portal. You could use these fields for different use cases, for example for auditing purposes.

| Field | Description |
|---|---|
| **Server** | Select your deployment region - US, EU, or SG (APAC) |
| **Token** | Specify the token value you noted in the Note token step |

| Field | Description |
| --- | --- |
| **Secret** | Specify the secret value you noted in the Note token step |
| **Merchant Reporting Criteria** | Specify the reportingCriteria parameter you configured in your Jumio setup. |
| **Customer Internal Reference** | Specify the customerInternalReference parameter you configured in your Jumio set up. |
| **Redirect URI** | Specify https://*tenant-env-fqdn*/am |

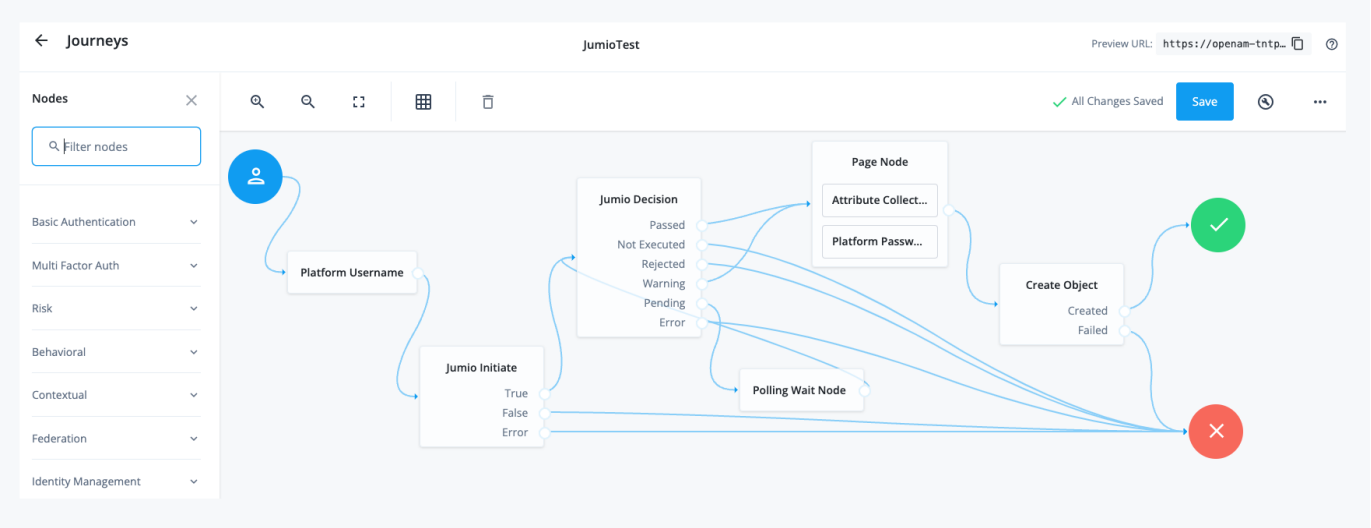
### Configuring a journey in PingOne Advanced Identity Cloud

The following example journey uses Jumio Initiate and Jumio Decision nodes to authenticate a user and create an identity:

- The Platform Username node gets the user's name.
- The Jumio Initiate node verifies if the user exists. If the user isn't found, the journey ends in a failure.
- If the user is found, Jumio Decision node is invoked to verify the level of authentication.
- If the outcome from Jumio Decision node is `Passed` or `Warning`:
  - The Attribute Collector node gathers additional user details.
  - The Platform Password node collects the user password.
  - The Create Object node creates a user identity based on the user details collected.

To create the sample journey described:

1. Log in to the Advanced Identity Cloud admin UI.
2. Go to **Journeys** > **+ New Journey**.
3. Enter a name for the journey, for example JumioTest. Select **Alpha realm-Users managed/alpha_user** as the **Identity Object**. Then click **Save**.
4. Drag and drop the nodes to create a journey as shown in the following diagram:




5. In the Jumio Decision node, map Jumio returned parameter attributes to the corresponding PingOne Advanced Identity Cloud IDM properties.

   The **Key** is the Jumio parameter, and the **Value** is the PingOne Advanced Identity Cloud IDM property. For example:

   - **Key** = firstName
   - **Value** = givenName

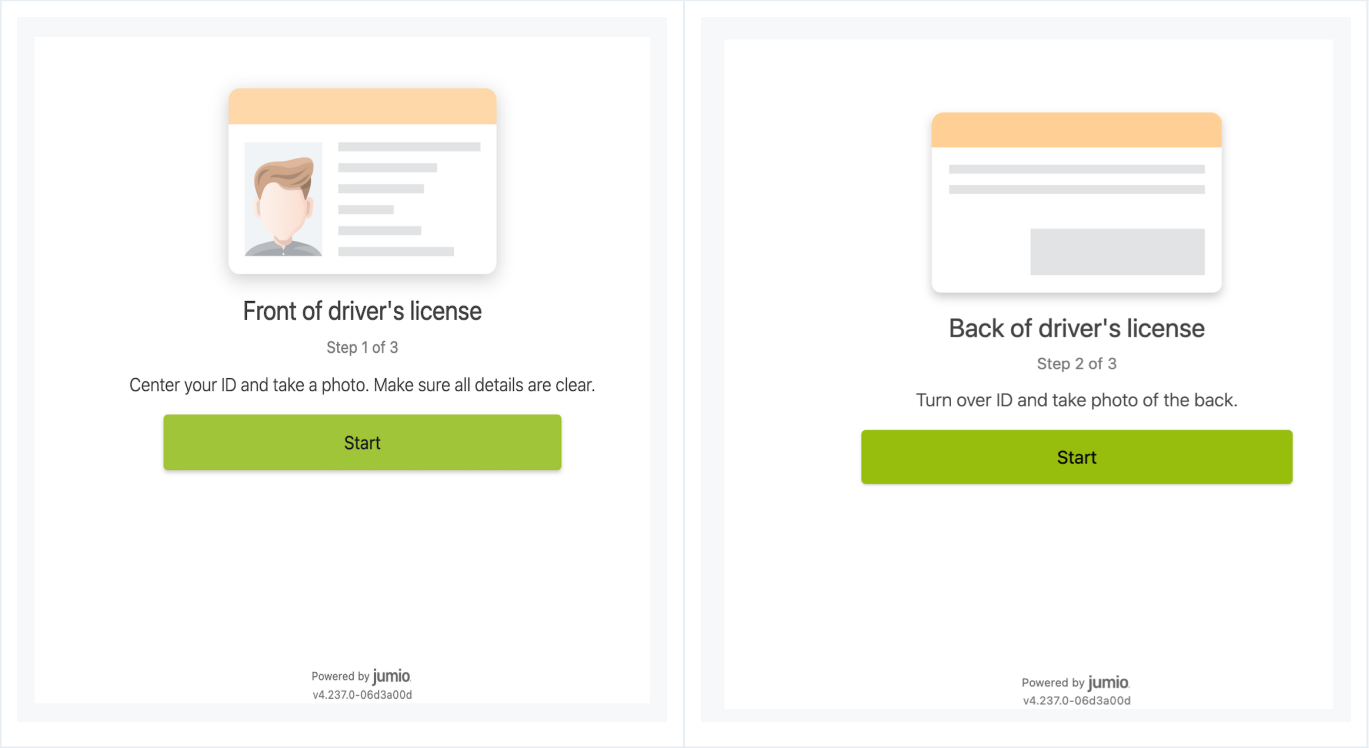
     In the above example, firstName is the Jumio parameter name for the first name of a user; and givenName is the PingOne Advanced Identity Cloud IDM property name for the first name of the user.

6. Click **Save**.

   The system generates a **Preview URL**. Use this preview URL for testing the journey and Jumio service you configured.

## Testing Jumio verification service integration

1. Copy and paste the preview URL in an incognito browser window.
2. Log in using your PingOne Advanced Identity Cloud credentials. Then click **Start** to start the verification process when the system prompt appears.
3. Select your **Region** and **ID type**.
4. Take a photo of the front and back of the ID using the webcam or upload the photo of the ID from your computer.




5. When prompted, complete the face verification step. If the verification is successful, the user is successfully authenticated.

## Jumio initiate node

Triggers Jumio transaction and redirects the user to the Netverify journey.

### Outcomes

**`True`**

The request completed successfully.

**`False`**

The request failed.

**`Error`**

An error occurred.

### Properties

This node has no configurable properties.

### Jumio decision node

This node decides if the user can successfully log in depending on the validity of their NetVerify scan, and upon successful login, maps the Jumio properties to the PingOne Advanced Identity Cloud PingIDM attributes. It also retrieves transaction result, updates the shared state and directs the user based on the outcome of the Netverify transaction. If the user's identity is successfully validated, the user is also authenticated.

**Outcomes**

**`Passed`**

Successfully verified the user identity.

**`Not Executed`**

The initiation node has not run successfully.

**`Rejected`**

Could not be successfully verified with the stored identity.

**`Pending`**

Identity verification is still pending.

**`Error`**

An error occurred during verification.

**Properties**

In the Jumio Decision node, Jumio returned parameter attributes are mapped to the corresponding PingOne Advanced Identity Cloud IDM properties.

The **Key** is the Jumio parameter, and the **Value** is the PingOne Advanced Identity Cloud IDM property.

## Microsoft Intune node

Lets you integrate with Microsoft Intune using Microsoft Graph APIs. Microsoft Intune lets you control features and settings on Android, Android Enterprise, iOS/iPadOS, macOS, and Windows 10/11 devices in your organization.

The Microsoft Intune node checks the device details and determines the device's compliance status. You can enable this node to save device information to shared state for subsequent use by other nodes in the journey.

For more information about Microsoft Intune, refer to the Microsoft Intune documentation.

## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Inputs

This node needs DeviceID to be in either the shared state or in the request header.

## Dependencies

You must set up Microsoft Intune before using this node in your PingOne Advanced Identity Cloud environment. Follow the Get started with your Microsoft Intune deployment document to set up and use a production version of Microsoft Intune.

Microsoft Intune requires a Microsoft Graph API application to be registered. Follow the register apps to use the Microsoft Graph API steps to complete the registration process.

Set the following Microsoft Intune and Graph API permissions:

| API / Permission | Type | Description |
|---|---|---|
| **Microsoft Intune** | | |
| `get_data_warehouse` | Delegated | Get data warehouse information from Microsoft Intune |
| `get_device_compliance` | Application | Get device state and compliance information from Microsoft Intune |
| `pfx_cert_provider` | Application | PFX certificate management |
| `scep_challenge_provider` | Application | SCEP challenge validation |
| `send_data_usage` | Application | Exchange device telecom and Wi-Fi data usage information with Microsoft Intune |
| `update_device_attributes` | Application | Send device attributes to Microsoft Intune |
| `update_device_health` | Application | Send device threat information to Microsoft Intune |
| **Microsoft Graph API** | | |
| `DeviceManagementManagedDevices.Read.All` | Delegated | Read Microsoft Intune devices |

| API / Permission | Type | Description |
| --- | --- | --- |
| `User.Read` | Delegated | Sign in and read user profile |

**Important**

The TLS handshake must be completed before the request accesses this node. During the TLS handshake, PingOne Advanced Identity Cloud obtains the device ID from the client certificate and sets it in the request header or in the shared state. You can use PingGateway to complete the TLS handshake.

### Verifying Intune setup

Use the following cURL commands to verify your Intune setup before using the node:

1. This first command retrieves the access token needed by the Intune node.
    1. Sample cURL request:

```
curl \
 --location 'https://login.microsoftonline.com/tenant ID/oauth2/v2.0/token' \
 --header 'Content-Type: application/x-www-form-urlencoded' \
 --data-urlencode 'client_id=application client ID' \
 --data-urlencode 'client_secret=client secret' \
 --data-urlencode 'scope=https://graph.microsoft.com/.default' \
 --data-urlencode 'grant_type=password' \
 --data-urlencode 'username=Azure admin username' \
 --data-urlencode 'password=Azure admin pwd'
```

    2. Sample response:

```
{
    "token_type": "Bearer",
    "scope": "profile openid email https://graph.microsoft.com/
DeviceManagementManagedDevices.Read.All https://graph.microsoft.com/User.Read https://
graph.microsoft.com/.default",
    "expires_in": 3971,
    "ext_expires_in": 3971,
    "access_token": "This will be the jwt token used to retrieve the device info cURL"
}
```

2. This second command retrieves the device info:
    1. Sample cURL request:

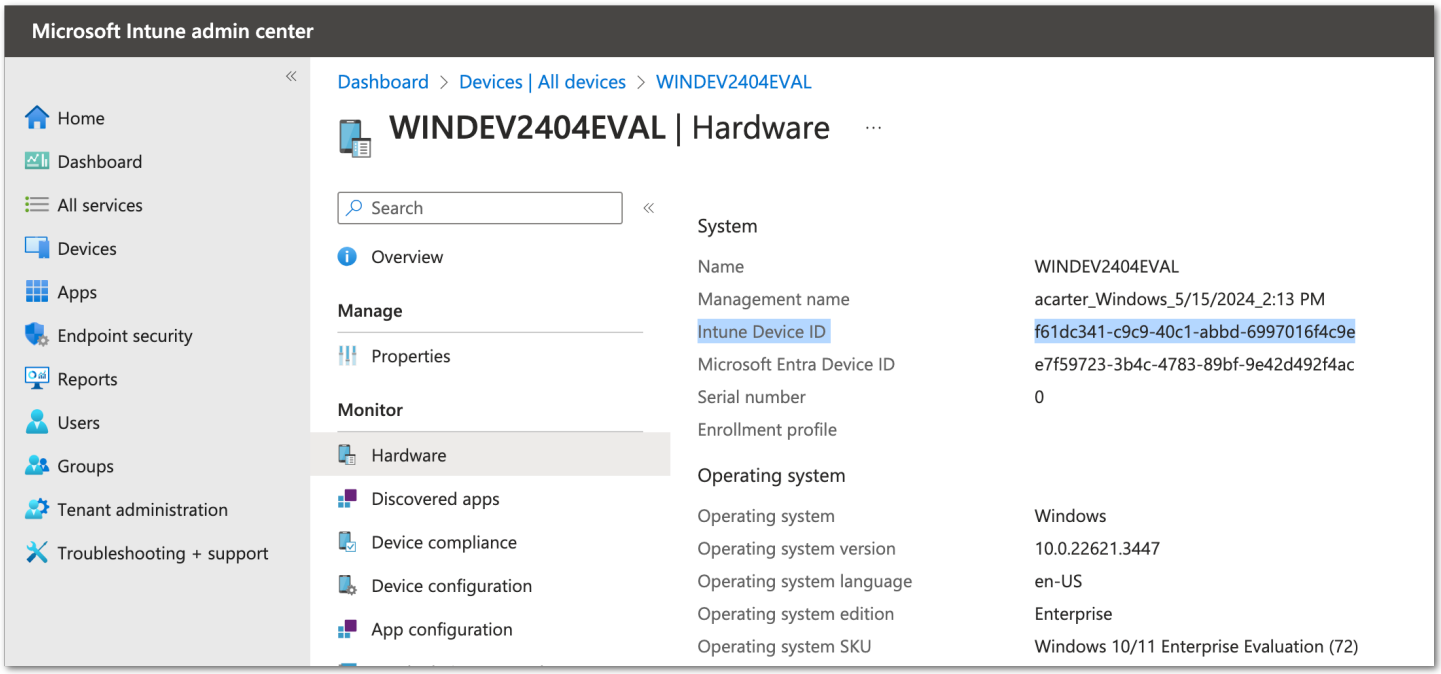
```
curl \
 --location 'https://graph.microsoft.com/v1.0/deviceManagement/manageddevices/Intune device ID'
\
 --header 'Authorization: Bearer This is the jwt token returned by the cURL command above'
```

2. Sample response:

```
{
    "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#deviceManagement/
managedDevices/$entity",
    "id": "f61dc341-c9c9-40c1-abbd-6997016f4c9e",
    "userId": "0f7e78ba-6e1f-485b-8b8b-fb86895f0498",
    "deviceName": "WINDEV2404EVAL",
    "managedDeviceOwnerType": "company",
    "enrolledDateTime": "2024-05-15T14:13:19.376941Z",
    "lastSyncDateTime": "2024-05-16T03:45:42.4309247Z",
    "operatingSystem": "Windows",
    "complianceState": "compliant",
    "jailBroken": "Unknown",
    "managementAgent": "mdm",
    "osVersion": "10.0.22621.3447",
    "easActivated": true,
    "easDeviceId": "24783A3196BAFE89291A012E4B607591",
    "easActivationDateTime": "2024-05-15T14:25:40.632661Z",
    "azureADRegistered": true,
    "deviceEnrollmentType": "windowsAzureADJoin",
    "activationLockBypassCode": null,
    "emailAddress": "",
    "azureADDeviceId": "e7f59723-3b4c-4783-89bf-9e42d492f4ac",
    "deviceRegistrationState": "registered",
    "deviceCategoryDisplayName": "",
    "isSupervised": false,
    "exchangeLastSuccessfulSyncDateTime": "0001-01-01T00:00:00Z",
    "exchangeAccessState": "none",
    "exchangeAccessStateReason": "none",
    "remoteAssistanceSessionUrl": "",
    "remoteAssistanceSessionErrorDetails": "",
    "isEncrypted": false,
    "userPrincipalName": "",
    "model": "VirtualBox",
    "manufacturer": "innotek GmbH",
    "imei": null,
    "complianceGracePeriodExpirationDateTime": "9999-12-31T23:59:59.9999999Z",
    "serialNumber": "0",
    "phoneNumber": null,
    "androidSecurityPatchLevel": null,
    "userDisplayName": "acarter",
    "configurationManagerClientEnabledFeatures": null,
    "wiFiMacAddress": null,
    "deviceHealthAttestationState": null,
    "subscriberCarrier": "",
    "meid": null,
    "totalStorageSpaceInBytes": 133661982720,
    "freeStorageSpaceInBytes": 74671194112,
    "managedDeviceName": "acarter_Windows_5/15/2024_2:13 PM",
    "partnerReportedThreatState": "unknown",
    "requireUserEnrollmentApproval": null,
    "managementCertificateExpirationDate": "2025-05-15T03:56:04Z",
    "iccid": "",
    "udid": "",
    "notes": null,
```

```
    "ethernetMacAddress": "08002732B5A2",
    "physicalMemoryInBytes": 0,
    "enrollmentProfileName": "",
    "deviceActionResults": []
}
```

Ensure you are using the correct device ID when performing the verification tests. You can retrieve the Intune device ID from the Hardware section:




## Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| DeviceID Attribute Name | Name of the header or shared state variable for storing the deviceID value. The value is the `SSL_Client_S_DN` in the client certificate presented at the TLS termination gateway. The format should be: `CN=f47d8e59-b60e-48a5-adc1-622cb2244zzz`. |
| DeviceID in SharedState | Boolean.<br>• If `enabled`, the node takes the device ID from shared state.<br>• If `disabled`, the node takes the device ID from the request header.<br>Default: `disabled`.<br>The DeviceID in SharedState toggle is used to determine where to look for the device ID value. If the DeviceID in SharedState toggle is enabled, the device ID is available in the shared state, and if it's disabled, the device ID is in the header. |

| Property | Usage |
| --- | --- |
| Tenant ID | Tenant's global unique identifier (GUID) in Azure Active Directory (AD). |
| Application (client) ID | The application ID or client ID is a value the Microsoft identity platform assigns to your application when you register it in Azure AD. |
| Client Secret | Sometimes called an application password, a client secret is a string value your application can use in place of a certificate to identify itself. |
| Azure Admin User Name | The administrative username in Azure. |
| Azure Admin User Password | The administrative password in Azure. |
| Save Device Properties to SharedState | If enabled, the device information is saved to the shared state with `INTUNE_` prepended to the key name. Null and empty string values are not placed into shared state.<br>Refer to the Properties table in Microsoft Intune documentation for the managed device properties. |
| Save installed apps to SharedState | If enabled, the apps installed on the Mobile Device are extracted and saved to the Shared State with the key name - `INTUNE_INSTALLED_APPS`. |

## Outputs

- If `Save Device Properties to SharedState` is enabled, the device property available in Microsoft Intune service is stored on the shared state.
- If `Save installed apps to ShatedState` is enabled, the details of application installed on the device are stored on the shared state.

## Outcomes

### `Compliant`

The device is compliant.

### `Not Compliant`

The device does not comply with the policy.

### `In Grace Period`

The device is not-compliant, but it's in the grace period defined by the administrator.

### `Config Manager`

The device is managed by Microsoft Intune's Configuration Manager.

`Conflict`

Multiple settings are applied to the same device and Intune can't sort out the conflict. An administrator should review.

`No Id`

No device ID is found in the header or shared state.

`Status Unknown`

The device is offline or failed to communicate with Intune or Azure AD.

`Error`

An error occurred within the node. Related stacktrace and message are placed in the shared state.

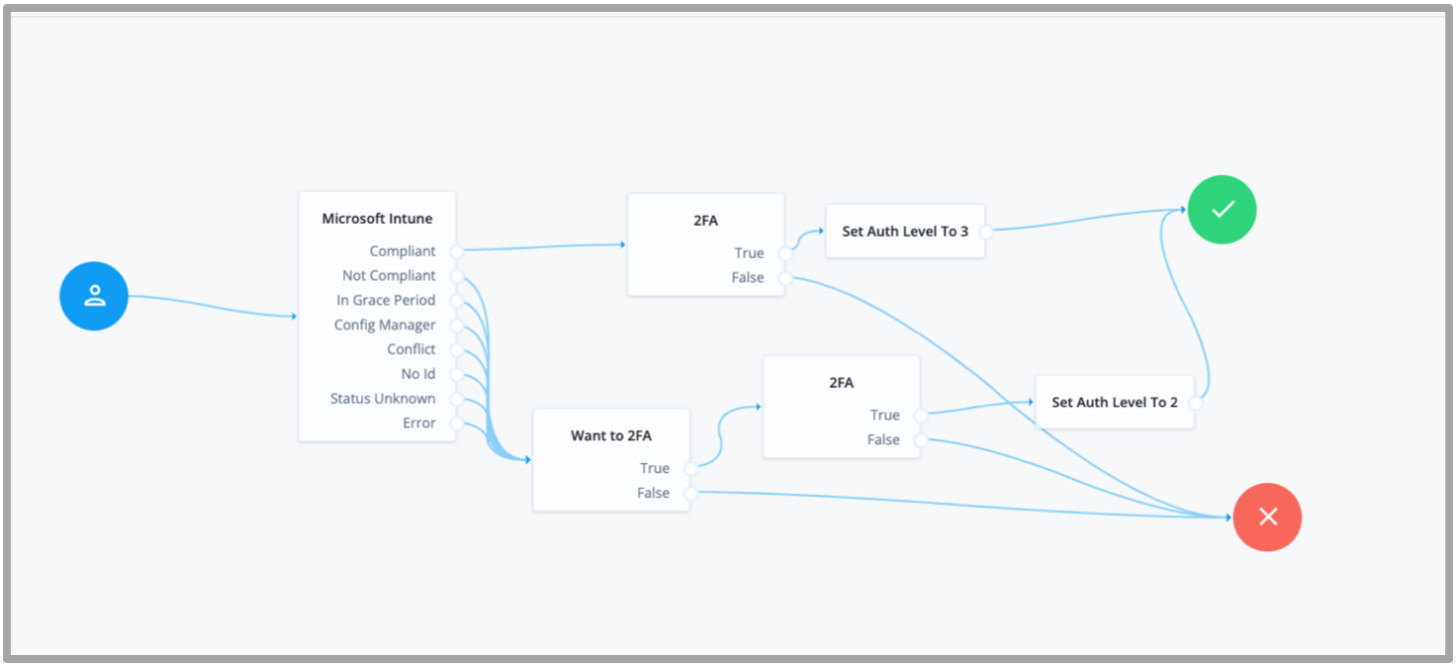
Learn more in the Microsoft Graph API documentation on Compliance State.

## Errors

Review the log messages to find the reason for the error and address the issue appropriately.

## Examples

An example journey using the Microsoft Intune node:




In this sample journey, the `Microsoft Intune` node is invoked to verify if the client device is compliant. If the device is found compliant, a two-factor authentication node is invoked to authenticate the user. Upon successful authentication, the user is set to higher authentication level. If the device is not found compliant, the user is still given an option to go through a two-factor authentication node. In this case, the authenticated user is set to a lower authentication level.

# LexisNexis One-Time Passcode (OTP)

LexisNexis One Time Password (OTP) is an out-of-band identity proofing method that provides stronger authentication for high risk, high-value transactions. It sends a unique time-sensitive random passcode via SMS text, email, or phone to a user's existing phone or personal computer. No additional hardware, such as an electronic fob, is required.

PingOne Advanced Identity Cloud lets you implement identity proofing with LexisNexis OTP using these authentication nodes:

- LexisNexis OTP Sender node
- LexisNexis OTP Collector node
- LexisNexis OTP Decision node

## Setup

The LexisNexis Dynamic Decision Platform (DDP) portal configuration is required for enabling integration with PingOne Advanced Identity Cloud. Complete the following configuration steps:

1. Get configuration details such as OrgID and API key for the REST API interface from the LexisNexis DDP portal.
2. Configure LexisNexis DDP portal policies to enable access to LexisNexis OTP services from PingOne Advanced Identity Cloud.

