Advanced Identity Cloud/PingAM Login Widget

July 30, 2026

LOGIN-WIDGET

Table of Contents

Advanced Identity Cloud/PingAM Login Widget

Advanced Identity Cloud/PingAM Login Widget

PingOne Advanced Identity Cloud | PingAM | JavaScript

The Advanced Identity Cloud/PingAM Login Widget is an all-inclusive UI component to help you add authentication, user registration, and other self-service journeys into your web applications.

You can use the Advanced Identity Cloud/PingAM Login Widget within React, Vue, Angular and a number of other modern JavaScript frameworks, as well as vanilla JavaScript.

It does not currently support server-side rendering (SSR), including Node.js.

The Advanced Identity Cloud/PingAM Login Widget uses `@forgerock/journey-client` for journey execution, and `@forgerock/oidc-client` for OAuth/OIDC tokens, user info, and logout. It adds a user interface and state management on top, eliminating the need to develop and maintain UI components for complex authentication experiences.

This rendering layer uses Svelte and Tailwind, but these are "compiled away" resulting in no runtime dependencies.

The resulting Advanced Identity Cloud/PingAM Login Widget is both library- and framework-agnostic.

Topics

Get started with the Advanced Identity Cloud/PingAM Login Widget in the following sections:

Tutorial

Learn how to install the Advanced Identity Cloud/PingAM Login Widget, add it to your applications and manage user authentication and self-service journeys.

Themes

Discover how to reconfigure the Advanced Identity Cloud/PingAM Login Widget to use different colors, fonts, or sizing, or select between the light and dark modes.

Advanced Identity Cloud/PingAM Login Widget

Use cases

Find out how to achieve some common use case scenarios using the Advanced Identity Cloud/PingAM Login Widget.

Integrations

Integrate the Advanced Identity Cloud/PingAM Login Widget into various different frameworks.

API

Access a list of the modules included in the Advanced Identity Cloud/PingAM Login Widget and the API they offer.

Functionality

The Advanced Identity Cloud/PingAM Login Widget supports the following PingOne Advanced Identity Cloud and PingAM features:

Advanced Identity Cloud/PingAM Login Widget

✓ Supported	✕ Unsupported

• Page node
• Username
• Password
• QR code scan-to-register
• Push authentication and registration
• One-time passwords and registration
• Recovery codes
• WebAuthn
• Device profiles
• Social login providers:
  ◦ Apple
  ◦ Facebook
  ◦ Google
• Email suspend, or "magic links"
• CAPTCHA display
  ◦ hCaptcha
  ◦ reCAPTCHA v2
  ◦ reCAPTCHA v3
• PingOne Protect
	• Centralized login
	• `TextOutputCallback` callbacks containing scripts
	• SAML federation

Requirements

The Advanced Identity Cloud/PingAM Login Widget is designed to work with the following:

- An ECMAScript module or CommonJS enabled client-side JavaScript app
- A "modern", fully-featured browser such as Chrome, Firefox, Safari, or Chromium Edge

The Advanced Identity Cloud/PingAM Login Widget supports vanilla JavaScript and many frameworks. It is tested against the following:

✓ Tested	✕ Unsupported
• React • Svelte • Vue	• Server-side rendering (SSR), including Node.js

The Advanced Identity Cloud/PingAM Login Widget is ***not designed or tested*** for use with the following:

- Internet Explorer
- Legacy Edge
- WebView

Advanced Identity Cloud/PingAM Login Widget

- Electron
- Modified, browser-like environments

What's New

Subscribe to get automatic updates:

- Ping (ForgeRock) SDKs Changelog RSS feed
- Ping (ForgeRock) SDKs Changelog email notifications

Latest updates

Advanced Identity Cloud/PingAM Login Widget 2.0

July 28, 2026 `major`

Key Features

- Added passkey autofill support (WebAuthn Conditional UI), letting users sign in by selecting a passkey from the browser's native autofill picker.

  Learn more in Implementing passkey autofill.

- Added invisible reCAPTCHA v2, invisible hCaptcha, and reCAPTCHA Enterprise support via the `captcha` configuration option.

  Learn more in Implement a CAPTCHA.

- Added runtime theming support: pass a `theme` object to `configure({ style: { theme } })` to apply brand colors and fonts without rebuilding the widget.

  Learn more in Set theme colors and fonts.

Added

- Added dedicated stages for MFA enrollment and recovery codes, and for the PingOne Advanced Identity Cloud admin registration flow (welcome, OTP verification, and privacy policy acceptance).
- Added experimental support for custom callback and stage components, letting you override or extend the widget's built-in UI. Refer to the Advanced Identity Cloud/PingAM Web Login Framework repository for details.
- Added `TextInputCallback` support.

Changed

- Migrated the Journey implementation from `@forgerock/javascript-sdk` to `@forgerock/journey-client`.
- Migrated the OIDC token, userinfo, and logout implementations from `@forgerock/javascript-sdk` to `@forgerock/oidc-client`.
- Migrated the PingOne Protect implementation from `@forgerock/ping-protect` to `@forgerock/protect`.
- Migrated the Advanced Identity Cloud/PingAM Login Widget internal build to Svelte 5 and a `pnpm` monorepo architecture.

Fixed

- Fixed a visual regression in the info alert that showed the wrong background color when runtime theming was active.
- Fixed a typo and some grammar issues in the default English locale text.

- Fixed an issue preventing ConfirmationCallback options beyond the second from being selectable.
- Fixed the passkey prompt appearing twice during WebAuthn sign-in.
- Fixed the `failureUrl` not being exposed in the journey error handler after login failure .

Breaking changes

- The synchronous `configuration()` export and its `.set()` method are replaced by a single async `configure()` function. `await` it before calling any other Widget API.
- The nested `forgerock` config object is replaced by a flat, top-level shape. Endpoint discovery is driven by a single top-level `wellknown` URL, and OAuth settings now live in an `oidcClient` sub-object.
- The `request` module export is removed. Get tokens from `user.tokens().get()` and call `fetch()` with a manual `Authorization` header.
- OAuth/OIDC, user info, and logout are now backed by `@forgerock/oidc-client`. PingOne Protect is backed by `@forgerock/protect`. `@forgerock/javascript-sdk` is no longer a dependency of the widget.
- The `serverConfig` object, `realmPath`, `tree`, `tokenStore`, `prefix`, `logLevel`, `logger`, `platformHeader`, `oauthThreshold`, and all `serverConfig.paths.*` overrides are removed.

For migration guidance and before/after examples, refer to Breaking changes.

Advanced Identity Cloud/PingAM Login Widget 1.3.0

June 5, 2024 `minor`

Added

- Added support for integration with PingOne Protect.
- Added the name of the device to the recovery codes page.

Fixed

- Corrected an issue that prevented use of the `logLevel` parameter in the Advanced Identity Cloud/PingAM Login Widget configuration.
- Fixed an issue with configuration literals that caused `ZodError` messages in the console.

Advanced Identity Cloud/PingAM Login Widget 1.2.1

January 8, 2024 `minor`

Added

- Support for CAPTCHA nodes.

Advanced Identity Cloud/PingAM Login Widget 1.1

July 17, 2023 `minor`

Added

- Support for device profiling callbacks (`DeviceProfileCallback`)
- Support for web authentication (WebAuthn) journeys and trees.

Advanced Identity Cloud/PingAM Login Widget 1.0

April 18, 2023 `major`

Changed

- First public release

Login Widget changelog

Subscribe to get automatic updates:

- Ping (ForgeRock) SDKs Changelog RSS feed
- Ping (ForgeRock) SDKs Changelog email notifications

Advanced Identity Cloud/PingAM Login Widget 2.0

July 28, 2026 `major`

Key Features

- Added passkey autofill support (WebAuthn Conditional UI), letting users sign in by selecting a passkey from the browser's native autofill picker.

  Learn more in Implementing passkey autofill.

- Added invisible reCAPTCHA v2, invisible hCaptcha, and reCAPTCHA Enterprise support via the `captcha` configuration option.

  Learn more in Implement a CAPTCHA.

- Added runtime theming support: pass a `theme` object to `configure({ style: { theme } })` to apply brand colors and fonts without rebuilding the widget.

  Learn more in Set theme colors and fonts.

Added

- Added dedicated stages for MFA enrollment and recovery codes, and for the PingOne Advanced Identity Cloud admin registration flow (welcome, OTP verification, and privacy policy acceptance).
- Added experimental support for custom callback and stage components, letting you override or extend the widget's built-in UI. Refer to the Advanced Identity Cloud/PingAM Web Login Framework repository for details.
- Added `TextInputCallback` support.

Changed

- Migrated the Journey implementation from `@forgerock/javascript-sdk` to `@forgerock/journey-client`.
- Migrated the OIDC token, userinfo, and logout implementations from `@forgerock/javascript-sdk` to `@forgerock/oidc-client`.
- Migrated the PingOne Protect implementation from `@forgerock/ping-protect` to `@forgerock/protect`.
- Migrated the Advanced Identity Cloud/PingAM Login Widget internal build to Svelte 5 and a `pnpm` monorepo architecture.

Fixed

- Fixed a visual regression in the info alert that showed the wrong background color when runtime theming was active.
- Fixed a typo and some grammar issues in the default English locale text.
- Fixed an issue preventing ConfirmationCallback options beyond the second from being selectable.
- Fixed the passkey prompt appearing twice during WebAuthn sign-in.
- Fixed the `failureUrl` not being exposed in the journey error handler after login failure .

Breaking changes

- The synchronous `configuration()` export and its `.set()` method are replaced by a single async `configure()` function. `await` it before calling any other Widget API.
- The nested `forgerock` config object is replaced by a flat, top-level shape. Endpoint discovery is driven by a single top-level `wellknown` URL, and OAuth settings now live in an `oidcClient` sub-object.
- The `request` module export is removed. Get tokens from `user.tokens().get()` and call `fetch()` with a manual `Authorization` header.
- OAuth/OIDC, user info, and logout are now backed by `@forgerock/oidc-client`. PingOne Protect is backed by `@forgerock/protect`. `@forgerock/javascript-sdk` is no longer a dependency of the widget.
- The `serverConfig` object, `realmPath`, `tree`, `tokenStore`, `prefix`, `logLevel`, `logger`, `platformHeader`, `oauthThreshold`, and all `serverConfig.paths.*` overrides are removed.

For migration guidance and before/after examples, refer to Breaking changes.

Advanced Identity Cloud/PingAM Login Widget 1.3.0

June 5, 2024 `minor`

Added

- Added support for integration with PingOne Protect.
- Added the name of the device to the recovery codes page.

Fixed

- Corrected an issue that prevented use of the `logLevel` parameter in the Advanced Identity Cloud/PingAM Login Widget configuration.

- Fixed an issue with configuration literals that caused `ZodError` messages in the console.

Advanced Identity Cloud/PingAM Login Widget 1.2.1

January 8, 2024 `minor`

Added

- Support for CAPTCHA nodes.

Advanced Identity Cloud/PingAM Login Widget 1.1

July 17, 2023 `minor`

Added

- Support for device profiling callbacks (`DeviceProfileCallback`)
- Support for web authentication (WebAuthn) journeys and trees.

Advanced Identity Cloud/PingAM Login Widget 1.0

April 18, 2023 `major`

Changed

- First public release

Breaking changes

This page summarizes breaking changes to the Advanced Identity Cloud/PingAM Login Widget, and how to migrate an existing integration past each one.

Advanced Identity Cloud/PingAM Login Widget 2.0

Advanced Identity Cloud/PingAM Login Widget 2.0 introduces breaking changes to configuration, module dependencies, and the public API. This section summarizes what changed, why it changed, and how to migrate an existing 1.x integration.

Summary of changes

- Configuration is now driven by a single async `configure()` function; the synchronous `configuration()` + `.set()` pattern is removed.
- Endpoint discovery is now driven by a single top-level `wellknown` URL that is shared by the journey and OIDC clients.
- The nested `forgerock` config object is replaced by a flat, top-level shape with an `oidcClient` sub-object.
- The `request` export, an alias to the legacy `HttpClient.request` , is removed.

- Underlying dependencies have moved: OAuth/OIDC is now backed by `@forgerock/oidc-client`, PingOne Protect is backed by `@forgerock/protect`, and `@forgerock/javascript-sdk` is no longer a dependency of the widget.

Migration checklist

1. Rename the `configuration` import to `configure`.
2. Replace calls to `configuration()` and `.set()` with a single `await configure({…})`.
3. Move endpoint discovery to a top-level `wellknown` URL. Remove `serverConfig`, `baseUrl`, `timeout`, `realmPath`, `tree`, and any `serverConfig.paths` overrides.
4. Wrap `clientId`, `redirectUri`, and `scope` in a new `oidcClient` sub-object. Add the `oidcClient` sub-object if you use OAuth/OIDC tokens, user info, or logout.
5. Remove any calls to the `request` export. Fetch tokens from `user.tokens().get()` and call `fetch` yourself.
6. Remove any of the following that appear in your configuration. They have no replacement: `oauthThreshold`, `tokenStore`, `prefix`, `logLevel`, `logger`, `platformHeader`.
7. `await` the returned promise before calling any other Widget API. `journey().start()`, `user.info().get()`, and `user.tokens().get()` all fail if called before `configure()` resolves.

Before and after

Configuration

Before (1.x)

```
import Widget, { configuration } from '@forgerock/login-widget';

const myConfig = configuration();

myConfig.set({
  forgerock: {
    serverConfig: {
      baseUrl: 'https://openam-forgerock-sdks.forgeblocks.com/am/',
      timeout: 3000,
    },
    clientId: '{edit_client_public}',
    realmPath: 'alpha',
    redirectUri: window.location.href,
    scope: '{edit_scopes}',
  },
});
```

After (2.0)

```
import '@forgerock/login-widget/widget.css';
import Widget, { configure } from '@forgerock/login-widget';

// configure() is async, so await it before any other Widget API
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    scope: 'openid profile email address',
  },
});
```

Starting a journey after configuration

Before (1.x)

```
import Widget, { configuration, journey } from '@forgerock/login-widget';

const myConfig = configuration();
myConfig.set({ forgerock: { /* ... */ } });

new Widget({ target: document.getElementById('widget-root') });

const journeyEvents = journey();
journeyEvents.start();
```

After (2.0)

```
import Widget, { configure, journey } from '@forgerock/login-widget';

await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: { /* ... */ },
});

new Widget({ target: document.getElementById('widget-root') });

journey().start();
```

Calling a protected resource

Before (1.x)

```
import { request } from '@forgerock/login-widget';

const response = await request({
  init: { method: 'GET' },
  url: 'https://protected.resource.com',
});
```

After (2.0)

```
import { user } from '@forgerock/login-widget';

const { response: tokens } = await user.tokens().get();

const response = await fetch('https://protected.resource.com', {
  method: 'GET',
  headers: {
    Authorization: `Bearer ${tokens.accessToken}`,
  },
});
```

Important

The legacy `request` alias automatically refreshed the access token on a 401 response and parsed Identity Gateway policy advice.
Neither behavior is provided by the Advanced Identity Cloud/PingAM Login Widget 2.0 or `@forgerock/oidc-client`.
If your app depends on these, you must implement them at the consumer level.

Custom endpoint paths

The following properties are all removed in 2.0. Every endpoint is now discovered from `wellknown`.

- `serverConfig.paths.authenticate`
- `serverConfig.paths.authorize`
- `serverConfig.paths.accessToken`
- `serverConfig.paths.revoke`
- `serverConfig.paths.userInfo`
- `serverConfig.paths.sessions`
- `serverConfig.paths.endSession`

If your deployment does not expose a standard `.well-known/openid-configuration` document, you must add one before upgrading to 2.0.

Removed with no replacement

The following configuration properties existed in 1.x and are removed in 2.0. None have direct replacements in the widget configuration.

Property	Notes
`request` (module export)	Use `user.tokens().get()` plus a manual `fetch` call.
`serverConfig` (nested object)	Endpoints are discovered from `wellknown`; `baseUrl` and `timeout` are gone.
`serverConfig.paths.*`	All seven per-endpoint overrides are gone.
`realmPath`, `tree`	The realm is encoded in the `wellknown` URL path. The `tree` option was inherited from the Ping Orchestration SDK for JavaScript and is not a widget concept.
`oauthThreshold`	Managed internally by `@forgerock/oidc-client`.
`tokenStore`, `prefix`	Token storage is managed internally by `@forgerock/oidc-client`.
`logLevel`, `logger`	Not currently configurable in the widget.
`platformHeader`	Not currently configurable in the widget.
`pingProtect` (as a `journey().start()` parameter)	Initialize PingOne Signals separately via `protect.start()`. Refer to Step 2. Configure the Advanced Identity Cloud/PingAM Login Widget for PingOne Protect.

Related pages

Tutorial

Updated for 2.0.

API reference

Updated for 2.0.

Login Widget changelog

The Advanced Identity Cloud/PingAM Login Widget 2.0 release entry.

Interface stability

Interfaces labelled as *Evolving* in the documentation may change without warning. In addition, the following rules apply:

- Interfaces that are not described in released product documentation should be considered *Internal/Undocumented*.

Product release levels

Ping Identity defines Major, Minor, Maintenance, and Patch product release levels. The version number reflects release level. The release level tells you what sort of compatibility changes to expect.

Release level definitions

Release Label	Version Numbers	Characteristics
Major	Version: x[.0.0] (trailing 0s are optional)	• Bring major new features, minor features, and bug fixes. • Can include changes even to Stable interfaces. • Can remove previously Deprecated functionality, and in rare cases remove Evolving functionality that has not been explicitly Deprecated. • Include changes present in previous Minor and Maintenance releases.
Minor	Version: x.y[.0] (trailing 0s are optional)	• Bring minor features, and bug fixes. • Can include backwards-compatible changes to Stable interfaces in the same Major release, and incompatible changes to Evolving interfaces. • Can remove previously Deprecated functionality. • Include changes present in previous Minor and Maintenance releases.
Maintenance, Patch	Version: x.y.z[.p] The optional *p* reflects a Patch version.	• Bring bug fixes • Are intended to be fully compatible with previous versions from the same Minor release.

Product stability labels

Ping Identity Platform software supports many features, protocols, APIs, GUIs, and command-line interfaces. Some of these are standard and very stable. Others offer new functionality that is continuing to evolve.

Ping Identity acknowledges you invest in these features and interfaces and so need to understand when they are expected to change. For that reason, we define stability labels and use these definitions in Ping Identity Platform products.

Stability label definitions

Stability Label	Definition
Stable	This documented feature or interface is expected to undergo backwards-compatible changes only for major releases. Changes may be announced at least one minor release before they take effect.

Stability Label	Definition
Evolving	This documented feature or interface is continuing to evolve and so is expected to change, potentially in backwards-incompatible ways even in a minor release. Changes are documented at the time of product release. While new protocols and APIs are still in the process of standardization, they are Evolving. This applies, for example, to recent Internet-Draft implementations and to newly developed functionality.
Legacy	This feature or interface has been replaced with an improved version, and is no longer receiving development effort from Ping Identity. You should migrate to the newer version, however the existing functionality will remain. Legacy features or interfaces will be marked as *Deprecated* if they are scheduled to be removed from the product.
Deprecated	This feature or interface is deprecated, and likely to be removed in a future release. For previously stable features or interfaces, the change was likely announced in a previous release. Deprecated features or interfaces will be removed from Ping Identity products.
Removed	This feature or interface was deprecated in a previous release, and has now been removed from the product.
Technology Preview	Technology previews provide access to new features that are considered as new technology that is not yet supported. Technology preview features may be functionally incomplete, and the function as implemented is subject to change without notice. *DO NOT DEPLOY A TECHNOLOGY PREVIEW INTO A PRODUCTION ENVIRONMENT.* Customers are encouraged to test drive the technology preview features in a non-production environment, and are welcome to make comments and suggestions about the features in the associated forums. Ping Identity does not guarantee that a technology preview feature will be present in future releases, the final complete version of the feature is liable to change between preview and the final version. Once a technology preview moves into the completed version, said feature will become part of Ping Identity Platform. Technology previews are provided on an "AS-IS" basis for evaluation purposes only, and Ping Identity accepts no liability or obligations for the use thereof.
Internal/ Undocumented	Internal and undocumented features or interfaces can change without notice. If you depend on one of these features or interfaces, contact support to discuss your needs.

Getting support

Ping Identity provides support services, professional services, training, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.pingidentity.com.

Ping Identity has staff members around the globe who support our international customers and partners. For details on Ping Identity's support offering, visit https://www.pingidentity.com/support.

Ping Identity publishes comprehensive documentation online:

- The Ping Identity Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage Ping Identity Platform software.

  While many articles are visible to everyone, Ping Identity customers have access to much more, including advanced information for customers using Ping Identity Platform software in a mission-critical capacity.

- Ping Identity product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Troubleshooting

For troubleshooting information, see the following articles in the Knowledge Base:

- Ping (ForgeRock) SDK Troubleshooting

Additional Articles

- How do I troubleshoot issues with the CORS filter in PingAM/OpenAM (All versions)?

Compatibility

Supported server versions

The Advanced Identity Cloud/PingAM Login Widget supports the following server versions:

- PingOne Advanced Identity Cloud
- PingAM 6.5, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 8.0, and later

Supported operating systems and browsers

The Advanced Identity Cloud/PingAM Login Widget supports the desktop and mobile browsers listed below.

Minimum supported Desktop browser versions

- Chrome 83
- Firefox 77
- Safari 13
- Microsoft Edge 83 (Chromium)

Supported Mobile browsers

- iOS (Safari) - Two most recent major versions of the operating system.
- Android (Chrome) - Two most recent major versions of the operating system.

JavaScript Compatibility with WebViews

A WebView allows you to embed a web browser into your native Android or iOS application to display HTML pages, and run JavaScript apps.

For example, the Android system WebView is based on the Google Chrome engine, and the iOS WebView is based on the Safari browser engine.

However, it is important to note that WebViews do not implement the full feature set of their respective browsers. For example, some of the browser-provided APIs that the Advanced Identity Cloud/PingAM Login Widget requires are not available in a WebView, such as the WebAuthn APIs.

In addition, there are concerns that a WebView does not provide the same level of security as their full browser counterparts.

As the Advanced Identity Cloud/PingAM Login Widget requires full, spec-compliant, browser-supplied APIs for full functionality we **do not** support usage within a WebView.

We also do not support or test usage with any wrappers around WebViews.

Whilst you might be able to implement simple use-cases using the Advanced Identity Cloud/PingAM Login Widget within a WebView, we recommend that you use an alternative such as opening a full browser, or using an in-app instance of a full browser such as Custom Tabs for Android or SFSafariViewController for iOS.

Supported authentication journey callbacks

The Advanced Identity Cloud/PingAM Login Widget support the following authentication journey callbacks when using the following servers:

- PingOne Advanced Identity Cloud
- PingAM

Callback name	Callback description	Supported?
`BooleanAttributeInputCallback` `SDK 2.1`	Collects true or false.	☑
`ChoiceCallback`	Collects single user input from available choices, retrieves selected choice from user interaction.	☑
`ConfirmationCallback`	Retrieve a selected option from a list of options.	☑
`ConsentMappingCallback` `SDK 2.0`	Prompts the user to consent to share their profile data.	✕
`DeviceBindingCallback`	Cryptographically bind a mobile device to a user account.	✕
`DeviceProfileCallback` `SDK 2.0`	Collects meta and/or location data about the authenticating device.	☑
`DeviceSigningVerifierCallback`	Verify ownership of a bound device by signing a challenge.	✕
`HiddenValueCallback`	Returns form values that are not visually rendered to the end user.	☑
`KbaCreateCallback` `SDK 2.0`	Collects knowledge-based answers. For example, the name of your first pet.	☑
`MetadataCallback` (1)	Injects key-value metadata into the authentication process. For example, the WebAuthn nodes use this callback to return the data the SDK requires to perform authentication and registration.	☑
`NameCallback`	Collects a username.	☑
`NumberAttributeInputCallback` `SDK 2.1`	Collects a number.	☑

Callback name	Callback description	Supported?
`PasswordCallback`	Collects a password or one-time pass code.	☑
`PingOneProtectEvaluationCallback` `SDK 4.4`	Collects captured contextual data from the client to perform risk evaluations.	☑
`PingOneProtectInitializeCallback` `SDK 4.4`	Instructs the client to start capturing contextual data for risk evaluations	☑
`PollingWaitCallback`	Instructs the client to wait for the given period and resubmit the request.	☑
`ReCaptchaCallback`	Provides data required to use a CAPTCHA in your apps.	☑
`ReCaptchaEnterpriseCallback`	Provides data required to use reCAPTCHA Enterprise in your apps.	☑ `SDK 4.6`
`RedirectCallback`	Redirects the user's browser or user-agent.	☑
`SelectIdPCallback`	Provides a list of identity providers (IdPs) users can choose from to perform social sign-on.	☑
`StringAttributeInputCallback` `SDK 2.0`	Collects the values of attributes for use elsewhere in a tree.	☑
`SuspendedTextOutputCallback` `SDK 2.1`	Pause and resume authentication, sometimes known as "magic links".	☑
`TermsAndConditionsCallback` `SDK 2.0`	Collects a user's acceptance of the configured Terms & Conditions.	☑
`TextInputCallback`	Collects text input from the end user. For example, a nickname for their account.	☑ `SDK 3.4`
`TextOutputCallback`	Provides a message to be displayed to a user with a given message type.	☑
`TextOutputCallback` *(`messageType` `===` `4`)*	Some nodes use the `TextOutputCallback` callback to include JavaScript that is intended to be run on the client. In this case the `mesageType` property equals `4`.	☑
`ValidatedCreatePasswordCallback` `SDK 2.0`	Collects a password value with optional password policy validation.	☑

Callback name	Callback description	Supported?
`ValidatedCreateUsernameCallback` `SDK 2.0`	Collects a username value with optional username policy validation.	☑

Show the nodes that might return each callback

The table below lists the nodes that might return supported callbacks.

The actual callbacks a node returns depends on its configuration. It might not return all the callbacks listed in this table.