### Get configuration details from the LexisNexis DDP portal

To retrieve the OrgID and API Key from the LexisNexis DDP portal, perform these steps:

1. Log in to your account on the LexisNexis DDP portal.
2. On the home page, select the user information drop-down to display username, OrgName, and OrgID. Make a note of the OrgID. You'll need it to configure LexisNexis OTP nodes.
3. Select Admin > API Keys tile to retrieve the API Key. You'll need it to configure LexisNexis OTP nodes. If no API key is listed, select Create New API Key to generate a new key.[1]

### Configure LexisNexis OTP policies on PingOne Advanced Identity Cloud

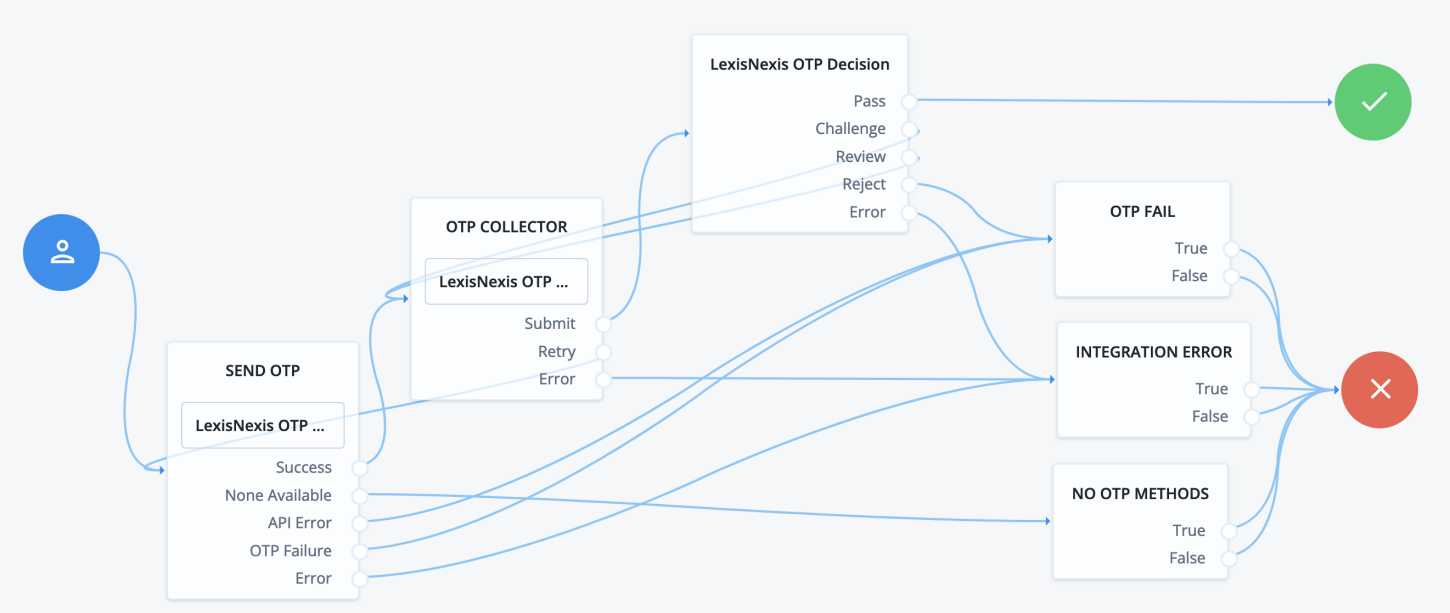
You must configure OTP policies before you can integrate PingOne Advanced Identity Cloud as the authentication hub. To get started with simple policy rules, perform the following steps:

1. Log into the LexisNexis DDP portal using your administrative account.
2. On the DDP portal home page, go to Policies > Create > New Policy (Standard).
3. On the Policy Summary page, the Properties tab displays. Enter OTP in the Policy Name field, select Active, and update the status thresholds for Reject and Review to -20 and 20 respectively.
4. Select the Rules tab, and set the OTP policy rules using the Authentication Rule Editor. The OTP policy is a single authentication rule that integrates the authentication hub.
5. Save the policy.

Consult with LexisNexis ThreatMetrix services for a more comprehensive policy configuration.

## Example

This example shows a sample authentication journey using LexisNexis OTP Nodes for identity proofing and multi-factor authentication (MFA). The sample journey lets the user select the OTP delivery method, generates an OTP code, collects and verifies the user-entered code and authenticates the user on success.




The flow is as follows:

- The LexisNexis OTP Sender node user selects the OTP delivery method.
- The LexisNexis OTP Collector node lets the user input the received OTP.
- The LexisNexis OTP Decision node verifies the user entered OTP and authenticates the user.
- OTP Fail node, a message node, displays the failure condition such as user rejection, API failure, or OTP failure.
- Integration Error node, a message node, displays the errors such as network time out at LexisNexis DDP portal.
- No OTP Methods node, a message node, displays if no OTP delivery method has been configured.

1. This API Key is required to configure the LexisNexis OTP nodes. The API Key is to be protected. Do not email or keep this value in clear text on any computer system.

## LexisNexis OTP Sender node

The LexisNexis One-Time Password (OTP) Sender node displays a list of available methods to send an OTP code. This supports sending OTP over email, SMS, and voice. The choice of OTP delivery method depends upon:

- Whether the capability is activated in the node
- If the necessary attribute is available in the user directory for the user

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

The LexisNexis OTP Sender node reads the `username` from shared state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

During the journey, the node queries the user directory with the username to fetch the attributes required by subsequent nodes in the journey.

### Dependencies

To use this node, you should have already set up PingOne Advanced Identity Cloud integration with the LexisNexis service.

### Configuration

| Property | Usage |
|---|---|
| Org ID | A unique ID associated with your organization in the Dynamic Decision Platform (DDP). |
| API Key | The unique API key generated in the DDP portal to associate with the organization. |
| OTP URL | The URL for the DDP authentication hub API endpoint. The default URL is the worldwide endpoint. This should be modified for specific regions such as EU, US, or IN (India). |
| Policy | The DDP portal policy for associating the DDP authentication hub with the OTP. |
| OTP Length | Length of the OTP |
| OTP Expire | OTP validity time in minutes. |
| Send Email | Toggle to enable or disable OTP delivery over email. |
| Email Title | The title of the email sent to the user when OTP over email is triggered. |
| Email Message | The message body of the email sent to the user when email is triggered. |
| Email Attribute | The attribute in the user directory to get the email address for sending the OTP code at runtime. If the value is present, then this OTP delivery method is displayed; otherwise, this method is not displayed. |

| Property | Usage |
| --- | --- |
| Send SMS Text | Toggle to enable or disable OTP delivery over SMS. |
| SMS Message | The message body of the SMS text message sent to the user when SMS/OTP is triggered. |
| SMS Attribute | The attribute to get the mobile phone number for sending the OTP over SMS. If the value is present, then this OTP delivery method is displayed; otherwise, this method is not displayed. |
| Send Voice | Toggle to enable or disable OTP delivery over voice. |
| Voice Attribute | The attribute in the user directory to get the user's phone number to convey the OTP code over automated voice. If the value is present, then this OTP delivery method is displayed; otherwise, this method is not displayed. |

### Outputs

In case of an `Error` outcome, the error message is output to the shared state.

### Outcomes

#### `Success`

The OTP code was successfully generated for the user. OTP generation does not guarantee delivery to the user's device. The LexisNexis OTP Collector node allows for retry if the user does not receive the OTP.

#### `None Available`

No OTP delivery methods are available because the required attributes required are not configured for that user.

#### `API Error`

There is an issue with the API request, such as a network timeout, or the service is unavailable.

#### `OTP Fail`

The API request is rejected by the LexisNexis DDP authentication hub. The actual error code is logged in the debug log for the node.

#### `Error`

There is an integration error or a new bug is discovered. Check the debug log files for any integration issues.

### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error. If the configuration looks accurate, then open a support case with LexisNexis.

### Example

Refer to the Example LexisNexis OTP journey.

## LexisNexis OTP Collector node

The LexisNexis One-Time Password (OTP) Collector node collects the OTP code sent to the user. This node lets the user submit an OTP code for decision and validation. The user can also request the OTP code be resent.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

There are no inputs for this node.

### Dependencies

Requires the LexisNexis OTP Sender node earlier in the journey.

### Configuration

| Property | Usage |
|---|---|
| Message Body | The message displayed on the collector interface. The variable otpDestination contains an email address or a phone number depending on the OTP delivery method selected by the user in the LexisNexis OTP Sender node. |
| Help Text | The help text displayed in the OTP text entry box for the user. |
| OTP Error Message | The message displayed when the user enters an incorrect OTP code. |
| OTP Blank Message | The message displayed when the user attempts to submit a blank OTP code. |

### Outputs

In case of an `Error` outcome, the error message is output to the shared state.

#### Outcomes

##### `Submit`

The user selected the "Submit" button. This must be linked to the LexisNexis OTP Decision node to validate the OTP Code submitted. If the LexisNexis OTP Decision node detects a blank or invalid OTP code, the journey returns to the LexisNexis OTP Collector node, and an appropriate message is displayed.

> **Note**
>
> The `Review` and `Challenge` outcomes from the LexisNexis OTP Decision node must be linked back to the LexisNexis OTP Collector node.

##### `Retry`

The user requests a new OTP code by selecting the "Retry" button on the interface.

##### `Error`

There is an integration error. Check the debug log files for any integration issues.

#### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error. If the configuration looks accurate, then open a support case with LexisNexis.

#### Example

Refer to the Example LexisNexis OTP journey.

### LexisNexis OTP Decision node

The LexisNexis OTP Decision node compares the OTP code entered by the user with the one in the shared state.

#### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

#### Inputs

There are no inputs to this node.

### Dependencies

Requires LexisNexis OTP Sender node and LexisNexis OTP Collector node to be configured in the journey. Typically, the LexisNexis OTP Collector node precedes this node in an authentication journey. The collector node places the OTP submitted by the user in shared state. Additionally, the LexisNexis OTP Sender node precedes the LexisNexis Collector node, which places the characteristics of the OTP into shared state.

### Configuration

| Property | Usage |
| --- | --- |
| Org ID | The unique ID associated with your organization in the Dynamic Decision Platform (DDP). |
| API Key | The unique API key generated by the DDP portal and associated with the `Org ID`. |

### Outputs

In case of an `Error` outcome, the error message is output to the shared state.

### Outcomes

#### `Pass`

The OTP code submitted by the user is valid, and MFA using OTP is successful.

#### `Challenge`

The OTP code submitted by the user failed validation, and the permitted retry limit has not been reached. This outcome must be linked to the LexisNexis OTP Collector node.

#### `Review`

The user submitted blank OTP code. This outcome must be linked to the LexisNexis OTP Collector node.

#### `Reject`

The user submitted OTP code is invalid and the retry limit has been reached.

#### `Error`

There is an integration error. Check the debug log files for any integration issues.

### Troubleshooting

If this node logs an error, review the log messages to find the reason for the error. If the configuration looks accurate, open a support case with LexisNexis.

### Example

Refer to the Example LexisNexis OTP journey.

# OneSpan

PingOne Advanced Identity Cloud lets you integrate with OneSpan Intelligent Adaptive Authentication (IAA) and Risk Analytics (RA) to help reduce fraud, improve customer experience, and meet compliance requirements.

Advanced Identity Cloud provides these artifacts for OneSpan authentication journeys:

- One configuration service
- Sixteen authentication nodes
- Four sample nodes that are used only for development and testing

## Quick start with sample journeys

PingOne Advanced Identity Cloud provides sample journeys to help you understand and address the most common OneSpan use cases. To use the samples, perform these steps:

1. Add the OneSpan Configuration service in your Advanced Identity Cloud environment.
2. Download the JSON files for sample journeys from here.
3. Import the downloaded sample journeys into your Advanced Identity Cloud environment.

For more information on sample journeys, refer to the OneSpan sample journeys.

## Set up

Before using OneSpan nodes in your PingOne Advanced Identity Cloud environment, you must set up:

- OneSpan tenant setup
- OneSpan Configuration auxiliary service.

### OneSpan tenant setup

This section gives you a brief introduction to set up your OneSpan tenant and get started with using OneSpan in PingOne Advanced Identity Cloud.

For OneSpan IAA and RA users:

1. Create a OneSpan Developer Community account.
2. Log in to the community portal and register a OneSpan Trusted Identity Platform Sandbox.
3. Set up a mobile application integrated using the Mobile Security Suite documentation. To get stated, install the OneSpan Mobile IAA Demo App on your phone.
4. Configure the Risk Analytics Presentation service.

For OneSpan OCA Users:

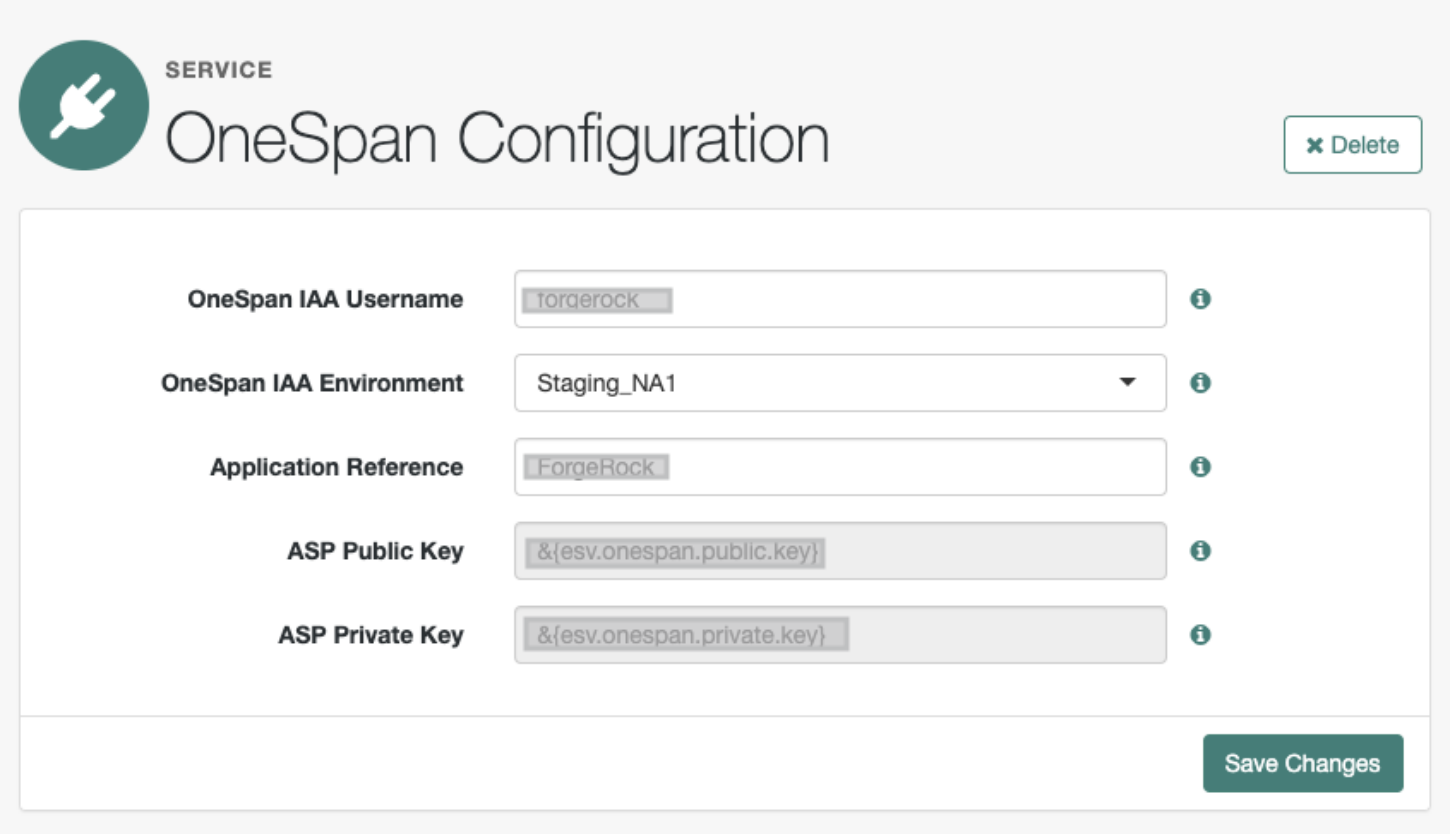
1. Create a OneSpan Developer Community account.

2. Log in to the community portal, and register a OneSpan Trusted Identity Platform Sandbox.
3. Install a mobile application to act as the Digipass authenticator by doing one the following:
    - Download a sample MAS app from the demo site, and then customize and build your own Mobile Authenticator Studio (MAS) app.
    - Download OneSpan Mobile Authenticator from Google Play Store or Apple App Store.

> **Note**
>
> To use the mobile authenticator from Apple App Store or Google Play Store, you need to configure a mobile authenticator license in your tenant.

**PingOne Advanced Identity Cloud service setup**

You must set up OneSpan Configuration service in your PingOne Advanced Identity Cloud environment before configuring and using OneSpan journeys.




1. In your AM admin UI navigate to REALMS > your realm.
2. Click Service Management in the dashboard.
3. Click Add a service and select OneSpan Configuration from the dropdown list.
4. Enter the following configuration details about the new service:
    - OneSpan IAA username: Username for the OneSpan tenant

- OneSpan IAA Environment: Type of environment for the OneSpan tenant
- Application Reference: A descriptive value for the integrated application
- ASP Public Key: Application service provider's public key
- ASP Private Key: Application service provider's private key
- Custom URL: URL to be used for OneSpan API calls when the IAA Environment parameter is set to CUSTOMIZED

5. Click Create.

After configuring the service, you can configure and use OneSpan authentication nodes.

## OneSpan nodes

| | | | |
|---|---|---|---|
| OneSpan Auth Activate Device | OneSpan Auth Add Device | OneSpan Auth Check Activation | OneSpan Auth Check Session Status |
| OneSpan Auth User Register | OneSpan Auth User Login | OneSpan Auth Generate Challenge | OneSpan Auth Validate Event |
| OneSpan Auth Validate Transaction | OneSpan Auth Visual Code | OneSpan Auth Hide Visual Code | OneSpan Risk Analytics Send Transaction |
| OneSpan Risk CDDC | OneSpan Auth VDP User Register | OneSpan Auth Generate VOTP | OneSpan Auth Assign Authenticator |
| | OneSpan Get User Authenticator | OneSpan Identity Verification | |

### OneSpan sample nodes

> **Note**
>
> The sample nodes are provided only for development and testing purposes. They must not be used in production environments.

| | |
|---|---|
| OneSpan Sample Attributes Collector | OneSpan Sample Error Display |
| OneSpan Sample Store Command | OneSpan Sample Transaction Collector |

#### OneSpan Auth Activate Device node

The OneSpan Auth Activate Device node prompts for the signature returned by the Digipass authenticator and finalizes the device activation process.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
| --- | --- | --- |
| Registration ID | As specified in the property | Shared state |
| Digipass signature | As specified in the property | Shared state |

### Dependencies

This node uses the device code and signature values collected by other OneSpan nodes in the journey to activate a device.

### Configuration

There are no configurable properties for this node.

### Outputs

There are no outputs from this node.

### Outcomes

*Success*

The device was successfully activated.

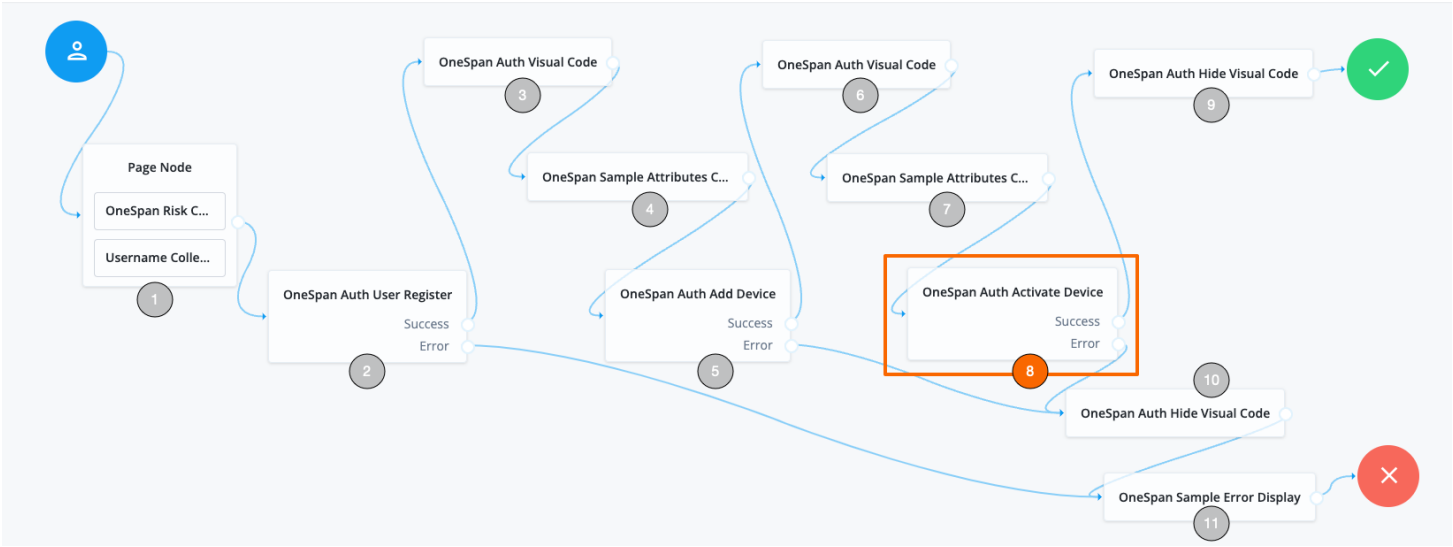
*`Error*

The device wasn't found in the pending activation list.

### Troubleshooting

Review the log messages and address the issue appropriately.

### Examples




This example shows the user registration and device activation journey and highlights the use of the OneSpan Auth Activate Device node:

1. The Username Collector and OneSpan Risk CDDC nodes collect the user's name and device fingerprint data and store them in the shared state.
2. The OneSpan Auth User Register node registers the user.
3. The OneSpan Auth Visual Code node uses the app to obtain a visual authentication code from OneSpan and places it in the shared state.
4. The OneSpan Sample Attributes Collector node collects custom device attributes and stores them in the shared state.
5. The OneSpan Auth Add Device node prompts user for the device code, to compare with the stored device code.
6. Upon successful device code comparison, the process continues to the next OneSpan Auth Visual Code node to get the device signature.
7. The OneSpan Sample Attributes Collector node collects the device signature.
8. The OneSpan Auth Activate Device node prompts the user for the device signature to authenticate the device.
9. When the device is authenticated successfully, the OneSpan Auth Hide Visual Code node stops displaying the visual code and completes the successful authentication.
10. If there's an error at the OneSpan Auth Add Device node or at the OneSpan Auth Activate Device node, the OneSpan Auth Hide Visual Code node is invoked to stop displaying the visual code.
11. The OneSpan Sample Error Display node displays the error message and the authentication fails.

### OneSpan Auth Add Device node

The OneSpan Auth Add Device node prompts for a device code to add the device for authentication.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the device code as input.

### Dependencies

This node depends on other OneSpan nodes that obtain and store the device code in the shared state.

### Configuration

This node has no configurable properties.

### Outputs

This node outputs the cronto signature and activation message.

### Outcomes

This node returns the following two outcomes:

*Success*

The device was added to the device list successfully.

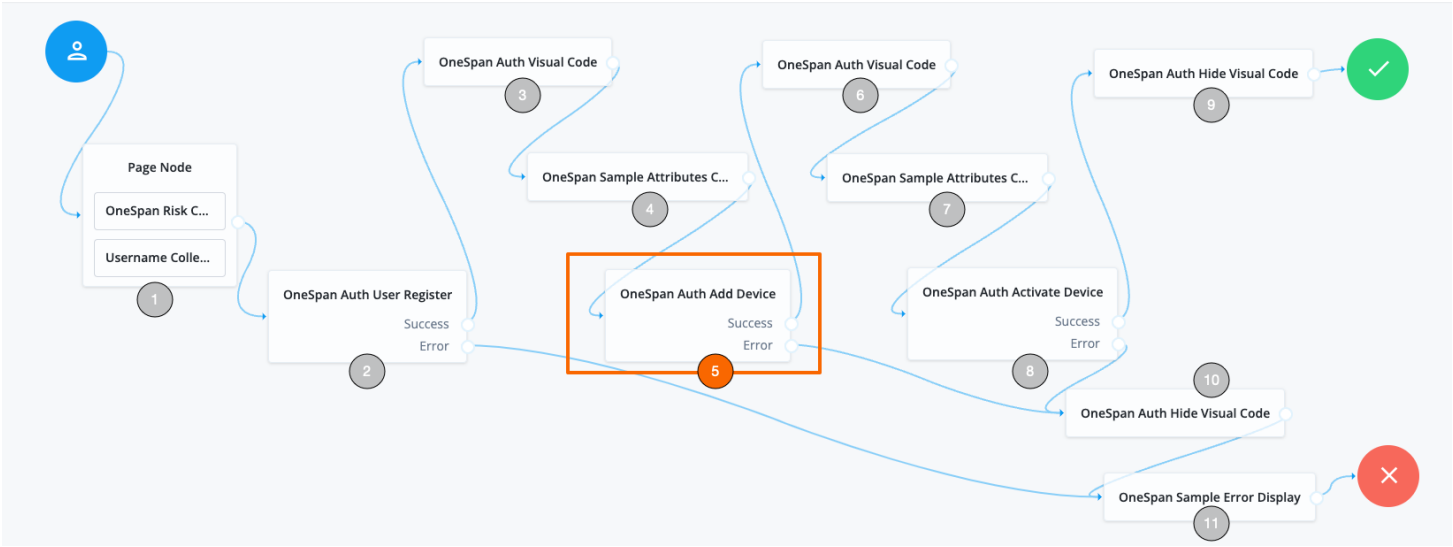
*Error*

There was an error in adding the device to the device list.