Callback	Auth nodes that might return callback
`BooleanAttributeInputCallback`	• Attribute Collector node
`ChoiceCallback`	• Choice Collector node
`ConfirmationCallback`	• LDAP Decision node • Message node • MFA Registration Options node • OATH Token Verifier node • Polling Wait node • Push Wait node • WebAuthn Authentication node • OATH Registration node
`ConsentMappingCallback`	• Consent Collector node
`DeviceBindingCallback`	• Device Binding node
`DeviceProfileCallback`	• Device Profile Collector node
`DeviceSigningVerifierCallback`	• Device Signing Verifier node
`HiddenValueCallback`	• Amster Jwt Decision node • Push Wait node • WebAuthn Authentication node • WebAuthn Registration node

Callback	Auth nodes that might return callback
`KbaCreateCallback`	• KBA Definition node
`MetadataCallback`	• WebAuthn Authentication node • WebAuthn Registration node
`NameCallback`	• Username Collector node • Datastore Decision node • OATH Token Verifier node • Platform Username node • Configuration Provider node
`NumberAttributeInputCallback`	• Attribute Collector node
`PasswordCallback`	• Create Password node • Password Collector node • Datastore Decision node • KBA Verification node • LDAP Decision node • One-time Password Collector Decision node • Platform Password node
`PingOneProtectEvaluationCallback`	• PingOne Protect Evaluation node
`PingOneProtectInitializeCallback`	• PingOne Protect Initialization node
`PollingWaitCallback`	• Combined MFA Registration node • Push Registration node
`ReCaptchaCallback`	• CAPTCHA node • Legacy CAPTCHA node (deprecated)
`ReCaptchaEnterpriseCallback`	• reCAPTCHA Enterprise node

Callback	Auth nodes that might return callback
`RedirectCallback`	• Provision IDM Account node • Identity Assertion node • Social Provider Handler node
`SelectIdPCallback`	• Select Identity Provider node
`StringAttributeInputCallback`	• Attribute Collector node
`SuspendedTextOutputCallback`	• Email Suspend node
`TermsAndConditionsCallback`	• Accept Terms and Conditions node
`TextInputCallback`	• Configuration Provider node
`TextOutputCallback`	• Create Password node • Display Username node • LDAP Decision node • Message node • MFA Registration Options node
`TextOutputCallback (messageType == 4)`	• WebAuthn Authentication node • WebAuthn Registration node
`ValidatedCreatePasswordCallback`	• Platform Password node
`ValidatedCreateUsernameCallback`	• Platform Username node

Show the callbacks each node might return

The table below lists the supported callbacks that a node might return.

The actual callbacks a node returns depends on its configuration. It might not return all the callbacks listed in this table.

Auth node	Callbacks the node might return

Accept Terms and Conditions node	`TermsAndConditionsCallback`
Amster Jwt Decision node	`HiddenValueCallback`

Attribute Collector node	`BooleanAttributeInputCallback`
	`NumberAttributeInputCallback`
	`StringAttributeInputCallback`

CAPTCHA node	`ReCaptchaCallback`
Choice Collector node	`ChoiceCallback`
Combined MFA Registration node	`PollingWaitCallback`
Configuration Provider node	`NameCallback` `TextInputCallback`
Consent Collector node	`ConsentMappingCallback`
Create Password node	`PasswordCallback` `TextOutputCallback`
Datastore Decision node	`NameCallback` `PasswordCallback`
Device Binding node	`DeviceBindingCallback`
Device Profile Collector node	`DeviceProfileCallback`
Device Signing Verifier node	`DeviceSigningVerifierCallback`
Display Username node	`TextOutputCallback`
Email Suspend node	`SuspendedTextOutputCallback`
Identity Assertion node	`RedirectCallback`
KBA Definition node	`KbaCreateCallback`
KBA Verification node	`PasswordCallback`

LDAP Decision node	`ConfirmationCallback`
	`PasswordCallback`
	`TextOutputCallback`

Legacy CAPTCHA node (deprecated)	`ReCaptchaCallback`
Message node	`ConfirmationCallback` `TextOutputCallback`

MFA Registration Options node	`ConfirmationCallback` `TextOutputCallback`
OATH Registration node	`ConfirmationCallback`
OATH Token Verifier node	`ConfirmationCallback` `NameCallback`
One-time Password Collector Decision node	`PasswordCallback`
Password Collector node	`PasswordCallback`
PingOne Protect Evaluation node	`PingOneProtectEvaluationCallback`
PingOne Protect Initialization node	`PingOneProtectInitializeCallback`
Platform Password node	`PasswordCallback` `ValidatedCreatePasswordCallback`
Platform Username node	`NameCallback` `ValidatedCreateUsernameCallback`
Polling Wait node	`ConfirmationCallback`
Provision IDM Account node	`RedirectCallback`
Push Registration node	`PollingWaitCallback`
Push Wait node	`ConfirmationCallback` `HiddenValueCallback`
reCAPTCHA Enterprise node	`ReCaptchaEnterpriseCallback`
Select Identity Provider node	`SelectIdPCallback`
Social Provider Handler node	`RedirectCallback`
Username Collector node	`NameCallback`

WebAuthn Authentication node	`ConfirmationCallback`
	`HiddenValueCallback`
	`MetadataCallback`
	`TextOutputCallback (messageType == 4)`

WebAuthn Registration node	`HiddenValueCallback`
	`MetadataCallback`
	`TextOutputCallback (messageType == 4)`

(1) The WebAuthn Authentication node and the WebAuthn Registration node both use a `MetadataCallback` when the **Return challenge as JavaScript** is *NOT* enabled.

The Advanced Identity Cloud/PingAM Login Widget handles either the `MetadataCallback` or the JavaScript-based payload.

Tutorial

This tutorial guides you through adding the Advanced Identity Cloud/PingAM Login Widget to your application in the *modal* or *inline* form factor.

Prerequisites

You need to set up your PingOne Advanced Identity Cloud or PingAM instance with an authentication journey, and a demo user. To obtain access tokens, you also need to create an OAuth 2.0 client.

You may need to edit the CORS configuration on your server.

Server configuration

This tutorial requires you to configure one of the following servers:

PingOne Advanced Identity Cloud

PingAM

PingOne Advanced Identity Cloud

Task 1. Configure CORS

Cross-origin resource sharing (CORS) lets user agents make cross-domain server requests. In PingOne Advanced Identity Cloud, you can configure CORS to allow browsers from trusted domains to access PingOne Advanced Identity Cloud protected resources. For example, you might want a custom web application running on your own domain to get an end-user's profile information using the PingOne Advanced Identity Cloud REST API.

The Ping (ForgeRock) SDK for JavaScript samples and tutorials use `https://localhost:8443` as the host domain, which you should add to your CORS configuration.

If you are using a different domain for hosting SDK applications, ensure you add them to the CORS configuration as accepted origin domains.

To update the CORS configuration in PingOne Advanced Identity Cloud, follow these steps:

1. Log in to your PingOne Advanced Identity Cloud tenant.
2. At the top right of the screen, click your name, and then select **Tenant settings**.
3. On the **Global Settings** tab, click **Cross-Origin Resource Sharing (CORS)**.
4. Perform one of the following actions:
    - If available, click **ForgeRockSDK**.
    - If you haven't added any CORS configurations to the tenant, click **+ Add a CORS Configuration**, select **Ping (ForgeRock) SDK**, and then click **Next**.

5. Add `https://localhost:8443` and any DNS aliases you use to host your Ping (ForgeRock) SDK for JavaScript applications to the **Accepted Origins** property.
6. Complete the remaining fields to suit your environment.

   This documentation assumes the following configuration, required for the tutorials and sample applications:

Property	Values
`Accepted Origins`	`https://localhost:8443`
`Accepted Methods`	`GET` `POST`

`Accepted Headers`	`accept-api-version`
	`x-requested-with`
	`content-type`
	`authorization`
	`if-match`
	`x-requested-platform`
	`iPlanetDirectoryPro` [1]
	`ch15fefc5407912` [2]

`Exposed Headers`	`authorization` `content-type`
`Enable Caching`	`True`
`Max Age`	`600`
`Allow Credentials`	`True`

> **Tip**
>
> Click **Show advanced settings** to be able to edit all available fields.

7. Click **Save CORS Configuration**.

Task 2. Create a demo user

The samples and tutorials in this documentation often require that you have an identity set up so that you can test authentication.

To create a demo user in PingOne Advanced Identity Cloud, follow these steps:

1. Log in to your PingOne Advanced Identity Cloud tenant.
2. In the left panel, click **Identities** > **Manage**.
3. Click **+ New Alpha realm - User**.

4. Enter the following details:
    - **Username** = `demo`
    - **First Name** = `Demo`
    - **Last Name** = `User`
    - **Email Address** = `demo.user@example.com`
    - **Password** = `Ch4ng3it!`
5. Click **Save**.

Task 3. Create an authentication journey

Authentication journeys provide fine-grained authentication by allowing multiple paths and decision points throughout the flow. Authentication journeys are made up of nodes that define actions taken during authentication.

Each node performs a single task, such as collecting a username or making a simple decision. Nodes can have multiple outcomes rather than just success or failure.

To create a simple journey for use when testing the Ping (ForgeRock) SDKs, follow these steps:

1. In your PingOne Advanced Identity Cloud tenant, navigate to **Journeys**, and click **+ New Journey**.
2. Enter a name, such as `sdkUsernamePasswordJourney` and click **Save**.

   The authentication journey designer appears.
3. Drag the following nodes into the designer area:
    - **Page Node**
    - **Platform Username**
    - **Platform Password**
    - **Data Store Decision**
4. Drag and drop the **Platform Username** and **Platform Password** nodes onto the **Page Node**, so that they both appear on the same page when logging in.
5. Connect the nodes as follows:

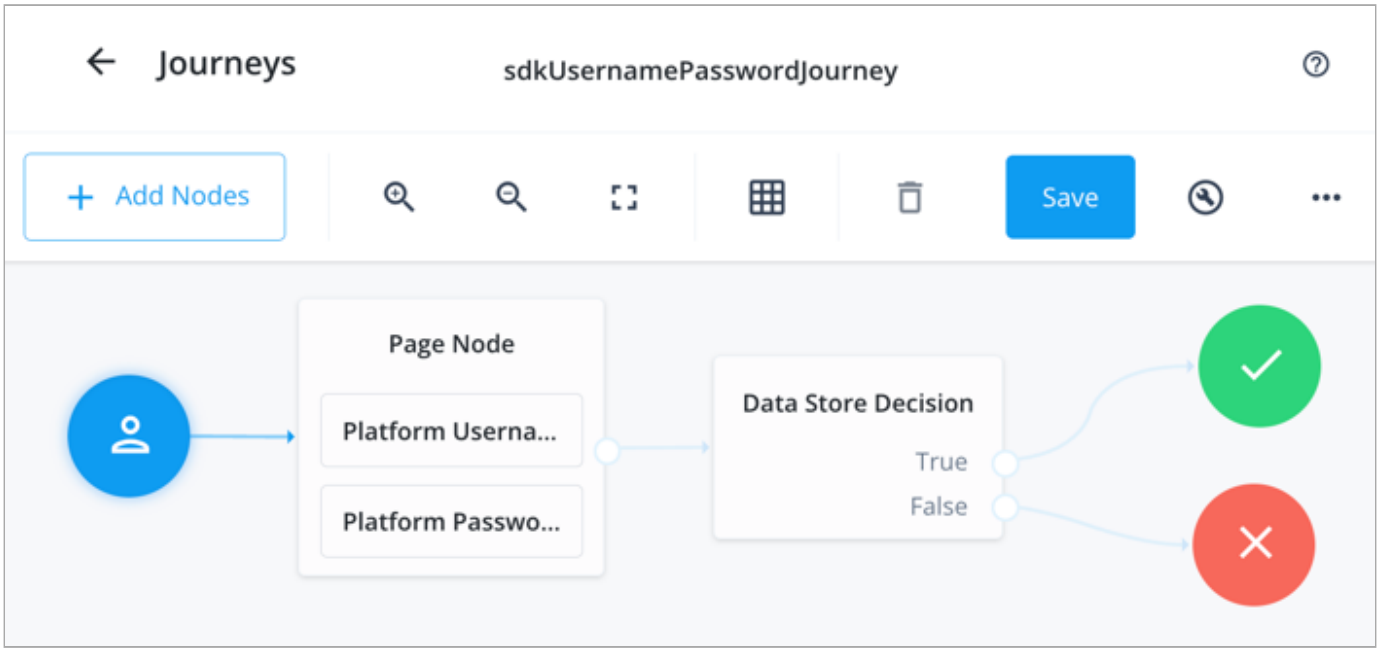


Figure 1. Example username and password authentication journey

6. Click **Save**.

Task 4. Register a public OAuth 2.0 client

Public clients do not use a client secret to obtain tokens because they are unable to keep them hidden. The Ping (ForgeRock) SDKs commonly use this type of client to obtain tokens, as they cannot guarantee safekeeping of the client credentials in a browser or on a mobile device.

To register a *public* OAuth 2.0 client application for use with the SDKs in PingOne Advanced Identity Cloud, follow these steps:

1. Log in to your PingOne Advanced Identity Cloud tenant.
2. In the left panel, click **Applications**.
3. Click **+ Custom Application**.
4. Select **OIDC - OpenId Connect** as the sign-in method, and then click **Next**.
5. Select **Native / SPA** as the application type, and then click **Next**.
6. In **Name**, enter a name for the application, such as `Public SDK Client`.
7. In **Owners**, select a user that is responsible for maintaining the application, and then click **Next**.

> **Tip**
>
> When trying out the SDKs, you could select the `demo` user you created previously.

8. In **Client ID**, enter `sdkPublicClient`, and then click **Create Application**.

   PingOne Advanced Identity Cloud creates the application and displays the details screen.

9. On the **Sign On** tab:
    1. In **Sign-In URLs**, enter the following values:

        > **Important**
        >
        > Also add any other domains where you host SDK applications.

    2. In **Grant Types**, enter the following values:

        `Authorization Code`

        `Refresh Token`

    3. In **Scopes**, enter the following values:

        `openid profile email address`

10. Click Show advanced settings, and on the **Authentication** tab:
    1. In **Token Endpoint Authentication Method**, select `none`.
    2. In **Client Type**, select `Public`.
    3. Enable the **Implied Consent** property.
11. Click **Save**.

The application is now configured to accept client connections from and issue OAuth 2.0 tokens to the example applications and tutorials covered by this documentation.

Task 5. Configure the OAuth 2.0 provider service

The provider specifies the supported OAuth 2.0 configuration options for a realm.

To ensure the PingOne Advanced Identity Cloud OAuth 2.0 provider service is configured for use with the Ping (ForgeRock) SDKs, follow these steps:

1. In your PingOne Advanced Identity Cloud tenant, navigate to **Native Consoles** > **Access Management**.
2. In the left panel, click **Services**.
3. In the list of services, click **OAuth2 Provider**.
4. On the **Core** tab, ensure **Issue Refresh Tokens** is enabled.
5. On the **Consent** tab, ensure **Allow Clients to Skip Consent** is enabled.
6. Click **Save Changes**.

PingAM

Task 1. Configure CORS

Cross-origin resource sharing (CORS) lets user agents make cross-domain server requests. In PingAM, you can configure CORS to allow browsers from trusted domains to access PingAM protected resources. For example, you might want a custom web application running on your own domain to get an end-user's profile information using the PingAM REST API.

The Ping (ForgeRock) SDK for JavaScript samples and tutorials all use `https://localhost:8443` as the host domain, which you should add to your CORS configuration.

If you are using a different URL for hosting SDK applications, ensure you add them to the CORS configuration as accepted origin domains.

To enable CORS in PingAM, and create a CORS filter to allow requests from your configured domain names, follow these steps:

1. Log in to the PingAM admin UI as an administrator.
2. Navigate to **Configure** > **Global Services** > **CORS Service** > **Configuration**, and set the **Enable the CORS filter** property to `true`.

   > **Important**
   >
   > If this property is not enabled, CORS headers are not added to responses from PingAM, and CORS is disabled entirely.

3. On the **Secondary Configurations** tab, click **Click Add a Secondary Configuration**.
4. In the **Name** field, enter `ForgeRockSDK`.
5. in the **Accepted Origins** field, enter any DNS aliases you use for your SDK apps.

   This documentation assumes the following configuration:

Property	Values
`Accepted Origins`	`https://localhost:8443`
`Accepted Methods`	`GET` `POST`

`Accepted Headers`	`accept-api-version`
	`x-requested-with`
	`content-type`
	`authorization`
	`if-match`
	`x-requested-platform`
	`iPlanetDirectoryPro` [1]
	`ch15fefc5407912` [2]

`Exposed Headers`	`authorization` `content-type`

6. Click **Create**.

   PingAM displays the configuration of your new CORS filter.

7. On the CORS filter configuration page:

   1. Ensure **Enable the CORS filter** is enabled.

2. Set the **Max Age** property to `600`
3. Ensure **Allow Credentials** is enabled.

8. Click **Save Changes**.

Task 2. Create a demo user

The samples and tutorials in this documentation often require that you have an identity set up so that you can test authentication.

To create a demo user in PingAM, follow these steps:

1. Log in to the PingAM admin UI as an administrator.
2. Navigate to **Identities**, and then click **+ Add Identity**.
3. Enter the following details:
   - **User ID** = `demo`
   - **Password** = `Ch4ng3it!`
   - **Email Address** = `demo.user@example.com`
4. Click **Create**.

Task 2. Create an authentication tree

Authentication trees provide fine-grained authentication by allowing multiple paths and decision points throughout the authentication flow. Authentication trees are made up of nodes that define actions taken during authentication.

Each node performs a single task, such as collecting a username or making a simple decision. Nodes can have multiple outcomes rather than just success or failure.

To create a simple tree for use when testing the Ping (ForgeRock) SDKs, follow these steps:

1. Under **Realm Overview**, click **Authentication Trees**, then click **Create Tree**.
2. Enter a tree name, for example `sdkUsernamePasswordJourney`, and then click **Create**.

   The authentication tree designer appears, showing the **Start** entry point connected to the **Failure** exit point.
3. Drag the following nodes from the **Components** panel on the left side into the designer area:
   - **Page Node**
   - **Username Collector**
   - **Password Collector**
   - **Data Store Decision**
4. Drag and drop the **Username Collector** and **Password Collector** nodes onto the **Page Node**, so that they both appear on the same page when logging in.
5. Connect the nodes as follows:

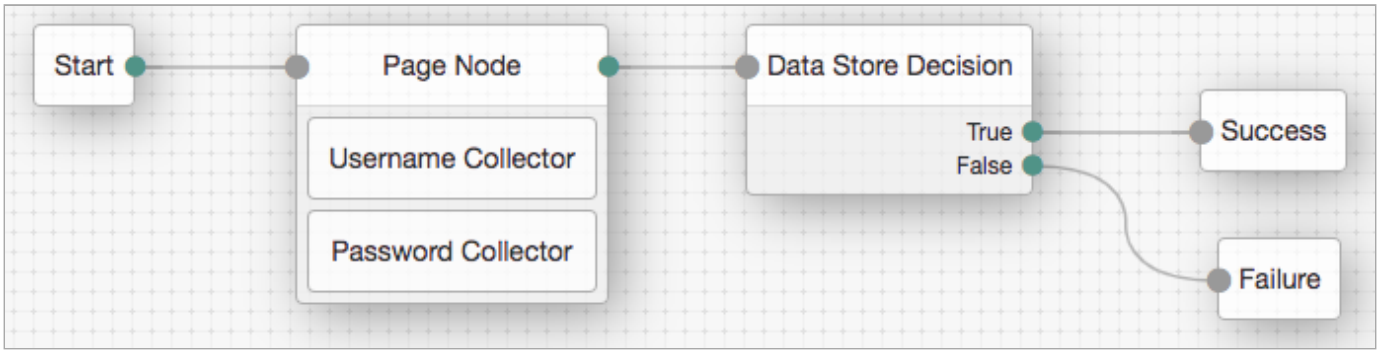


Figure 2. Example username and password authentication tree

6. Select the **Page Node**, and in the **Properties** pane, set the **Stage** property to `UsernamePassword`.

> **Tip**
>
> You can configure the node properties by selecting a node and altering properties in the right-hand panel.

One of the samples uses this specific value to determine the custom UI to display.

7. Click **Save**.

Task 3. Register a public OAuth 2.0 client

Public clients do not use a client secret to obtain tokens because they are unable to keep them hidden. The Ping (ForgeRock) SDKs commonly use this type of client to obtain tokens, as they cannot guarantee safekeeping of the client credentials in a browser or on a mobile device.

To register a *public* OAuth 2.0 client application for use with the SDKs in AM, follow these steps:

1. Log in to the PingAM admin UI as an administrator.
2. Navigate to **Applications** > **OAuth 2.0** > **Clients**, and then click **+ Add Client**.
3. In **Client ID**, enter `sdkPublicClient`.
4. Leave **Client secret** empty.
5. In **Redirection URIs**, enter the following values:

> **Important**
>
> Also add any other domains where you will be hosting SDK applications.

6. In **Scopes**, enter the following values:

```
openid profile email address
```

7. Click **Create**.

PingAM creates the new OAuth 2.0 client, and displays the properties for further configuration.

8. On the **Core** tab:

    1. In **Client type**, select `Public`.

2. Disable **Allow wildcard ports in redirect URIs**.
3. Click **Save Changes**.

9. On the **Advanced** tab:
    1. In **Grant Types**, enter the following values:

    ```
    Authorization Code
    Refresh Token
    ```

    2. In **Token Endpoint Authentication Method**, select `None`.
    3. Enable the **Implied consent** property.
10. Click **Save Changes**.

Task 4. Configure the OAuth 2.0 provider service

The provider specifies the supported OAuth 2.0 configuration options for a realm.

To ensure the PingAM OAuth 2.0 provider service is configured for use with the Ping (ForgeRock) SDKs, follow these steps:

1. Log in to the PingAM admin UI as an administrator.
2. In the left panel, click **Services**.
3. In the list of services, click **OAuth2 Provider**.
4. On the **Core** tab, ensure **Issue Refresh Tokens** is enabled.
5. On the **Consent** tab, ensure **Allow Clients to Skip Consent** is enabled.
6. Click **Save Changes**.

Steps

Step 1. Install the widget

In this step, you use `npm` to add the Advanced Identity Cloud/PingAM Login Widget to your project. It also covers how to download and build the Advanced Identity Cloud/PingAM Login Widget to support custom requirements.

Step 2. Configure the CSS

In this step, you add the default CSS to your app, and learn how to use layers to control the CSS cascade.

Step 3. Import the widget

In this step, you import the modules from the Advanced Identity Cloud/PingAM Login Widget you want to use in your app.

Step 4. Configure the widget

In this step, you provide the configuration necessary for the Advanced Identity Cloud/PingAM Login Widget to contact your server. You pass a well-known URI, and the widget derives all endpoints from it.

Step 5. Instantiate the widget

In this step, you choose where in your app to mount the Advanced Identity Cloud/PingAM Login Widget, and then instantiate an instance, choosing either the inline or modal form factor.

Step 6. Start a journey

In ths step, you start a journey so that the Advanced Identity Cloud/PingAM Login Widget can display the UI for the first callback.

Step 7. Subscribe to events

In this step, you subscribe to *observables* to capture and react to events that occur during use of the Advanced Identity Cloud/PingAM Login Widget.

---

1. Cookie name value in PingAM servers.
2. In PingOne Advanced Identity Cloud tenants, go to **Tenant Settings > Global Settings > Cookie** to find this dynamic cookie name value.

Step 1. Install the widget

You can add the Advanced Identity Cloud/PingAM Login Widget to your app by using npm, or you can download it from GitHub and build it yourself, adding results to your project directly.

Install the Advanced Identity Cloud/PingAM Login Widget with npm

The easiest way to add the Advanced Identity Cloud/PingAM Login Widget to your project.

Build a customized Advanced Identity Cloud/PingAM Login Widget

If you want to customize the themes included in the Advanced Identity Cloud/PingAM Login Widget, you need to download the Advanced Identity Cloud/PingAM Web Login Framework source, make your modifications, and build a customized package.

Install the Advanced Identity Cloud/PingAM Login Widget with npm

Add the Advanced Identity Cloud/PingAM Login Widget to your project using npm as follows:

```
npm install @forgerock/login-widget
```

Next, you can Step 2. Configure the CSS.

Build a customized Advanced Identity Cloud/PingAM Login Widget

The following steps show how to download the Advanced Identity Cloud/PingAM Web Login Framework and build the Advanced Identity Cloud/PingAM Login Widget:

1. Download the Advanced Identity Cloud/PingAM Web Login Framework from the Git repository:

   ```
   git clone https://github.com/ForgeRock/forgerock-web-login-framework.git
   ```

2. In a terminal window, navigate to the root of the Advanced Identity Cloud/PingAM Web Login Framework:

```
cd forgerock-web-login-framework
```

3. Run `pnpm` to download and install the required packages and modules:

   ```
   pnpm install
   ```

4. Build the Advanced Identity Cloud/PingAM Login Widget with `pnpm`:

   ```
   pnpm run build:widget
   ```

5. Copy the built `dist/` directory into your app project

6. Import the `Widget` component into your app:

   ```
   import Widget from '../path/to/dist/index.js';
   ```

   > **Note**
   >
   > The exact syntax for importing the widget into your app varies depending on the technologies your app uses.

Next

Next, you can Step 2. Configure the CSS.

Step 2. Configure the CSS

You can use any of the following methods to add the default Advanced Identity Cloud/PingAM Login Widget styles to your app:

1. Import it into your JavaScript project as a module.
2. Import it using a CSS preprocessor, like Sass, Less, or PostCSS.

If you decide to import the CSS into your JavaScript, make sure your bundler is able to import and process the CSS as a module. If using a CSS preprocessor, configure your preprocessor to access files from within your package or from a directory.

Examples

Import the CSS into your JavaScript:

npm

```
// app.js
import '@forgerock/login-widget/widget.css';
```

Local

```
// app.js
import '../path/to/widget.css';
```

Import the CSS into your CSS:

npm

```
/* style.css */
@import '@forgerock/login-widget/widget.css';
```

Local

```
/* style.css */
@import '../path/to/widget.css';
```

Controlling the CSS cascade

How the browser applies styles to an app can depend on the order you import or declare CSS into it, referred to as the *cascade*.

You can use the `@layer` CSS rule to declare a cascade layer, ensuring Advanced Identity Cloud/PingAM Login Widget styles apply separately from your own.

For more information, refer to `@layer` in the MDN docs.

> **Note**
>
> The Advanced Identity Cloud/PingAM Login Widget styles will not overwrite any of your CSS.
> They are namespaced to help prevent collisions and use a CSS selector prefix of `tw_`.

To create a cascade layer for the Advanced Identity Cloud/PingAM Login Widget styles:

1. Wrap your existing styles in a new layer, for example, "app":

```
@layer app {
    /* Your app's existing CSS */
}
```

2. Declare the order of layers in your index HTML file before loading any CSS.

> **Note**
>
> The Widget has multiple `@layer` declarations in its CSS files

```
<style type="text/css">
  /* Your existing "app" CSS layer first */
  @layer app;

  /* List the Widget layers after your own styles */
  @layer 'fr-widget.base';
  @layer 'fr-widget.utilities';
  @layer 'fr-widget.components';
  @layer 'fr-widget.variants';
</style>
```

> **Note**
>
> The CSS imported for the Widget will not overwrite any of your app's CSS. It's all namespaced to ensure there are no collisions. To help achieve this, the Advanced Identity Cloud/PingAM Login Widget uses a selector naming convention with a `tw_` prefix.

Next

Next, you can Step 3. Import the widget

Step 3. Import the widget

To use the Advanced Identity Cloud/PingAM Login Widget, import the modules you want to use into your app:

```
// Import the Login Widget
import Widget, { configure } from '@forgerock/login-widget';
```

The exact syntax for importing the widget depends on the module system you are using.

The Advanced Identity Cloud/PingAM Login Widget exports a number of different modules, each providing different functionality.

Advanced Identity Cloud/PingAM Login Widget modules

Module	Description	API reference
`Widget`	Use this main class to instantiate the Advanced Identity Cloud/PingAM Login Widget, mount it into the DOM, and set up event listeners.	Widget API reference

Module	Description	API reference
`configure`	Use this async function to configure the Advanced Identity Cloud/PingAM Login Widget. You call it once with a well-known URI — from which the widget derives all server endpoints — plus OIDC client settings and any optional style, content, or link overrides.	Configuration API reference
`journey`	Use this module to configure and start an authentication journey.	Journey API reference
`component`	Use this module to subscribe to events triggered by the Advanced Identity Cloud/PingAM Login Widget and for controlling the modal form factor.	Component API reference
`user`	Use this module for managing users in the Advanced Identity Cloud/ PingAM Login Widget, such as obtaining user or token information, and logging users out.	User API reference
`protect`	Use this module to manually control the PingOne Signals SDK for risk evaluations.	Protect API reference

Next

Next, you can Step 4. Configure the widget.

Step 4. Configure the widget

The Advanced Identity Cloud/PingAM Login Widget needs the URL of your server's `.well-known/openid-configuration` endpoint. If you use OAuth/OIDC tokens, user info, or logout, it also needs an OIDC client configuration.

To provide these settings, import and `await` the async `configure()` function:

Example Advanced Identity Cloud/PingAM Login Widget configuration

```
// Import the modules
import Widget, { configure } from '@forgerock/login-widget';

// configure() is async, so await it before calling any other Widget API
await configure({
  // REQUIRED; the well-known URL, shared by the journey and OIDC clients
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  // REQUIRED if you use OAuth/OIDC tokens, user info, or logout
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    // OPTIONAL; defaults to 'openid'
    scope: 'openid profile email address',
  },
});
```

> **Tip**
>
> Call `configure()` once at the top level of your application, such as its `index.js` or `app.js` file, and `await` the result before any other Widget API.
> The Advanced Identity Cloud/PingAM Login Widget constructs its internal clients during this call. Any Widget API, including `journey().start()`, `user.info().get()`, or `user.tokens().get()`, throws an error if it runs before `configure()` resolves.

Widget configuration properties

The properties accepted by `configure()` are as follows:

Server

Properties

Property	Description

`wellknown`	**Required.** The full URL to your server's `.well-known/openid-configuration` endpoint.
	The Advanced Identity Cloud/PingAM Login Widget uses this single URL to discover the endpoints for both the journey client and the OIDC client. The realm is encoded in the path.
	PingOne Advanced Identity Cloud example:
	`https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-configuration`
	Self-hosted example:
	`https://openam.example.com:8443/openam/oauth2/realms/root/.well-known/openid-configuration`

OAuth 2.0

The `oidcClient` object is required if you use OAuth/OIDC tokens, user info, or logout.

Properties

Property	Description
`oidcClient: { clientId }`	**Required within `oidcClient`.** The `client_id` of the OAuth 2.0 client profile to use.

Property	Description
`oidcClient: { redirectUri }`	**Required within `oidcClient`.** The `redirect_uri` as configured in the OAuth 2.0 client profile. Must be a full URL. **Tip** The Ping (ForgeRock) SDK for JavaScript attempts to load the redirect page to capture the OAuth 2.0 `code` and `state` query parameters that the server appended to the redirect URL. If the page you redirect to does not exist, takes a long time to load, or runs any JavaScript you might get a timeout, delayed authentication, or unexpected errors. To ensure the best user experience, we ***highly recommend*** that you redirect to a static HTML page with minimal HTML and no JavaScript when obtaining OAuth 2.0 tokens. For example, `https://localhost:8443/callback.html`.
`oidcClient: { scope }`	**Optional within `oidcClient`.** A list of scopes to request when performing an OAuth 2.0 authorization flow, separated by spaces. For example, `openid profile email address`. If not provided, the default value is `openid`.

Optional configuration objects

Properties

Property	Description
`captcha`	Configure CAPTCHA rendering. Currently supports `mode: 'visible'` (default) or `mode: 'invisible'`. Refer to Implement a CAPTCHA.
`content`	Override the widget's default text content, or provide localized strings. Refer to Localizing the widget.
`journeys`	Map HREF values rendered by the widget to journeys so that clicking a link starts a journey instead of navigating. Refer to Journeys configuration options.
`links`	Set the full canonical URL to your terms and conditions page. Refer to Links configuration options.
`style`	Configure the widget's visual appearance, including labels, logos, stage icons, password visibility, and theme colors and fonts. Refer to Style configuration options.

Next

Next, you can Step 5. Instantiate the widget.

Step 5. Instantiate the widget

To use the Advanced Identity Cloud/PingAM Login Widget in your app you must choose an appropriate place to mount it. Then, you need to choose which form factor to implement, either inline, or modal.

With those decisions made, you can instantiate the Advanced Identity Cloud/PingAM Login Widget in your app, ready for your users to start their authentication or self-service journey.

Choose where to mount the Advanced Identity Cloud/PingAM Login Widget

To implement the Advanced Identity Cloud/PingAM Login Widget, we recommend you add a new element into your HTML file.

For most single page applications (SPA) this is your `index.html` file.

This new element should be a direct child element of `<body>` and not within the element where you mount your SPA.

Example HTML structure

```
<!DOCTYPE html>
<html lang="en">
<head>
<!-- ... -->
</head>
<body>
    <!-- Root element for main app -->
    <div id="root"></div>

    <!-- Root element for Widget -->
    <div id="widget-root"></div>

    <!-- scripts ... -->
  </body>
</html>
```

Tip

We recommend that you do not inject the element into which you mount the modal form factor in your app. This can cause virtual DOM issues.
Instead, manually hard-code the element in your HTML file.

Instantiate the modal form factor

To use the default Advanced Identity Cloud/PingAM Login Widget modal form factor, import the modules into your app and instantiate the widget as follows:

Instantiate the modal form factor

```
// Import the Login Widget
import Widget, { configure } from '@forgerock/login-widget';

// configure() is async, so await it before instantiating the widget or calling any other API
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    scope: 'openid profile email address',
  },
});

// Get the element in your HTML into which you will mount the widget
const widgetRootEl = document.getElementById('widget-root');

// Instantiate Widget with the `new` keyword
new Widget({
  target: widgetRootEl,
});
```

This mounts the Advanced Identity Cloud/PingAM Login Widget into the DOM. The modal form factor is the default and is hidden when first instantiated.

To open the modal, import the `component` module, assign the function, and call its `open()` method:

Open the modal

```
// Import the Login Widget
import Widget, { configure, component } from '@forgerock/login-widget';

await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    scope: 'openid profile email address',
  },
});

// Get the element in your HTML into which you will mount the widget
const widgetRootEl = document.getElementById('widget-root');

// Instantiate Widget with the `new` keyword
new Widget({
  target: widgetRootEl, // Any existing element from static HTML file
});

// Assign the component function
const componentEvents = component();

// Call the open() method, for example after a button click
const loginButton = document.getElementById('loginButton');

loginButton.addEventListener('click', () => {
  componentEvents.open();
});
```

The modal form factor opens and displays a spinner graphic until you start a journey.

> **Tip**
>
> The modal form factor closes itself when a journey completes successfully.
> You can also close it by calling `componentEvents.close();`

Instantiate the inline form factor

You override the default Advanced Identity Cloud/PingAM Login Widget modal form factor and instead use the inline form factor. The inline form factor mounts within your application's controlled DOM, so it is important to consider how your framework mounts elements to the DOM.

For example, the inline form factor cannot mount into a virtual DOM element, such as those used by React. In this scenario, you must wait until the element has been property mounted to the real DOM before instantiating the Advanced Identity Cloud/PingAM Login Widget.

To use the inline form factor, instantiate the widget with a `type: 'inline'` property, as follows:

Instantiate the inline form factor

```
// Import the Advanced Identity Cloud/PingAM Login Widget
import Widget, { configure } from '@forgerock/login-widget';

import { useRef } from 'react';

await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    scope: 'openid profile email address',
  },
});

// Target needs to be an actual DOM element, so ref is needed with inline type
const widgetElement = useRef(null);

// Instantiate Widget with the `new` keyword
new Widget({
  target: widgetElement.current,
  props: {
    type: 'inline', // Override the default 'modal' form factor
  },
});
```

The inline form factor loads into the specified DOM element and displays a spinner graphic until you start a journey.

Next

Next, you can Step 6. Start a journey.

Step 6. Start a journey

The Advanced Identity Cloud/PingAM Login Widget displays a loading spinner graphic if it does not yet have a callback from the server to render.

You must specify and start a journey to make the initial call to the server and obtain the first callback.

To start a journey, import the `journey` function and execute it to receive a `journeyEvents` object. After you have this `journeyEvents` object, you can call the `journeyEvents.start()` method, which starts making requests to the server for the initial form fields.

You can call the `journeyEvents.start()` method anywhere in your application, or anytime, as long as it is after `await configure()` has resolved and the Widget is instantiated.

Start the default journey

```
// Import the Login Widget
import Widget, { configure, journey } from '@forgerock/login-widget';

// Configure the widget; always await this before any other Widget API
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    scope: 'openid profile email address',
  },
});

// Get the element in your HTML into which you will mount the widget
const widgetRootEl = document.getElementById('widget-root');

// Instantiate Widget with the `new` keyword
new Widget({
  target: widgetRootEl,
});

// Assign the journey function
const journeyEvents = journey();

// Ensure you call `.start` *AFTER* instantiating the Widget
journeyEvents.start();
```

This starts the journey configured as the default in your server and renders the initial callback.

To specify which journey to use and other parameters, refer to Configure journey() parameters and Configure start() parameters.

Configure journey() parameters

If you do not pass any parameters when calling the `journey()` function the Advanced Identity Cloud/PingAM Login Widget will attempt to retrieve OAuth 2.0 tokens and user information by default.

You can override this behavior by passing in the following JSON parameters:

Optional `journey()` *parameters*

Parameter	Description
`oauth`	Set to `false` to prevent the Advanced Identity Cloud/PingAM Login Widget attempting to obtain OAuth 2.0 tokens after successfully completing a journey. The default is `true`.

Parameter	Description
`user`	Set to `false` to prevent the Advanced Identity Cloud/PingAM Login Widget attempting to obtain user information by calling the `/oauth2/userinfo` endpoint after successfully completing a journey. The default is `true`.

Example - obtaining only a user session token:

```
const journeyEvents = journey({
    oauth: false,
    user: false,
});
```

For more information, refer to journey API reference

Configure start() parameters

If you do not pass any parameters when calling the `start()` method the Advanced Identity Cloud/PingAM Login Widget will use whichever journey is marked as the default in your server.

You can override this behavior by passing in JSON parameters:

Optional `start()` *parameters*

Parameter	Description
`journey`	Specify the name of the journey to use. If not specified, the Advanced Identity Cloud/PingAM Login Widget uses whichever journey is marked as the default.
`query`	An object of additional query parameters to include when starting the journey.
`resumeUrl`	Specify the full URL to visit if resuming a suspended journey, for example `window.location.href`. The server uses this to return your users to your application after clicking a "magic link" in an email, for example. Omit this property when starting a new journey.
`recaptchaAction`	Tag reCAPTCHA v3 or reCAPTCHA Enterprise events with a custom action string. Falls back to the journey name.

Example of specifying the journey to use:

```
// Specify a different journey
journeyEvents.start({
  journey: 'sdkRegistrationJourney',
});
```

For more information, refer to journey in the API reference.

Listen for journey completion

Use the `journeyEvents.subscribe` method to know when a user has completed their journey.

A summary of the events for a journey and their order is as follow:

1. Journey is loading
2. Journey is complete
3. Tokens are loading
4. Tokens are complete
5. Userinfo is loading
6. Userinfo is complete

Pass a callback function into this method to run on journey related events, of which there will be many, and each event object you receive contains a lot of information about the event.

You conditionally check for the events you are interested in and ignore what you do not need.

Example - subscribe to journey events

```
journeyEvents.subscribe((event) => {
  // Called multiple times, filtering by event data is recommended
  if (event.journey.successful) {
    // Only output successfull journey log entries
    console.log(event);
  }
});
```

Handle journey errors

When authentication fails, `event.journey.error` is set to an object describing the failure. Check for this alongside the success condition:

Example - handle journey success and failure

```
journeyEvents.subscribe((event) => {
  if (event.journey.successful) {
    console.log('Authentication succeeded');
  } else if (event.journey.error) {
    console.error('Authentication failed:', event.journey.error.message);
  }
});
```

Redirect to a failure URL

If your journey includes the Failure URL node, the server returns a `failureUrl` value inside the `detail` property of the error object. You can read this value and redirect the user to that URL:

Example - redirect to failure URL

```
journeyEvents.subscribe((event) => {
  if (event.journey.error) {
    const failureUrl = event.journey.error.detail?.failureUrl;

    if (failureUrl) {
      window.location.assign(failureUrl);
    }
  }
});
```

> **Note**
>
> The `detail` property is only present when the server includes it in the failure response.
> Always access it with optional chaining (`detail?.failureUrl`) as it may not be present for all failure types.

Next

Next, you can learn more information about observables and how to Step 7. Subscribe to events.

Step 7. Subscribe to events

The Advanced Identity Cloud/PingAM Login Widget has a number of asynchronous APIs, which are designed around an event-centric *observable* pattern. It uses Svelte's simplified, standard observable implementation called a "store".

> **Note**
>
> These Svelte stores are embedded into the Advanced Identity Cloud/PingAM Login Widget itself.
> They are not a dependency that your app layer needs to import or manage.

For more information on Svelte stores, refer to the Svelte documentation.

This observable pattern is optimal for UI development as it allows for a dynamic user experience. You can update your application in response to the events occurring within the Advanced Identity Cloud/PingAM Login Widget. For example, the Advanced Identity Cloud/PingAM Login Widget has events such as "loading", "completed", "success", and, "failure".

Assign an observable

You can create a variable and assign the observable to it:

Assign an observable

```
const userInfoEvents = user.info();
```

Subscribe to observable events

An observable is a stream of events over time. The Advanced Identity Cloud/PingAM Login Widget invokes the callback for each and every event from the observable, until you unsubscribe from it.

Use the `subscribe()` method on your variable to observe the event stream:

Example userInfoEvents observable

```
userInfoEvents.subscribe((event) => {
  if (event.loading) {
    console.log('User info is being requested from server');
  } else if (event.successful) {
    console.log('User info request was successful');
    console.log(event.response);
  } else if (event.error) {
    console.error('User info request failed');
    console.error(event.error.message);
  }
});
```

For information on the events each observable returns, refer to the API reference.

Unsubscribe from an observable

Unlike a JavaScript *promise*, an observable does not resolve and then get cleaned up after completion.

You need to unsubscribe from an observable if it is no longer needed. This is especially important if you are subscribing to observables in a component that gets created and destroyed many times over. Subscribing to an observable over and over without unsubscribing creates a memory leak.

To unsubscribe, assign the function that is returned from calling the `subscribe()` method to a variable. Call this variable at a later time to unsubscribe from the obeservable.

Example unsubscribe from an observable

```
const unsubUserInfoEvents = userInfoEvents.subscribe((event) => console.log(event));

// ...

// Unsubscribe when no longer needed
unsubUserInfoEvents();
```

You *do not need* to unsubscribe from observables if you subscribe to observables in a top-level component of your app that is only initiated once, and is retained over the lifetime of your application.

A good location in which to subscribe to observables might be the central state management component or module of your application.

Get current local values

The Advanced Identity Cloud/PingAM Login Widget stores a number of important values internally.

You can get the current values stored within the Advanced Identity Cloud/PingAM Login Widget without subscribing to any future events or their resulting state changes by calling `subscribe()` and then immediately calling its unsubscribe method:

Get current stored values and unsubscribe

```
// Create variable for user info
let userInfo;

// Call subscribe, get the current local value, and then immediately call the returned function
userInfoEvents.subscribe((event) => (userinfo = event.response))(); // <-- notice the second pair of parentheses
```

Get updated values

You can ask the Advanced Identity Cloud/PingAM Login Widget to request new, fresh values from the server, rather than just what it has stored locally, by calling the observable action methods, such as `get`.

Get latest values from the server

```
userInfoEvents.get();
```

When using the observable pattern, you can call this method and forget about it. The get causes any `subscribe` callback functions you have for the observable to receive the events and new state.

The `subscribe` can exist before or after this `get` call, and it will still capture the resulting events.

Use promises rather than observables

We recommend observables, but the choice is up to you.

All of the Advanced Identity Cloud/PingAM Login Widget APIs that involve network calls have an alternative promise implementation that you can use.

The following example again shows `userInfoEvents` but converted to use promises:

Using promises rather than observables

```
// async-await
let userInfo;
async function example() {
  try {
    userInfo = await userInfoEvents.get();
  } catch (err) {
    console.log(err);
  }
}

// Promise
let userInfo;
userInfoEvents
  .get()
  .then((data) => (userInfo = data))
  .catch((err) => console.log(err));
```

Customizing the Advanced Identity Cloud/ PingAM Login Widget

The **Advanced Identity Cloud/PingAM Login Widget** is designed to be flexible and can be customized to suit many different situations.

Learn more about customizing the widget in the sections below:

Localization

Alter the default text that appears in the Advanced Identity Cloud/PingAM Login Widget, including how to respond to different locales.

Read more »

Theming

Learn how to alter the default light and dark themes by using the Tailwind configuration file.

Read more »

Localizing the widget

The Advanced Identity Cloud/PingAM Login Widget uses variables with default values for all strings it displays when rendering the user interface.

Show variable names and default values

Default localization values

Variable	Default text
`adminRegInvalidInviteHeader`	Invitation not valid
`adminRegInvalidInviteDescription`	This invitation can't be used. Please contact your administrator or Ping Support for help.
`adminRegOtpHeader`	Admin Invite Code
`adminRegOtpDescription`	We've sent a 6-digit code to your email. Enter it below to continue.

Variable	Default text
`adminRegOtpError`	That code isn’t correct or may have expired. Please try again.
`adminRegOtpResendPrompt`	Didn’t get a code?
`adminRegOtpResend`	Resend
`adminRegPrivacyPolicyHeader`	Accept Privacy Policy
`adminRegPrivacyPolicyDescription`	Select your region of residence to review the applicable privacy policy.
`adminRegPrivacyPolicySelectRegion`	Region of residence
`adminRegPrivacyPolicyAgreement`	I agree to the Ping Identity privacy policy.
`adminRegSendVerificationCode`	Send Verification Code
`adminRegWelcomeHeader`	Welcome to PingOne Advanced Identity Cloud
`adminRegWelcomeDescriptionPre`	You’ve been invited to administer the
`adminRegWelcomeDescriptionPost`	tenant in PingOne Advanced Identity Cloud.
`adminRegWelcomeVerification`	Before you can begin, we need to verify your email address. Click the button below and we’ll send you a one-time verification code.
`alreadyHaveAnAccount`	Already have an account? Sign in here!
`backToDefault`	Back to Sign In
`backToLogin`	Back to Sign In
`captchaError`	CAPTCHA verification failed. Please try again.
`closeModal`	Close
`charactersCannotRepeatMoreThan`	Character cannot repeat more than {max} times
`charactersCannotRepeatMoreThanCaseInsensitive`	Character cannot repeat more than {max} times (case insensitive)
`checkYourEmail`	Check your email
`chooseDifferentUsername`	Please choose a different username.

Variable	Default text
`chooseYourDeviceForIdentityVerification`	Choose your device for identity verification.
`confirmPassword`	Confirm password
`constraintViolationForPassword`	Password does not meet the requirements.
`constraintViolationForValue`	Value does not meet the requirements.
`continueButton`	Continue
`continueWith`	Continue with
`copyAndPasteUrlBelow`	Or, copy and paste the URL below into your authentictor app.
`copyUrl`	Copy URL
`customSecurityQuestion`	Custom security question
`deviceName`	Device name
`doesNotMeetMinimumCharacterLength`	At least {min} character(s)
`downloadTheApp`	Download the app
`dontGetLockedOut`	Don’t get locked out of your account!
`dontHaveAnAccount`	No account? Register here!
`ensurePasswordIsMoreThan`	Password must contain at least {minPasswordLength} character(s).
`ensurePasswordHasOne`	Password must contain at least 1 capital letter, 1 number, and 1 special character.
`enterVerificationCode`	Enter verification code
`exceedsMaximumCharacterLength`	Exceeds maximum of {max} characters
`fieldCanNotContainFollowingCharacters`	Cannot contain these character(s): {chars}
`fieldCanNotContainFollowingValues`	Cannot contain these value(s): {fields}
`forgotPassword`	Forgot password?
`forgotUsername`	Forgot username?
`getAuthenticatorApp`	Set up the ForgeRock Authenticator

Variable	Default text
`getAuthenticatorAppDescription`	To get started, you need to register your device using the ForgeRock Authenticator app.
`getAuthenticatorAppLinks`	Get the app from the Apple App Store or on Google Play Store
`givenName`	First name
`inputRequiredError`	Value is required
`loading`	Loading …
`loginButton`	Sign in
`loginFailure`	Sign in failed
`loginHeader`	Sign in
`loginSuccess`	Sign in successful!
`mail`	Email address
`minimumNumberOfNumbers`	At least {num} number(s)
`minimumNumberOfLowercase`	At least {num} lowercase letter(s)
`minimumNumberOfUppercase`	At least {num} uppercase letter(s)
`minimumNumberOfSymbols`	At least {num} symbol(s)
`nameCallback`	Username
`nameYourDevice`	Name your device
`next`	Next
`nextButton`	Next
`notToExceedMaximumCharacterLength`	No more than {max} characters
`noLessThanMinimumCharacterLength`	At least {min} character(s)
`onMobileOpenInAuthenticator`	On mobile? Open link in Authenticator.

Variable	Default text
`optionallyNameDevice`	Optionally name your device
`passwordCallback`	Password
`passwordCannotContainCommonPasswords`	Password cannot contain common passwords
`passwordCannotContainCommonPasswordsOrBeReversible`	Password cannot contain common passwords or reversible text
`passwordCannotContainCommonPasswordsOrBeReversibleStringsLessThan`	Password cannot contain common passwords or reversible text less than {min} characters
`passwordRequirements`	Password requirements:
`pleaseCheckValue`	Please check this value
`pleaseConfirm`	Please confirm
`preferencesMarketing`	Send me special offers and services
`preferencesUpdates`	Send me news and updates
`provideCustomQuestion`	Provide custom security question
`qrCodeNotWorking`	Not working or need an alternative method?
`qrCodeFailedToRender`	QR Code failed to render. Please notify your support administrator. You are welcome to use the alternative methods below.
`qrCodeImportFailure`	We are unable to render your QR Code. Please use one of the alternative methods below.
`redirectingTo`	Redirecting you to
`registerButton`	Register
`registerHeader`	Register
`registerSuccess`	Registration successful!
`requiredField`	Value is required
`registerYourDevice`	Register {name}
`scanQrCodeWithAuthenticator`	Scan the QR code image below with the ForgeRock Authenticator app to register your device with your login.

Variable	Default text
`securityAnswer`	Security answer
`securityQuestions`	Security question(s)
`securityQuestionsPrompt`	Provide security question(s) and answer(s):
`setupTwoStepVerification`	Set up 2-step verification
`setupTwoStepVerificationButton`	Set up
`setupTwoStepVerificationDescription`	To protect your account, add a second authentication method.
`setupTwoStepVerificationWarning`	Starting April 2, 2024, you must sign in using 2-step verification. Learn more.
`shouldContainANumber`	Should contain a number
`shouldContainAnUppercase`	Should contain an uppercase letter
`shouldContainALowercase`	Should contain a lowercase letter
`shouldContainASymbol`	Should contain a symbol
`showPassword`	Show password
`signalsEvaluation`	Evaluating device information.
`skipButton`	Skip
`skipForNow`	Skip for now
`sn`	Last name
`submit`	Submit
`submitButton`	Submit
`successMessage`	Success!
`termsAndConditions`	Please accept our Terms & Conditions
`termsAndConditionsLinkText`	View full Terms & Conditions

Variable	Default text
`tryAgain`	Please try again
`twoFactorAuthentication`	Two factor authentication
`useThisNewMfaToHelpVerifyYourIdentity`	Use this new device or Multi-Factor Authentication method to help verify your identity.
`useValidEmail`	Please use a valid email address.
`unrecoverableError`	There was an error in the form submission.
`unknownLoginError`	Unknown login failure has occurred.
`unknownNetworkError`	Unknown network request failure has occurred.
`url`	URL:
`useDeviceForIdentityVerification`	Use your device for identity verification.
`useOneOfTheseCodes`	Use one of these codes to authenticate if you lose your device, which has been named: {name}
`userName`	Username
`usernameRequirements`	Username requirements:
`useTheAuthenticatorAppOnYourPhone`	Find the verification code using the authenticator app on your phone.
`validatedCreatePasswordCallback`	Password
`validatedCreateUsernameCallback`	Username
`valueRequirements`	Value requirements:
`verifyYourIdentity`	Verify your identity
`yourDevice`	Your device
`yourMultiFactorAuthIsEnabled`	Your new device or MFA is enabled
`yourRecoveryCodesToAccessAccountForLostDevice`	If you lose your device, or don’t have it with you, a recovery code is the only way to sign in to your account with 2-step verification enabled. It’s strongly recommended that you print and store these codes in a safe place. Each code can only be used once."