### Troubleshooting

If this node logged an error, review the log messages and the error display node if you have configured in the journey, and then address the issue appropriately.

### Examples




This example shows the user registration and device activation journey and highlights the use of the OneSpan Auth Add Device node:

1. The Username Collector and OneSpan Risk CDDC nodes collect the user's name and device fingerprint data and store them in the shared state.
2. The OneSpan Auth User Register node registers the user.
3. The OneSpan Auth Visual Code node uses the app to obtain a visual authentication code from OneSpan and places it in the shared state.
4. The OneSpan Sample Attributes Collector node collects custom device attributes in the shared state.
5. The OneSpan Auth Add Device node prompts user for the device code, to compare with the stored device code.
6. Upon successful device code comparison, the process continues to the next OneSpan Auth Visual Code node to get the device signature.
7. The OneSpan Sample Attributes Collector node collects the device signature.
8. The OneSpan Auth Activate Device prompts user for the device signature to authenticate the device.
9. When the device is authenticated successfully , the OneSpan Auth Hide Visual Code node stops displaying the visual code and completes the successful authentication.
10. If there's an error at the OneSpan Auth Add Device node or at the OneSpan Auth Activate Device node, the OneSpan Auth Hide Visual Code node is invoked to stop displaying the visual code.
11. The OneSpan Sample Error Display node displays the error message and the authentication fails.

## OneSpan Auth Check Activation node

The OneSpan Auth Check Activation node checks the status of a pending device activation.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node gets the username, event expiry date, and cronto image as inputs from the shared state.

### Dependencies

This node requires that you have already configured OneSpan Risk CDDC, OneSpan Auth User Register, and OneSpan Auth Visual Code nodes.

### Configuration

This node has no configurable properties.

### Outputs

This node has no outputs.

### Outcomes

*Pending*

The device is in the process of activation but not yet activated.

*Activated*

The device is activated and available for authentication.

*Timeout*

The process of verifying device activation has timed out.

*Unknown*

The device can’t be located in the registered device list.

*Error*

An error occurred while verifying device activation.

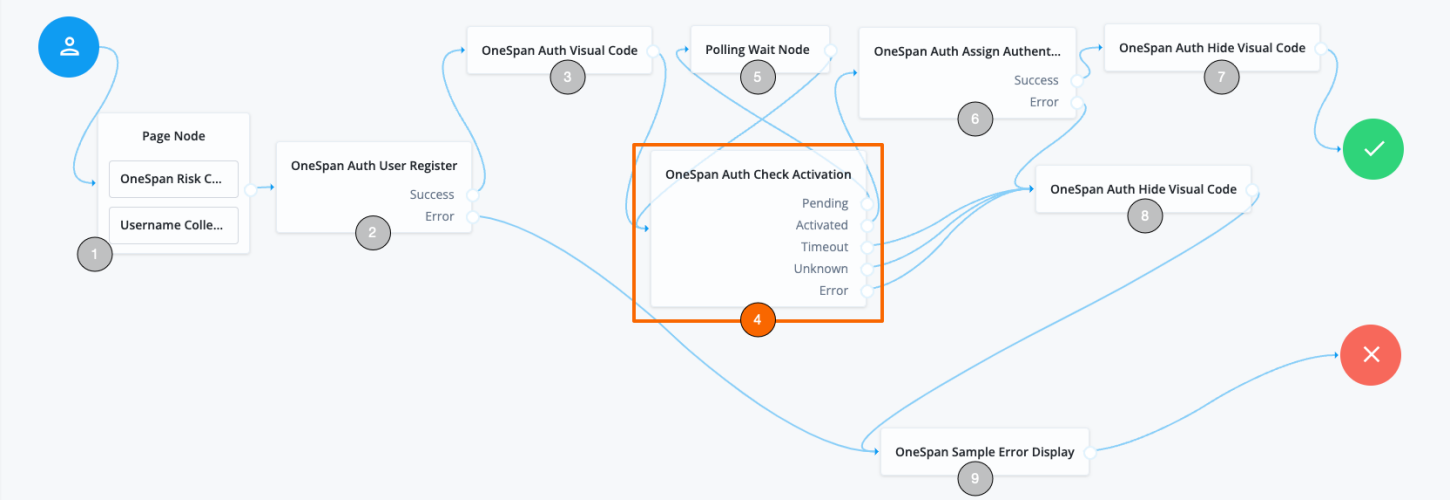
### Errors

If any error is encountered, then the error condition is stored in the shared state.

### Troubleshooting

If this node logged an error, review the log messages and address the issue appropriately.

### Examples




This example shows the device activation verification journey, highlighting the use of the OneSpan Auth Check Activation node:

1. The Username Collector and OneSpan Risk CDDC nodes collect the user's name and device fingerprint data and store them in the shared state.
2. The OneSpan Auth User Register node registers the user.
3. The OneSpan Auth Visual Code node uses the app to obtain a visual authentication code from OneSpan and places it in the shared state.
4. The OneSpan Auth Check Activation node verifies the status of device activation.
5. If device activation is in a pending state, the Polling Wait Node pauses the journey to wait and repeat the OneSpan Auth Check Activation node set number of times or activation is complete.
6. Upon successful device activation, the process continues to the OneSpan Auth Asssign Authenticator node.
7. When the device is assigned an authenticator successfully, the OneSpan Auth Hide Visual Code node stops displaying the visual code and completes the journey.
8. If there's an error outcome at the OneSpan Auth User Register node, or there's a timeout, unknown or error outcome at the OneSpan Auth Check Activation node; the OneSpan Auth Hide Visual Code node is invoked to stop displaying the visual code.
9. The OneSpan Sample Error Display node displays the error message and the authentication fails.

### OneSpan Auth Check Session Status node

The OneSpan Auth Check Session Status node checks the status of an authentication request session.

#### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

#### Inputs

This node gets the username, expiry date, and register code as inputs from the shared state.

#### Dependencies

This node requires that you have already configured OneSpan Risk CDDC and OneSpan Auth Visual Code nodes.

#### Configuration

This node has no configurable properties.

#### Outputs

This node has no outputs.

#### Outcomes

*Pending*

The session validation was not complete yet.

*Acccepted*

The session was authenticated successfully.

*Refused*

Access was refused by the user.

*Failure*

The session failed authentication.

*Timeout*

Session expiry date was reached before authentication.

### *Unknown*

OneSpan didn't find the pending activation session.

### *Error*

The session resulted in erroneous step-up authentication.

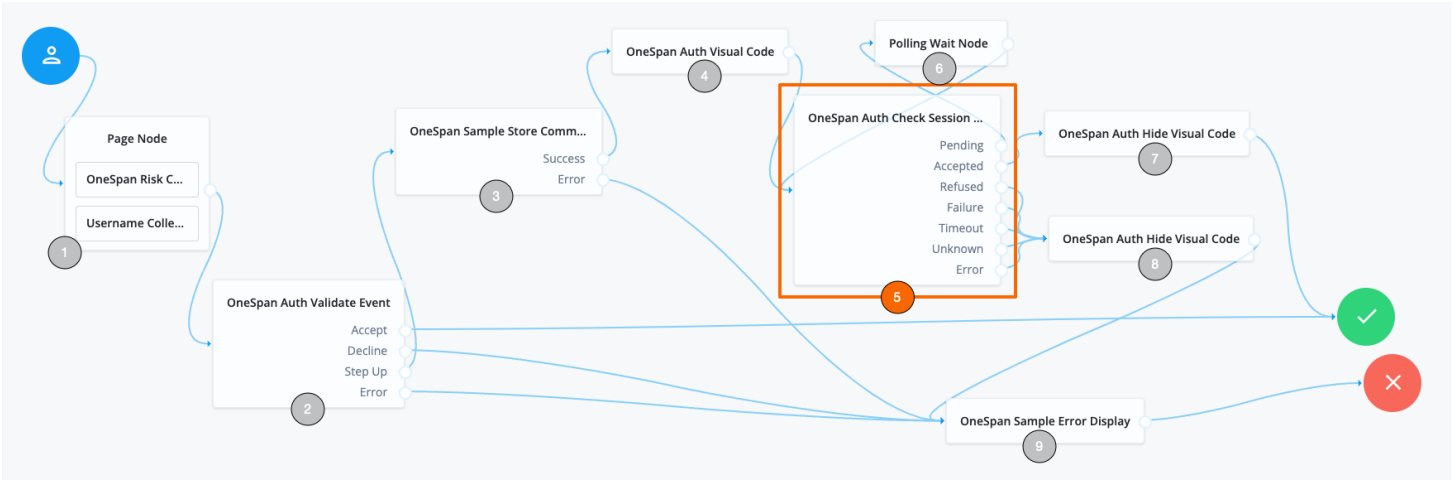
### Errors

Error codes are stored in shared state, and can be displayed using the OneSpan Sample Error Display node.

### Troubleshooting

If there's an error outcome from this node, review the logged messages and errors displayed in the OneSpan Sample Error Display node. Based on the error messages appropriately address the issue and retry the session.

### Examples




This example shows an authentication journey, highlighting the use of the OneSpan Auth Check Session Status node:

1. The Username Collector and OneSpan Risk CDDC nodes collect the user's name and device fingerprint data and stores them in the shared state.
2. If session is successfully validated, the journey is successfully completed.
3. If the session requires a step-up authentication, the OneSpan Sample Store Command node is invoked.
4. If OneSpan Sample Store Command node completes successfully, the OneSpan Auth Visual Code node is invoked.
5. The OneSpan Auth Visual Code node obtains the required step-up authentication code.
6. The OneSpan Auth Check Session Status node validates the step-up authentication.
7. If the step-up authentication is in a pending state, the Pending Wait Node is invoked to retry the step-up authentication.
8. If the step-up authentication ends in failure, timeout, unknown, or error status, then the journey is unsuccessful and the visual code is removed.

9. If the journey is unsuccessful, the OneSpan Sample Error Display node displays the error message.

### OneSpan Auth VDP User Register node

This node registers users to authenticate using the virtual one-time password (VOTP). You can design a user registration journey with this node on its own. You can also position this node after the OneSpan Auth User Register node. Both IAA and OCA authentication can use VDP delivery.

#### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

#### Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
|---|---|---|
| Username | As specified in the property | Shared state |
| Password (optional) | As specified in user attributes | Transient state |
| User attributes | As specified in the property | Shared state |

This node uses input from OneSpan Auth VDP User Registration node and another node that collects the VOTP delivery method.

#### Dependencies

To use this node, you should have already set up Advanced Identity Cloud integration with OneSpan, as mentioned in Set up.

#### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| IAA Domain | The domain in which the user account resides. In a sandbox, the domain is the same as your tenant name. |

| Property | Usage |
| --- | --- |
| User Name In SharedState | The key parameter in shared state that represents the OneSpan IAA username. |
| VDP Delivery Method | The mode in which VOTP is delivered. The available values are SMS, Email, Voice, and Default. |
| User Attributes | Supplementary information for registration in the key-value pair form:<br>1. Click Add.<br>2. In the Key field, enter the JSON attribute as defined in API schema.<br>3. In the Value field, enter the name of the shared state attribute.<br>`For example, given a pair such as `emailAddress` : `"emailAddress"`, the node searches for the first occurrence of the key `emailAddress` in the shared state, and adds a pair `emailAddress` : `"valueInSharedState"` to the OneSpan API payload.`<br>To edit an entry, click the Pencil icon (✎).<br>To remove an entry, click the Delete icon (🗑). |

### Outputs

If an error occurs, an error message with the key `ostid_error_message` is output to the shared state.

### Outcomes

#### Success

If the user exists, then this node invokes the Update user API. If the user entry isn't found, this node invokes the Create a user API.

#### Error

An error message with the key `ostid_error_message` is output to the shared state.

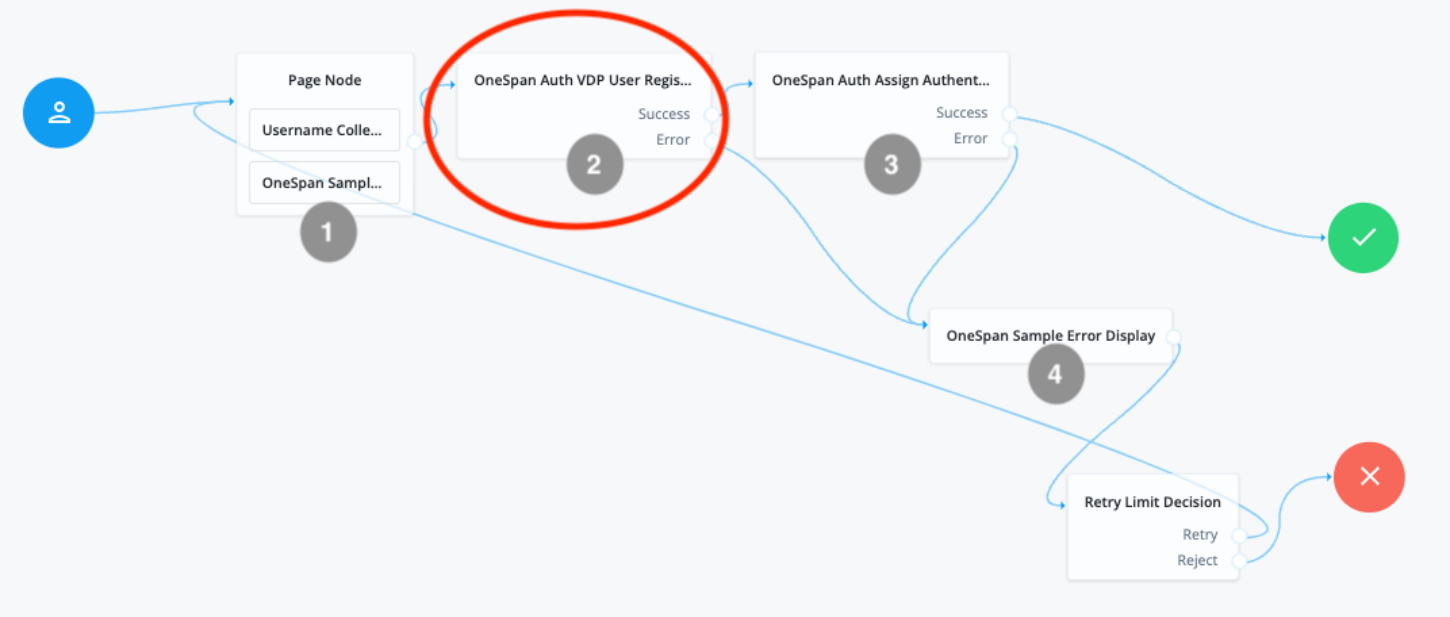
### Errors

This node logs an error message with the key `ostid_error_message` and the reason why the user registration for VDP wasn't successful.

## Troubleshooting

If this node logged an error, review the log messages for the transaction to find the reason for the exception.

### Examples




This example describes an authentication journey to register a user for VOTP authentication:

1. In the initial login page, the user enters their username and the required VDP delivery information, such as their virtual email or phone number.
2. The OneSpan VDP User Register node determines if there's an unassigned VIR10 authenticator available in the tenant. It also determines if the user isn't already assigned a VIR10 authenticator.
3. If the user hasn't been assigned a VIR10 authenticator, the OneSpan Auth Assign Authenticator node assigns a VIR10 authenticator to the user.
4. The OneSpan Sample Error Display node displays if the VIR10 authenticator assignment failed and lets you retry registration.

#### OneSpan Auth Assign Authenticator node

The OneSpan Auth Assign Authenticator node assigns VIR10 authenticator to the user when:

1. There's a VIR10 authenticator available in the tenant, and
2. The user isn't assigned a VIR10 authenticator.

The node fails when it can't find an unassigned VIR10 authenticator or encounters a node process exception.

The node first retrieves the user data. If the user is assigned an authenticator, the node loops through the authenticator list to determine if the user is assigned a VIR10 authenticator. If the user isn't assigned a VIR10 authenticator, it assigns the first unassigned VIR10 authenticator to the user.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
|---|---|---|
| Username | As specified in the property | Shared state |
| Delivery mode | As specified in the property | Shared state |

### Dependencies

This node uses input from OneSpan Auth VDP User Registration node and any other node that collects the VOTP delivery method.

### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| IAA Domain | The domain in which the user account resides. In a sandbox, the domain is the same as your tenant name. |

### Outputs

There are no outputs from this node.

### Outcomes

#### Success

User is successfully assigned to the VR10 authenticator.

#### Error

`OS_Auth_VDPUserRegisterNode` Exception, Date `ostid_error_message` OneSpan Auth Assign process.

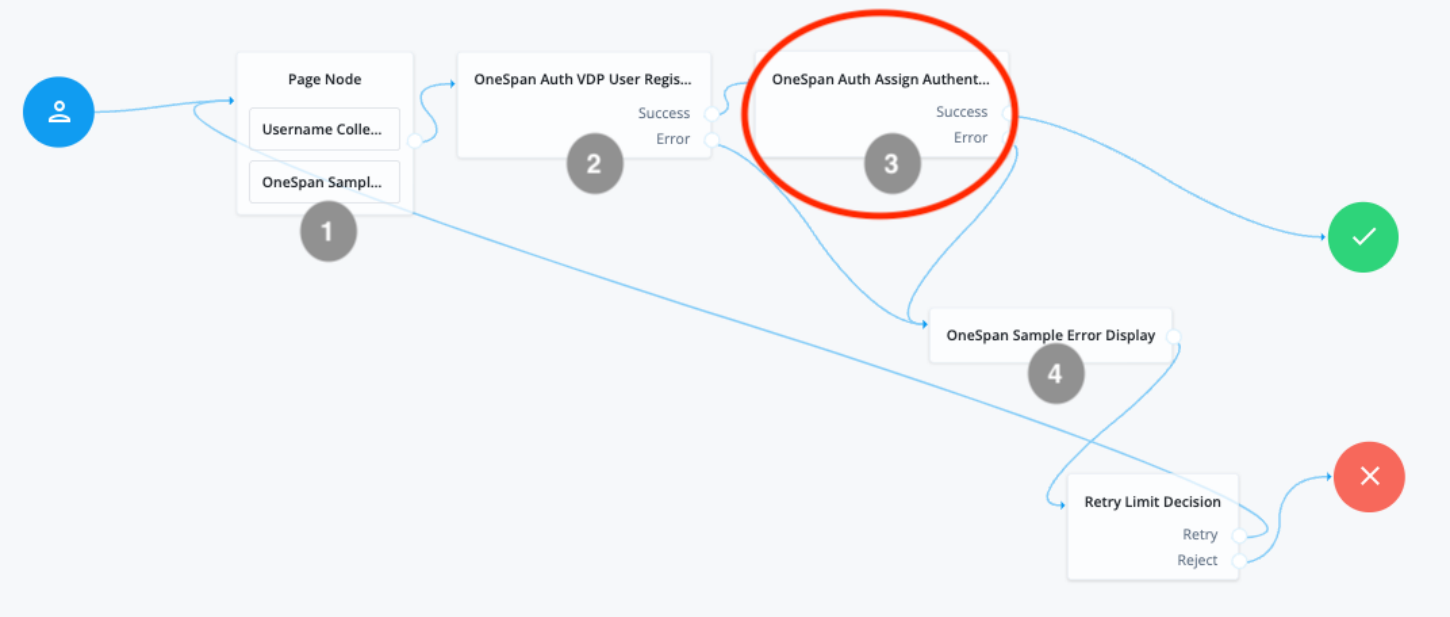
### Errors

This node logs an error message with the key `ostid_error_message` and the reason why assigning a VIR10 authenticator wasn't successful.

## Troubleshooting

If this node logged an error, review the log messages for the transaction to find the reason for the exception.

### Examples




This example describes an authentication journey to register a user for VOTP authentication, and assign a VIR10 authenticator:

1. In the initial login page, the user enters their username and the required VDP delivery information, such as their virtual email or phone number.
2. The OneSpan VDP User Register node determines if there's an unassigned VIR10 authenticator available in the tenant. It also determines if the user isn't already assigned a VIR10 authenticator.
3. If the user hasn't been assigned a VIR10 authenticator, the OneSpan Auth Assign Authenticator node assigns a VIR10 authenticator to the user.
4. The OneSpan Sample Error Display node displays the error if the VIR10 authenticator assignment failed. It lets you retry assigning VIR10 authenticator.

### OneSpan Auth Generate VOTP node

This node generates and delivers a virtual one-time passcode (VOTP) through the delivery method configured in the node if there's a VIR10 authenticator assigned to the user.

This node first retrieves user data, then loops through the authenticator list to verify if the user is assigned a VR10 authenticator. If the user has been assigned a VIR10 authenticator, the node triggers the Generate VOTP API.

#### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

#### Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
|---|---|---|
| Username | As specified in the property | Shared state |
| Password (optional) | As specified in user attributes | Transient state |
| User attributes | As specified in the property | Shared state |

#### Dependencies

To use this node, you should have already set up Advanced Identity Cloud integration with OneSpan, as mentioned in Set up, and enabled user authentication through an OneSpan VIR10 authenticator.

#### Configuration

The configurable properties for this node are:

| Property | Usage |
|---|---|
| IAA Domain | The domain in which the user account resides. In a sandbox, the domain is the same as your tenant name. |
| User Name In SharedState | The key parameter in shared state that represents the OneSpan IAA username. |

| Property | Usage |
| --- | --- |
| VDP Delivery Method | The mode in which VOTP is delivered. The available values are SMS, Email, Voice, and Default. |
| User Attributes | Supplementary information for registration in the key-value pair form:<br>1. Click Add.<br>2. In the Key field, enter the JSON attribute as defined in API schema.<br>3. In the Value field, enter the name of the shared state attribute.<br>`For example, given a pair such as `emailAddress` : `"emailAddress"`, the node searches for the first occurrence of the key `emailAddress` in the shared state, and adds a pair `emailAddress` : `"valueInSharedState"` to the OneSpan API payload.`<br>To edit an entry, click the Pencil icon (✎).<br>To remove an entry, click the Delete icon (🗑). |

### Outputs

This node adds the User Attributes values to the OneSpan API payload.

### Outcomes

#### Success

VOTP sent successfully in the delivery mode configured.

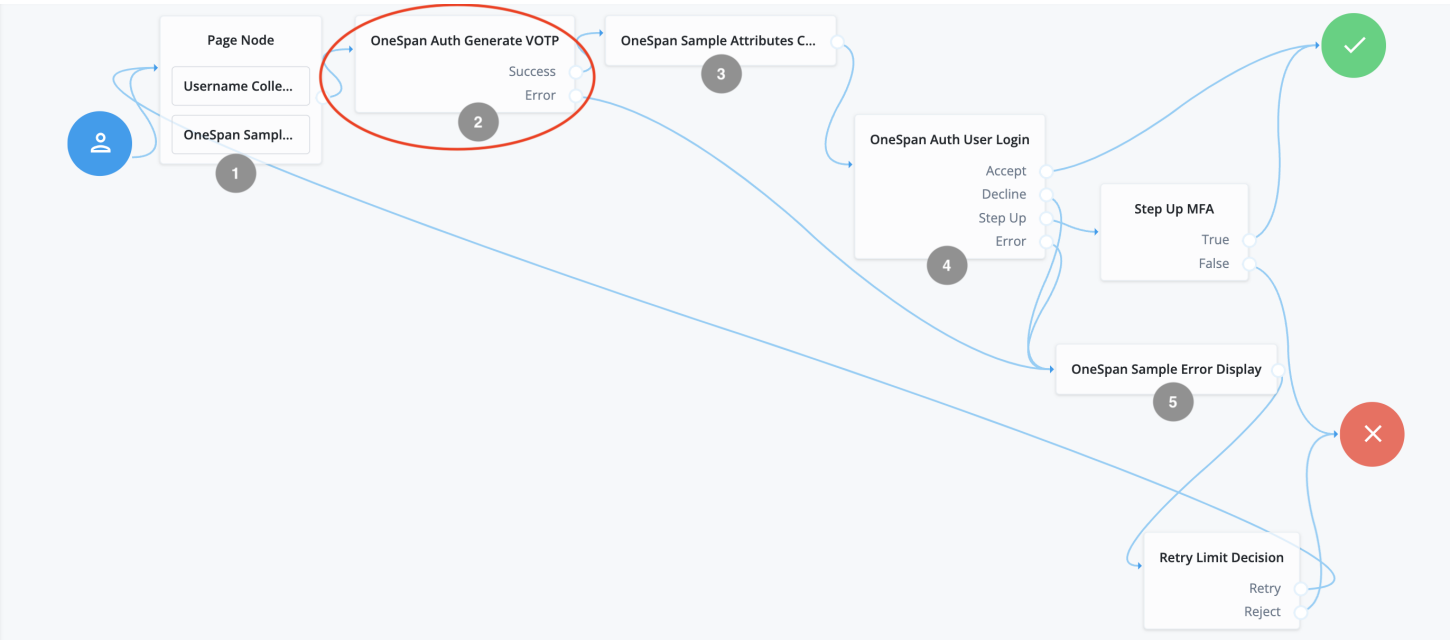
#### Error

- An error message with the key `ostid_error_message` is output to the shared state.