You can use the `content` configuration element to specify custom values for these variables, or to localize text content.

To localize the text the Advanced Identity Cloud/PingAM Login Widget displays, provide the localized strings in the `content` configuration element when you call `configure()`. If you don't override the content strings, the widget uses the default `en-us` strings.

The `content` object is passed once during `configure()`. To select strings based on the user's browser locale, build the content object first and then pass it to a single `await configure()` call:

Example `content` configuration

```
import { configure } from '@forgerock/login-widget';

const userLang = navigator.language || navigator.userLanguage;

const localizedContent = userLang === 'fr-CA'
  ? {
      userName: 'Identifiant',
      passwordCallback: 'Phrase secrète',
      nextButton: 'Nous allez en route!',
    }
  : {
      userName: 'Identifier',
      passwordCallback: 'Passphrase',
      nextButton: "Let's go!",
    };

await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-configuration',
  content: localizedContent,
});
```

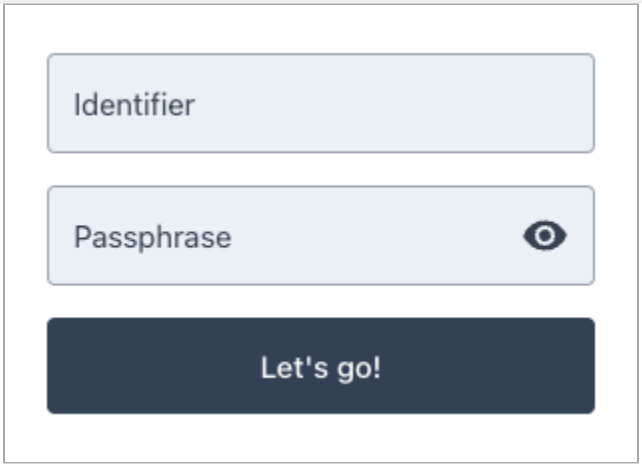


Figure 1. Custom values displayed when locale is not `fr-CA`

Theming the widget

The Advanced Identity Cloud/PingAM Login Widget provides a default theme with both light and dark modes.

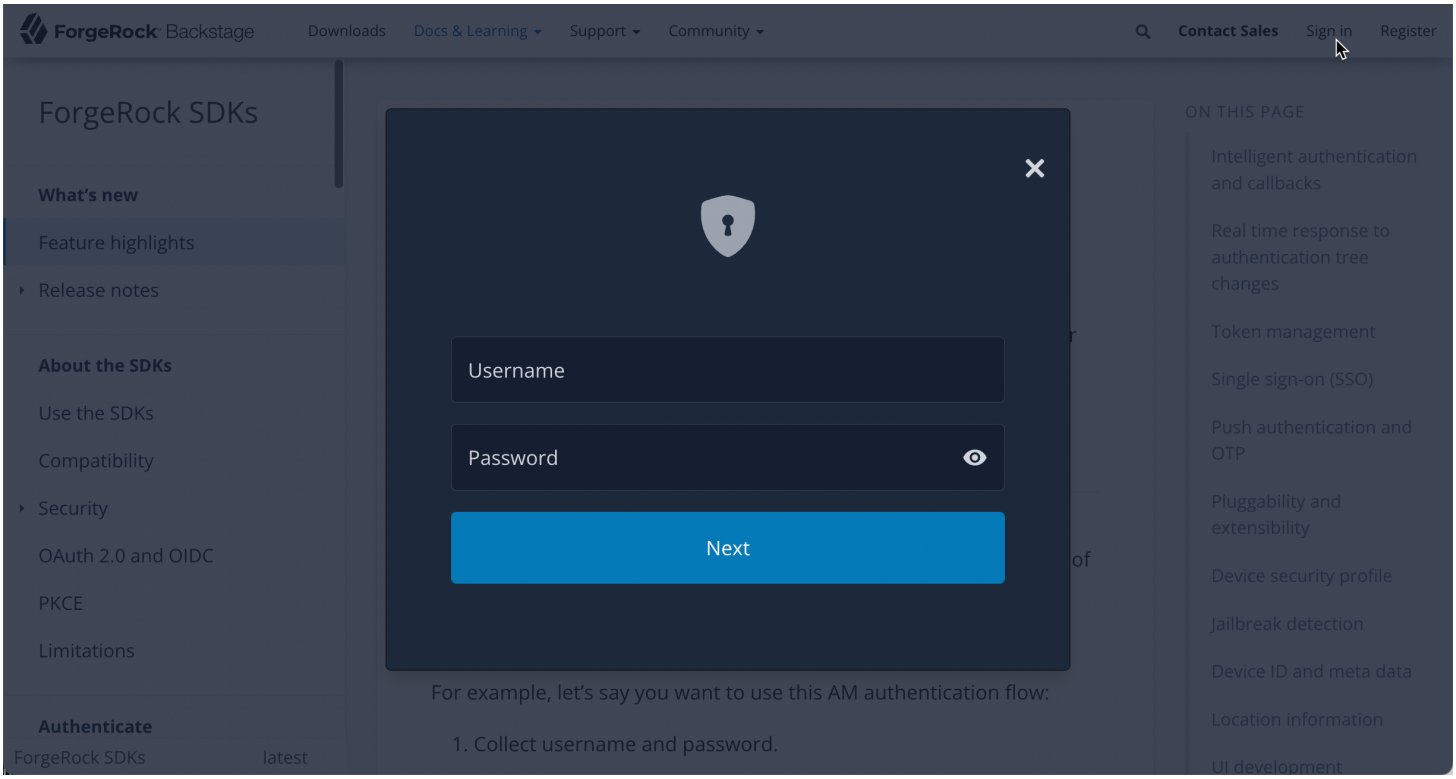


Figure 1. Example of the modal component in dark mode

You can alter these themes by using the Tailwind configuration file.

Note

If your organization's UX or brand requires pixel-perfect UIs developed in accordance to Photoshop, Figma or other "design mocks", Advanced Identity Cloud/PingAM Login Widget might not be able to support such requirements.

Tip

For quick brand color and font overrides, you do not need to rebuild the widget. Use the `style/theme` configuration element described in Set theme colors and fonts.
Reserve the Tailwind rebuild approach on this page for changes beyond colors and fonts, such as spacing or breakpoints.

Switch between light and dark themes

To switch to the dark mode, you can manually add `tw_dark` to the `<body>` element:

```
<body class="tw_dark"></body>
```

The Advanced Identity Cloud/PingAM Login Widget defaults to the light mode if the class is not present.

You can programmatically apply the class if required:

```
const prefersDarkTheme = window.matchMedia('(prefers-color-scheme: dark)').matches;
if (prefersDarkTheme) {
    document.body.classList.add('tw_dark');
}
```

Customize the theme

To reconfigure the theme to use different colors, fonts, or sizing you can provide new values to the Tailwind configuration file and rebuild the Advanced Identity Cloud/PingAM Login Widget, as follows:

1. Download the Advanced Identity Cloud/PingAM Web Login Framework from the GitHub repository:

   `git clone https://github.com/ForgeRock/forgerock-web-login-framework.git`

2. In a terminal window, navigate to the root of the Advanced Identity Cloud/PingAM Web Login Framework:

   `cd forgerock-web-login-framework`

3. Run `pnpm` to download and install the required packages and modules:

   `pnpm install`

4. Run the development script:

   `pnpm run dev`

5. Run Storybook:

   `pnpm run storybook`

   Make a note of the URLs to the Storybook UI listed in the terminal output:

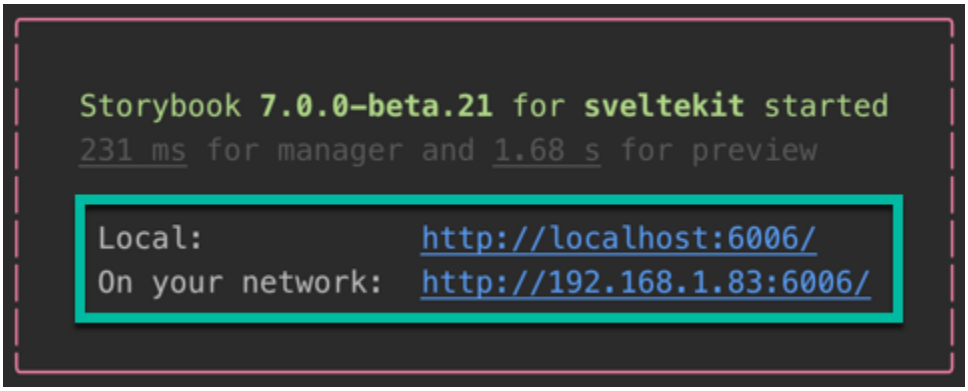


Figure 2. URLs to the Storybook UI

6. Open the `tailwind.config.cjs` file in the root of the Advanced Identity Cloud/PingAM Web Login Framework and adjust your theme by adding them under the `extend` property:

```
// tailwind.config.cjs
module.exports = {
  content: ['./src/**/*.{html,js,svelte,ts}', './.storybook/preview-head.html'],
  darkMode: 'class',
  presets: [require('./themes/default/config.cjs')],
  theme: {
    extend: {
      // Add your customizations here
      colors: {
        body: {
          light: 'darkblue',
        },
        primary: {
          dark: 'darkorange',
        },
        background: {
          light: "gainsboro",
        },
      },
      fontFamily: {
        sans: ['Impact'],
      },
    },
  },
};
```

7. Navigate to the Storybook UI provided in the terminal output earlier to view your changes:

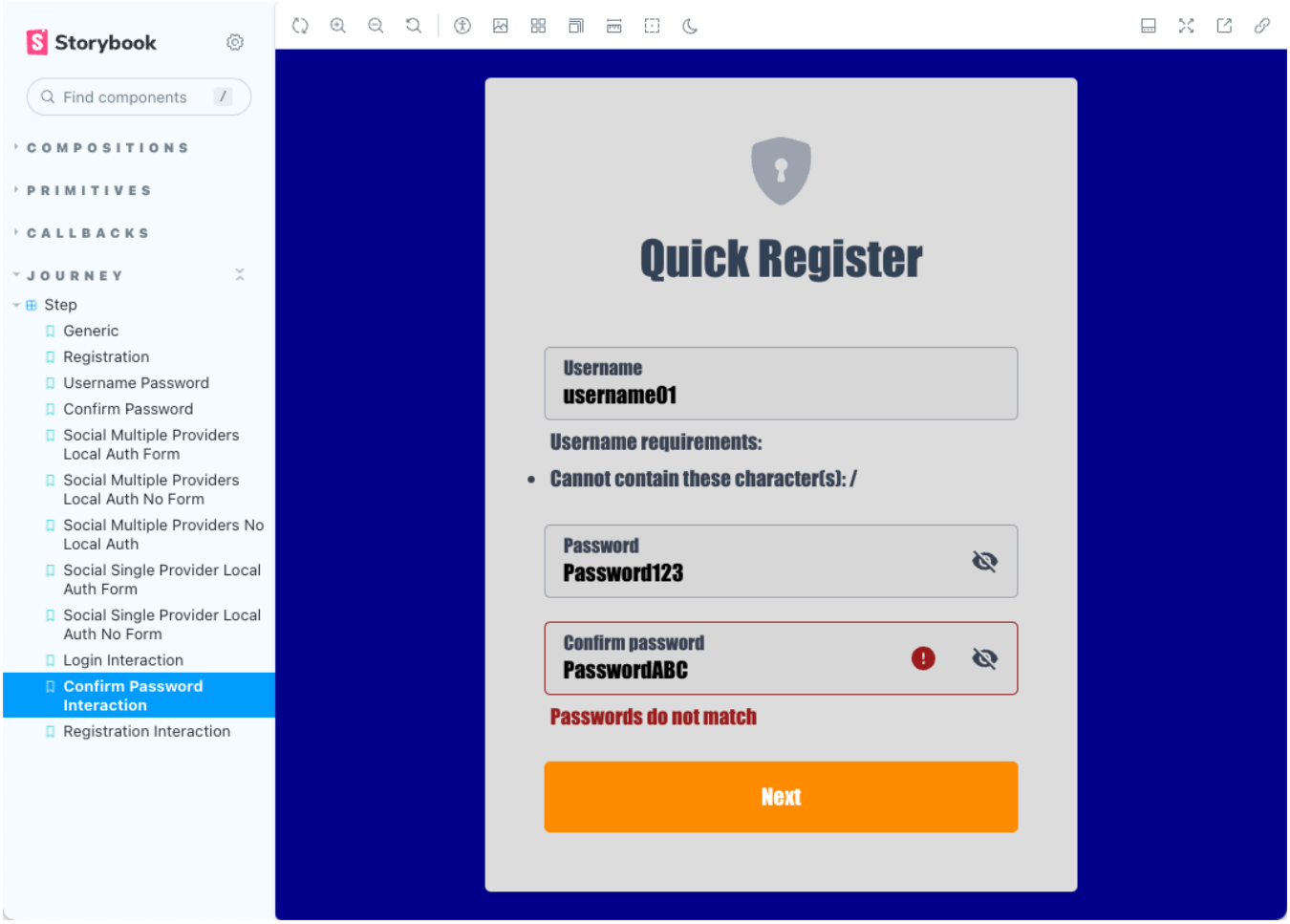


Figure 3. Customized registration modal in Storybook

Changes you make to the `tailwind.config.cjs` file are automatically reflected in the Storybook UI when you save them to disk.

Recommendations

Anything configurable in Tailwind is also configurable in the theme. The custom properties the default theme uses are stored in the `/themes/default/tokens.cjs` file.

We recommend you do not directly modify these default theme files. Only modify the root `tailwind.config.cjs`.

Supported customization

Using the methods on this page, you can customize:

- Colors
- Fonts
- Type sizes
- Spacing
- Breakpoints

Implement your use cases with the Advanced Identity Cloud/PingAM Login Widget

The Advanced Identity Cloud/PingAM Login Widget supports many common authentication and self-service use cases out of the box.

Passkeys

Register passkeys, and let users sign in with autofill

Multi-factor authentication

Register OATH and Push devices with a QR code, and verify one-time passwords

Social sign-on

Authenticate users with external identity providers

Magic links

Suspend and resume authentication journeys via email

CAPTCHA

Detect and block automated authentication attempts

Passkeys

Passkeys let users sign in without a password, using a FIDO2 credential backed by their device, a browser-saved passkey, or a hardware security key.

The Advanced Identity Cloud/PingAM Login Widget supports both sides of the passkey lifecycle:

Registering a passkey

Let a signed-in user register a new passkey for their account.

Implementing passkey autofill

Let users sign in by selecting a registered passkey directly from the browser's native autofill picker on the username field.

Registering a passkey

The Advanced Identity Cloud/PingAM Login Widget provides UI elements for the WebAuthn Registration node node, letting a signed-in user register a new passkey backed by their device, browser, or a hardware security key.

No additional widget configuration is required. The Advanced Identity Cloud/PingAM Login Widget detects a registration step from its callback shape and renders the appropriate UI automatically.

How it works

When the Advanced Identity Cloud/PingAM Login Widget encounters a WebAuthn registration step, it takes the following actions:

1. Prompts the user to optionally name the new device, for example **My Work Laptop**.
2. Calls the browser's `navigator.credentials.create()` API to create a new passkey, using the challenge and relying party details from the server.
3. Submits the resulting credential to the server to complete registration.

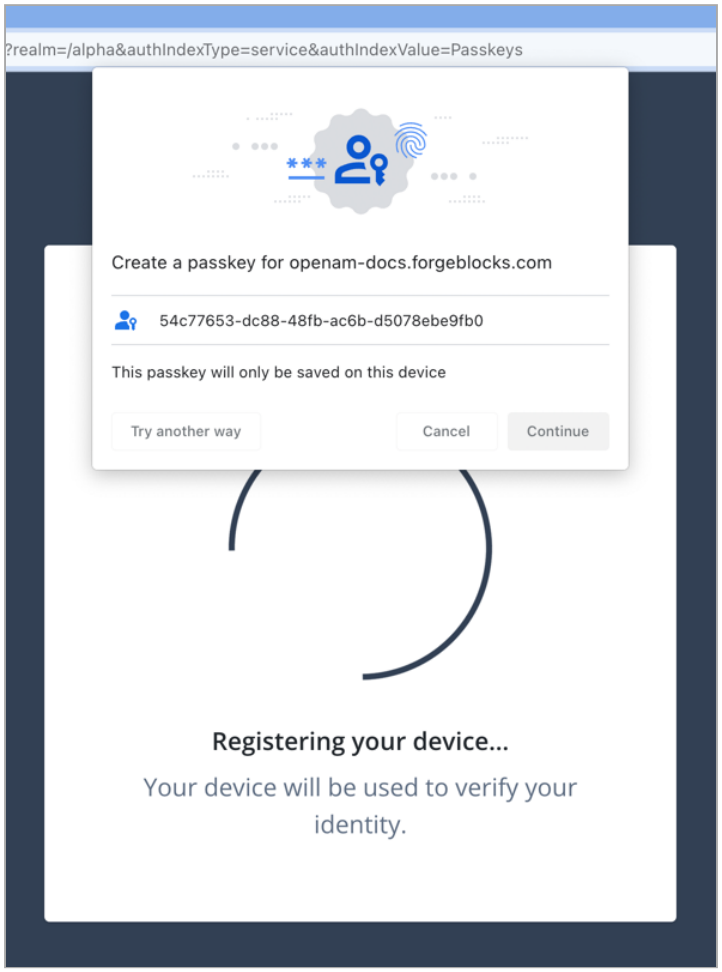


Figure 1. Creating a passkey

The exact prompt the user sees depends on their browser, operating system, and available authenticators. For example, the browser might show a platform biometric prompt, such as Touch ID or Windows Hello, for a device-bound passkey, or a picker for a hardware security key.

> **Note**
>
> If the user cancels the browser prompt, or the browser reports another WebAuthn error, the Advanced Identity Cloud/PingAM Login Widget does not display its own error message.
> It still submits the step to the server with the error recorded, so the WebAuthn Registration node node's **Client Error** outcome fires as expected. Route that outcome to a message or fallback in your journey.

Configure your authentication journey

Because registering a passkey requires an authenticated session, add the WebAuthn Registration node node **after** a successful sign-in step in your journey, rather than as part of the sign-in journey itself.

> **Important**
>
> Use the same configuration values in the WebAuthn Registration node node as you use in any WebAuthn Authentication node nodes in your sign-in journeys.
> Configuration mismatches between these nodes cause authentication to fail.

In the WebAuthn Registration node node:

1. In **Origin domains**, enter the URL where you host your app, for example `https://app.example.com`.

   > **Note**
   >
   > If you leave **Origin domains** empty, the server uses the origin of incoming requests as an accepted origin.

2. In **Accepted signing algorithms**, include one or more of `ES256` and `RS256`.
3. Ensure **Limit registrations** is **not enabled**, so returning users can register more than one passkey.
4. Optionally, use **Authentication attachment** to control which type of authenticator the browser offers:

Value	Description
`PLATFORM`	Only allow authenticators built into the user's device, such as Face ID, Touch ID, or Windows Hello.
`CROSS_PLATFORM`	Only allow external authenticators, such as hardware security keys.
`UNSPECIFIED`	Allow either type. This is the default.

5. Optionally, to support usernameless sign-in later, enable **Username to device**.

Test the registration flow

Sign in to your app, then start a journey that includes the WebAuthn Registration node node.

The Advanced Identity Cloud/PingAM Login Widget prompts for an optional device name, then invokes the browser's passkey creation flow. Complete the browser's prompt to finish registration.

After a device is registered, configure passkey autofill to let the user sign in with it, or add a WebAuthn Authentication node node without conditional mediation to require an explicit passkey sign-in step.

Displaying recovery codes

Use the Recovery Code Display node node after a device registration step to give users backup codes they can use to authenticate if they lose access to their registered device.

When the Advanced Identity Cloud/PingAM Login Widget encounters this node's callback, it automatically renders the device name and a formatted grid of recovery codes for the user to save or print.

Configure your authentication journey

Add the Recovery Code Display node node to your registration journey, after the node that registers the device.

To let users authenticate with a previously issued recovery code, add the Recovery Code Collector Decision node node to your login journey.

Test the flow

Complete a device registration journey that includes the Recovery Code Display node node. After registration succeeds, the Advanced Identity Cloud/PingAM Login Widget displays:

- A header confirming that MFA is now enabled
- The name of the registered device
- A list of one-time-use recovery codes

> **Important**
>
> The server returns recovery codes only once, immediately after registration.
> Advise your users to save or print them before continuing, as the Advanced Identity Cloud/PingAM Login Widget does not provide a way to view previously issued codes again.

Related use cases

Implementing passkey autofill

Let users sign in with a registered passkey from the browser's autofill picker.

Implementing passkey autofill

Passkey autofill, sometimes known as **WebAuthn Conditional UI**, lets users sign in by selecting a passkey directly from the browser's native autofill picker on the username field.

When the user clicks or tabs into the username field, the browser presents any passkeys registered for that site.

Selecting one completes authentication immediately, with no password required. If the user has no passkeys, or dismisses the picker, the login form continues to work as normal, so the experience degrades gracefully with no disruption.

The Advanced Identity Cloud/PingAM Login Widget activates passkey autofill automatically when it detects the right combination of callbacks from the server.

No additional widget configuration is required.

> **Important**
>
> Passkey autofill requires PingAM 8.1 or later, or PingOne Advanced Identity Cloud.
> It depends on version 2 of both the WebAuthn Authentication node and the Page node, which were introduced in that release.

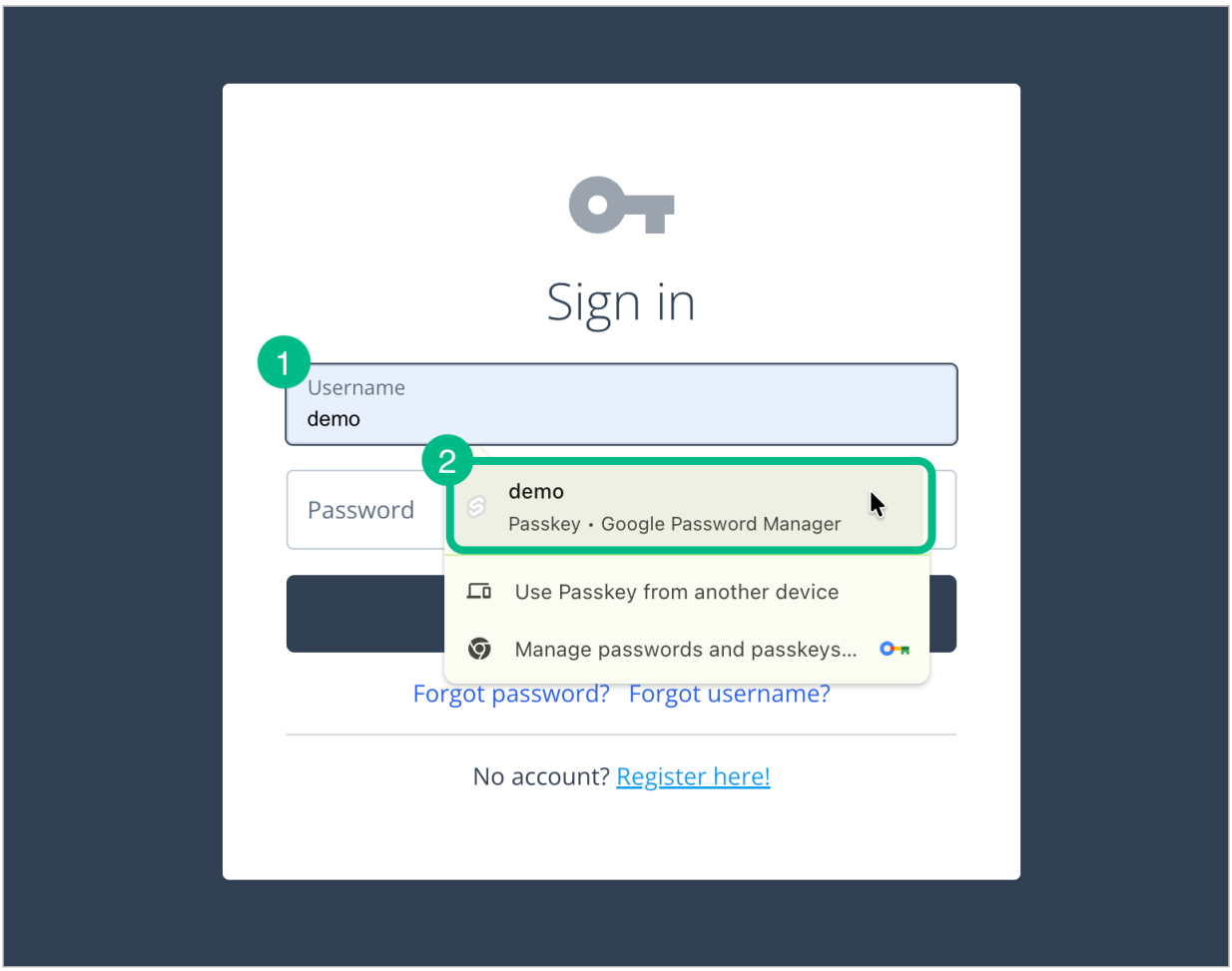
> **Note**
>
> Before a user can sign in with passkey autofill, they need a registered passkey. Refer to Registering a passkey.

How it works

When the Advanced Identity Cloud/PingAM Login Widget encounters a page node configured for passkey autocomplete, it takes the following actions:

1. Adds `autocomplete="username webauthn"` to the username field, which signals to the browser that passkey autofill should be offered.
2. Starts a background conditional credential request using the WebAuthn challenge from the `MetadataCallback`.

The browser then displays its native passkey picker attached to the username field:



When the user selects a passkey, the widget completes the WebAuthn assertion and submits the step automatically.