## Troubleshooting

If this node logged an error, review the log messages for the transaction to find the reason for the exception.

### Example




This example describes an authentication journey which generates VOTP to authenticate a user:

1. In the initial login page, the user enters their username and the required VDP delivery information, such as their virtual email or phone number.
2. The OneSpan Auth Generate VOTP node then generates and delivers the VOTP.
3. The OneSpan Sample Attributes Collect node accepts the VOTP for validation.
4. The OneSpan Auth User Login node invokes the User Login API, to validate the user's login. Depending upon the outcome, it appropriately directs the further authentication flow.
5. If an error is encountered, the OneSpan Sample Error Display node displays the error.

## OneSpan Get User Authenticator node

The OneSpan Get User Authenticator node retrieves the authenticators assigned to a user and helps in appropriately enabling the user's authentication and security levels.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
| --- | --- | --- |
| Username | As specified in the property | Shared state |

### Dependencies

To use this node, you must integrate OneSpan in your PingOne Advanced Identity Cloud tenant environment and enable authentication through a OneSpan VIR10 authenticator.

### Configuration

The configurable properties for this node are:

| Property | Usage |
| --- | --- |
| IAA Domain | The domain in which the user account resides. In a sandbox, the domain is the same as your tenant name. For example, master. |
| User Name In SharedState | The key parameter in shared state that represents the OneSpan IAA username. For example, username. |

### Outputs

There are no outputs from this node.

### Outcomes

The outcome specifies the authenticator configured:

**TYP**

A `TYP` authenticator is configured.

**VIR10**

A `VIR10` authenticator is configured.

**Both**

Both `TYP` and `VIR10` are configured.

**None**

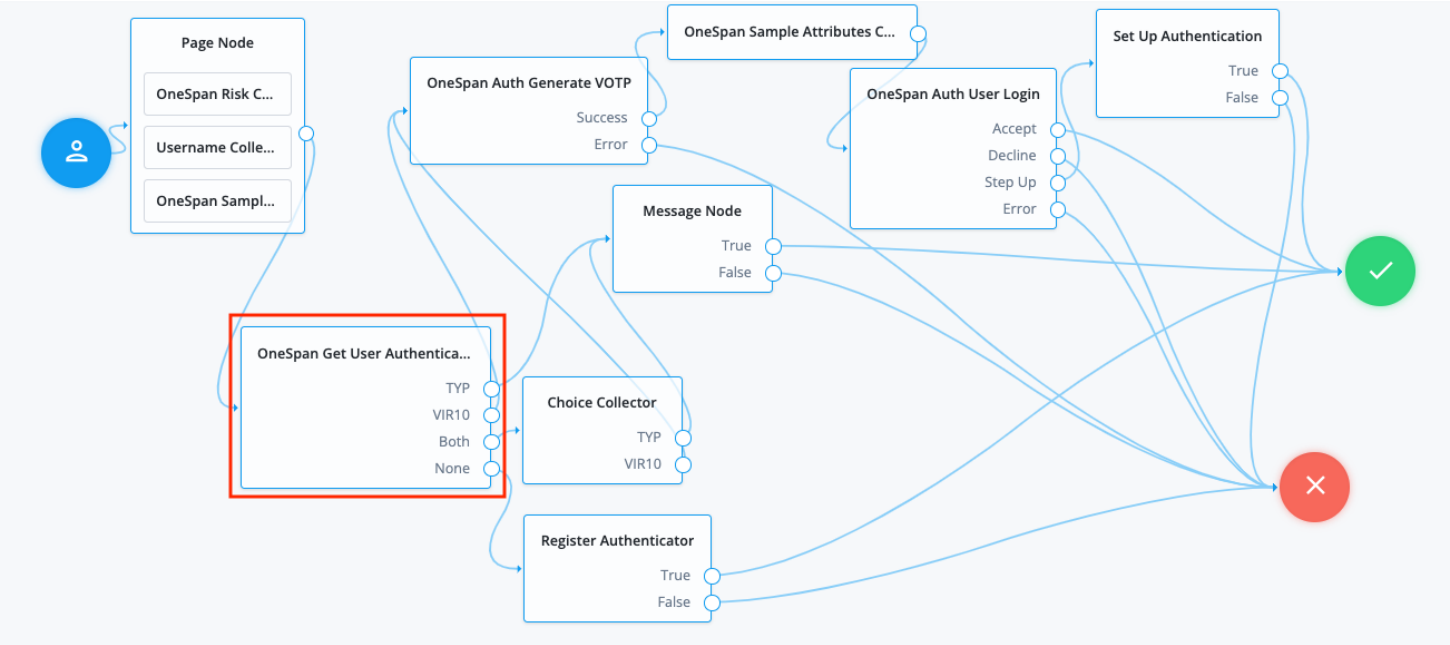
No authenticator is configured.

### Troubleshooting

If this node logs an error, review the log messages for the transaction to find the reason for the exception and address the issue appropriately.

### Examples

This example journey shows user registration and login. It highlights the use of the OneSpan Get User Authenticator node:




The OneSpan Get User Authenticator node gets the authenticators assigned to a user and appropriately directs the flow in the journey:

| Outcome | Description |
|---|---|
| `TYP` | The flow is directed to the Message node. |
| `VR10` | The flow is directed to the OneSpan Auth Generate VOTP node. |
| `Both` | The flow is directed to the Choice Collector node. The Choice Collector node forwards the flow to the OneSpan Auth Generate VOTP node or the Message node based on the choice made. |
| `None` | The flow is directed to Register Authenticator node so the user can be assigned an authenticator. |

### OneSpan Identity Verification node

The OneSpan Identity Verification node performs document verification using OneSpan. The user is redirected to OneSpan to verify their identity and returned to the authentication journey.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node requires the following inbound data:

| Description | Attribute name | Source |
|---|---|---|
| First Name | objectAttributes.givenName | Shared state |
| Last Name | objectAttributes.sn | Shared state |
| Phone Number | objectAttributes.telephoneNumber | Shared state |

### Dependencies

To use this node, you must integrate OneSpan in your PingOne Advanced Identity Cloud tenant environment and enable OneSpan identity verification.

### Configuration

| Property | Usage |
|---|---|
| OneSpan Auth Token | The authentication token used in OneSpan identity verification. |
| OneSpan tenant ID | The OneSpan identity verification tenant ID. |
| OneSpan IDV URL | The OneSpan URL to which this node will redirect to get identity verification. |
| OneSpan IDV workflow ID | The workflow ID for the OneSpan identity verification tenant. |
| BrandID | The brand ID for the OneSpan identity verification tenant. |
| Language | The default language for the OneSpan identity verification tenant. |
| failUrl | The URL to be returned to after OneSpan identity verification fails. |
| passUrl | The URL to be returned to after OneSpan successfully verifies the identity. |

| Property | Usage |
| --- | --- |
| defaultUrl | The default URL to be returned to after OneSpan identity verification. |
| errorUrl | The URL to be returned to if errors occur during OneSpan identity verification. |
| sessionTimeoutUrl | The URL to be returned to after the OneSpan identity verification session times out. |
| role | The role to be sent for OneSpan identity verification. |

### Outputs

| Output variable | Description |
| --- | --- |
| OneSpan IDV Results | Results of OneSpan identity verification. |

### Outcomes

`Pass`

Identity is successfully verified.

`Fail`

Identity verification failed.

`Error`

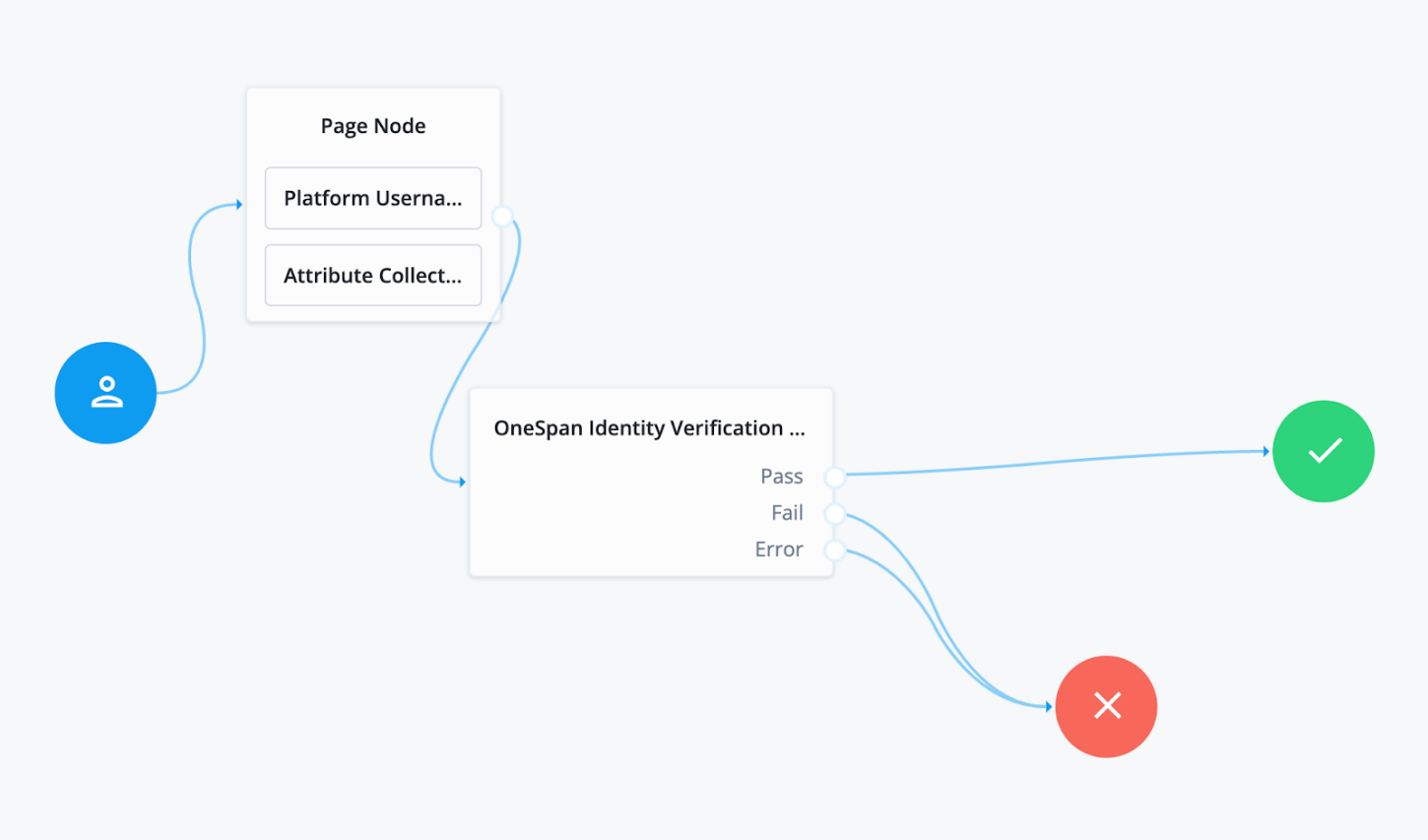
An error occurred while verifying identity. An error message is output to the shared state.

### Troubleshooting

If this node logs an error, review the log messages for the transaction to find the reason for the exception and address the issue appropriately.

### Examples

This example journey highlights the use of the OneSpan Identity Verification node:



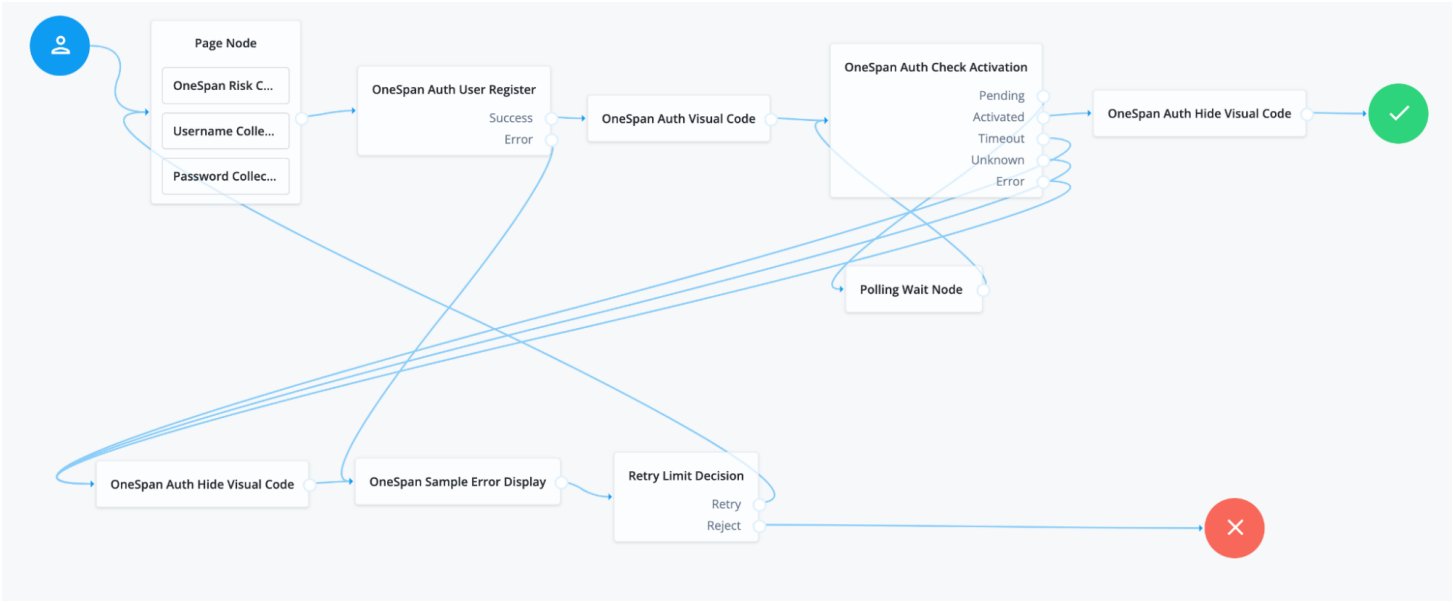

## OneSpan Sample journeys

PingOne Advanced Identity Cloud provides sample journeys that you can download from here to help you understand and address the most common OneSpan use cases.

> **Note**
>
> These samples are provided only for development and testing, don't use them in production environments.

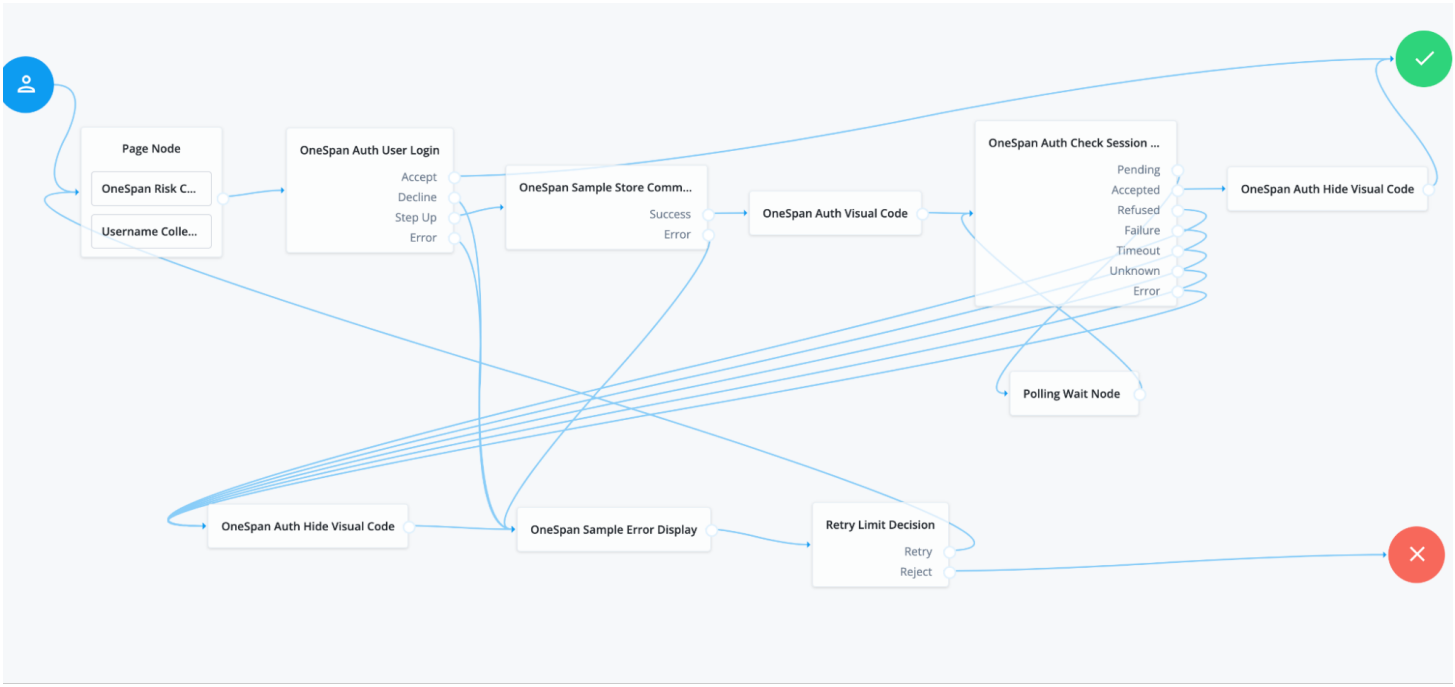
### OneSpan IAA user registration

The OneSpan IAA user registration journey interacts with the IAA service to create and activate a Digipass account. You can download the JSON file for this sample journey from here.




### OneSpan IAA user login

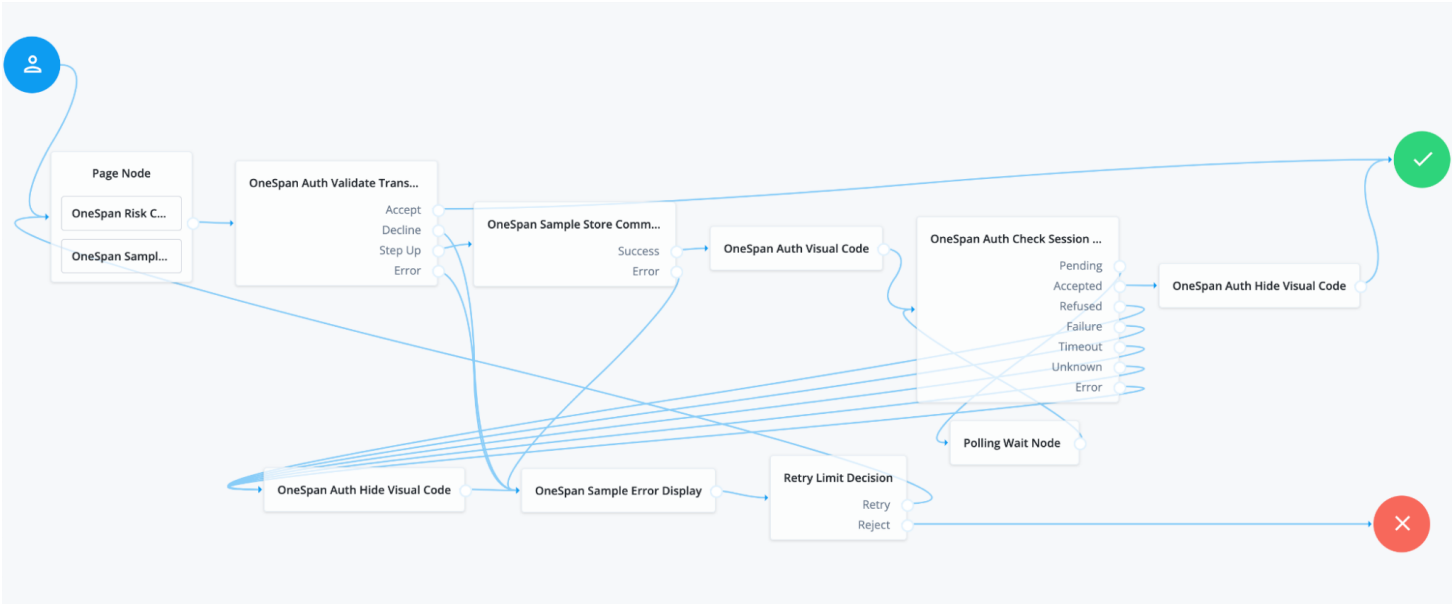
The OneSpan IAA user login journey checks the browsing context and analyzes the risk of the end-user login. You can download the JSON file for this sample journey from here.




### OneSpan IAA validate transaction event

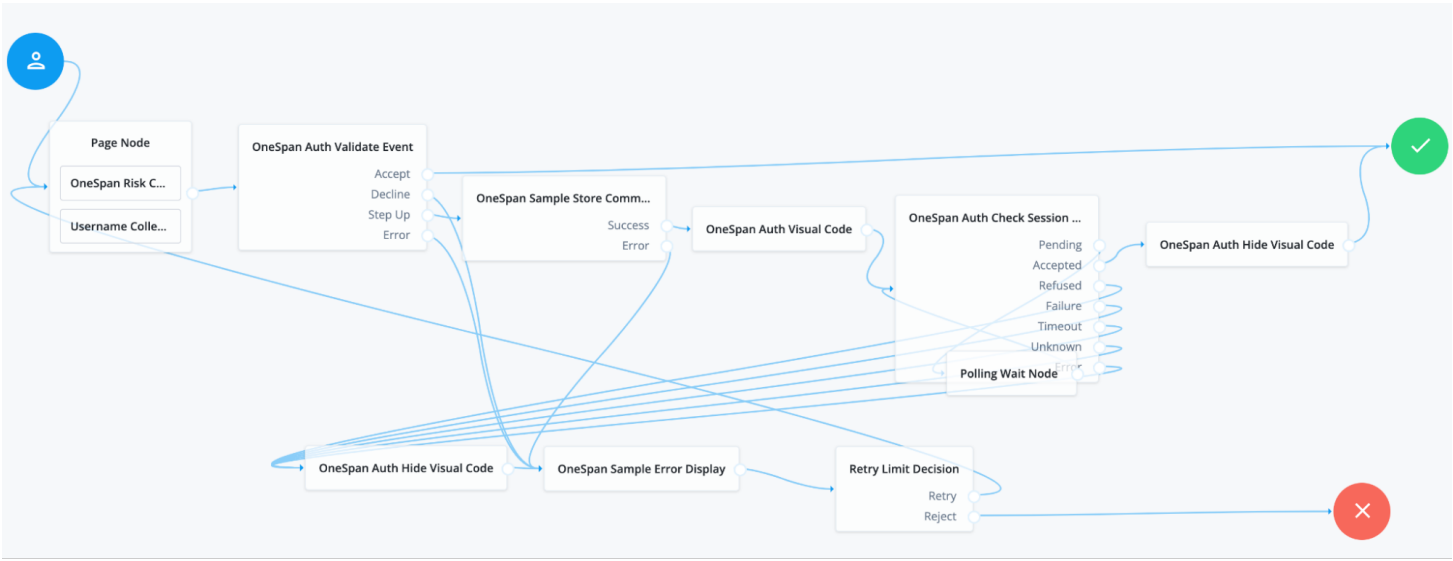
The OneSpan IAA validate transaction event journey evaluates the risk before an end-user tried to send a transaction, based on the transaction details and the browser or mobile's context.

Download the JSON file for this sample journey.



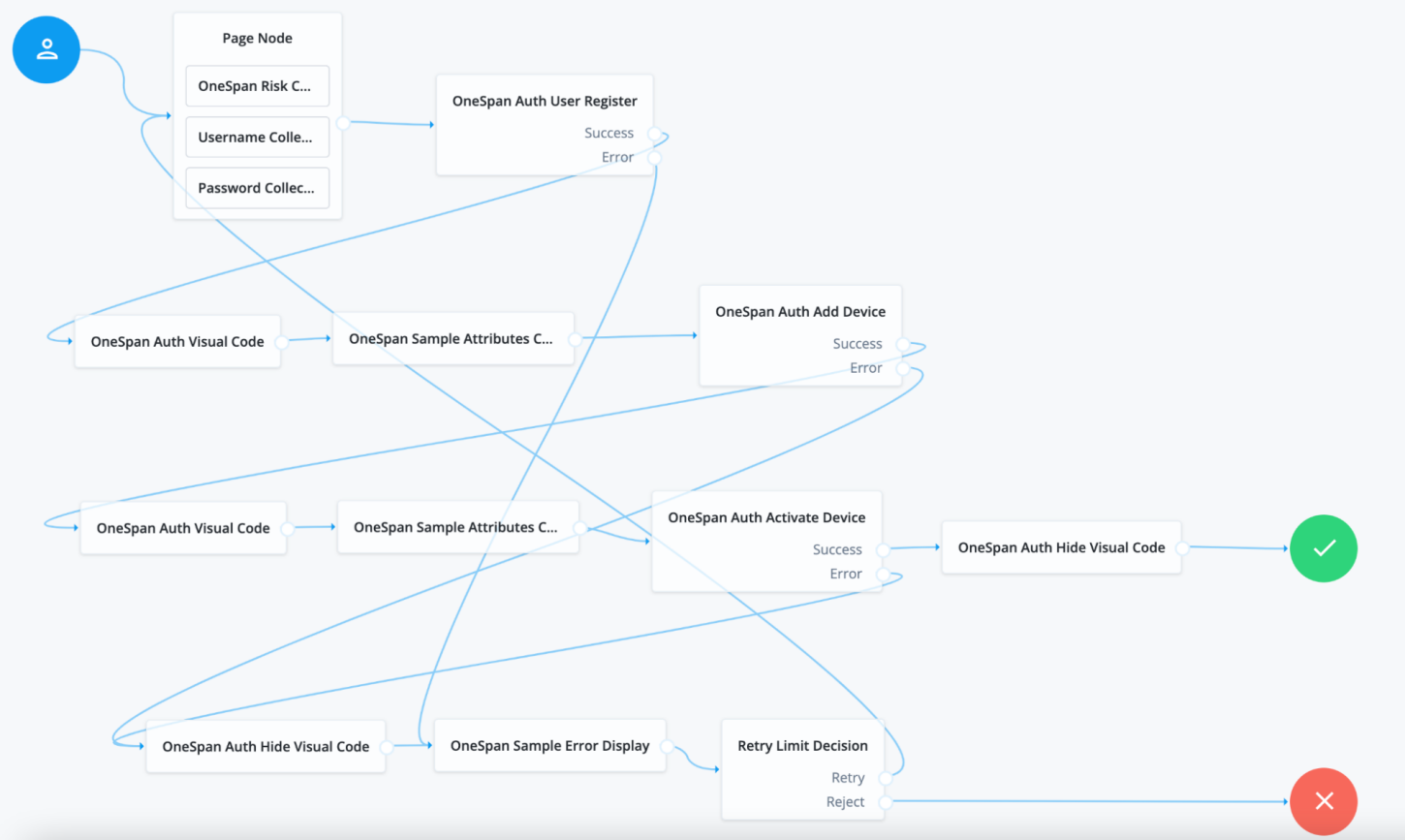

### OneSpan IAA validate non-monetary events

The OneSpan IAA validate transaction events journey provides a generic validation for non-monetary events. You can download the JSON file for this sample journey from here.



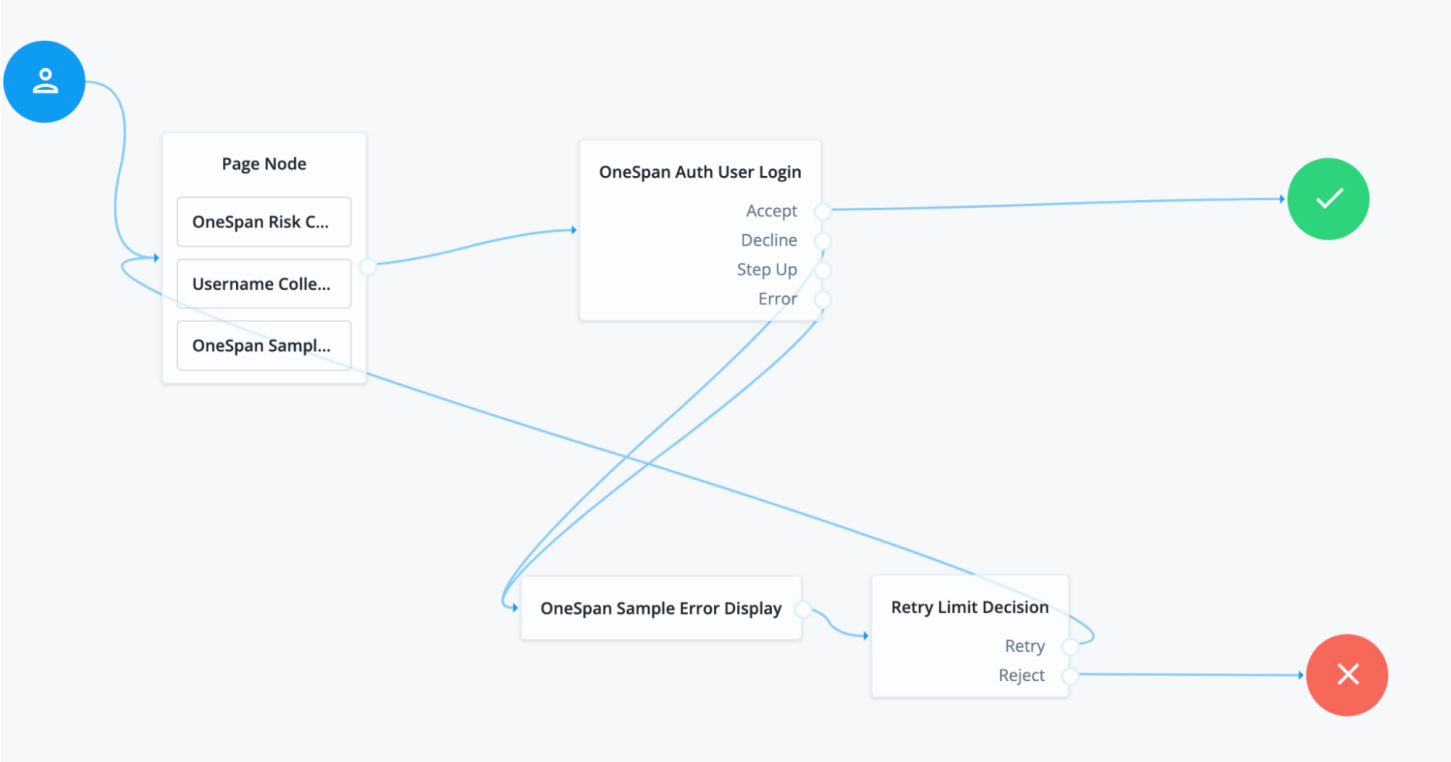

### OneSpan CA offline user registration and Digipass activation

The OneSpan CA offline user registration and Digipass activation journey interacts with the OneSPAN OCA service which creates a Digipass user account and awaits a Digipass Authenticator to activate the license. You can download the JSON file for this sample journey from here.



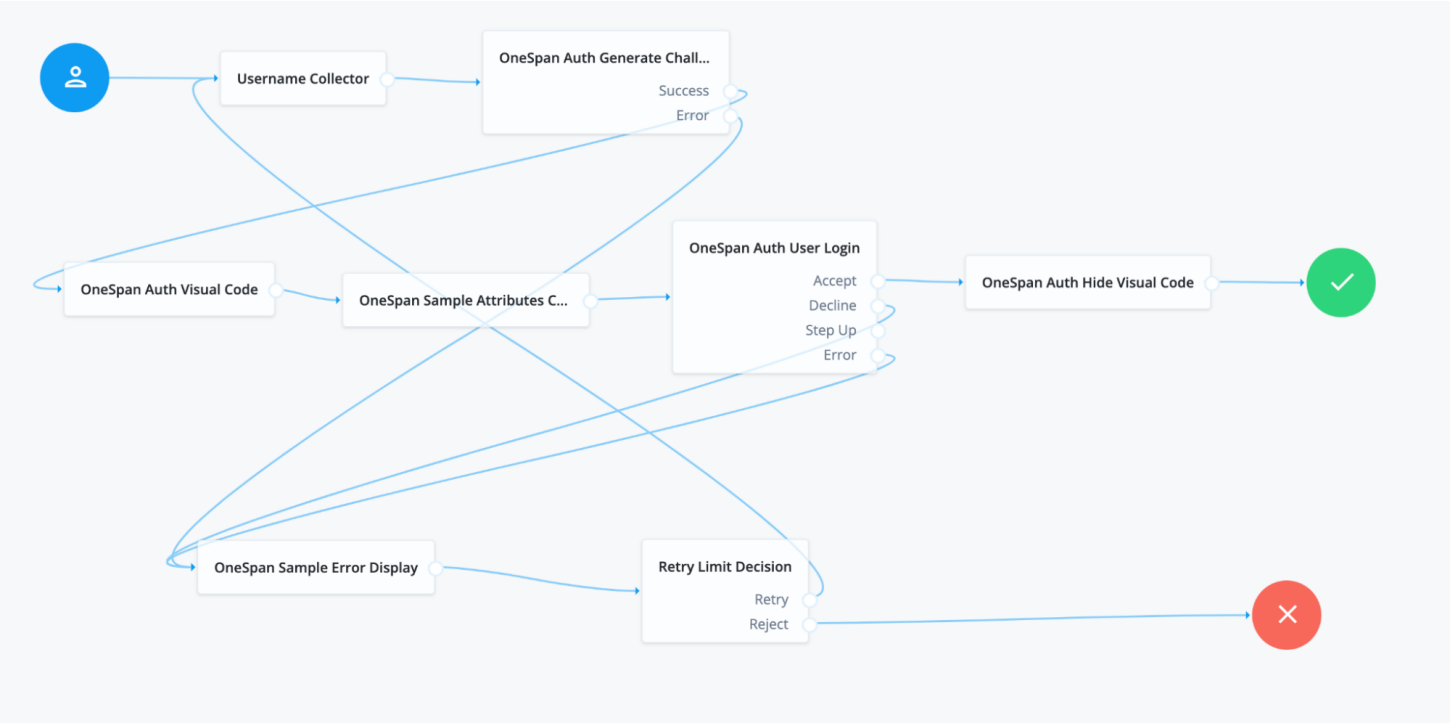

**OneSpan CA user login with one-time passcode**

**The OneSpan CA user login with one-time passcode journey validates the one-time passcode and returns the validation result. If the authentication has succeeded, the browser will be redirected to the success URL. You can download the JSON file for this sample journey from here.**



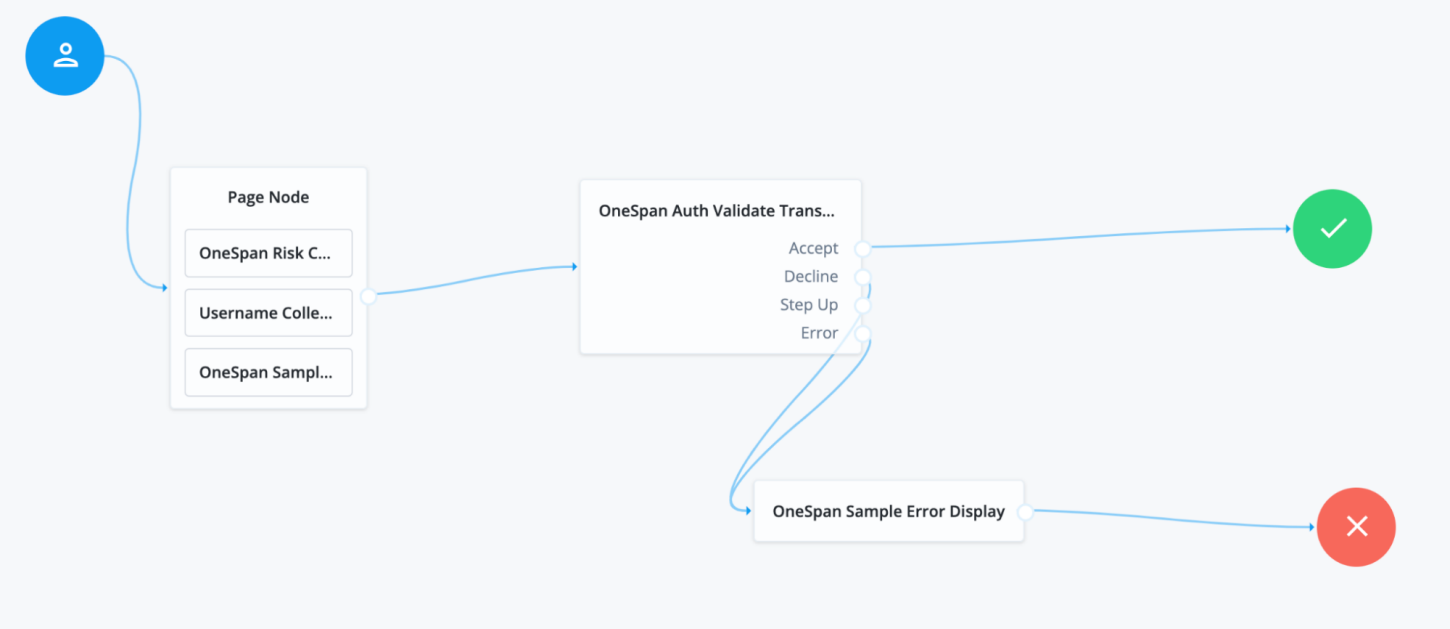

**OneSpan CA user login with challenge / response (CR)**

**The OneSpan CA user login with challenge / response (CR) journey validates the OTP and returns the validation result using the Challenge/Response mechanism. If the authentication has succeeded, the browser will be redirected to the success URL. You can download the JSON file for this sample journey from here.**



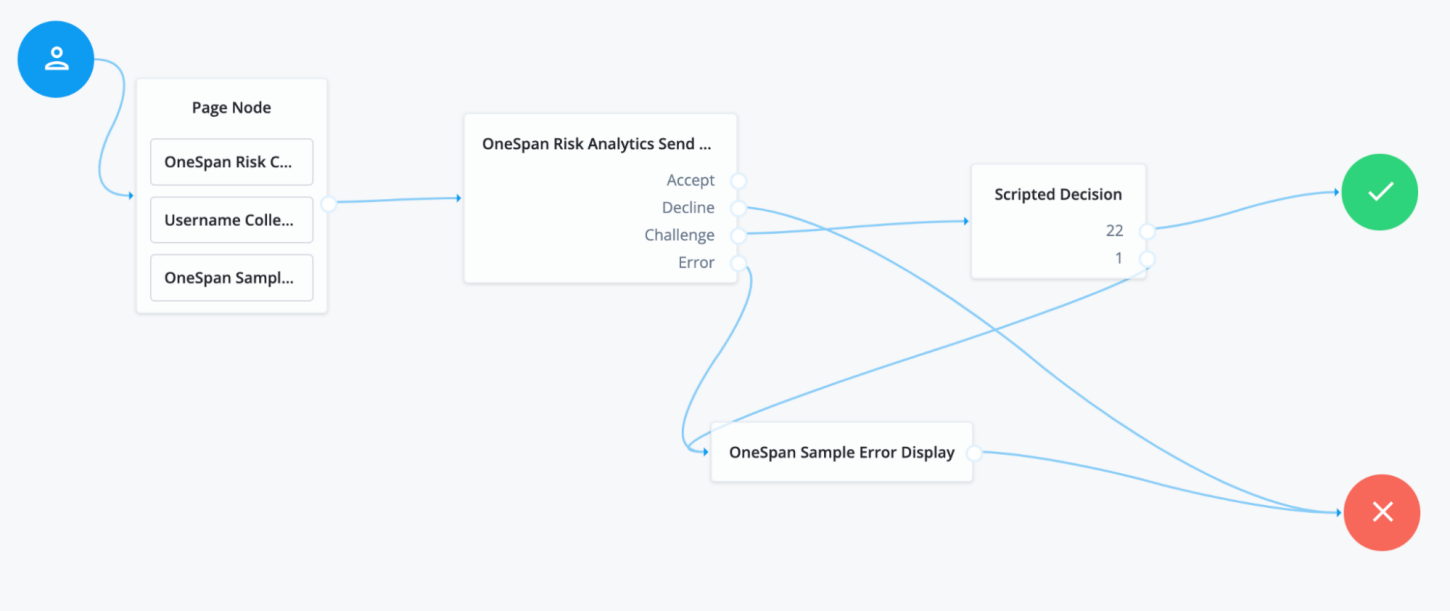

**OneSpan CA offline transaction data signing**

The OneSpan CA offline transaction data signing journey validates the signature and returns the validation result. If the authentication has succeeded, the browser will be redirected to the success URL. You can download the JSON file for this sample journey from here.




### OneSpan RA insert transaction

The OneSpan RA insert transaction journey leverages OneSpan Risk Analytics and gets a response code. You can download the JSON file for this sample journey from here.



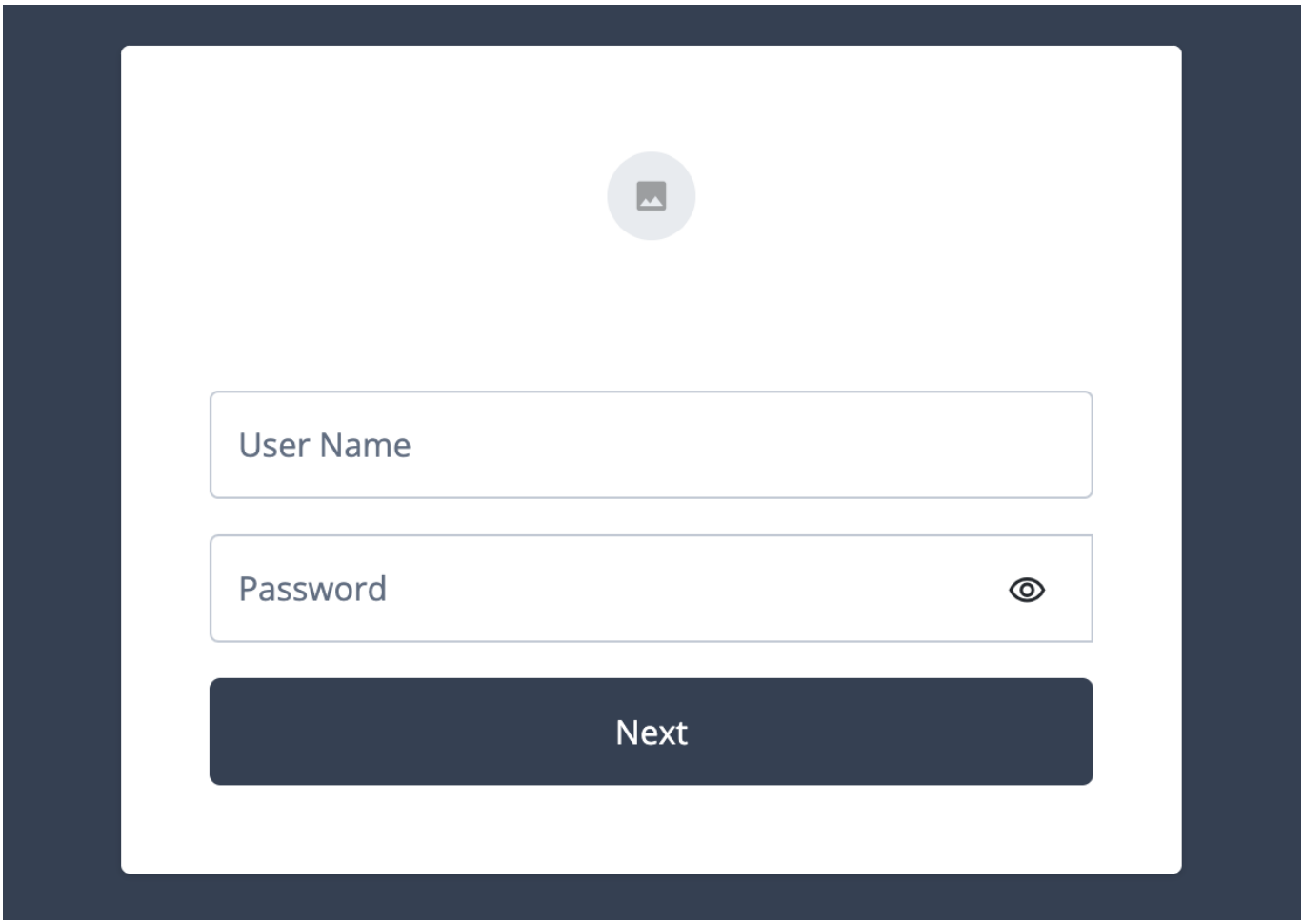

## Authentication with OneSpan

This section describes how to use the OneSpan IAA user register authentication journey and how the authentication nodes work.

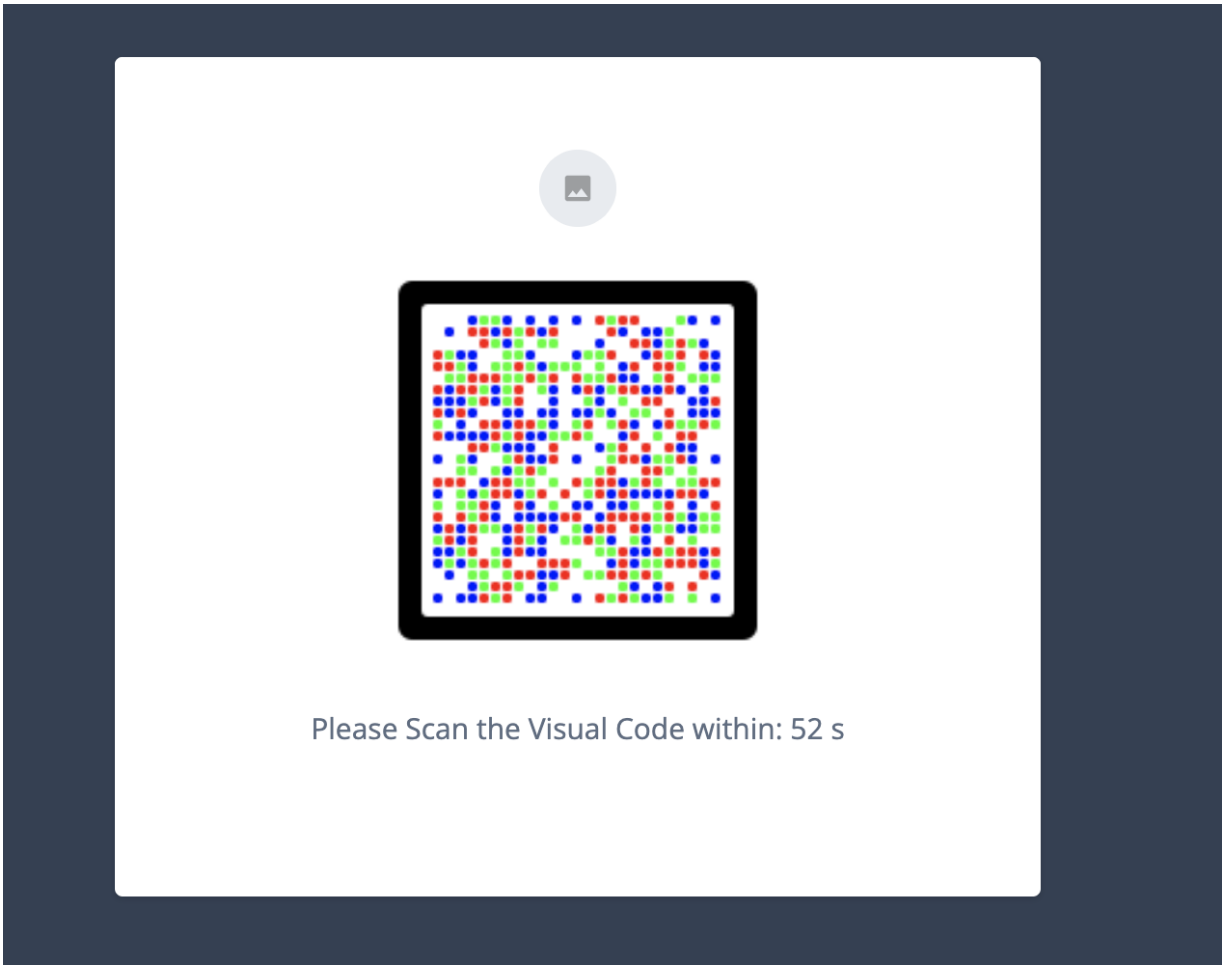
To start the authentication process:

1. Access https://tenant-env-fqdn/am/XUI/?realm=alpha&authIndexType=service&authIndexValue=OneSpan-XUI-Adapative-Authentication-User-Register-Sample-Tree in your browser.
2. Enter the username and password.