Configuring your authentication journey

To configure an authentication journey for passkey autocomplete, add the following nodes together inside a Page node **(version 2)**:

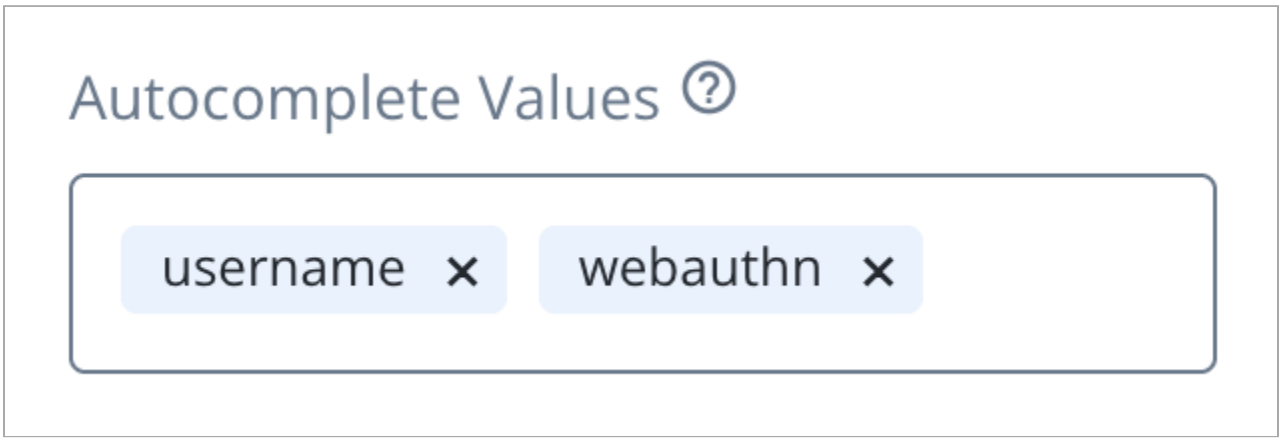
Platform Username node

Collects the username.

To add the required `autocomplete` attributes to the rendered field in the Login Widget, add the following values to the **Autocomplete Values** property of the node, in this order:

1. `username`
2. `webauthn`

The result resembles the following:



WebAuthn Authentication node ***(version 2)***

Issues the WebAuthn challenge and handles the assertion response.

The following settings are required for passkey autofill:

Mediation

Set to `CONDITIONAL`.

This is what instructs the browser to show passkeys inline in the autofill UI rather than triggering a separate popup.

Username from device

Must be **Enabled**.

Version

Must be `v2.0` or later. This version contains an outcome path so that users without a passkey can fall through to a normal username and password flow.

Connect the **Outcome** outcome to nodes that handle username and password authentication.

Connect the **Success** outcome to nodes that handle the successful authentication using the passkey. For example, you could connect directly to the green checkmark **Success** node to complete the journey.

The result resembles the following:

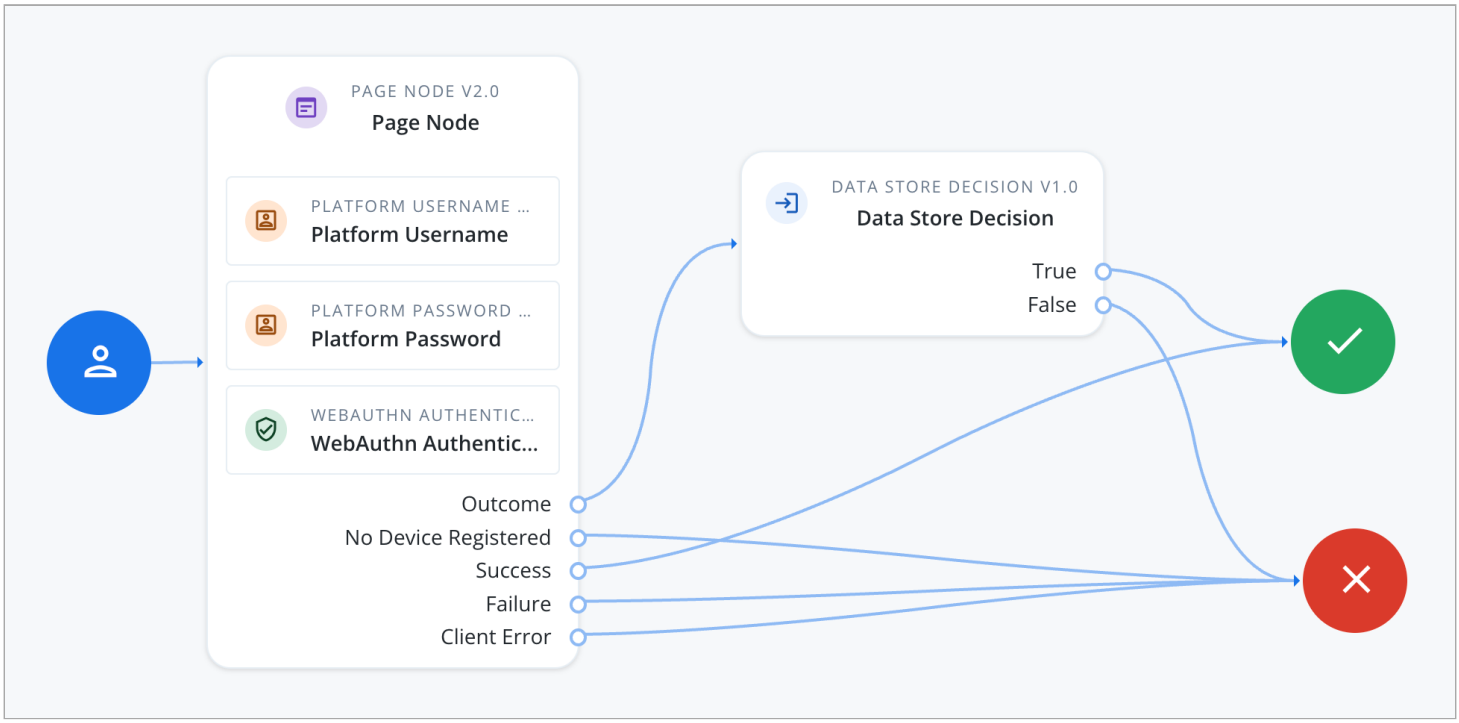


Figure 1. Basic authentication journey for passkey autofill

Browser and HTTPS requirements

Passkey autofill is a browser-native capability that requires a secure context:

- **HTTPS** is required in production. Browsers will not expose passkeys on plain HTTP origins.
  - `localhost` is treated as a secure context by browsers, so passkey autofill works without HTTPS during local development.
- The browser must support WebAuthn Conditional UI. Current versions of Chrome, Edge, Safari, and Firefox all support it.

  The widget checks for support at runtime; if the browser does not support conditional mediation, the username field behaves as normal with no error.

Configuring the Login Widget for passkey autofill

No additional widget-specific configuration is required.

The Advanced Identity Cloud/PingAM Login Widget detects and enables passkey autofill automatically.

Related use cases

Registering a passkey

Let a signed-in user register a new passkey.

Multi-factor authentication with OATH and Push

The Advanced Identity Cloud/PingAM Login Widget provides UI elements for registering a new OATH or Push device by scanning a QR code with the ForgeRock Authenticator app, and for verifying a one-time password from an already-registered device.

This page covers:

- Registering a device with a QR code
- Verifying a one-time password
- Displaying recovery codes

Registering a device with a QR code

Use the Combined MFA Registration node or Push Registration node node to let a user register a new device by scanning a QR code with the ForgeRock Authenticator app, instead of typing in a shared secret manually.

When the Advanced Identity Cloud/PingAM Login Widget encounters a step containing QR code data, it automatically:

1. Renders a QR code image the user can scan with their authenticator app.
2. Provides a fallback link and copyable URL for users who cannot scan the code, for example if testing on the same device that renders the widget.
3. If the step also includes a polling callback, polls the server until the user completes registration on their device, then continues the journey automatically.

No additional widget configuration is required. This behavior is automatic whenever the journey step matches this shape.

Configure your authentication journey

Add the Combined MFA Registration node or Push Registration node node to a registration journey.

Connect the node's outcomes to handle both successful registration and any failure paths your journey requires.

> **Tip**
>
> The Combined MFA Registration node node lets users choose between Push and OATH registration in a single step. Use the Push Registration node node if you only want to offer Push registration.

Test the registration flow

With your journey configured, start it in an app using the Advanced Identity Cloud/PingAM Login Widget. When you reach the registration step, the widget displays:

- A QR code image
- A collapsible section with a direct link to open the registration URL in the ForgeRock Authenticator app, and a copyable URL, useful when testing in a browser on the same device as the authenticator app
- A loading indicator while the widget polls the server for registration completion

Scan the QR code with the ForgeRock Authenticator app (or open the fallback link on a mobile device) to complete registration. The journey continues automatically once the server detects the new device.

Verifying a one-time password

The Advanced Identity Cloud/PingAM Login Widget provides UI elements for the **OATH Token Verifier** node but not currently the standalone **OATH Registration** node. Use the QR code registration flow above instead.

Once a user has registered the ForgeRock Authenticator for one-time passwords, for example using the flow above, the Advanced Identity Cloud/PingAM Login Widget can accept a one-time password from the authenticator app.

The Advanced Identity Cloud/PingAM Login Widget requires that the **OATH Token Verifier** node is contained within a **Page node** configured with a specific **Stage** property.

In the containing **Page Node**, set the **Stage** property to `OneTimePassword`:

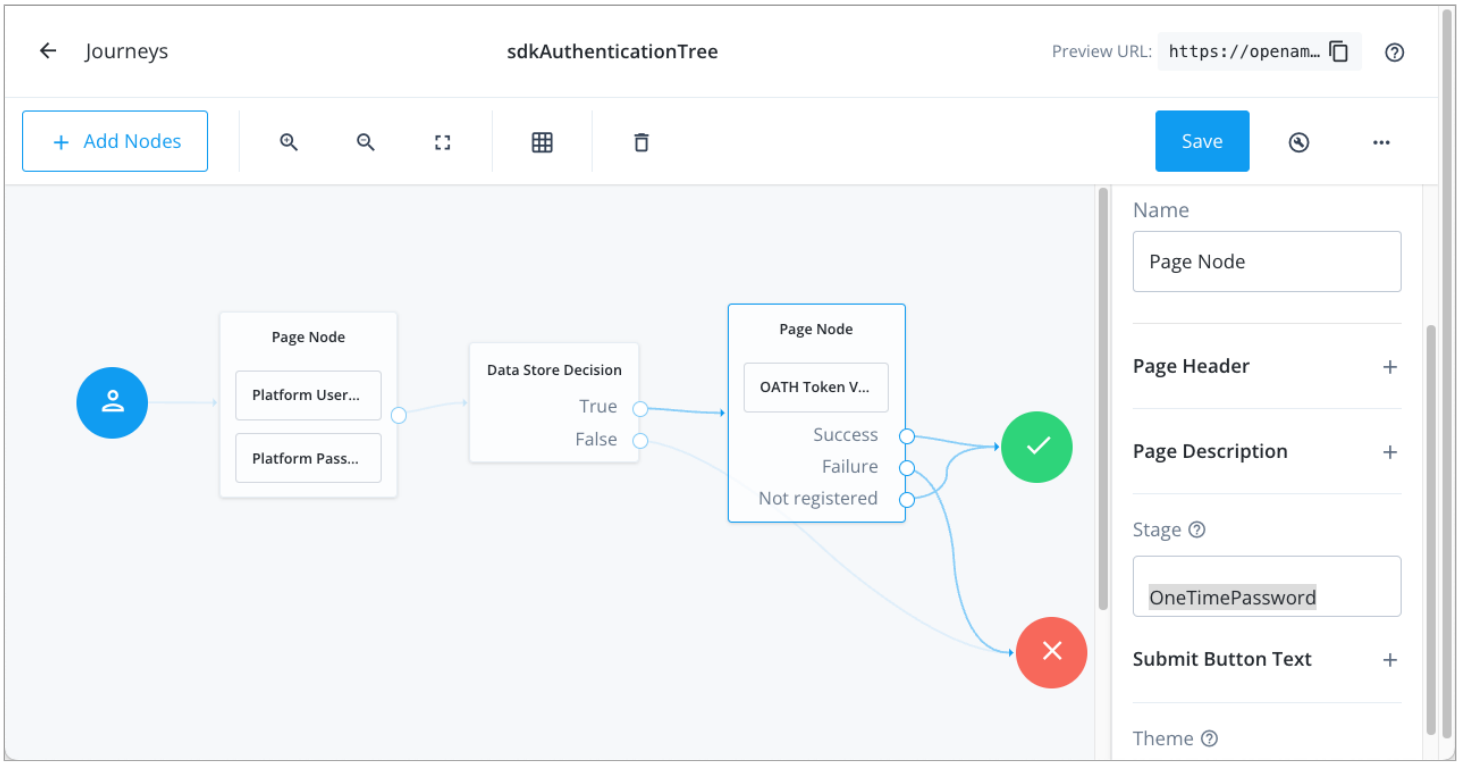


Figure 1. OATH journey example

The Advanced Identity Cloud/PingAM Login Widget detects that stage value as a special case and renders the appropriate UI:

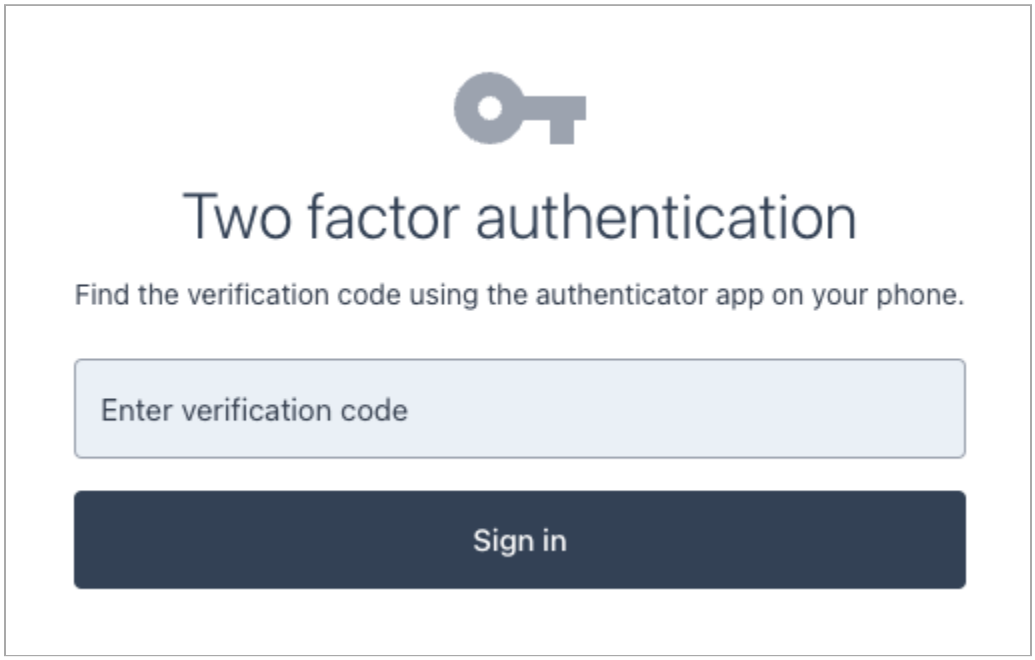


Figure 2. Rendering with the `OneTimePassword` stage property

If you do not put the **OATH Token Verifier** node within a **Page node**, the Advanced Identity Cloud/PingAM Login Widget will not render the UI correctly:

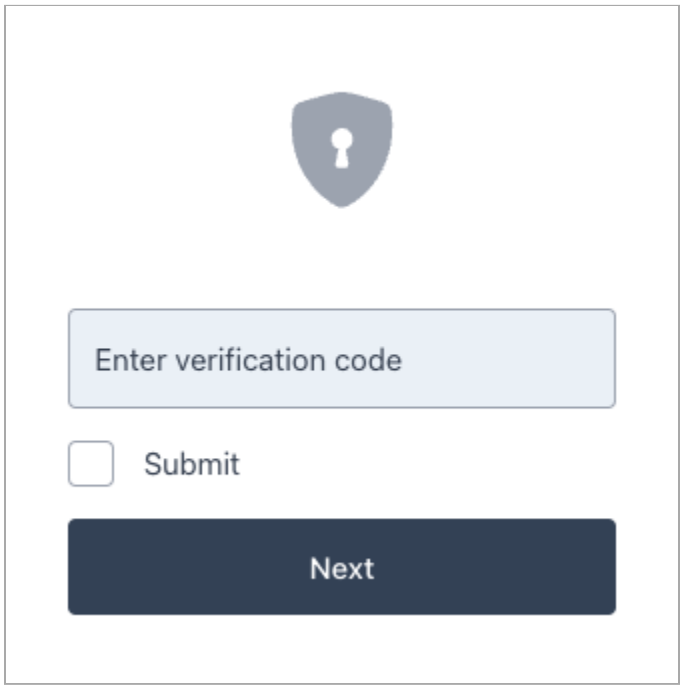


Figure 3. Rendering a lone OATH Token Verifier node

Displaying recovery codes

Use the Recovery Code Display node node after a device registration step to give users backup codes they can use to authenticate if they lose access to their registered device.

When the Advanced Identity Cloud/PingAM Login Widget encounters this node's callback, it automatically renders the device name and a formatted grid of recovery codes for the user to save or print.

Configure your authentication journey

Add the Recovery Code Display node node to your registration journey, after the node that registers the device.

To let users authenticate with a previously issued recovery code, add the Recovery Code Collector Decision node node to your login journey.

Test the flow

Complete a device registration journey that includes the Recovery Code Display node node. After registration succeeds, the Advanced Identity Cloud/PingAM Login Widget displays:

- A header confirming that MFA is now enabled
- The name of the registered device
- A list of one-time-use recovery codes

> **Important**
>
> The server returns recovery codes only once, immediately after registration.
> Advise your users to save or print them before continuing, as the Advanced Identity Cloud/PingAM Login Widget does not provide a way to view previously issued codes again.

Related use cases

Passkeys

Passwordless sign-in and registration with passkeys.

Log in with social authentication

Social authentication provides your users with a choice of ways to sign in that suits them.

The Advanced Identity Cloud/PingAM Login Widget supports social authentication with the following providers:

- Apple
- Facebook
- Google

Selecting one of these providers in a journey initiates an OAuth 2.0 flow allowing them to authenticate themselves with the social provider before returning to the original journey.

To enable this flow you need to:

1. Offer a choice of social identity providers using the **Select Identity Provider** node.
    - Optionally, you can allow users to skip social authentication and enter their credentials in the same form, provided nodes such as a username collector are also present.

2. Handle the OAuth 2.0 flow for your users using the **Social Provider Handler Node**.

3. Determine if the user signed in to the social provider maps to a known user by adding the **Identify Existing User Node**.

The following is an example journey for social authentication:

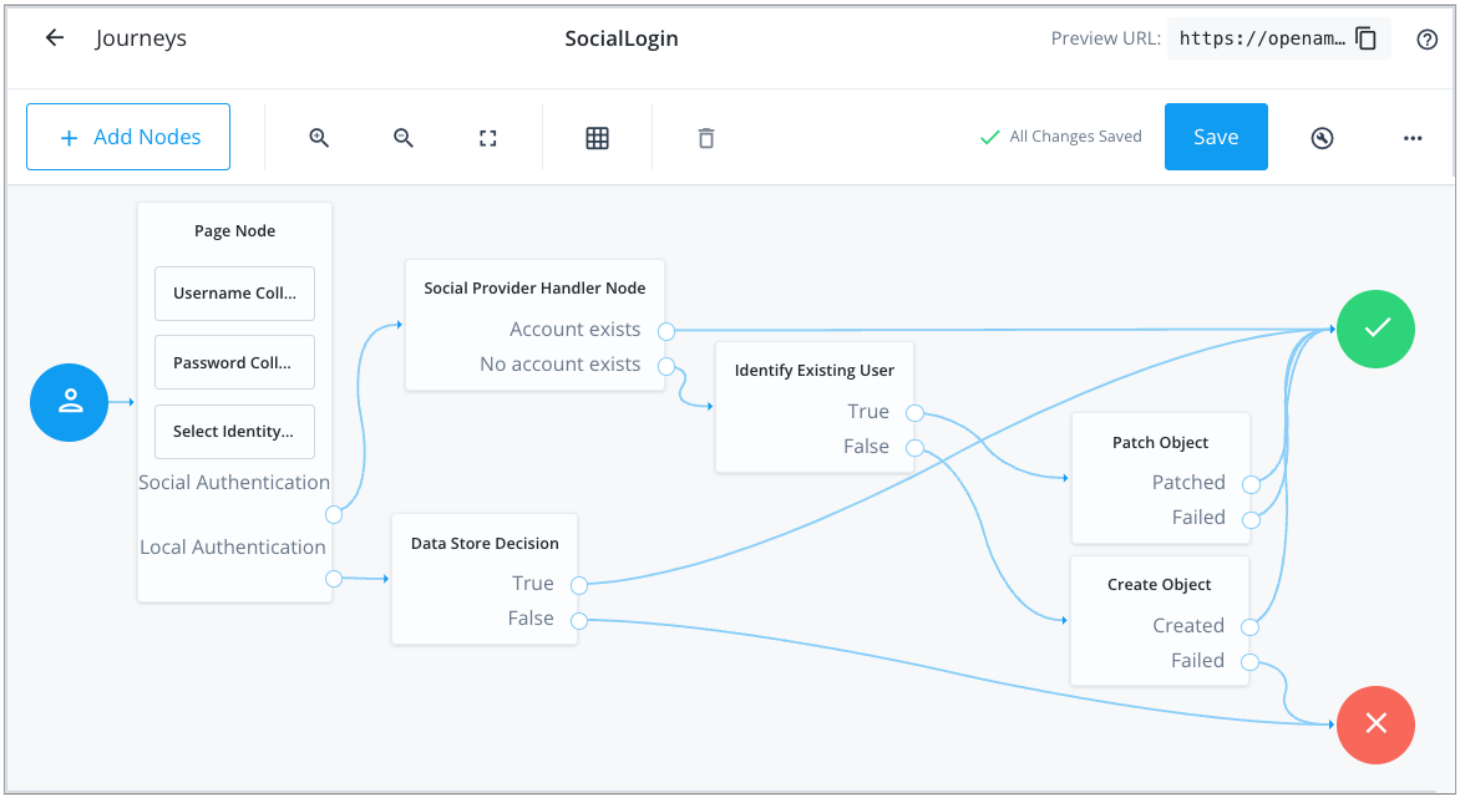


Figure 1. Example social login journey

> **Tip**
>
> For a detailed guide covering the creation of social authentication journeys, refer to How do I create end user journeys for social registration and login in PingOne Advanced Identity Cloud? in the Backstage Knowledge Base.

On the client side, the Advanced Identity Cloud/PingAM Login Widget handles the selection of the identity provider and redirection to the provider.

You need to ensure your app manages the return back from the provider. To handle the return from a social provider, detect `code`, `state` and `form_post_entry` query parameters, as these instruct the Advanced Identity Cloud/PingAM Login Widget to resume authentication using the current URL:

Detect social authentication query parameters and continue the journey

```
import { journey } from '@forgerock/login-widget';

// This snippet assumes the widget has already been configured elsewhere in your
// app via `await configure({...})`. Refer to the tutorial for the full setup.
const journeyEvents = journey();

const url = new URL(location.href);
const codeParam = url.searchParams.get('code');
const stateParam = url.searchParams.get('state');
const formPostEntryParam = url.searchParams.get('form_post_entry');

if (formPostEntryParam || (codeParam && stateParam)) {
    journeyEvents.start({ resumeUrl: location.href });
}
```

The `location.href` value includes any query parameters returned from the social provider. Without all the query parameters, the Advanced Identity Cloud/PingAM Login Widget might not be able to rehydrate the journey and continue as needed.

Suspend journeys with "magic links"

You can use the Email Suspend Node within your journeys to support a number of experiences, including verifying a user's email address, building a "forgot password" flow or using an email address for multifactor authentication.

The node suspends the journey until the user clicks a link, referred to as a *magic link*, in their email. This link contains a generated unique code that can continue the suspended journey.

This page shows how to configure the Advanced Identity Cloud/PingAM Login Widget to take advantage of this feature.

Configure the authentication server

1. Add the Email Suspend Node to the journey to suspend it until the user continues the journey from the link found in their email.

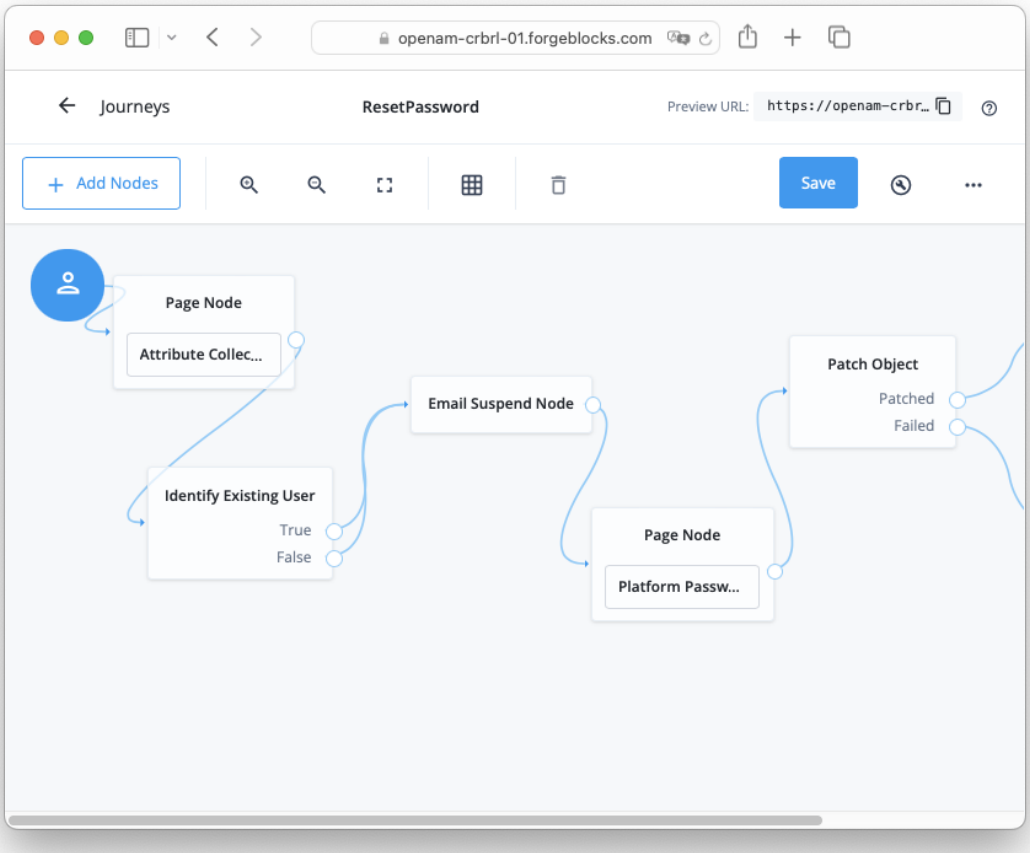


Figure 1. Insert the Email Suspend Node into your journey

2. Configure the **External Login Page URL** property in the Access Management native console to match your custom app's URL. This ensures the magic links are able to redirect users to your app to resume the journey. If not specified, the default behavior is to route users to the login page.

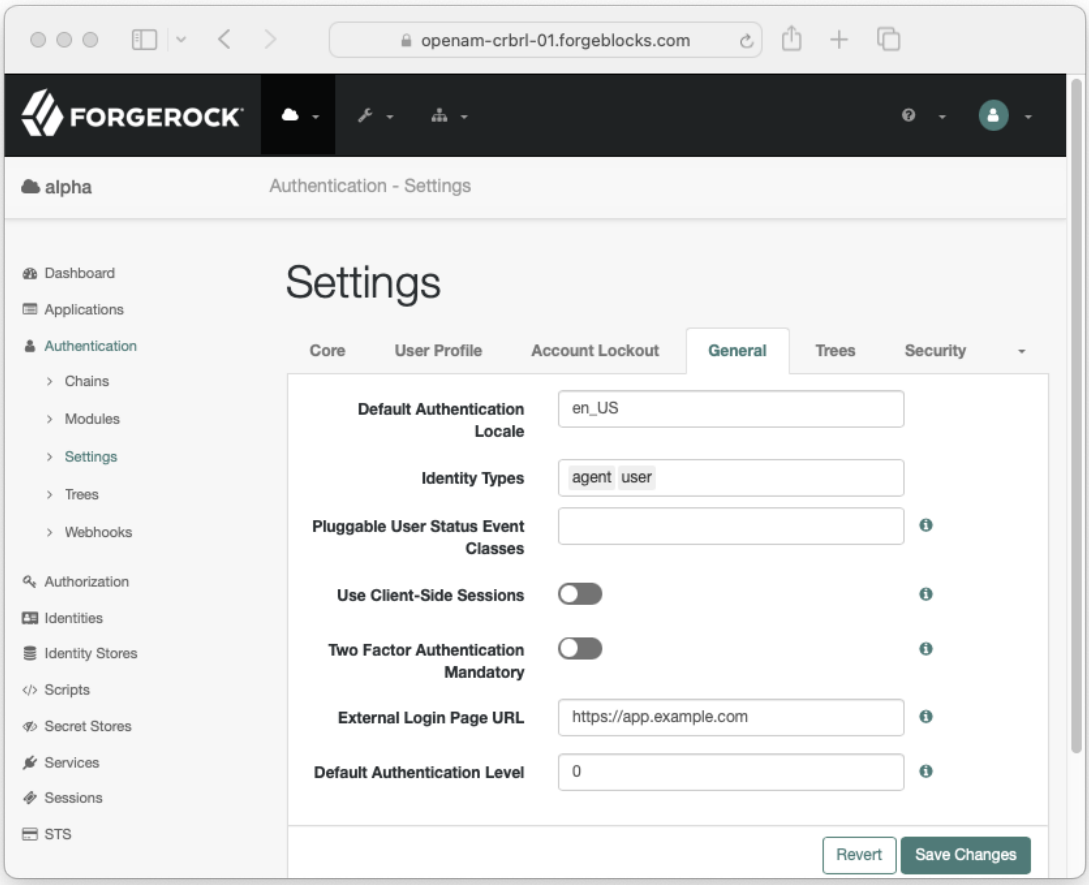


Figure 2. Configure external login URL in the PingAM native console

3. When the Advanced Identity Cloud/PingAM Login Widget encounters the Email Suspend Node, it renders the string configured in the **Email Suspend Message** property configured in the node. The user is not able to continue the journey until they click the link emailed to them.

Handle suspend IDs in your app

When your app handles a magic link, it needs to recognize it as a special condition and provide the Advanced Identity Cloud/PingAM Login Widget with the full URL that the user clicked in their email.

Return this URL, including all query parameters, to the server as the value of the `resumeUrl` parameter:

```
import { journey } from '@forgerock/login-widget';

// This snippet assumes the widget has already been configured elsewhere in your
// app via `await configure({...})`. Refer to the tutorial for the full setup.
const journeyEvents = journey();

const url = new URL(location.href);
const suspendedId = url.searchParams.get('suspendedId');

if (suspendedId) {
  journeyEvents.start({ resumeUrl: location.href });
}
```

The `location.href` value includes all query parameters included in the magic link. Without all the query parameters, the Advanced Identity Cloud/PingAM Login Widget might not be able to rehydrate the journey and continue as needed.

Implement a CAPTCHA

The Advanced Identity Cloud/PingAM Login Widget supports the use of a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart), which helps to prevent automated scripts from attempting to authenticate to your servers.

To use a CAPTCHA in the Advanced Identity Cloud/PingAM Login Widget, add a CAPTCHA node or reCAPTCHA Enterprise node to your authentication journey.

Supported CAPTCHA variants

Advanced Identity Cloud/PingAM Login Widget supports the following CAPTCHA variants:

CAPTCHA variant	Support	Node
hCaptcha	✓ Active ✓ Invisible ✗ Passive / 99% Passive	CAPTCHA node
reCAPTCHA v2	✓ Full, including Invisible reCAPTCHA.	CAPTCHA node
reCAPTCHA v3	✓ Full	CAPTCHA node
reCAPTCHA Enterprise	✓ Full, including invisible (score-based) mode.	reCAPTCHA Enterprise node

Visible and invisible CAPTCHA modes

CAPTCHA providers typically offer two rendering modes:

Visible (active) mode

The CAPTCHA renders a visible widget, usually a checkbox, inside the login form. The user must interact with it before they can submit the form.

Invisible mode

No visible widget appears in the login form. Instead, the CAPTCHA runs an automated risk assessment in the background when the page loads. If the provider's scoring determines the request looks suspicious, it may surface a challenge popup for the user to complete. Otherwise, authentication proceeds without any visible interruption.

The Advanced Identity Cloud/PingAM Login Widget defaults to visible mode. To enable invisible mode, pass `mode: 'invisible'` in the `captcha` option when configuring the widget:

```
import { configure } from '@forgerock/login-widget';

await configure({
    wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
    // ... other config options ...
    captcha: {
        mode: 'invisible', // 'visible' (default) | 'invisible'
    },
});
```

The `mode` value applies to whichever CAPTCHA provider is configured in your authentication journey node, for example hCaptcha, reCAPTCHA v2, or reCAPTCHA Enterprise.

When invisible mode is active and the CAPTCHA challenge fails or expires, the Advanced Identity Cloud/PingAM Login Widget displays an inline error alert prompting the user to try again.

Configure your app

The Advanced Identity Cloud/PingAM Login Widget automatically injects the CAPTCHA provider script into your page when it encounters a CAPTCHA callback. You do not need to add a `<script>` tag manually.

If you prefer to load the script yourself, for example to control when it loads relative to other resources, you can add it to the `<head>` of your page. The widget detects that the provider is already available and skips its own injection.

hCaptcha

```
<script src="https://js.hcaptcha.com/1/api.js" async defer></script>
```

reCAPTCHA v2

```
<script async src="https://www.google.com/recaptcha/api.js"></script>
```

reCAPTCHA v3

If pre-loading reCAPTCHA v3, append your site key as a `render` query parameter:

```
<script async src="https://www.google.com/recaptcha/api.js?render={reCAPTCHA_site_key}"></script>
```

You can add a `recaptchaAction` property with a custom value when calling the journey. That value tags the event in the reCAPTCHA console so that you can track different usage:

```
import { journey } from '@forgerock/login-widget';

const journeyEvents = journey();

journeyEvents.start({
    journey: 'reCAPTCHAv3journey',
    // ...any other journey config required...
    recaptchaAction: 'loginVIPArea', // optional; falls back to journey name
});
```

If you do not provide a `recaptchaAction` value, the Advanced Identity Cloud/PingAM Login Widget attempts to use the name of the journey instead, if available.

reCAPTCHA Enterprise

The Advanced Identity Cloud/PingAM Login Widget handles the `ReCaptchaEnterpriseCallback` automatically when the journey contains the reCAPTCHA Enterprise node.

No additional script setup is required beyond configuring the node in your authentication journey.

For invisible (score-based) Enterprise keys, also set `captcha: { mode: 'invisible' }` in your widget configuration. Learn more in Visible and invisible CAPTCHA modes.

You can add a `recaptchaAction` property with a custom value when calling the journey. That value tags the event in the reCAPTCHA Enterprise console so that you can track different usage:

```
import { journey } from '@forgerock/login-widget';

const journeyEvents = journey();

journeyEvents.start({
    journey: 'reCAPTCHAEnterpriseJourney',
    // ...any other journey config required...
    recaptchaAction: 'loginVIPArea', // falls back to journey name
});
```

> **Note**
>
> For reCAPTCHA Enterprise, either `recaptchaAction` or `journey` must be provided for the CAPTCHA to execute.

Test a CAPTCHA

With your app configured, you can visit any journey that contains a CAPTCHA node or reCAPTCHA Enterprise node to test a CAPTCHA with the Advanced Identity Cloud/PingAM Login Widget.

For example, the following image shows how the Advanced Identity Cloud/PingAM Login Widget displays a reCAPTCHA v2 challenge, alongside a platform username and platform password nodes, all within a single page node:

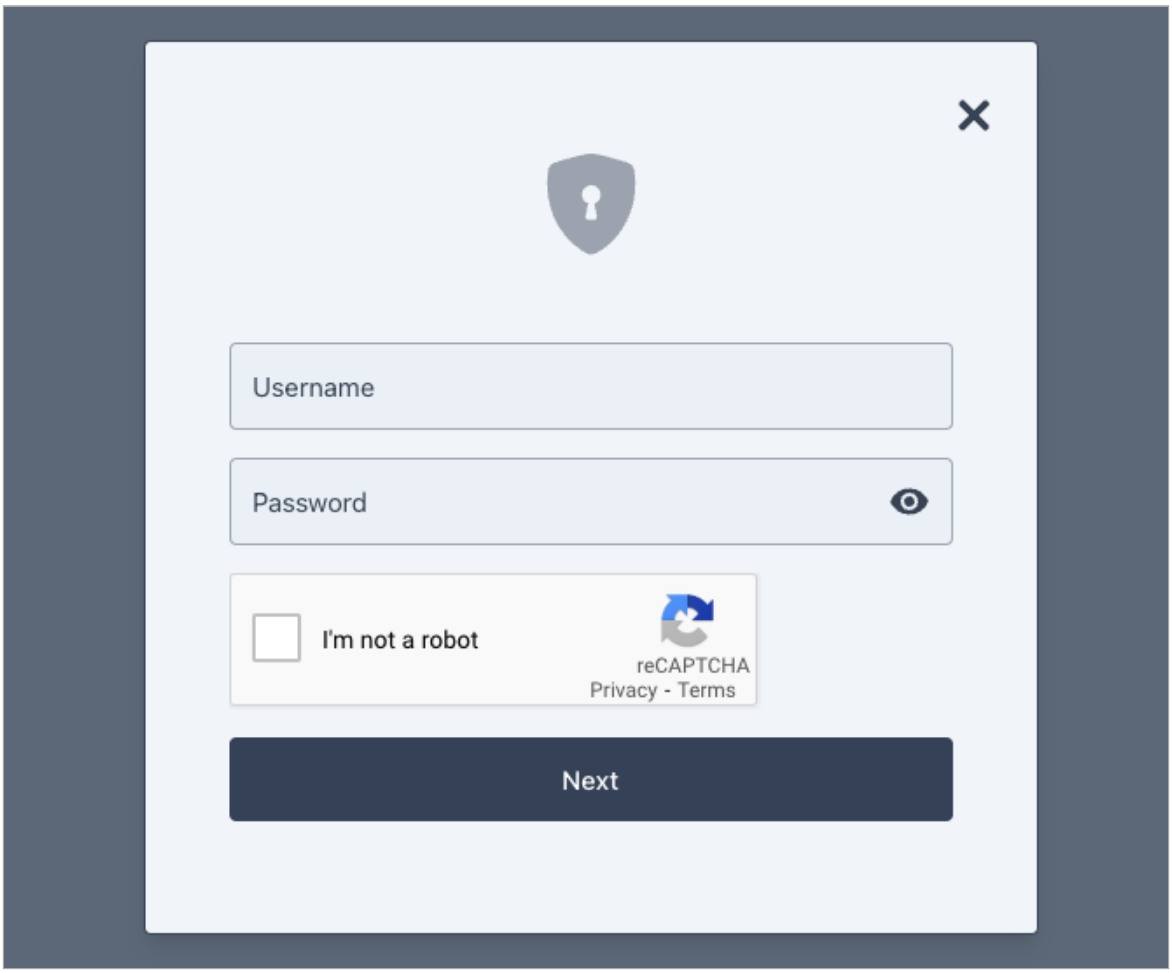


Figure 1. Login Widget handling reCAPTCHA v2 in a page node

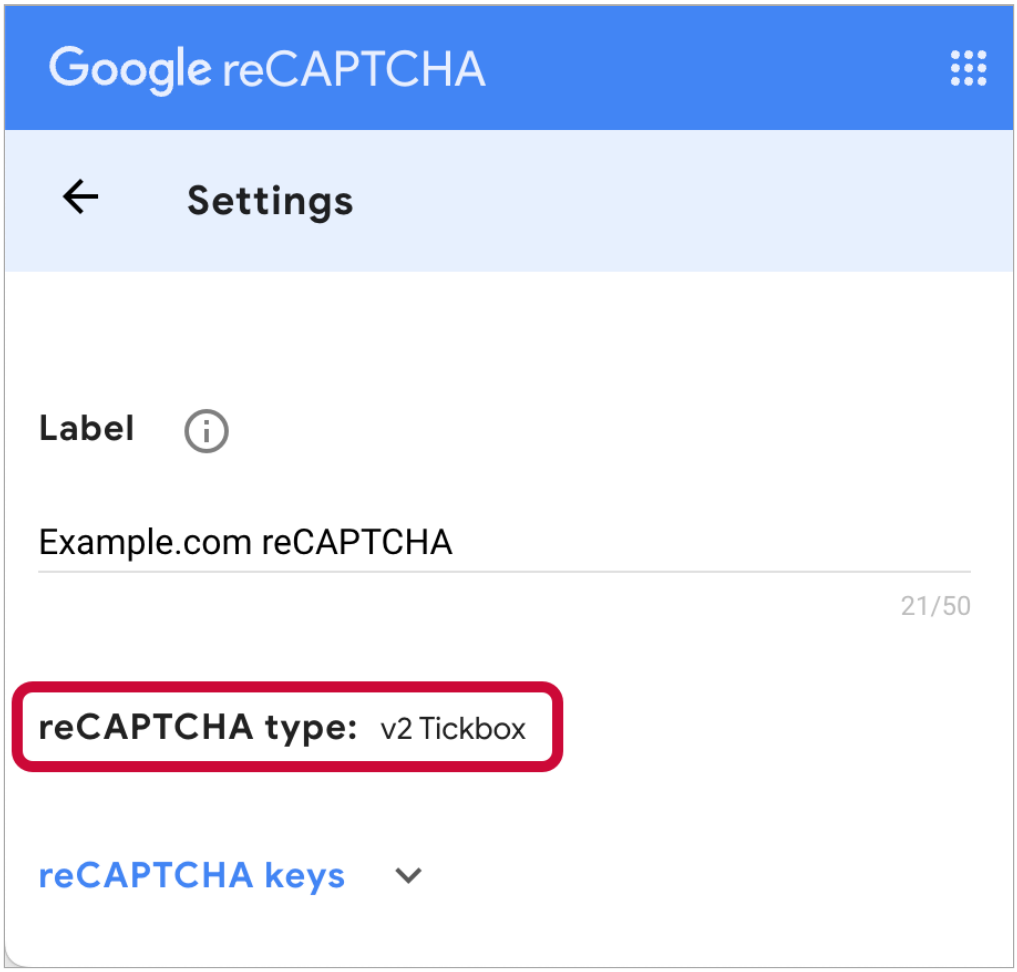
Troubleshooting

This section contains information on how to diagnose issues when using a CAPTCHA with the Advanced Identity Cloud/PingAM Login Widget.

Why does the reCAPTCHA display "ERROR for site owner: Invalid key type"?

Ensure the site key you have specified in the CAPTCHA node is configured for the version of reCAPTCHA type you want to use.

For example, the following image shows a configuration for **v2 Tickbox**:



When using a v2 site key, do not select **ReCaptcha V3 Node** in the CAPTCHA node configuration.

Why does the browser console display "Error: Invalid site key or not loaded in api.js"?

Ensure you have added the correct site key value as a `render` query parameter of the Google `api.js` script.

For example:

```
<script async src="https://www.google.com/recaptcha/api.js?render=1249672216234"></script>
```

Why does the reCAPTCHA display "Localhost is not in the list of supported domains for this site key."?

The `localhost` domain is blocked from working with reCAPTCHA by default.

You can add the domain to the site key configuration for testing purposes if required.

> **Note**
>
> It can take several minutes for changes to the allowed domains to take effect.

For more information, refer to Google's reCAPTCHA documentation.

Integrating the Advanced Identity Cloud/ PingAM Login Widget

Find out how you can integrate the Advanced Identity Cloud/PingAM Login Widget with different frameworks and libraries.

Integrate with PingOne Protect for risk evaluations

The Advanced Identity Cloud/PingAM Login Widget can integrate with PingOne Protect to evaluate the risk involved in a transaction.

Use PingOne Protect in your authentication journeys to help prevent identity fraud by incorporating advanced features and real-time detection.

Integrate Login Widget into a React app

Learn how to integrate the Advanced Identity Cloud/PingAM Login Widget into a simple React app that you scaffold using Vite.

Integrate with PingOne Protect for risk evaluations

The Advanced Identity Cloud/PingAM Login Widget can integrate with PingOne Protect to evaluate the risk involved in a transaction.

Important

PingOne Protect is supported in the following servers:

Advanced Identity Cloud

Use the official PingOne Protect nodes

PingAM 7.5 and later

Use the official PingOne Protect nodes

PingAM 7.2 - 7.4

Use the marketplace PingOne Protect nodes

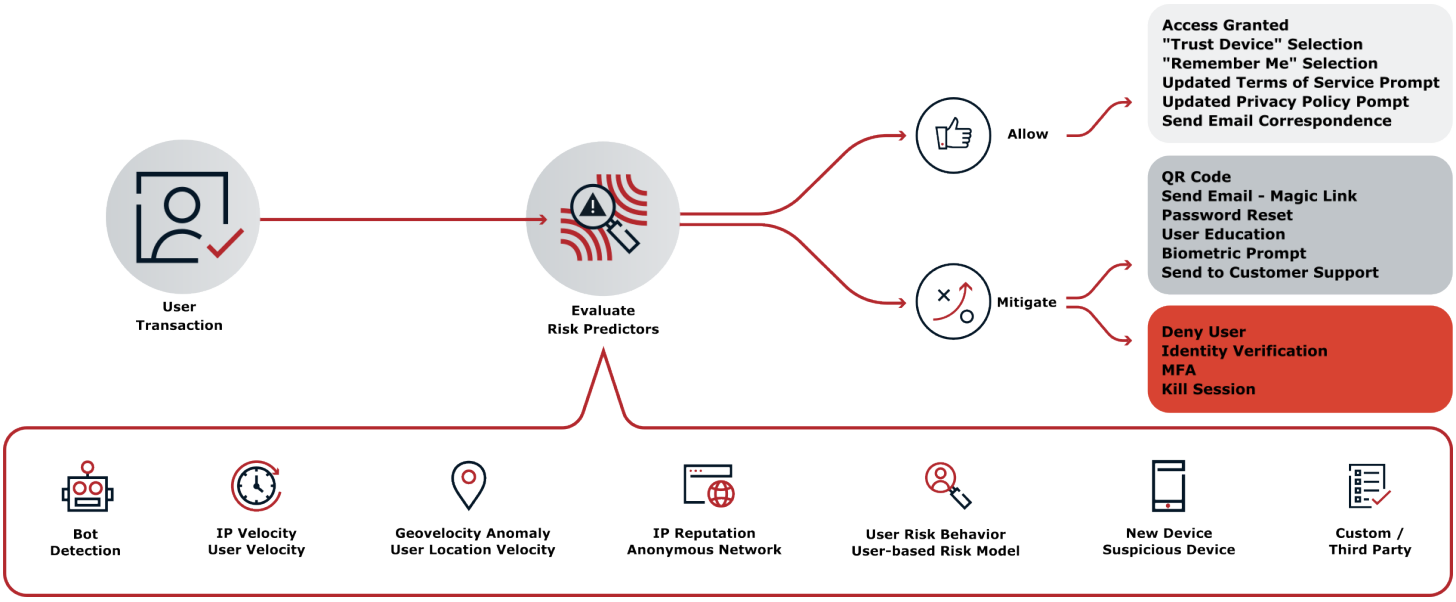


Figure 1. A flowchart illustrating how risk predictors evaluate many different data points.

You can instruct the Advanced Identity Cloud/PingAM Login Widget to gather information during a transaction. Your app can then collate this information together and request a risk evaluation from PingOne.

Based on the response from Protect, you can choose whether to allow or deny the transaction or perform additional mitigation, such as bot detection measures.

You can use the audit functionality in PingOne to view the risk evaluations:

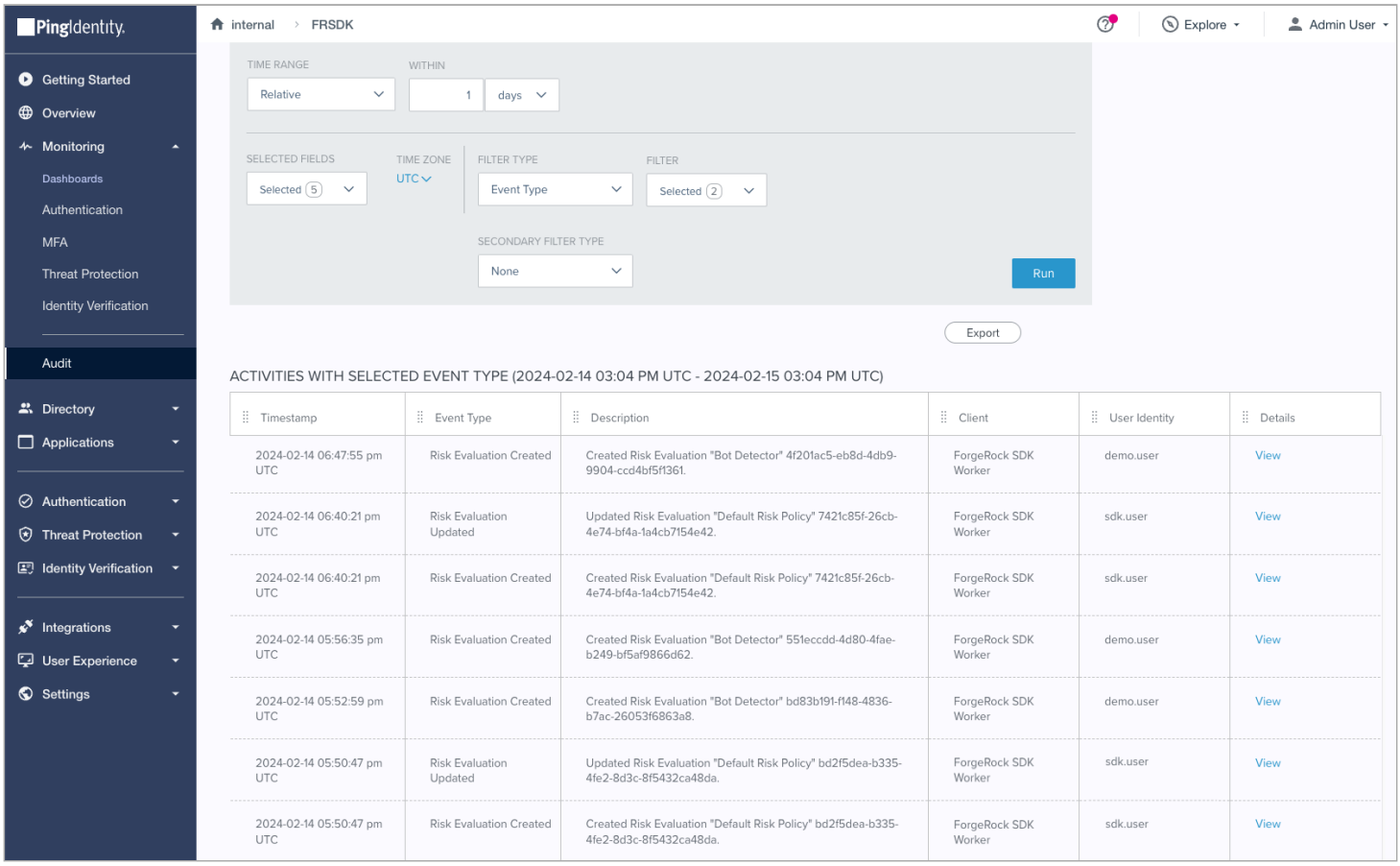


Figure 2. Risk evaluation records in the PingOne audit viewer.

Steps

Step 1. Set up the servers

In this step, you set up your PingOne Advanced Identity Cloud or PingAM server, and your PingOne instance to perform risk evaluations.

For example, you create a worker application in PingOne and configure your server to access it. You also create an authentication journey that uses the relevant nodes.

Step 2. Configure the Advanced Identity Cloud/PingAM Login Widget for PingOne Protect

With everything prepared, you can now configure the Advanced Identity Cloud/PingAM Login Widget to evaluate risk by using PingOne Protect.

Step 1. Set up the servers

In this step, you set up your PingOne Advanced Identity Cloud or PingAM server, and your PingOne instance to perform risk evaluations.

1. Create a worker application in PingOne
2. Configure the PingOne Worker service in your server
3. Configure a journey to perform PingOne Protect risk evaluations

Create a worker application in PingOne

To allow your server to access the PingOne administration API you must create a worker application in PingOne.