> **Note**
>
> Password should be at least eight characters long and include at least one lowercase, one uppercase, and one number. Also it and should not include part of the username for any three characters.

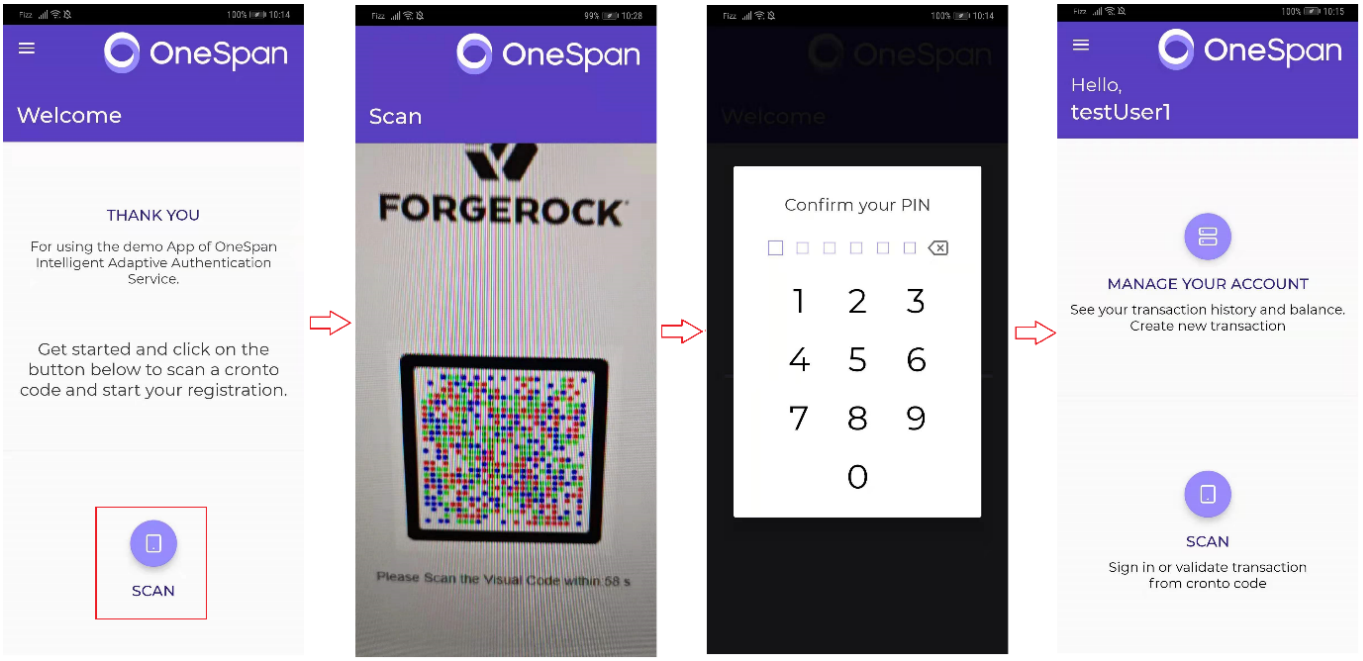



3. Once the Risk Analytics has accepted the user registration, the IAA service creates a Digipass user account and waits for a trusted device to activate the license with an activation token, which is rendered as a visual code.

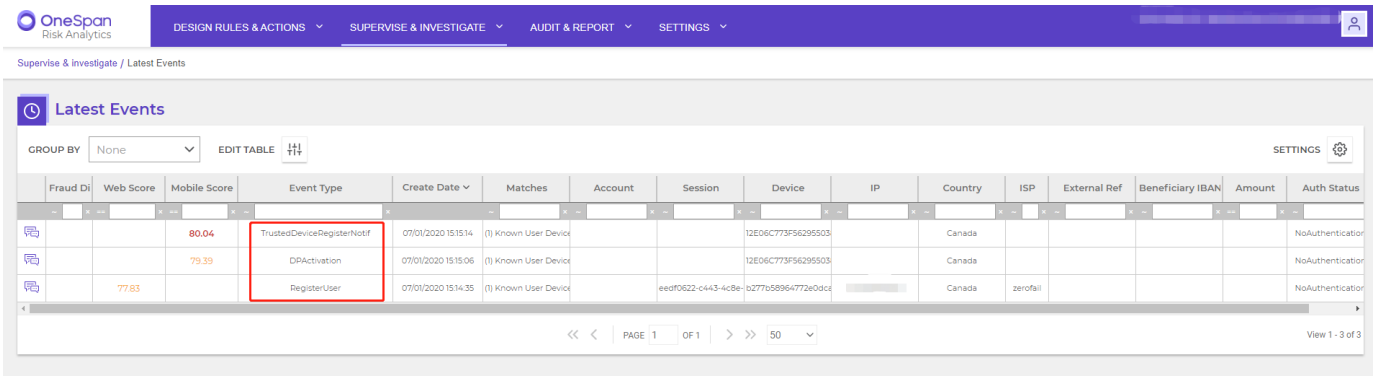



4. Launch the AAS Demo App, click SCAN and use the camera to scan the above visual code.
5. Once the code is detected, the app prompts you to enter a 6-digit security PIN and confirm the same.

   After completing the registration process, the demo app displays the user page and the browser redirects to the success URL.




6. To verify that the user registration process has been logged by the system, log in to your IRM system and navigate to SUPERVISE & INVESTIGATE > Latest Events.




# Onfido Check node

Integrates an Onfido check for identity verification, matching a user with their official identification documents.

## Outcomes

The node returns the status of the Onfido check:

`Clear`

Successful verification; all underlying verifications pass.

`Consider`

The check returned information to evaluate.

**Deny**

Failed verification.

**Pending**

The check is still in progress.

**Error**

Other failure.

### Properties

| Property | Usage |
| --- | --- |
| Onfido Live Token | The Live API token from the Token pane of the Onfido dashboard. |
| Onfido API URL | Regional base URL for Onfido API requests.<br>Default: `https://api.onfido.com/v3/` |
| Onfido Check ID Attribute | The user profile attribute that stores the Onfido Check ID. The node verifies that this matches the ID in the Identity Verification Report from Onfido after verification.<br>Default: `description` |

### Examples

For examples in context, refer to the Onfido examples.

## Onfido Registration node

Register a user with Onfido using identity verification, optionally collecting biometrics.

This node uses the Onfido Autofill endpoint to retrieve user attributes for provisioning from the user's identity document.

It returns the outcome when the registration completes.

### Outcomes

**True**

Success

**Error**

Failure

## Properties

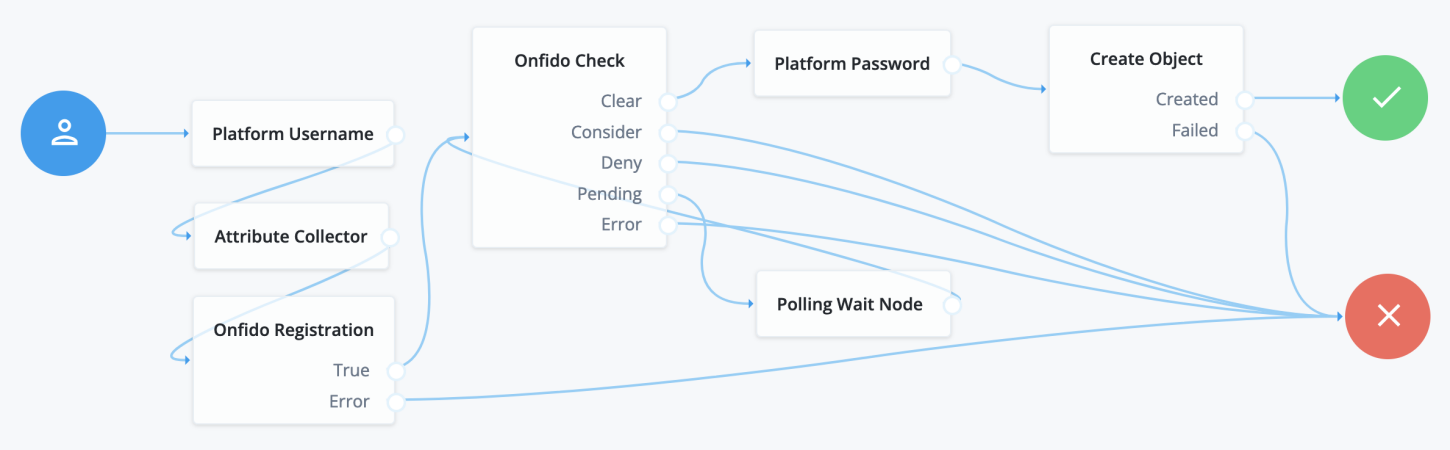
| Property | Usage |
| --- | --- |
| Onfido Live Token | The Live API token from the Token pane of the Onfido dashboard. |
| Just-in-Time Provisioning | If enabled, create a shared JSON object for the Create Object node.<br>Enable this when using registration as a service (RaaS).<br>Leave this disabled when using this node to provide a level of assurance.<br>Default: Disabled |
| Referrer for JWT | Referrer for JWT SDK token used to enforce that the JWT SDK token is only used by the correct caller.<br>Default: `*://*/*` |
| Onfido API URL | Regional base URL for Onfido API requests.<br>Default: `https://api.onfido.com/v3/` |
| Biometric Check | Determines whether and how to check the user against their ID photo.<br>***None***<br>Neither (default).<br>***Selfie***<br>Check a selfie of the user.<br>***Live***<br>Check live video of the user. |
| Onfido Applicant ID Attribute | The user profile attribute that stores the Onfido Applicant ID.<br>Default: `title` |

| Property | Usage |
| --- | --- |
| Attribute Map | This maps the attributes in the response from the Onfido Autofill endpoint to user profile attributes.<br>The keys are the user profile attributes. The values are the Onfido attributes.<br>Default mapping:<br>**city**<br>`address_line_3`<br>**sn**<br>`last_name`<br>**postalCode**<br>`address_line_4`<br>**postalAddress**<br>`address_line_1`<br>**givenName**<br>`first_name`<br>**stateProvince**<br>`address_line_5`<br>**cn**<br>`first_name` |
| Modal Title | The title to display in the modal dialog presented to the user.<br>Default: `Identity Verification` |
| Modal Subtitle | The subtitle to display in the modal dialog presented to the user.<br>Default: `Thank you for using Onfido for Identity Verification` |
| Onfido SDK URL | The URL for the Onfido JavaScript SDK.<br>Default: `https://assets.onfido.com/web-sdk-releases/6.7.1/onfido.min.js` |
| Onfido CSS URL | The URL for the Onfido JavaScript SDK CSS.<br>Default: `https://assets.onfido.com/web-sdk-releases/6.7.1/style.css` |
| Onfido Check ID Attribute | The user profile attribute that stores the Onfido Check ID.<br>The node verifies the value matches the ID in the Identity Verification Report from Onfido after verification.<br>Default: `description` |

## Examples

### Registration as a service

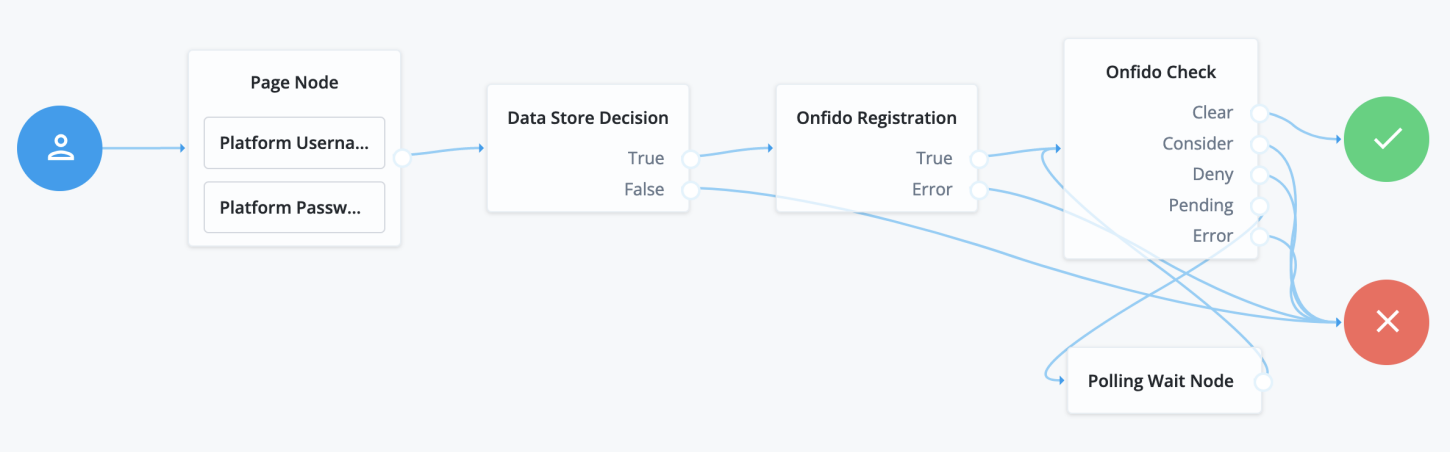
The following example checks the user's official identity documentation with Onfido and uses the response to provision their account.




**Level of assurance**

The following example redirects the authentication flow through Onfido identity verification.

Use this flow when a user has an account and you require a higher level of assurance to confirm the user's identity. For example, the user who signed up to use a bank's digital identity to review and track offers now wants to open a checking or savings account.




## RSA SecurID

The RSA SecurID node lets you use the RSA Cloud Authentication Service (RSA ID Plus) or RSA Authentication Manager from within an authentication journey on your PingOne Advanced Identity Cloud environment.

This node lets users authenticate using their registered RSA authenticators, including:

- SecurID OTP (hardware and software tokens), including new PIN mode
- SecurID Authenticate OTP
- Emergency Access Code
- Approve (Push Notifications)
- Device Biometrics

- QR Code
- SMS OTP
- Voice OTP

## Quick start with sample journeys

PingOne Advanced Identity Cloud provides sample journeys to help you understand the most common RSA SecurID use cases. To use the samples, download the JSON files for sample journeys and import the downloaded sample journeys into your Advanced Identity Cloud environment.

### Dependencies

To use this node, you must:

- Enroll RSA authenticators. Refer to RSA SecurID setup for more information.
- Ensure the username on the shared node state matches one of the following:
  - The username, alternate username, or email address of the user in the RSA Cloud Authentication.
  - The username in RSA Authentication Manager.

## RSA SecurID setup

The RSA SecurID node in Advanced Identity Cloud can be used with the RSA Cloud Authentication Service or RSA Authentication Manager. Depending on which integration you choose, the RSA setup differs slightly.

### Setup with RSA Cloud Authentication Service

1. Configure the following using the RSA Cloud Administration Console:
   1. Assurance Levels: Refer to the Configure Assurance Levels page in the RSA documentation.
   2. Policies: Refer to the Manage Access Policies page in the RSA documentation.

      Note the policy name you will use when configuring the RSA SecurID node in your Advanced Identity Cloud journey.
   3. Authentication API Keys: Refer to the Manage the SecurID API Keys in the RSA documentation.

      Note the SecurID Authentication API REST URL and Authentication API key you will use when configuring the RSA SecurID node in your Advanced Identity Cloud journey.
   4. End users enroll their RSA authenticators: Refer to the Manage My Page in the RSA documentation.

### Setup with RSA Authentication Manager

1. Using the RSA Authentication Manager Security console, configure the following:
   1. Go to Access > Authentication Agents > Add New and add a new access agent.

Note the Authentication Agent name. You will need this when configuring the RSA SecurID node in your Advanced Identity Cloud journey. For additional information, refer to the Add an Authentication Agent page in RSA documentation.

2. Go to Setup > System Settings > RSA SecurID Authentication API, and note the access key. You will use this key in the SecurID node configuration. For additional information, refer to the Configure the RSA SecurID Authentication API for Authentication Agents page in the RSA documentation.

2. You'll need the REST API URL for your RSA environment. Get the REST API URL from your RSA Authentication Manager administrator.

### RSA SecurID node implementation

Refer to the implementation details of RSA SecurID node here.

### RSA SecurID node

The RSA SecurID node lets users authenticate using their registered RSA authenticators.

#### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | No |
| Ping Identity Platform (self-managed) | No |

#### Inputs

The `username` attribute must exist in the shared node state as an input to the node.

#### Configuration

| Property | Usage |
|---|---|
| Base URL | The RSA endpoint.<br>• For connections to the RSA Cloud Authentication Service, such as https://companyname.auth.securid.com:443/mfa/v1_1<br>• For connections through RSA Authentication Manager, such as https://RSA.AM.server:5555/mfa/v1_1 |

| Property | Usage |
| --- | --- |
| Client ID | The name used by this node as the client ID for connecting to the RSA endpoint. This can contain alphanumeric English characters only.<br>• For connections to the RSA Cloud Authentication Service, this can be any string. End users will see this value as part of push notification messages, and administrators will see this as the application name in the User Event Monitor of the RSA Cloud Administration Console.<br>Example: Example Login Journey.<br>• For connections to RSA Authentication Manager, this value must match an Authentication Agent name configured in the RSA Authentication Manager Security Console.<br>Example: MyAgentName |
| Assurance Policy ID | The name of the RSA Cloud Authentication Service policy to use. This name can contain alphanumeric English characters only. This name is required for connections to RSA Authentication Manager only when RSA AM acts as a proxy for connections to the cloud.<br>Example: All Users Medium Assurance Level. |
| Client Key | The API key for connecting to the RSA endpoint.<br>• For the RSA Cloud Authentication Service, this value can be generated or obtained using the RSA Cloud Administration Console, My Account > Company Settings > Authentication API Keys.<br>• For RSA Authentication Manager, this value can be found in the RSA Security Console, Setup > System Settings > RSA SecurID Authentication API (Access Key). |
| Verify SSL | A boolean to verify the SSL connection. It is enabled by default. If disabled, the node ignores SSL/TLS errors, including hostname mismatch and certificates signed by an unknown Certificate Authority, such as self-signed certificates. |
| Prompt for MFA Choice | The string to display to end users on the MFA selection input page.<br>Example: Select your preferred Authentication Method. |
| Waiting Message | The string to display to end users when a push notification has been sent to the user's registered device.<br>Example: Please check your registered mobile device for an authentication prompt. |

### Outputs

None

### Outcomes

#### Success

The user completed the RSA authentication process and does not require any further steps according to the RSA Assurance Policy this node references.

#### Failure

The user has failed the RSA MFA authentication.

`Not Enrolled`

The user is not enrolled in any RSA authentication methods required by the specified policy.

`Cancel`

The user pressed the cancel button.

`Error`

An error occurred. Refer to Troubleshooting.

### Troubleshooting

Review the log messages to find the reason for the error and address the issue appropriately.

### Limitations and known issues

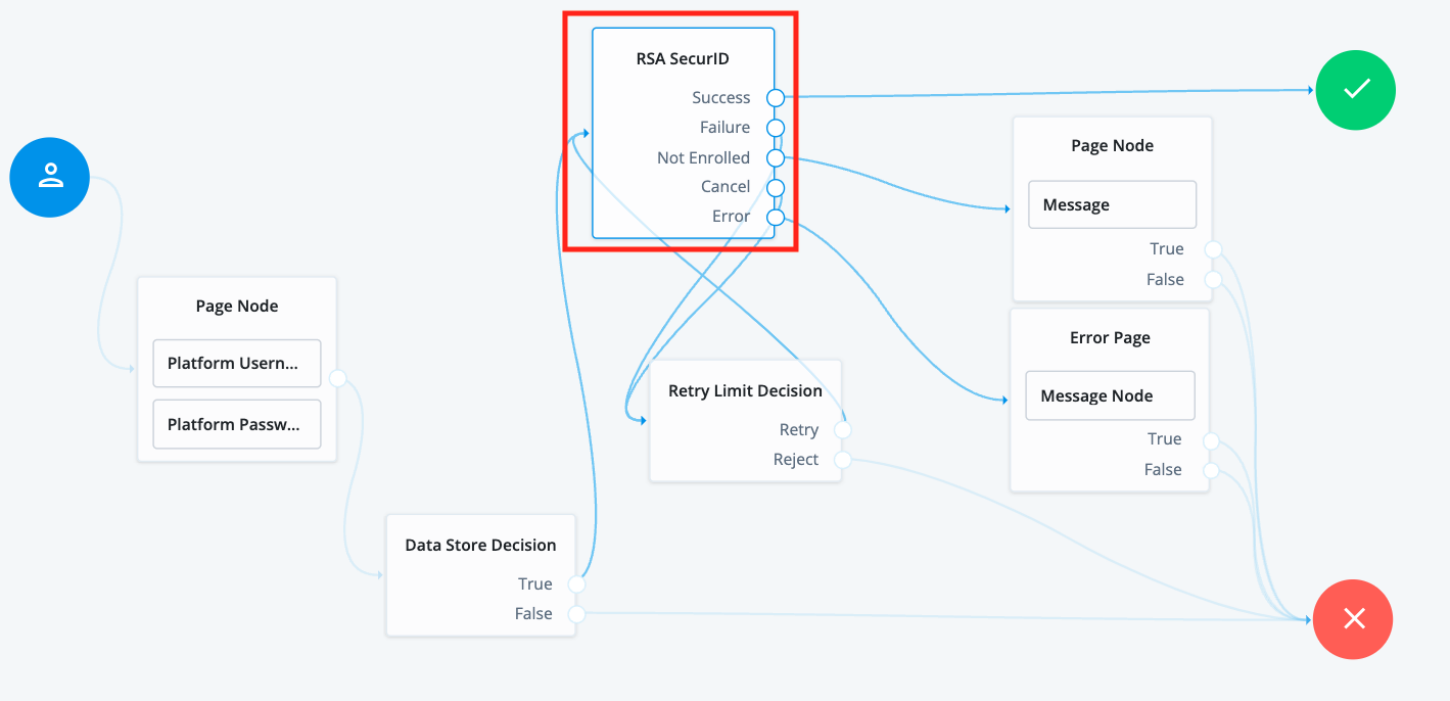
- The RSA SecurID node supports most RSA authentication methods; however, the following RSA authentication methods are not supported:
  - FIDO: Customers can consider using the WebAuthn nodes as an alternative.
  - LDAP Directory Password or RSA Cloud Authentication Service password: Customers can consider using the Platform Password node and Data Store Decision node, or Pass-through Authentication node as alternatives.
- SecurID tokens are not supported in Next Tokencode mode.
- RSA API returns multiple authentication options when only the New PIN mode option should be returned. This situation occurs when all these conditions are met:
  - The RSA SecurID node connects to the RSA Cloud Authentication Service directly or through RSA Authentication Manager as a proxy.
  - The configured policy & assurance level & user-enrolled authenticators include SecurID and other authentication methods.
  - The user selects the SecurID option, and their SecurID token is in new PIN mode.

    Seeing multiple options can be confusing when *only* the New PIN option is expected. The RSA team is aware of this RSA API behavior and is evaluating ways to correct the behavior to ensure that the REST API returns only the SecurID new PIN and passcode prompts.
- The RSA SecurID node only supports English characters for:
  - Client ID
  - Assurance Policy ID

### Examples

PingOne Advanced Identity Cloud provides sample journeys. You can download the JSON file to understand and implement the most common RSA SecurID use cases.

This example journey highlights using the RSA SecurID node to authenticate users:




# Secret Double Octopus (SDO) nodes

Integrates with Secret Double Octopus (SDO) to provide high-assurance, passwordless authentication system engineered to address the diverse authentication needs of a real-world, working enterprise.

## SDO prerequisite

Before configuring the SDO nodes in PingOne Advanced Identity Cloud, you complete the following prerequisite steps using the Octopus Management Console:

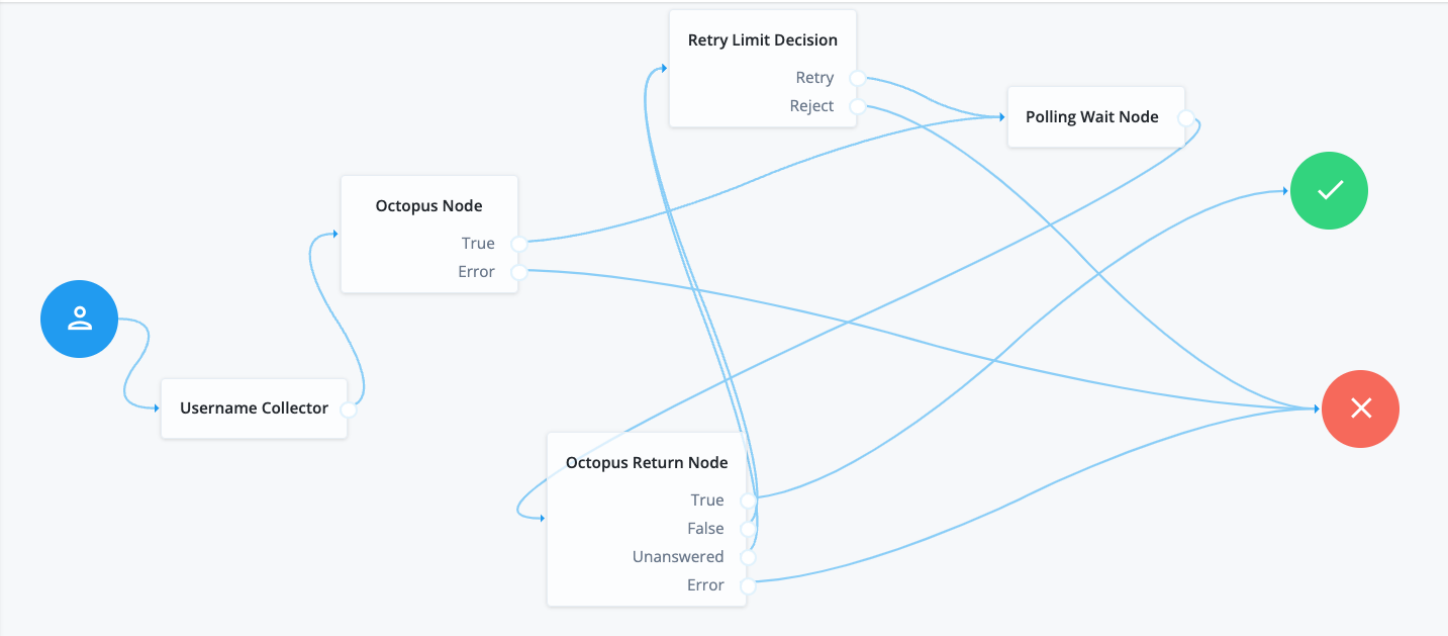
- Integrate your corporate directory.
- Create a REST API service.

After the prerequisite configurations are completed, you can obtain the API token, service URL, and service certificate from the Octopus Admin console. These three values are essential to configure SDO nodes in PingOne Advanced Identity Cloud.

## Setting up

To set up SDO with PingOne Advanced Identity Cloud, configure Octopus node and Octopus return node.

A typical authentication journey using SDO nodes:




### Octopus node configuration

The Octopus node sends the Octopus authentication request.

### Outcomes

#### `True`

Successfully obtained device ID.

#### `Error`

An error occurred when obtaining device ID.

### Properties

| Property | Usage |
|---|---|
| API token | The API Token of the Octopus Authenticator REST service from the Octopus Admin console. |
| Service URL | The URL of the Octopus Authenticator REST service from the Octopus Admin console. |
| Message | The message to be sent on authentication by push. |

### Octopus Return node configuration

The Octopus Return node checks that the Octopus authentication request was successfully approved.

### Outcomes

**`True`**

Successfully verified the device.

**`False`**

The device cannot be verified.

**`Unregistered`**

The device is not registered yet.

**`Error`**

An error occurred during verification.

### Properties

| Property | Usage |
| --- | --- |
| Service Certificate | The certificate of the Octopus Authenticator REST service from the Octopus Admin console. |

## SpyCloud Auth Node

Use the SpyCloud Auth node to integrate the SpyCloud service with Advanced Identity Cloud and assess if a user's password is compromised. If a user's password is compromised, the administrator can take remedial action, such as denying access and asking the user to reset their password, or forcing a password reset.

### Availability

| Product | Available? |
| --- | --- |
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

> **Note**
>
> AM customers can get the SpyCloud node here.

### Inputs

This node reads a customizable user identifier from shared state. The identifier is set in the `identifierSharedStateKey` property in the node configuration.

### Dependencies

To use this node, you should have already set up integration with the SpyCloud service in PingOne Advanced Identity Cloud. Refer to the SpyCloud documentation to set up the SpyCloud service in your environment.

### Configuration

| Property | Usage |
|---|---|
| API URL | The SpyCloud API URL. |
| API Key | The SpyCloud API key. |
| Severity | Allows you to filter based on the numeric severity code. You can find more information about severity codes in SpyCloud documentation. |
| identifierSharedStateKey | The shared state key to find the identifier value. |

### Outputs

Only when an error occurs does this node store the error in shared state.

### Outcomes

**`Compromised`**

Compromised password detected.

**`Not compromised`**

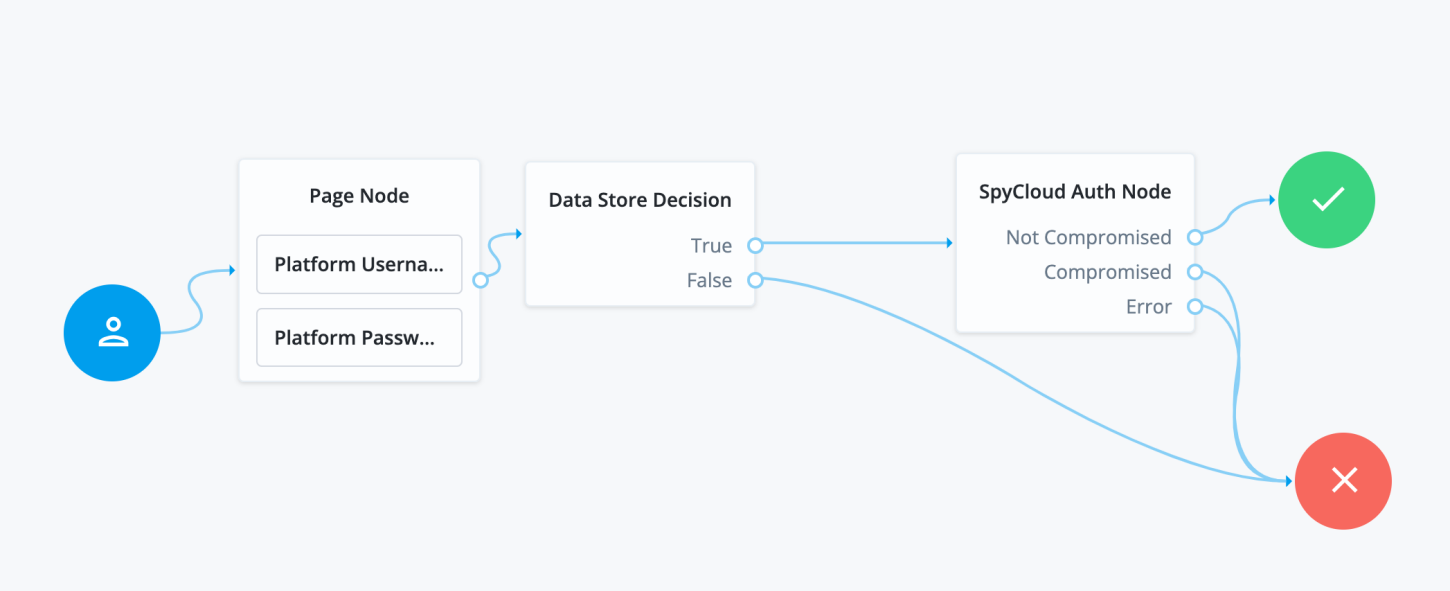
Password is not compromised.

**`Error`**

There was an error while assessing if the password was compromised.

### Example

The following sample journey illustrates the use of this node to assess if a user's password is compromised:




### Troubleshooting

If this node logs an error, then an error message is sent to shared state. For example:

- `"[SpyCloud] StackTrace", new Date() + ": " + stackTrace`
- `"[SpyCloud] Exception", new Date() + ": " + ex.getMessage()`

Review the log messages to find the reason for the error and address the issue appropriately.

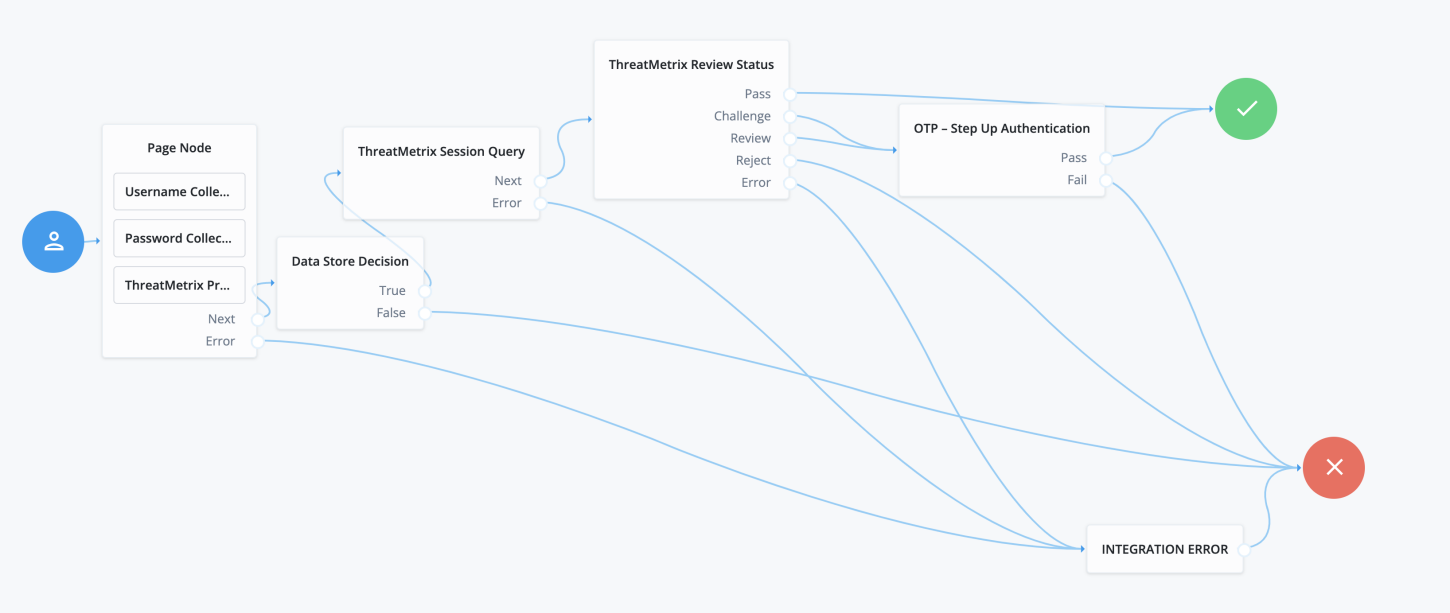
## ThreatMetrix Authentication nodes

Integrates Lexis-Nexis ThreatMetrix decision tools and enable device intelligence and risk assessment in PingOne Advanced Identity Cloud.

You can implement ThreatMetrix authentication using these nodes:

- ThreatMetrix Profiler node
- ThreatMetrix Session Query node
- ThreatMetrix Review Status node
- ThreatMetrix Reason Code node
- ThreatMetrix Update Review node

The following screenshot shows a typical authentication journey using ThreatMetrix nodes:



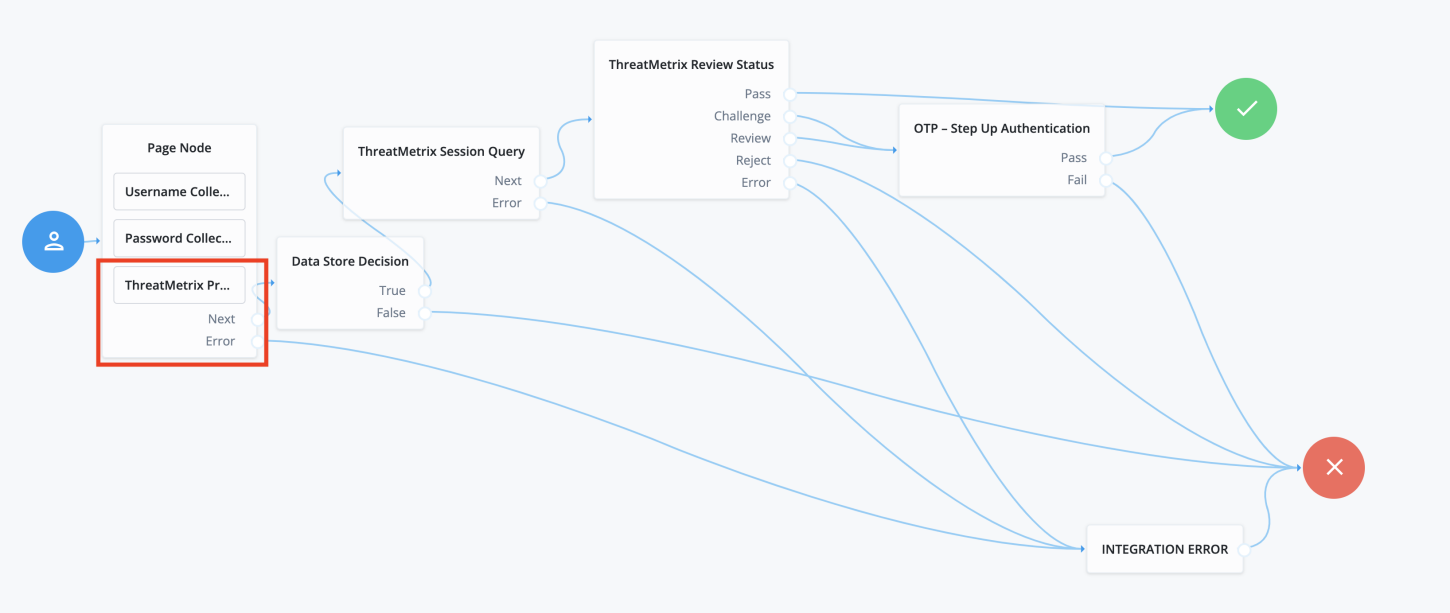

- The journey starts with a Page node to capture username and password. Simultaneously, the ThreatMetrix Profiler node captures device intelligence information associated to a unique session ID.
- The ThreatMetrix Session Query node uses the Session ID, along with user personal identifiable information (PII) to make an API call to the LexisNexis Dynamic Decision Platform (DDP) for risk assessment.
- Risk assessment is performed at the DDP to capture suspicious activity.
- The result of the risk assessment is captured as an API response in the ThreatMetrix Session Query node.
- The ThreatMetrix Review Status node interprets the API risk assessment response.
- The possible outcomes to continue the journey are `Pass`, `Challenge`, `Review`, or `Reject`.
- The `Challenge` and `Review` outcomes start a step-up second factor authentication.
- Regardless of the step-up authentication, the ThreatMetrix Update Review node adds truth data to the login event which enriches the information in the network and enhance fraud detection.

> **Note**
>
> The example described above shows initialization of step-up MFA along with outcome of either pass or fail. If a step-up authentication is initialized but is not completed, then it is a sign of potential fraud in a login event.

## ThreatMetrix Profiler node

Tags the PingOne Advanced Identity Cloud login page with the ThreatMetrix JavaScript to collect information about the event as part of a risk assessment use case. This node fetches the session ID and puts it in the shared state for use by other ThreatMetrix nodes in the journey.

A typical use of the ThreatMetrix Profiler node in a ThreatMetrix authentication journey:




## Properties

| Property | Usage |
|---|---|
| Org ID | ThreatMetrix generated unique identifier for your organization. |
| Page ID | Page identifier used when you set ThreatMetrix tag on multiple pages. |
| Profiler URI | ThreatMetrix Profiler URI. This can be the basic profiling URL or the enhanced profiling — hosted SSL URL. The default is the basic profiling URL. |
| Use Client Generated Session IDs | Enable this optional toggle if you have separately integrated ThreatMetrix JS tags in the application web page outside the PingOne Advanced Identity Cloud platform. When enabled, the ThreatMetrix Profiler node fetches the ThreatMetrix session ID from the application. |

## Outcomes

### `Next`

The request moves to the next node in the journey.

### `Error`

An error occurred.

## Other ThreatMetrix nodes

- ThreatMetrix Session Query node
- ThreatMetrix Review Status node
- ThreatMetrix Reason Code node

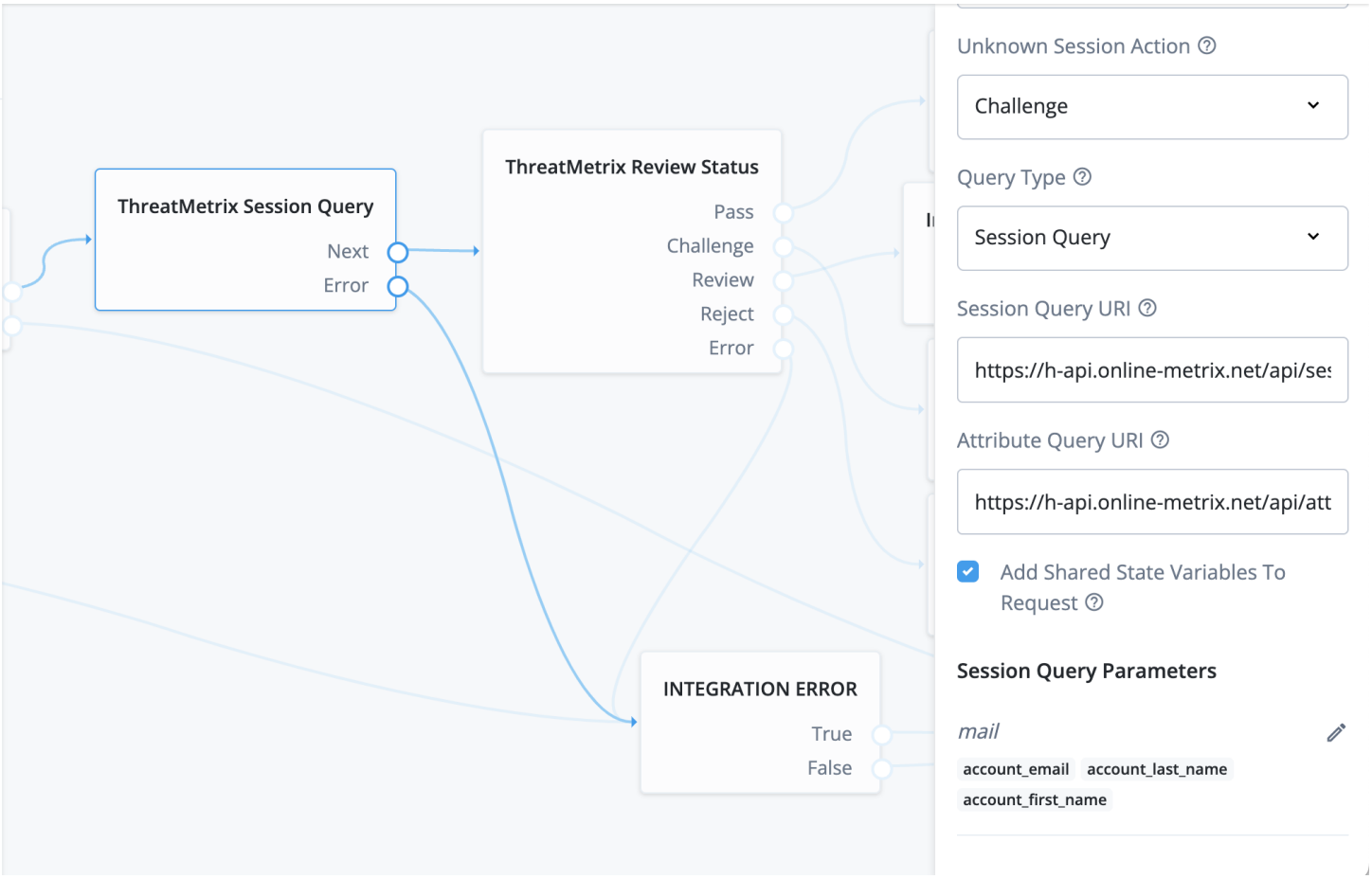
- ThreatMetrix Update Review node

## ThreatMetrix Session Query node

**Obtains the policy decision about a user session.**

**This node:**

- **Queries the DDP**
- **Retrieves a policy decision about the user session**
- **Stores the configured session query parameters in shared state**




### Properties

| Property | Usage |
|---|---|
| Org ID | ThreatMetrix-generated unique identifier for your organization. |
| API Key | Unique key allocated by ThreatMetrix and associated with an Org ID. |
| Service Type | Defines the API Response output fields returned from the API Request. The default is `session-policy`. |

| Property | Usage |
| --- | --- |
| Event Type | Specifies the type of transaction or event. The default is the login event. |
| Policy | The ThreatMetrix policy used to query the DDP. |
| Unknown Session Action | Unknown sessions occur for a variety of reasons where the device profiling has failed. The unknown sessions action parameter lets the system administrator to define a response action when an unknown session occurs at runtime. |
| Query Type | Defines the query type to send to ThreatMetrix. Session query requires device profile information and the attribute query does not require device profile information. |
| Session Query URI | ThreatMetrix Session Query URI, which is ignored when the `Query Type` property is not set to `Session Query`. |
| Attribute Query URI | ThreatMetrix Attribute Query URI, which is ignored when the `Query Type` property is not set to `Attribute Query`. |
| Add Shared State Variables To Request | A boolean property. Enable this property to add additional parameters to the ThreatMetrix API request. In general,we recommend that you add as much data as possible to the API requests to improve fidelity of the risk assessment. |
| Session Query Parameters | A list of key-value pairs. Key is a ThreatMetrix attribute and the value is a AM attribute. The values are read from the authenticated identity store. |

### Outcomes

#### `Next`

The request moves to the next node in the journey.

#### `Error`

An error such as a misconfigured URI, or network failure has occurred.

### Other ThreatMetrix authentication nodes:

- ThreatMetrix Profiler node
- ThreatMetrix Review Status node
- ThreatMetrix Reason Code node
- ThreatMetrix Update Review node

## ThreatMetrix Review Status node

Analyzes the response from the auth-node-threat-metrix-query.adoc and routes based on the review_status API Response. The possible outcomes to route are `Pass`, `Challenge`, `Review` or `Reject`.

### Properties

There are no properties to configure in this node.

### Outcomes

**`Pass`**

The request is successfully validated.

**`Challenge`**

The request requires step-up authentication, accordingly directed to appropriate node.

**`Review`**

The request needs to be further reviewed and so may be sent to step-up authentication or another appropriate node.

**`Reject`**

The request has failed validation and is rejected.

**`Error`**

An error condition has occurred in validating the request.

### Other ThreatMetrix authentication nodes:

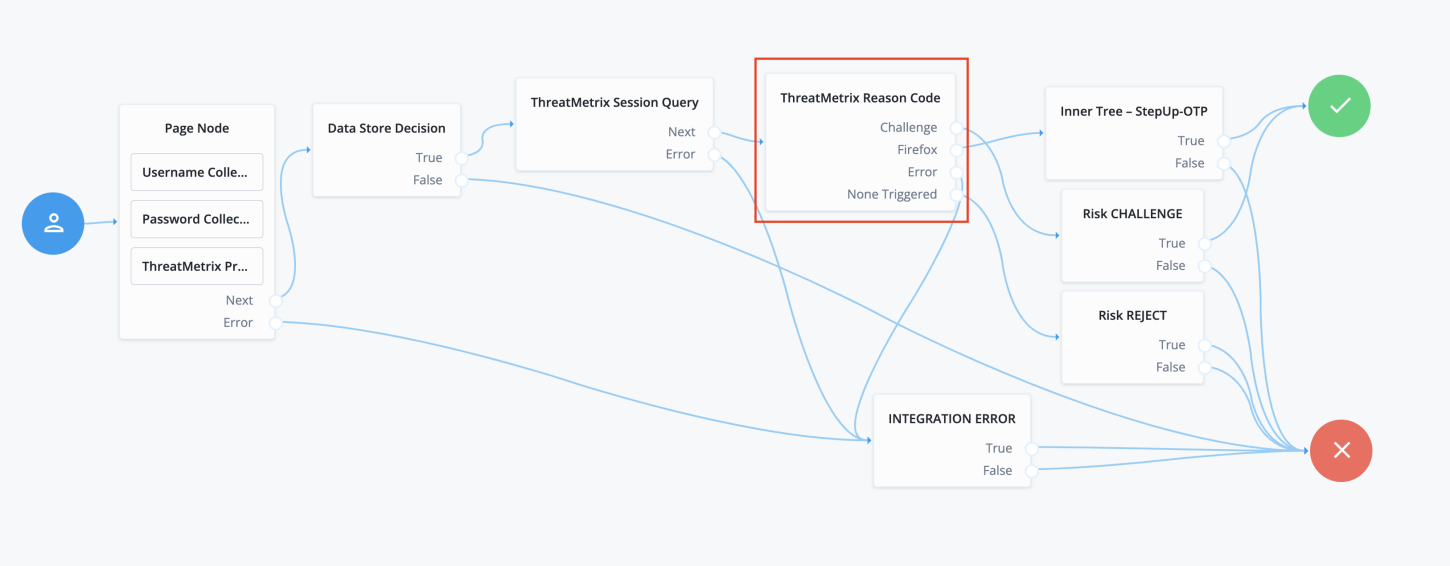
- ThreatMetrix Profiler node
- ThreatMetrix Session Query node
- ThreatMetrix Reason Code node
- ThreatMetrix Update Review node

## ThreatMetrix Reason Code node

Analyzes the response from the ThreatMetrix Session Query node and checks to see if an individual reason code has been returned. These reason codes correspond to the rule names you configured in LexisNexis Dynamic Decision Platform (DDP) using the ThreatMetrix console.

The ThreatMetrix Reason Code node enable you to dynamically configure outcomes and manage journeys quickly. Often this node is used in place of the ThreatMetrix Review Status node.