The worker application provides the client credentials your server uses to communicate with the PingOne admin APIs using the OpenID Connect protocol.

To create a worker application in PingOne:

1. In the PingOne administration console, navigate to **Applications** > **Applications**, and then click **Add** (+).
2. In the **Add Application** panel:
    1. In **Application name**, enter a unique identifier for the worker application.

       For example, `Ping (ForgeRock) SDK Worker`.
    2. Optionally, enter a **Description** for the application and select an **Icon**.

       These do not affect the operation of the worker application but do help you identify it in the list.
    3. In **Application Type**, select **Worker**.
    4. Click **Save**.
3. In the application properties panel for the worker application you created:
    1. On the **Roles** tab, click **Grant Roles**.
    2. On the **Available responsibilities** tab, select the **Identity Data Admin** row, and ensure the environment is correct.
    3. Click **Save**.
    4. On the **Overview** tab, ensure your worker application resembles the following image, and then enable it by using the toggle (1):

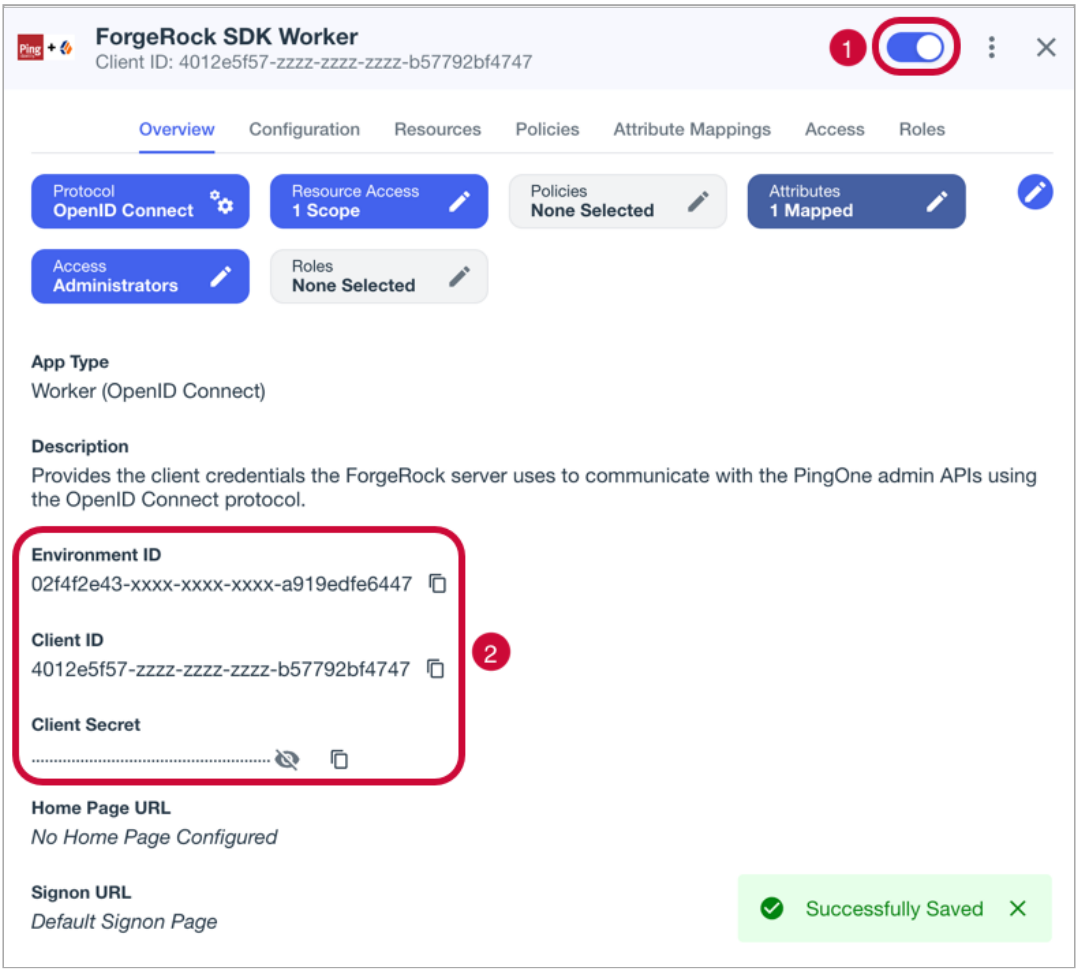


Figure 1. Example worker application in PingOne

5. Make a note of the **Environment ID**, **Client ID**, and **Client Secret** values (2).

   You need these values in the next step when you Configure the PingOne Worker service in your server.

Configure the PingOne Worker service in your server

After you create a worker application in PingOne, you must configure the PingOne Worker service in your server with the credentials.

Prerequisites

You need the following values from the PingOne Worker application you created in PingOne:

Client ID

Client ID of the worker application in PingOne.

Example: `6c7eb89a-66e9-ab12-cd34-eeaf795650b2`

Client Secret

Client secret of the worker application in PingOne.

> **Tip**
>
> Use the **Secret Mask** (👁) or **Copy to Clipboard** (📋) buttons to obtain the value in the PingOne administration console.

Example: `Ch15~o5Hm8N4_eS_m8~ARrV0KQAIQS6d.sJWe8TMXurEb~KWexY_p0gelR`

Environment ID

Identifier of the environment that contains the worker application in PingOne.

Example: `3072206d-c6ce-ch15-m0nd-f87e972c7cc3`

> **Important**
>
> The PingOne Worker Service requires a configured OAuth2 provider service in your server.
>
> - If you are using a self-managed AM server, you must configure the OAuth2 Provider service in a realm to expose the OAuth 2.0 endpoints and OAuth 2.0 administration REST endpoints.
> - The **OAuth2 provider service** is preconfigured in Advanced Identity Cloud.

Register the client secret in the server

You need to make the client secret of the worker application in PingOne available for use in the PingOne worker service.

Advanced Identity Cloud

If you are using Advanced Identity Cloud you will need to create an environment secret to hold the client secret value, as follows:

1. In the Advanced Identity Cloud admin UI, go to ⚙ **Tenant Settings** > **Global Settings** > **Environment Secrets & Variables**.
2. Click the **Secrets** tab.
3. Click **+ Add Secret**.
4. In the **Add a Secret** modal window, enter the following information:

Name	Enter a secret name. For example, `ping-protect-client-secret`. **Note** Secret names cannot be modified after the secret has been created.
Description	(optional) Enter a description of the purpose of the secret.
Value	Enter the **Client Secret** value you obtained when creating the worker application in PingOne. For example, `Ch15~o5Hm8N4_eS_m8~ARrV0KQAIQS6d.sJWe8TMXurEb~KWexY_p0gelR`. The field obscures the secret value by default. You can optionally click the visibility toggle (👁) to view the secret value as you enter it.

5. Click **Save** to create the variable.
6. Click **View Update**, check the details of the new secret, and then click **Apply Update**.

   Advanced Identity Cloud displays a final confirmation page.

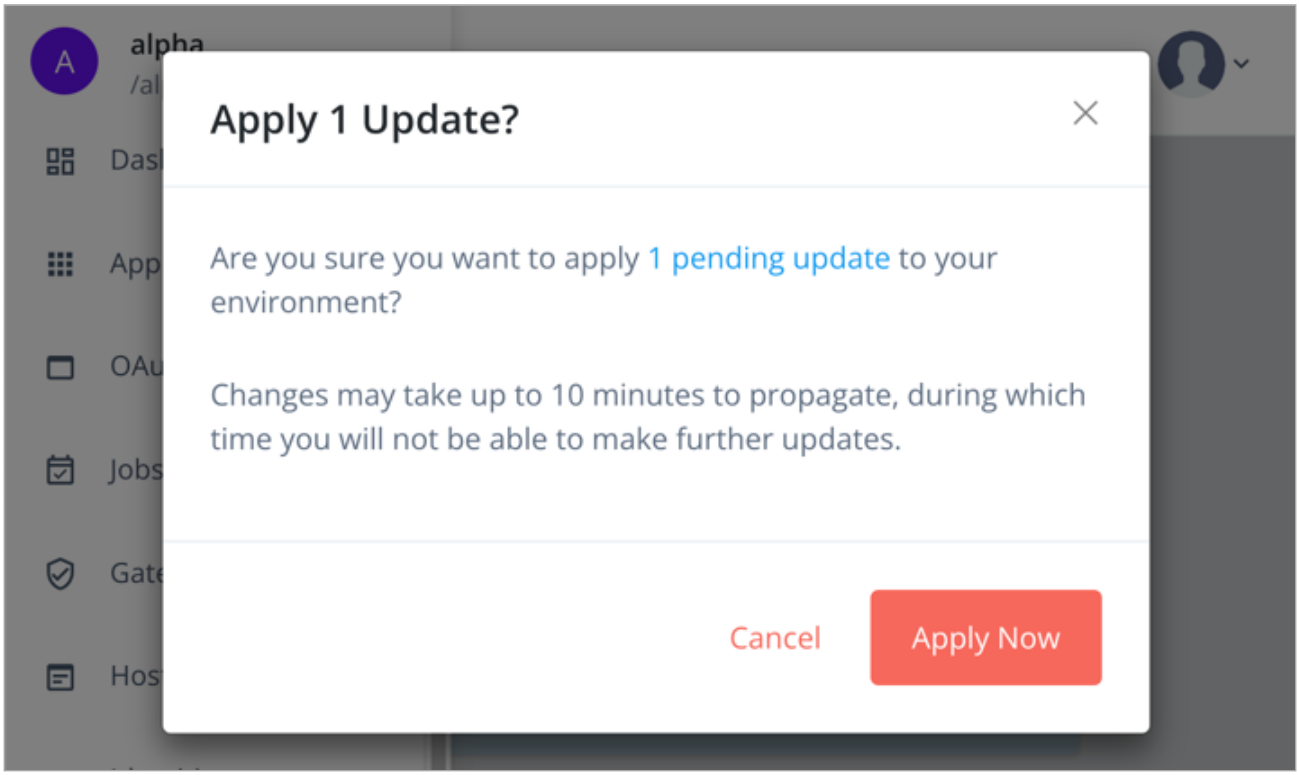


Figure 2. Apply updated secrets in Advanced Identity Cloud

7. Click **Apply Now**.

   Advanced Identity Cloud propagates the new secret and its value to all servers. You **must** wait until the secrets have propagated throughout the environment before attempting to use the secret.

   The **Environment Secrets & Variables** page displays the following message while the update is in progress:

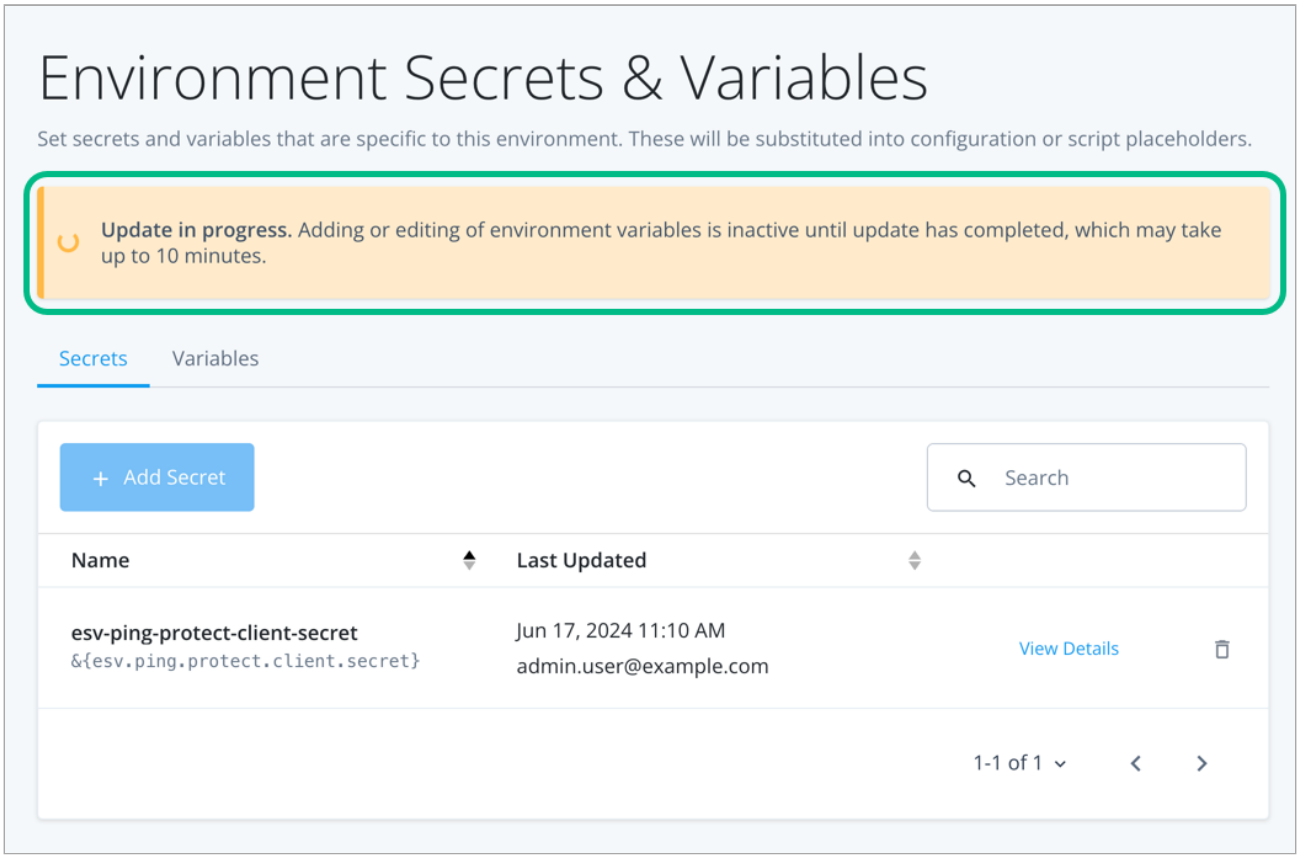


Figure 3. Propagating secrets in progress in Advanced Identity Cloud.

Self-managed AM

For information on adding secret values for use in services in a self-managed AM instance, refer to Create key aliases in the AM documentation.

Configure the PingOne worker service

To configure the PingOne worker service:

1. If you are using PingOne Advanced Identity Cloud, in the administration console navigate to **Native Consoles** > **Access Management**.
2. In the AM admin UI, click **Services**.
3. If the **PingOne Worker Service** is in the list of services, select it.
4. If you do not yet have a **PingOne Worker Service**:
    1. Click **+ Add a Service**.
    2. In **Choose a service type**, select `PingOne Worker Service`, and then click **Create**.
5. On the **Secondary Configurations** tab, click **+ Add a Secondary Configuration**.
6. On the **New workers configuration** page:
    1. Enter a **Name** for the configuration.

For example, `SDK PingOne Worker`.

You use this value when you configure an authentication journey that performs risk evaluations.

2. In **Client ID**, enter the client ID of the PingOne Worker application you created earlier.
3. In **Client Secret Label Identifier**, enter an identifier to create a specific secret label to represent the client secret of the worker application.

   For example, `workerAppClientSecret`.

   The secret label uses the template `am.services.pingone.worker.identifier.clientsecret` where identifier is the **Client Secret Label Identifier** value.

   This field can only contain characters `a-z`, `A-Z`, `0-9`, and `.` and can't start or end with a period.
4. In **Environment ID**, enter the environment ID containing the PingOne Worker application you created earlier.
5. Click **Create**

7. On the **Workers Configuration** page, ensure that the **PingOne API Server URL** and **PingOne Authorization Server URL** are correct for the region of your PingOne servers:

PingOne URLs by region

Region	Authorization URL	API URL
North America (Excluding Canada)	`https://auth.pingone.com`	`https://api.pingone.com/v1`
Canada	`https://auth.pingone.ca`	`https://api.pingone.ca/v1`
Europe	`https://auth.pingone.eu`	`https://api.pingone.eu/v1`
Asia-Pacific	`https://auth.pingone.asia`	`https://api.pingone.asia/v1`

8. Confirm your configuration resembles the image below, and then click **Save changes**.

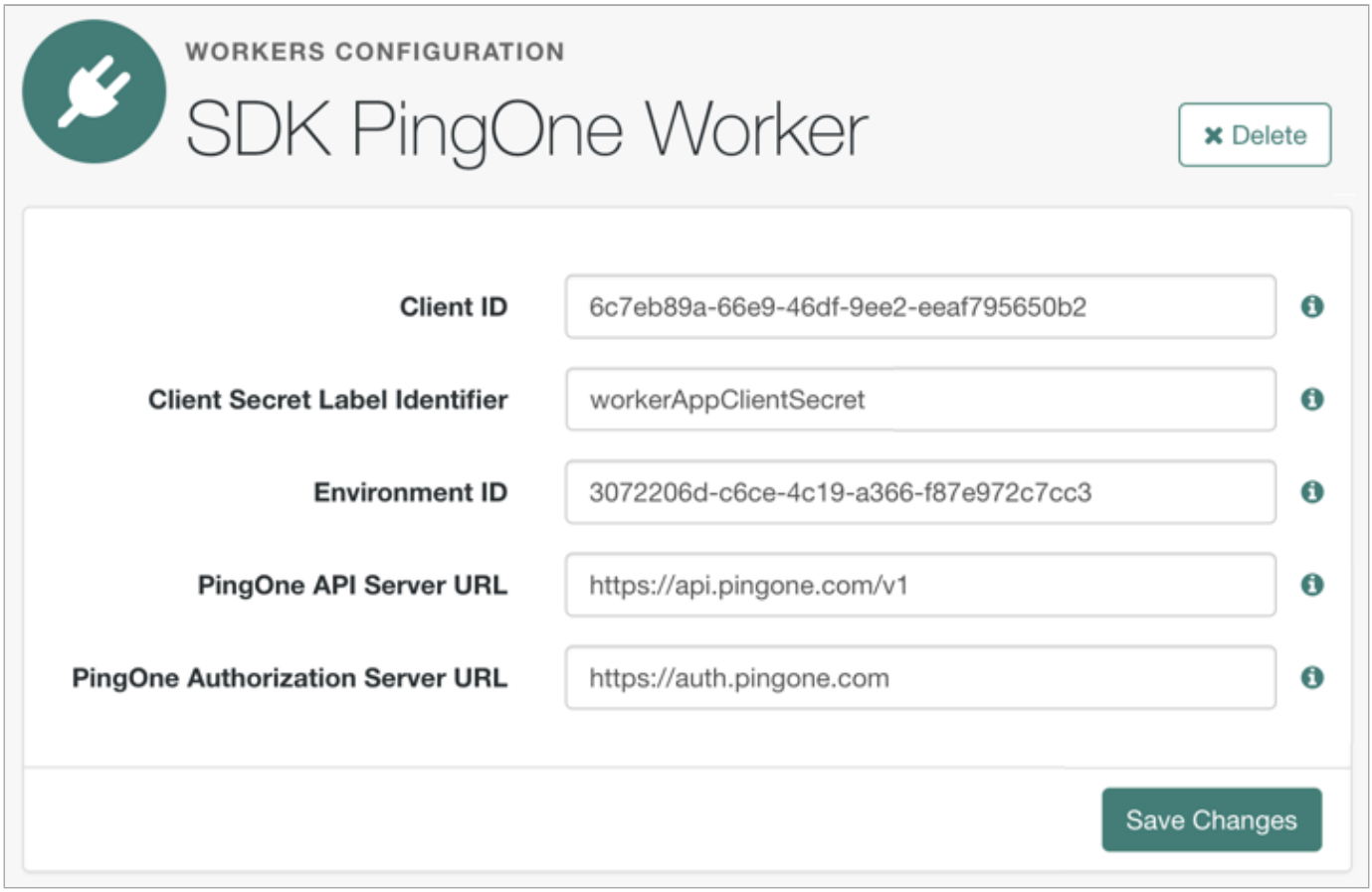


Figure 4. Example worker application in PingOne

Map the Client Secret Label Identifier to a secret

To make the client secret available to the PingOne Worker Service, you must map the secret to the ID created.

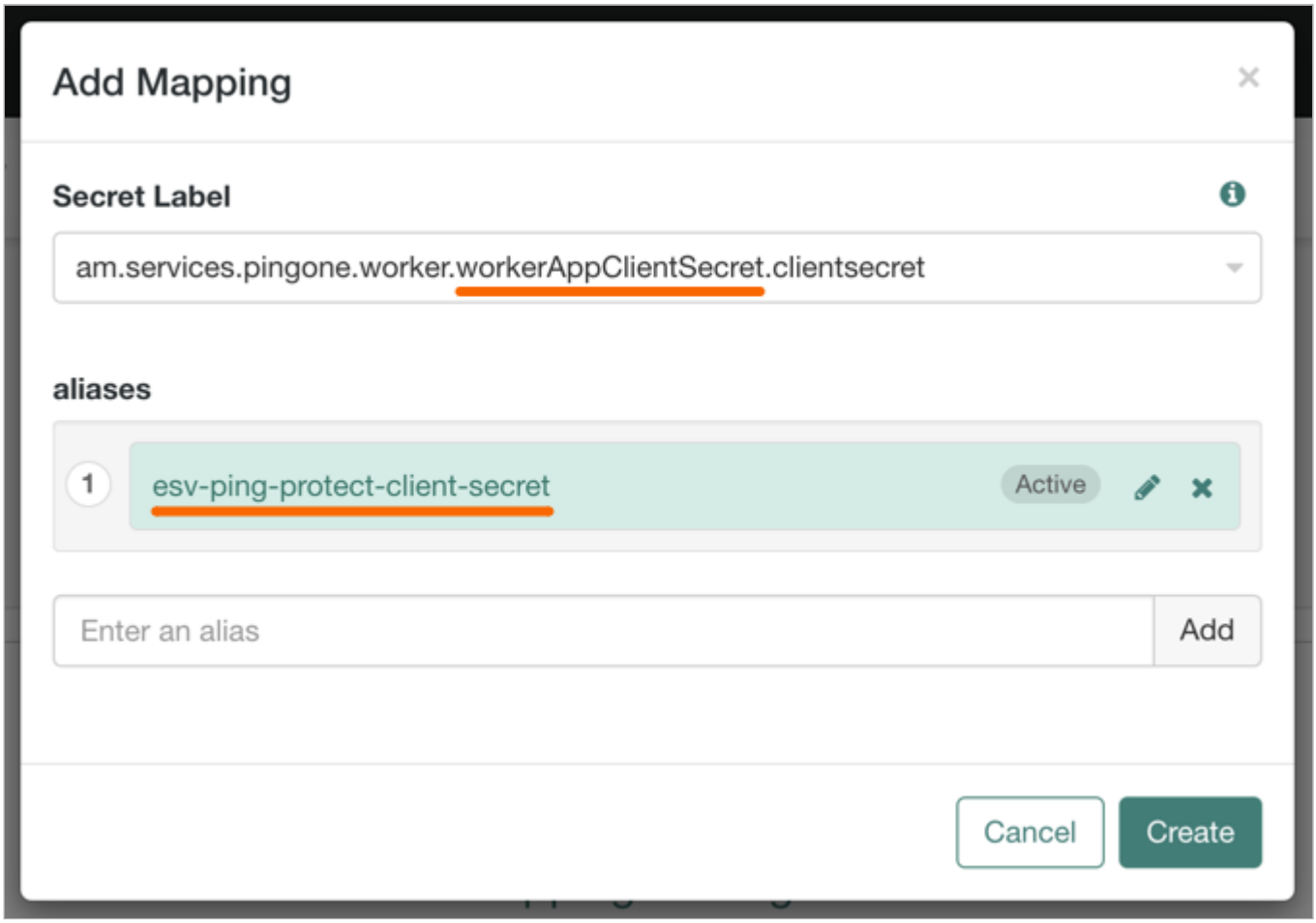
Map secrets in Advanced Identity Cloud

1. In the Advanced Identity Cloud admin UI, click **Native Consoles > Access Management**.
2. In the AM admin UI (native console), go to **Realm > Secret Stores**.
3. Click the **ESV** secret store, then click **Mappings**.
4. Click **+ Add Mapping**.
    1. In **Secret Label**, select the label generated when you entered the **Client Secret Label Identifier** previously.

       For example, `am.services.pingone.worker.workerAppClientSecret.clientsecret`.
    2. In **aliases**, enter the name of the ESV secret you created earlier, including the `esv-` prefix, and then click **Add**.

       For example, `esv-ping-protect-client-secret`

    The result resembles the following:



5. Click **Create**.

To learn more about mapping secrets and label identifiers in Advanced Identity Cloud, refer to Secret labels.

Map secrets in self-managed AM

To learn about mapping secrets in self-managed AM, refer to Map and rotate secrets.

You have now configured the PingOne Worker service in your server. You can now Configure a journey to perform PingOne Protect risk evaluations.

Configure a journey to perform PingOne Protect risk evaluations

To make risk evaluations in PingOne, you must configure an authentication journey in your server.

The following table covers the authentication nodes and callbacks for integrating your authentication journeys with PingOne Protect.

Node	Callback	Description
PingOne Protect Initialization node	PingOneProtectInitializeCallback	Instruct the embedded PingOne Signals SDK to start gathering contextual information.
PingOne Protect Evaluation node	PingOneProtectEvaluationCallback	Returns contextual information that the server can send to your PingOne Protect instance to perform a risk evaluation.
PingOne Protect Result node	Non-interactive	Inform the PingOne Protect instance about the status of the transaction.

In your server, log in as an administrator and create a new authentication journey similar to the following example:

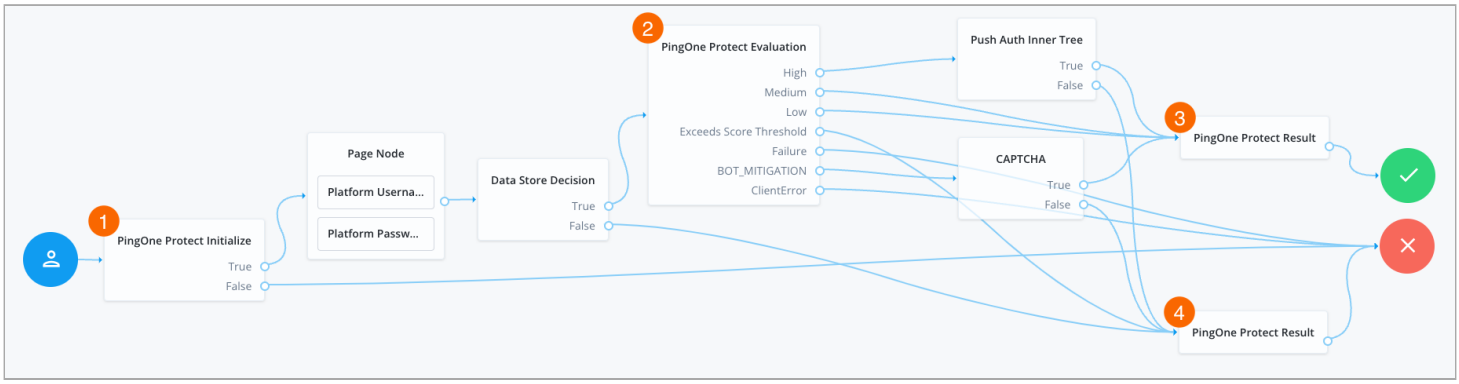


Figure 5. An example PingOne Protect journey

- The PingOne Protect Initialize node 1 instructs the SDK to initialize the PingOne Protect Signals API with the configured properties.