A typical journey using ThreatMetrix Reason Code node is shown here:




In the journey shown, the ThreatMetrix Reason Code node is configured with the Firefox reason code to appropriately route the flow to a step-up authentication node.

### Properties

A list of reason codes you would like to check and apply ThreatMetrix policy. When a reason code is added to this list, a new outcome is presented on the node. The node iterates through the configured reason codes until a code is found and returns the obtained reason as the outcome. If no reason code is found, None Triggered outcome is returned.

In the example diagram shown above the following reason codes have been added.

| Property | Usage |
|---|---|
| Challenge | Route to the Risk Challenge node. |
| Firefox | Route to a step-up authentication node |
| Error | Route to an integration error handler node |
| None Triggered | Route to a rejection node |

### Outcomes

#### `Challenge`

Add a challenge step in authentication - for additional security.

#### `Firefox`

Add a step-up authentication when the request comes from Firefox browser.th

#### `None Triggered`

No configured reason code is found.

**`Error`**

An error occurred.

**Other ThreatMetrix authentication nodes:**

- ThreatMetrix Profiler node
- ThreatMetrix Session Query node
- ThreatMetrix Review Status node
- ThreatMetrix Update Review node

## ThreatMetrix Update Review node

Provides retrospective truth data from an authentication event to the ThreatMetrix portal. Truth data is beneficial for tuning policies and overall fraud detection.

**Properties**

| Property | Usage |
|---|---|
| Org ID | ThreatMetrix generated unique identifier for your organization. |
| API Key | Unique ThreatMetrix allocated key associated with an organization identifier. |
| Update URI | ThreatMetrix update URI. |
| Event Tag | Represents the event disposition and outcome of the ThreatMetrix Query. Generally, the `Step-Up Initialize` tag is configured prior to sending a step-up authentication request. Following a step-up authentication, either `Step-Up Pass` or `Step-Up Fail` tag is sent to the ThreatMetrix platform. |
| Step-Up Method | The chosen step-up method from one of ThreatMetrix's predefined set of global methods. |
| Notes | An optional parameter that allows you to append any textual detail as the reason for updating the review status. |

**Outcomes**

**`Next`**

The request moves to the next node in the journey.

**`Error`**

An error occurred.

**Other ThreatMetrix authentication nodes:**

- ThreatMetrix Profiler node
- ThreatMetrix Session Query node
- ThreatMetrix Review Status node
- ThreatMetrix Reason Code node

# Twilio Identifier node

Pulls an attribute from the user's profile and stores it in the shared state.

The attributes should hold the telephone number or the email address depending on the delivery channel for sending the one-time passcode.

## Outcomes

- `True` : Found user and user identifier.
- `False` : User isn't found.
- `Error` : Exception occurred, and the error message stored in shared store.

## Properties

| Property | Usage |
| --- | --- |
| Identifier Attribute | The IDM attribute to find in the user profile:<br>• `mail`<br>• `telephoneNumber` |
| Identifier Shared State | The shared state variable in which to store the value. |

## Examples

For examples in context, refer to the Twilio examples.

# Twilio Verify Collector Decision node

Collects a one-time passcode from the user and validates it against the Twilio Verify service.

Place this node after the Twilio Verify Sender node in the authentication flow.

### Outcomes

- `True` OTP is approved successfully.
- `False` OTP isn't approved.
- `Error` Exception occurred, and the error message is stored in shared store.

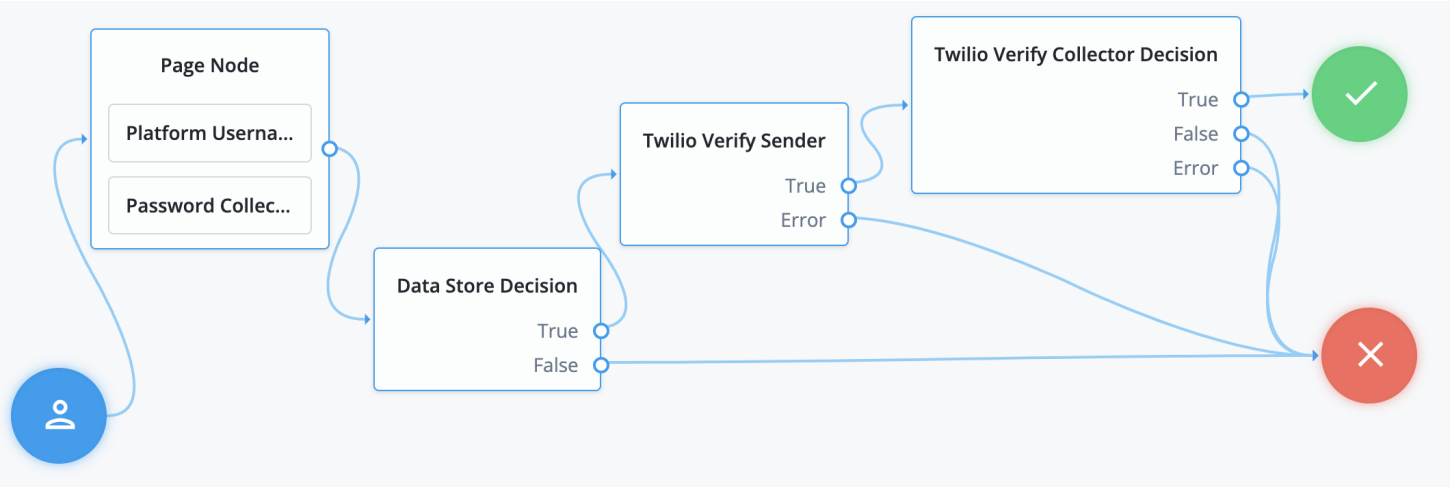
### Properties

| Property | Usage |
|---|---|
| Hide Code Text | If enabled, hides the code text from the end user similar to a password collector.<br>Default: enabled |
| Identifier Shared State | The shared state variable to search for the user identifier. |

### Twilio examples

#### Prompt user for phone number

The following example validates the user's username and password, asks the user for their phone number, and sends a one-time passcode over SMS to that phone number. In this example, the Twilio Verify Sender node has the Request Identifier enabled. In production, alter this flow to pull the user's `userIdentifierphoneNumber` from their profile:




#### Find phone number in user profile

The following example validates the user's username and password, identifies the user, finds the user's phone number in their profile, and sends a one-time passcode over SMS to that phone number.

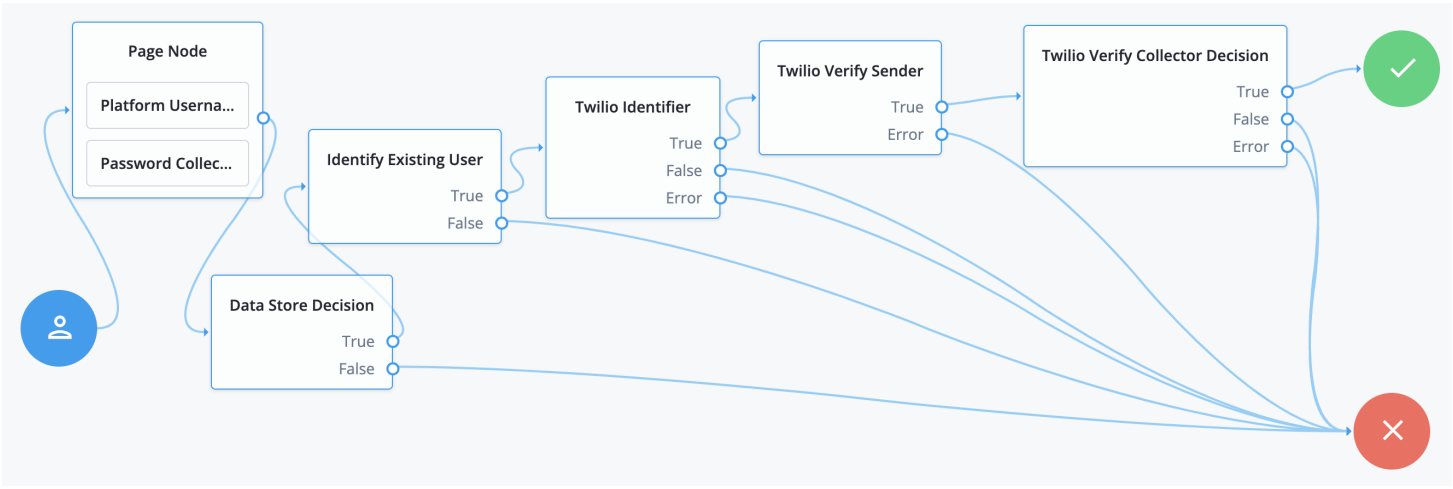
The Identify Existing User node has the following configuration:

#### *Identifier*

`_id`

### *Identify Attribute*

`username`




**Verify the phone number is a mobile carrier**

The following example validates the user's username and password, identifies the user, finds the user's phone number in their profile, verifies the phone number is a mobile carrier, and sends a one-time passcode over SMS to that phone number.

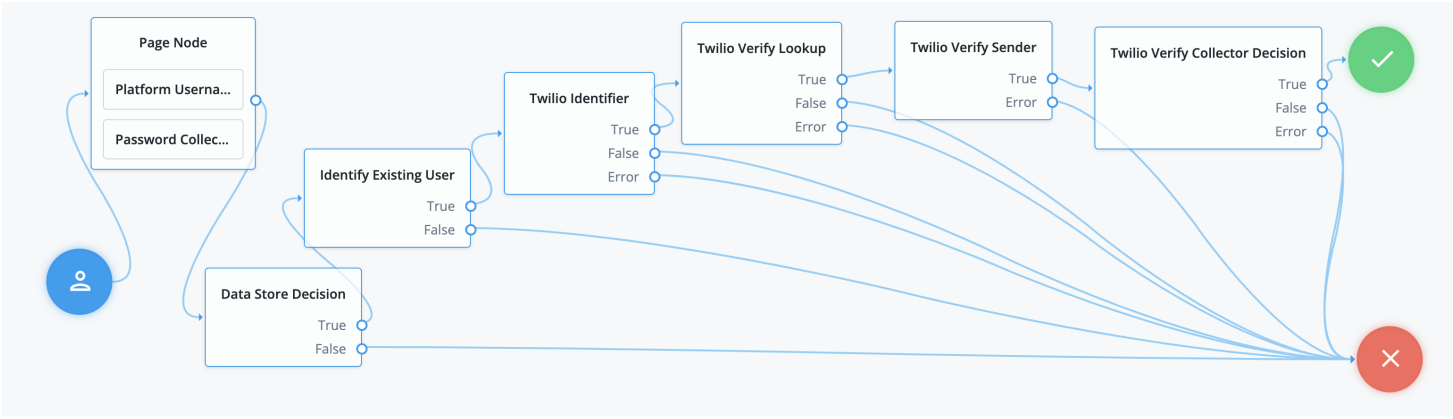
The Identify Existing User node has the following configuration:

### *Identifier*

`_id`

### *Identify Attribute*

`username`




## Twilio Verify Lookup node

Checks if the provided phone number reflects a mobile carrier with the Twilio Verify service.

### Outcomes

- `True`: Mobile phone number is found.
- `False`: Phone number isn't found, or phone number isn't a mobile number.
- `Error`: Exception occurred, and the error message is stored in shared store.

### Properties

| Property | Usage |
|---|---|
| Account SID | The unique string to identify the account found in the Twilio account dashboard. |
| Authentication Token | The authentication token found in the Twilio account dashboard. |
| Identifier Shared State | The shared state variable to search for the user identifier. |

### Examples

For examples in context, refer to the Twilio examples.

## Twilio Verify Sender node

Initiates a Twilio Verify request to send a one-time passcode to the user's device as an additional authentication factor.

If you do not enable the Request Identifier property, make sure the shared node state has a `userIdentifier` field set with a valid phone number or an email ID when the flow reaches this node.

Follow this node with the Twilio Verify Collector Decision node in the authentication flow.

### Outcomes

- `True`: OTP sent successfully.
- `Error`: Exception occurred, and the error message is stored in the shared store.

### Properties

| Property | Usage |
|---|---|
| Account SID | The unique string to identify the account found in the Twilio account dashboard. |

| Property | Usage |
| --- | --- |
| Authentication Token | The authentication token found in the Twilio account dashboard. You can use ESVs to set credentials appropriately for each environment in a multi-environment setup. Refer to Introduction to ESVs for more information on ESVs. |
| Service SID | The unique string to identify the service in the Twilio account dashboard. |
| Channel | The delivery channel for sending the one-time passcode:<br>**`SMS` *(default)***<br>Phone numbers must contain the country code and can optionally contain special characters. The node removes all special characters before making the Twilio Verify request.<br>Valid identifiers include `+15553231234` or `1(555)323-1234`. Invalid identifiers include `(555) 323-1234` as the country code is missing.<br>**`Call`**<br>The same rules apply for phone numbers.<br>**`Email`**<br>To use this channel, first integrate the Twilio Verify with SendGrid services, as described in Send Email Verifications with Verify and Twilio SendGrid.<br>**`WHATSAPP`**<br>The same rules apply for phone numbers. |
| Request Identifier | Whether to prompt the user for their identifier.<br>Optionally, this node can prompt the user for their identifier. By default, the authentication flow shared node state must contain a `userIdentifier` field that holds a valid phone number or email address.<br>Default: disabled |
| Identifier Shared State | The shared state variable to search for the user identifier. |

### Examples

For examples in context, refer to the Twilio examples.

## TypingDNA

The TypingDNA nodes provide two-factor authentication using proprietary AI-based typing biometrics (keystroke dynamics) and PingOne Advanced Identity Cloud. Use the TypingDNA nodes to create alternative authentication workflows to record and verify an end user's typing behavior.

Advanced Identity Cloud supports the following TypingDNA nodes:

- TypingDNA Recorder node

- TypingDNA Short Phrase Collector node
- TypingDNA Decision node
- TypingDNA Reset Profile node

Learn more about TypingDNA solutions in the TypingDNA API documentation.

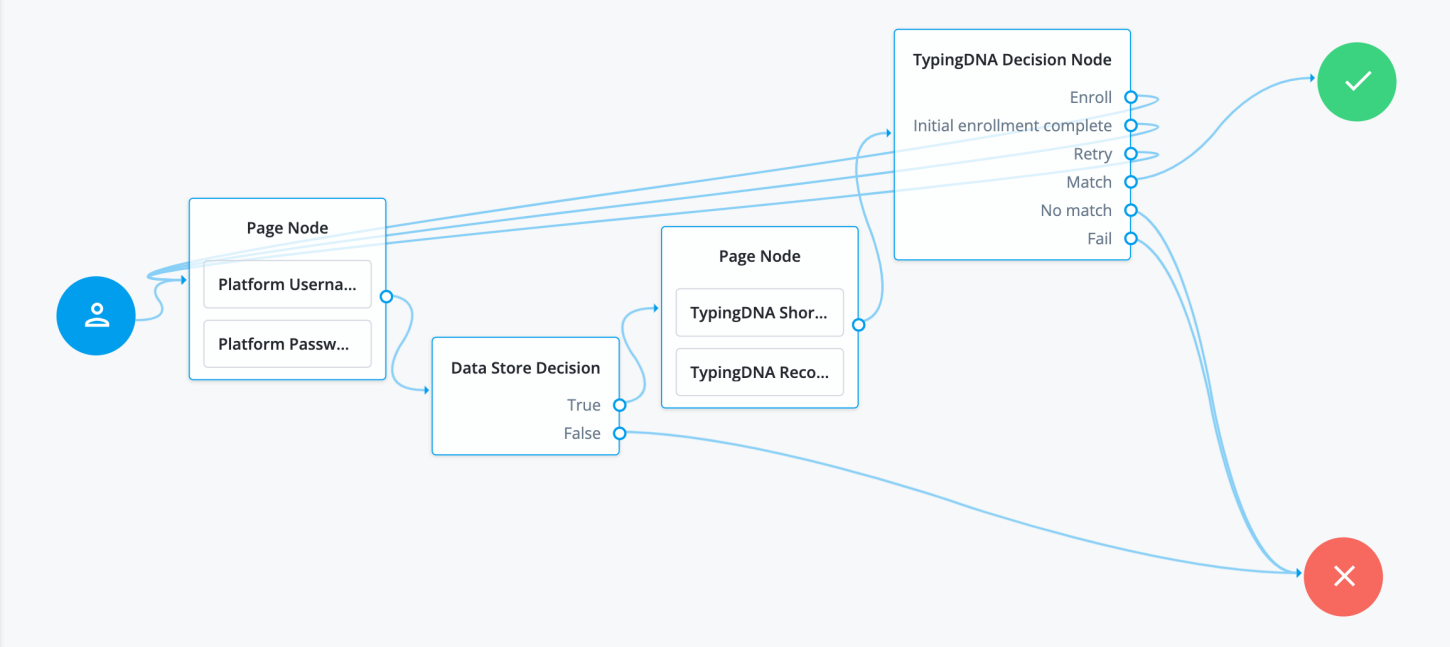
## Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

## Dependencies

Before using the TypingDNA nodes, you must configure PingOne Advanced Identity Cloud integration with TypingDNA as described in Step-by-step: TypingDNA as a 2FA factor in Ping.

## Example




The main highlights of the example journey are:

- The TypingDNA Short Phrase Collector node displays a phrase and prompts the user to type it in.
- The TypingDNA Recorder records the end user's keystrokes.

- The TypingDNA Decision node analyzes the typed data and provides an outcome. Depending on the outcome, the further journey proceeds:
  - `Enroll`, Initial enrollment is incomplete and the user is prompted to enter credentials again.
  - `Retry`: The user is prompted to enter credentials.
  - `Match`: The journey ends in a successful outcome.
  - `No match` or `Fail`: The journey ends in an unsuccessful outcome.

## TypingDNA Decision node

The TypingDNA Decision node handles the authentication logic by communicating with the TypingDNA Authentication API. To perform this, the TypingDNA Decision node uses the API key and API secret from the TypingDNA user account dashboard.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads `TDNA_TEXT_TO_ENTER`, `TDNA_DEVICE_TYPE`, `TDNA_TEXT_ID`, and `TDNA_TYPING_PATTERN` parameters from the shared state.

### Dependencies

Before using the TypingDNA nodes, you must set up PingOne Advanced Identity Cloud integration with TypingDNA as described in the Step-by-step: TypingDNA as a 2FA factor in Ping. This node also requires that the `TypingDNA Recorder` node be configured earlier in the journey.

### Properties

| Property | Usage |
|---|---|
| API URL | The URL for TypingDNA API, for example, https://api.typingdna.com |
| API key | The API key from your TypingDNA account |
| API secret | The API secret from your TypingDNA account |
| Retries | The number of times a user is allowed to retry authentication if it fails. Default: `0`, meaning no retries are allowed. |

| Property | Usage |
| --- | --- |
| Authentication API Configuration | Two options are available:<br>• `Basic` : Uses the default settings of Authentication API for auto-enroll, minimum number of enrollments, and thresholds for auto-enroll and verification. All requests will use the `/auto` endpoint that is free for all types of Authentication API clients.<br>• `Advanced` : Uses the `/verify` endpoint. The authentication behavior can be managed by using the API Settings menu in the TypingDNA Dashboard for Authentication API. Don't use the `Advanced` configuration with the free starter account. |
| Hash algorithm | The hash-algorithm used to anonymize user IDs before sending them to the TypingDNA Authentication API. |
| Salt | A string that is used to anonymize the user ID for additional security. For example, username or user email.<br>Default: null. |
| Request identifier | An optional parameter that may be used to identify requests coming from a specific PingOne Advanced Identity Cloud authentication tree. The identifier also appears in the TypingDNA logs.<br>Default: ForgeRock. |
| Request time out | Time in milliseconds after which each request to the TypingDNA Authentication API times out if no response was received.<br>Default: 8000 ms. |

### Outputs

This node doesn't store any output in the shared state.

### Outcomes

#### `Enroll`

This occurs if the user's number of saved patterns was lower than the number of enrollments. The newly presented typing pattern will be saved to the profile. In this case, no authentication is actually performed.

For passive enrollment, you need to continue the flow to an alternative authentication node or to success. For active enrollment, you need to link this outcome back to the page node, such as the login page or the short phrase page, where the typing patterns are collected.

#### `Initial enrollment complete`

This occurs when the user's number of saved patterns is equal to that needed for enrollment. The minimum number of patterns for initial enrollment can be configured from the API Settings menu in the TypingDNA Dashboard for Authentication API.

The API Settings menu is available only for paid Authentication API plans.

**`Retry`**

This occurs if the authentication fails and the number of times the user has retried is lower than the Retries property value configured. The authentication can fail because the `Match` threshold hasn't been reached or because a non-critical error, which could be overcome by trying again, has occurred. To effectively allow retry effectively, link this outcome back to the page node that collects typing patterns.

**`Fail`**

This occurs when a critical error occurs, such as if invalid API credentials are entered. This outcome should be linked to an alternative authentication node or `Failure`.

**`Match`**

This occurs when the authentication is successful. For this, the net score of the authentication must exceed the Match threshold. This outcome is usually linked to `Success`.

**`No match`**

This occurs if the authentication fails and the allowed number of retries has reached. This outcome would be linked to an alternative authentication node.

### Example

Sample journey using this node.

### Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

## TypingDNA Recorder node

This node collects typing behaviors and transforms them into typing patterns.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node does not read any input from the shared state.

### Dependencies

Before using the TypingDNA nodes, you must set up PingOne Advanced Identity Cloud integration with TypingDNA as described in Step-by-step: TypingDNA as a 2FA factor in Ping.

### Properties

| Property | Usage |
|---|---|
| Display Messages | Specify if the node should display messages for retry and enroll. |
| Show Visualizer | A visual representation of how the user types. |
| Disable copy and paste | Prevent the end user from copying and pasting the short phrase, to enforce typing analysis. |
| Recorder Target IDs | The identifiers of the typing patterns recorded by the TypingDNA recorder. |
| Submit button ID | When the user clicks the button, it triggers the TypingDNA Recorder node to record the typing pattern of the user. When the user submits the credentials, the recorded pattern is sent to the backend. |

### Outcomes

Single outcome path.

### Outputs

This node stores the following properties in the shared state:

- `TDNA_TEXT_TO_ENTER`: The short phrase the end user was prompted to type.
- `TDNA_DEVICE_TYPE`: The type of device from which the end user initiated the request
- `TDNA_TEXT_ID`: The identifier of the typing pattern recorded.
- `TDNA_TYPING_PATTERN`: The recorded typing pattern.

### Example

Sample journey using this node.

### Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

## TypingDNA Reset Profile node

Use the TypingDNA Reset Profile node to reset an end user's TypingDNA profile. When a user's profile is reset, all the patterns recorded for the user are deleted, and the user is enrolled again the next time they sign on. To reset a profile, you must use the same API key, API secret, and the salt as configured in the TypingDNA Decision node.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node reads the `username` of the authenticating user from the shared state.

Implement a Username Collector node (standalone AM) or Platform Username node (Advanced Identity Cloud and Ping Identity Platform deployments) earlier in the journey.

### Dependencies

The dependencies for TypingDNA nodes are explained here.

### Properties

| Property | Usage |
|---|---|
| API URL | The URL for TypingDNA API, for example, https://api.typingdna.com. |
| API key | The API key from your TypingDNA account. |
| API secret | The API secret from your TypingDNA account. |
| Hash algorithm | The hashing algorithm that is used for anonymizing usernames before sending them to the TypingDNA Authentication API. |
| Salt | A string that is used to anonymize the user ID for additional security. For example, username or user email.<br>Default is null. |
| Request identifier | An optional parameter that can be used to identify requests coming from a specific PingOne Advanced Identity Cloud authentication journey. The identifier also appears in the TypingDNA logs.<br>Default: ForgeRock. |
| Request time out | Time in milliseconds after which requests to the TypingDNA Authentication API time out if no response is received.<br>Default: 8000 ms |

### Outputs

This node does not store any output to the shared state.

### Outcomes

#### Success

This outcome is achieved when the profile was successfully deleted.

#### Error

This outcome is achieved when a critical error occurs, such as if invalid API credentials are entered. You can link this outcome to an alternative node, such as a `Retry Limit Decision node`, or to the Failure node.

### Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.

## TypingDNA Short Phrase Collector node

This node provides an alternative authentication method based on typing biometrics. End users are requested to type a short phrase so that their typing patterns are collected and verified.

### Availability

| Product | Available? |
|---|---|
| PingOne Advanced Identity Cloud | Yes |
| PingAM (self-managed) | Yes |
| Ping Identity Platform (self-managed) | Yes |

### Inputs

This node doesn't read any input from the shared state.

### Dependencies

Before using the TypingDNA nodes, you must set up Advanced Identity Cloud integration with TypingDNA as described in Step-by-step: TypingDNA as a 2FA factor in Ping.

### Configuration

| Property | Usage |
|---|---|
| Text to enter | The short phrase the end user must type to record their keystrokes. |

### Outputs

This node does not store any output in the shared state.

### Outcomes

#### `Next`

Single outcome path.

### Troubleshooting

If this node logged an error, review the transaction log to find the reason for the error.