  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.

  > **Tip**
  >
  > You can initialize the PingOne Protect Signals API whenever you want to start collecting data. This could be at application startup, or when a particular page or view is visited.

- The user enters their credentials, which are verified against the identity store.
- The PingOne Protect Evaluation node 2 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.

`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.

- An instance of the PingOne Protect Result node 3 returns the `Success` result to PingOne, which can be viewed in the audit console to help with analysis and risk policy tuning.
- A second instance of the PingOne Protect Result node 4 returns the `Failed` result to PingOne, which can be viewed in the audit console to help with analysis and risk policy tuning.

You have now configured a suitable authentication journey in your server. You can now proceed to Step 2. Configure the Advanced Identity Cloud/PingAM Login Widget for PingOne Protect.

Step 2. Configure the Advanced Identity Cloud/PingAM Login Widget for PingOne Protect

Integrating the Advanced Identity Cloud/PingAM Login Widget with PingOne Protect enables you to perform risk evaluations during your customer's journey.

Complete the following tasks to fully integrate with PingOne Protect:

1. Initialize data collection
2. Pause and resume behavioral data capture
3. Return collected data for a risk evaluation

Initialize data collection

You must initialize the PingOne Signals SDK so that it collects the data needed to evaluate risk.

The earlier you can initialize the PingOne Signals SDK, the more data it can collect to make a risk evaluation.

There are two options for initializing the PingOne Signals SDK in the Advanced Identity Cloud/PingAM Login Widget:

1. The Advanced Identity Cloud/PingAM Login Widget automatically initializes the PingOne Signals SDK on receipt of a `PingOneProtectInitializeCallback` callback from a journey you have started.

2. Manually initialize the PingOne Signals SDK, import the module and pass in any configuration parameters you need, as follows:

```
import Widget, { configure, journey, protect } from '@forgerock/login-widget';

new Widget({ target: widgetEl });

// Start PingOne Protect Signals SDK
await protect.start({
  envId: 3072206d-c6ce-ch15-m0nd-f87e972c7cc3,
  behavioralDataCollection: true,
  consoleLogEnabled: true,
});
```

The PingOne Signals SDK supports a number of parameters which you can supply yourself, or are contained in the `PingOneProtectInitializeCallback` callback.

Show PingOne Signals SDK `start` parameters

Parameter			Description
Android	***iOS***	***JavaScript***	
`envId`			Required. Your PingOne environment identifier.
`deviceAttributesToIgnore`			Optional. A list of device attributes to ignore when collecting device signals. For example, `AUDIO_OUTPUT_DEVICES` or `IS_ACCEPT_COOKIES`.
`isBehavioralDataCollection`	`behavioralDataCollection`		When `true`, collect behavioral data. Default is `true`.
`isConsoleLogEnabled`	`consoleLogEnabled`		When `true`, output SDK log messages in the developer console. Default is `false`.
`isLazyMetadata`	`lazyMetadata`		When `true`, calculate metadata on demand rather than automatically after calling `start`. Default is `false`.
N/A		`deviceKeyRsynCIntervals`	Number of days that device attestation can rely upon the device fallback key. Default: `14`
N/A		`disableHub`	When `true`, the client stores device data in the browser's `localStorage` only. When `false` the client uses an iframe. Default is `false`.

N/A	`disableTags`	When `true`, the client does not collect tag data. Tags are used to record the pages the user visited, forming a browsing history. Default is `false`.
N/A	`enableTrust`	When `true`, tie the device payload to a non-extractable crypto key stored in the browser for content authenticity verification. Default is `false`.
N/A	`externalIdentifiers`	Optional. A list of custom identifiers that are associated with the device entity in PingOne Protect.
N/A	`hubUrl`	Optional. The iframe URL to use for cross-storage device IDs.
N/A	`waitForWindowLoad`	When `true`, initialize the SDK on the `load` event, instead of the `DOMContentLoaded` event. Default is `true`.
N/A	`universalDeviceIdentification`	Optional. When `true`, device data in the payload returned to the server is provided as a signed JWT.
N/A	`agentIdentification`	Set to `true` when using risk policies that contain the PingID Device Trust predictor. Default is `false`.
N/A	`agentTimeout`	If you have enabled `agentIdentification`, use `agentTimeout` to specify a connection timeout, in milliseconds. Specifying a value overrides the default. Default is `1000`.
N/A	`agentPort`	If you have enabled `agentIdentification`, use `agentPort` to specify the port for connecting to the trust agent. Specifying a value overrides the default. Default is `9400`.

Return collected data for a risk evaluation

To perform risk evaluations, the PingOne server requires the captured data.

There are two options for returning data in the Advanced Identity Cloud/PingAM Login Widget:

1. On receipt of a `PingOneProtectEvaluationCallback` callback within a journey, the Advanced Identity Cloud/PingAM Login Widget automatically returns the captured data.
2. Use the `getData()` method to manually return the captured data:

```
import Widget, { configure, journey, protect } from '@forgerock/login-widget';

new Widget({ target: widgetEl });

// Start PingOne Protect Signals SDK
await protect.start({
  envId: 3072206d-c6ce-ch15-m0nd-f87e972c7cc3,
  behavioralDataCollection: true,
  consoleLogEnabled: true,
});

// Return gathered data to the server
await protect.getData();
```

Pause and resume behavioral data capture

The PingOne Protect Signals SDK can capture behavioral data, such as how the user interacts with the app, to help when performing evaluations.

There are scenarios where you might want to pause the collection of behavioral data. For example, the user might not be interacting with the app, or you only want to use device attribute data to be considered when performing PingOne Protect evaluations. You can then resume behavioral data collection when required.

There are two options for pausing and resuming behavioral data capture in the Advanced Identity Cloud/PingAM Login Widget:

1. The `PingOneProtectEvaluationCallback` callback can include a flag to pause or resume behavioral capture, which the Advanced Identity Cloud/PingAM Login Widget automatically responds to.
2. Use the `pauseBehavioralData()` and `resumeBehavioralData()` methods to manually pause or resume the capture of behavioral data:

```
import Widget, { configure, journey, protect } from '@forgerock/login-widget';

new Widget({ target: widgetEl });

// Start PingOne Protect Signals SDK
await protect.start({
  envId: 3072206d-c6ce-ch15-m0nd-f87e972c7cc3,
  behavioralDataCollection: true,
  consoleLogEnabled: true,
});

// Return gathered data to the server
await protect.getData();

// Pause behavioral data collection
protect.pauseBehavioralData();

// Resume behavioral data collection
protect.resumeBehavioralData();
```

Integrate the Advanced Identity Cloud/PingAM Login Widget into a React app

In this tutorial, you will learn how to integrate the Advanced Identity Cloud/PingAM Login Widget into a simple React app that you scaffold using Vite.

You install the Advanced Identity Cloud/PingAM Login Widget using `npm`, add an element to the HTML file for mounting the modal form factor, and wrap the app's CSS in a layer.

With the app prepared, you then configure the widget at the top level of your app, import and instantiate its components, and start a journey. You subscribe to the events the Advanced Identity Cloud/PingAM Login Widget emits so that the app can respond and display the appropriate UI.

When you have successfully authenticated a user, you add code to log the user out and invalidate their tokens, as well as update the UI to alter the button state.

Requirements

1. Node 18+
2. NPM 8+

Configure your server

Configure your PingOne Advanced Identity Cloud or self-managed PingAM server by following the steps in the Advanced Identity Cloud/PingAM Login Widget Tutorial.

- When creating the OAuth 2.0 client, add the URL that you are using to host the app to the **Sign-In URLs** property.

> **Tip**
>
> The URL is output to the console when you run the `npm run dev` command, and defaults to `http://localhost:5173/`

- If your instance has the default **Login** journey, you can use that instead of creating a new journey as described in the tutorial.

Create a Vite app

1. In a terminal window, create a Vite app with React as the template:

   ```
   npm create vite@latest login-widget-react-demo -- --template react
   ```

   For more information, refer to Scaffolding Your First Vite Project in the Vite developer documentation.

2. When completed, change to the new directory, for example `login-widget-react-demo`, and then install dependencies with `npm`:

   ```
   npm install
   ```

3. Run the app in developer mode:

   ```
   npm run dev
   ```

4. In a web browser, open the URL output by the previous command to render the app. The URL is usually `http://localhost:5173`

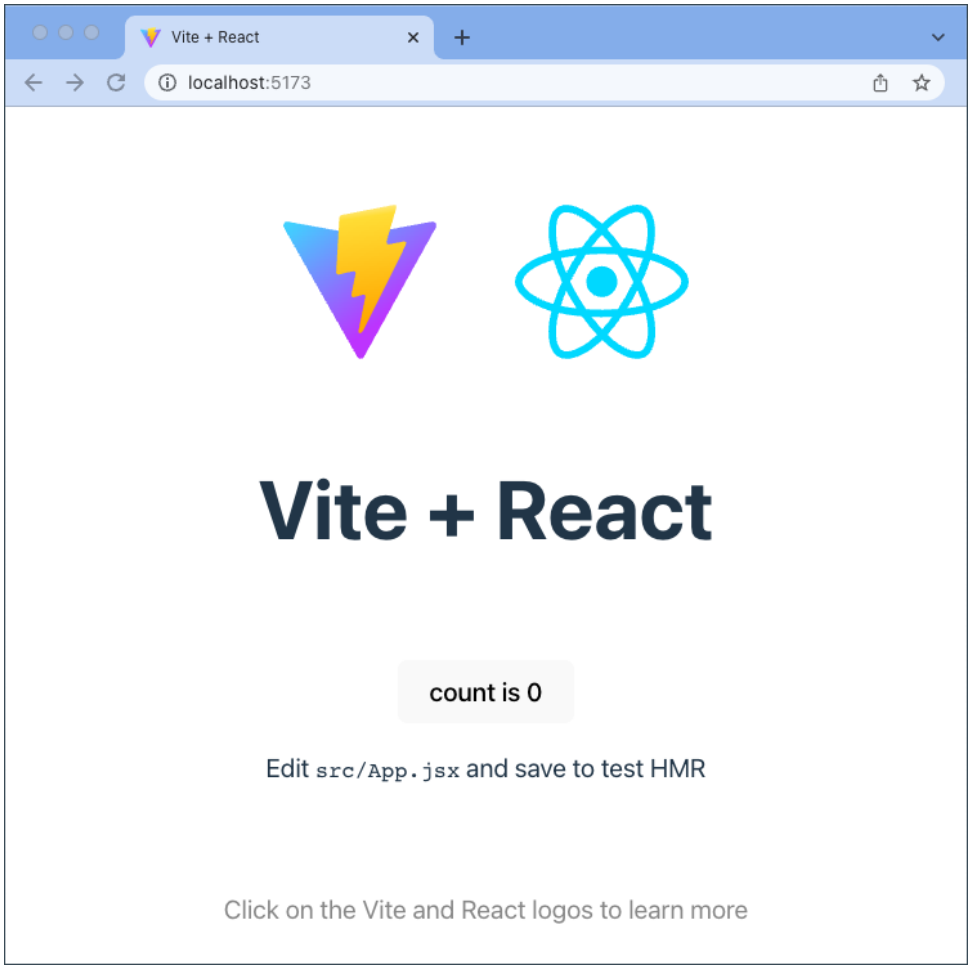


Figure 1. Example Vite + React app

> **Tip**
>
> Use a different browser for development testing than the one you use to log into PingOne Advanced Identity Cloud or PingAM.
> This prevents admin user and test user sessions colliding and causing unexpected authentication failures.

Install the Advanced Identity Cloud/PingAM Login Widget

In a new terminal window, install the Advanced Identity Cloud/PingAM Login Widget using `npm`:

```
npm install @forgerock/login-widget
```

Prepare the HTML

In your preferred IDE, open the directory where you created the Vite app, and then open the `index.html` file.

To implement the modal form factor, create a root element to contain the Advanced Identity Cloud/PingAM Login Widget.

Add `<div id="widget-root"></div>` toward the bottom of the `<body>` element but before the `<script>` tag:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <!-- Widget mount point -->
    <div id="widget-root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Prepare the CSS

You should wrap the app's CSS using `@layer`. This helps control the CSS cascade.

To wrap the app's CSS, in your IDE, open `src/index.css` and `src/App.css` and wrap them both with the following code:

```
@layer app {
    /* existing styles */
    #root {
        max-width: 1280px;
        margin: 0 auto;
        padding: 2rem;
        text-align: center;
    }

    .logo {
        height: 6em;
        padding: 1.5em;
        will-change: filter;
        transition: filter 300ms;
    }
    /* ... */
}
```

You can then specify the order of the various layers as follows:

```
<style>
  @layer app;
  /* List the Widget layers last */
  @layer 'fr-widget.base';
  @layer 'fr-widget.utilities';
  @layer 'fr-widget.components';
  @layer 'fr-widget.variants';
</style>
```

Configure the Advanced Identity Cloud/PingAM Login Widget at app boot

`configure()` is asynchronous and must complete before any other Widget API is called. The cleanest place to `await` it in a Vite + React app is in `src/main.jsx`, before rendering the app.

In your IDE, open `src/main.jsx` and update it to import the widget CSS, `await configure()`, and only then render the app:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { configure } from '@forgerock/login-widget';
import '@forgerock/login-widget/widget.css';
import App from './App.jsx';
import './index.css';

await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    scope: 'openid profile email address',
  },
});

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

> **Note**
>
> Vite supports top-level `await` in modules by default.
> If you use a bundler that does not, wrap the `configure()` call and the render in an async IIFE.

Instantiate and mount the Advanced Identity Cloud/PingAM Login Widget

Now that the widget is configured at app boot, `App.jsx` can instantiate it against the widget root element you added to `index.html`.

Open `src/App.jsx`. Import `Widget` from `@forgerock/login-widget` and `useEffect` from React:

```
import React, { useEffect, useState } from 'react';
import Widget from '@forgerock/login-widget';
```

Inside the `App` component, write a `useEffect` with an empty dependency array to instantiate the widget once when the component mounts and destroy it on unmount:

```
function App() {

  useEffect(() => {
    const widget = new Widget({ target: document.getElementById('widget-root') });

    return () => {
      widget.$destroy();
    };
  }, []);

  // ...
```

> **Note**
>
> The empty dependency array tells React this effect has no dependencies and should only run once.
> The returned function destroys the widget when the component unmounts, so a remount does not add two widgets to the DOM.

In your browser, the app doesn't look any different. This is because the widget, by default, is invisible at startup.

To ensure it is working as expected, inspect the DOM in the browser developer tools.

Open the `<div id="widget-root">` element in the DOM, and you should see the Advanced Identity Cloud/PingAM Login Widget mounted within it:

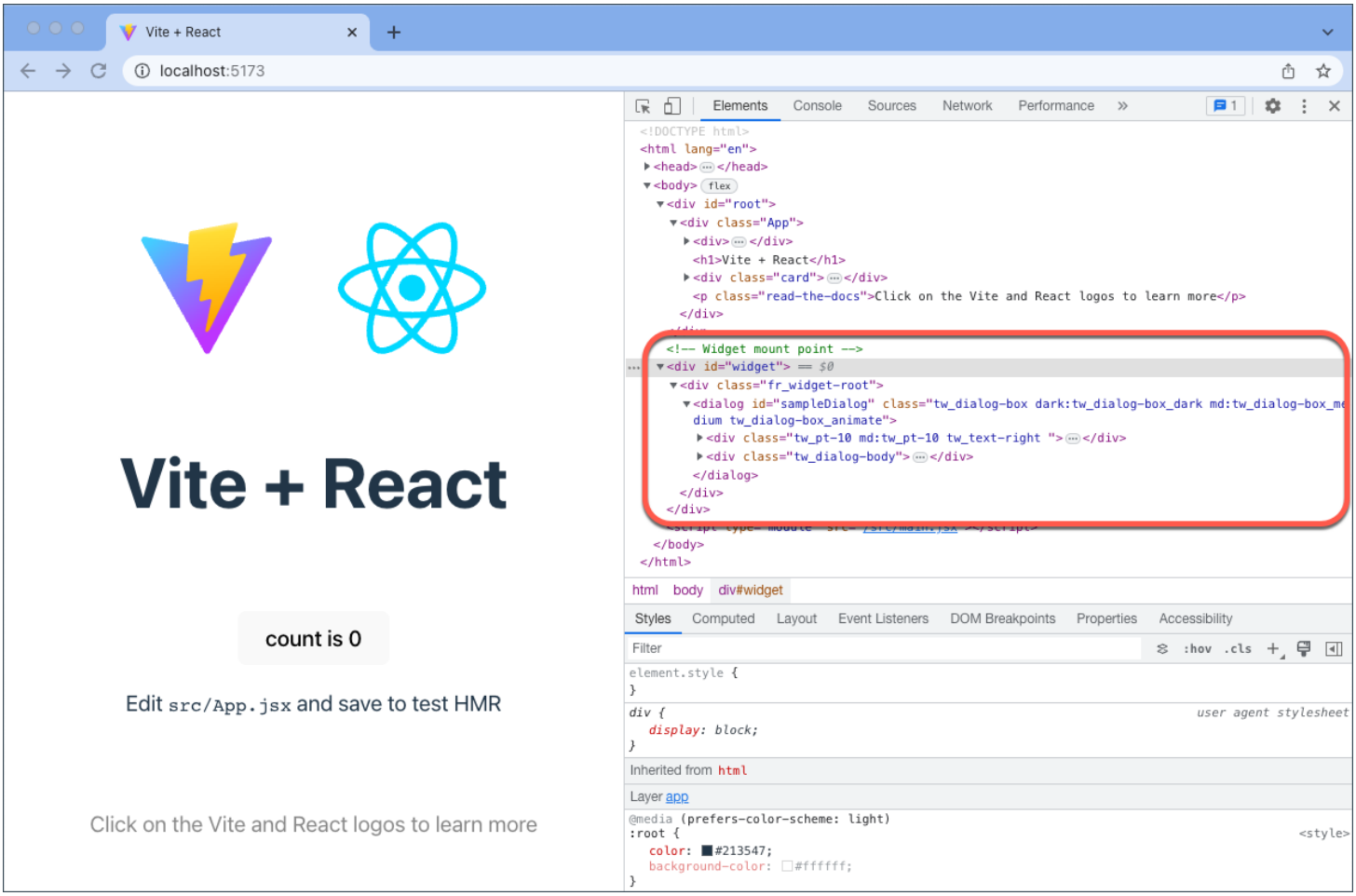


Figure 2. Instantiated and mounted modal form factor

Controlling the component

An invisible Advanced Identity Cloud/PingAM Login Widget is not all that useful, so your next task is to pull in the `component` module to manage the component's events.

Add `component` to the list of imports and call it inside the component:

```
import Widget, { component } from '@forgerock/login-widget';

// ...

function App() {
  const componentEvents = component();

  // ...
```

Now that you have a reference to the `component` events observable, you can trigger an event such as `open`, and you can also listen for events.

Repurpose the existing **count is 0** button within the `App` component.

1. Within the button's `onClick` handler, change the `setCount` function to `componentEvents.open`
2. Change the button text to read "Login"

The result resembles the following:

```
<button
  onClick={() => {
    componentEvents.open();
  }}>
  Login
</button>
```

You can now revisit your test browser and click the **Login** button. The modal opens and displays a "spinner" animating on repeat.

This is expected, because the Advanced Identity Cloud/PingAM Login Widget does not yet have a journey to render.

Click the button in the top-right to close the modal. The modal should be dismissed as expected.

Start a journey

Now that you have the modal mounted and functional, add the `journey` module so you can start an authentication flow.

Add `journey` to the imports and call it inside the component to get a `journeyEvents` observable:

```
import Widget, { component, journey } from '@forgerock/login-widget';

// ...

function App() {
  const componentEvents = component();
  const journeyEvents = journey();

  // ...
```

Within the **Login** button's `onClick` handler, call `journeyEvents.start()` so that clicking the button both opens the modal and requests the first authentication step from the server:

```
<button onClick={() => {
    journeyEvents.start();
    componentEvents.open();
  }}>
  Login
</button>
```

You are now capable of authenticating a user. With an existing user, authenticate as that user and see what happens.

If successful, you'll notice the modal dismisses itself but your app is not capturing anything from this action. Proceed to the next step to capture user data.

Authenticating a user

There are multiple ways to capture the event of a successful login and access the user information.

In this guide, you use the `journeyEvents` observable created previously.

Within the existing `useEffect` function:

1. Call the `subscribe` method and assign its return value to an unsubscribe variable
2. Pass in a function that logs the emitted events to the console
3. Call the unsubscribe function within the return function of `useEffect`

```
// ...

useEffect(() => {
  const widget = new Widget({ target: document.getElementById('widget-root') });

  const journeyEventsUnsub = journeyEvents.subscribe((event) => {
    console.log(event);
  });

  return () => {
    widget.$destroy();
    journeyEventsUnsub();
  };
}, []);
```

> **Note**
>
> Unsubscribing from the observable is important to avoid memory leaks if the component mounts and unmounts frequently.

Revisit your app in the test browser, but remove all of the browser's cookies and Web Storage to ensure you have a fresh start.

> **Tip**
>
> In Chromium browsers, you can find it under the "Application" tab of the developer tools.
> In Firefox and Safari, you can find it under the "Storage" tab.

Once you have deleted all the cookies and storage, refresh the browser and try to log in your test user.

You will notice in the developer tools console that a lot of events are emitted.

Initially, you may not have much need for all this data, but over time, this information might become more valuable to you.

To narrow down all of this information, capture just one piece of the event: the user response after successfully logging in.

To do that, you can add a conditional, as follows:

- Add an `if` condition within the `subscribe` callback function that tests for the existence of the user response.

```
const journeyEventsUnsub = journeyEvents.subscribe((event) => {
  if (event.user.response) {
    console.log(event.user.response);
  }
});
```

With the above condition, the Advanced Identity Cloud/PingAM Login Widget only writes out the user information when it's *truthy*. This helps narrow down the information to what is useful right now.

Remove all the cookies and Web Storage again and refresh the page. Try logging in again, and you should see only one log of the user information when it's available:

Example `user.event.response` output

```
{
    email: 'sdk.demo-user@example.com',
    sub: '54c77653-dc88-48fb-ac6b-d5078ebe9fb0',
    subname: '54c77653-dc88-48fb-ac6b-d5078ebe9fb0'
}
```

Next, use React's `useState` hook to save the user information.

1. Add a `useState` call at the top of the `App` component, defaulting to `null`
2. Change the condition from just truthy to `userInfo !== event.user.response`
3. Replace the `console.log` with the `setUserInfo` function
4. Add the `userInfo` variable in the dependency array of the `useEffect`

The top part of your `App` function component should resemble the following:

```
function App() {
  const [userInfo, setUserInfo] = useState(null);
  const componentEvents = component();
  const journeyEvents = journey();

  useEffect(() => {
    // Instantiate the Widget and assign it to a variable
    const widget = new Widget({ target: document.getElementById('widget-root') });

    // Subscribe to journey observable and assign unsubscribe function to variable
    const journeyEventsUnsub = journeyEvents.subscribe((event) => {
      if (userInfo !== event.user.response) {
        setUserInfo(event.user.response);
      }
    });

    // Return a function that destroys the Widget and unsubscribes from the journey observable
    return () => {
      widget.$destroy();
      journeyEventsUnsub();
    };
  }, [userInfo]);

  // ...
```

Note

The condition comparing `userInfo` to `event.user.response` reduces the number of times the `setUserInfo` is called as it will now only be called if what is set in the hook is different than what is emitted from the Advanced Identity Cloud/PingAM Login Widget.

Now that you have the user data set into the React component, print it out into the DOM.

1. Replace the paragraph tag containing the text `Edit <code>src/App.jsx</code> and save to test HMR` with a `<pre>` tag
2. Within the `<pre>` tag, write a pair of braces: `{}`
3. Within these braces, use the `JSON.stringify` method to serialize the `userInfo` value

Your JSX should now look like this:

```
<pre>{JSON.stringify(userInfo, null, ' ')}</pre>
```

Tip

The `null` and `' '` (literal space character) help format the JSON to be more reader-friendly.

After clearing the browser data, try logging the user in and observe the user info get rendered onto the page after success.

Figure 3. User info displaying after successful log in

Logging a user out

The final step is to log the user out, clearing all the user-related cookies, storage, and cache.

To do this, add the `user` module to the list of imports from the Advanced Identity Cloud/PingAM Login Widget:

```
import Widget, { component, journey, user } from '@forgerock/login-widget';
```

Next, configure the app to display the button as a **Login** button when the user has not yet authenticated and a **Logout** button when the user has already logged in:

1. Wrap the button element with braces containing a ternary, using the *falsiness* of the `userInfo` as the condition
2. When no `userInfo` exists—the user is logged out—render the **Login** button
3. Write a **Logout** button with an `onClick` handler to run the `user.logout` function

The resulting JSX should resemble this:

```
<h1>Vite + React</h1>
<div className="card">
  {
    !userInfo ? (
      <button
        onClick={() => {
          journeyEvents.start();
          componentEvents.open();
        }}>
        Login
      </button>
    ) : (
      <button
        onClick={() => {
          user.logout();
        }}>
        Logout
      </button>
    )
  }
  <pre>{JSON.stringify(userInfo, null, ' ')}</pre>
</div>
```

Tip

You do not have to add code to reset the `userInfo` with the `setUserInfo` function, because you are already "listening" to events emitted from the `user` observable nested within the `journeyEvents` subscribe.

If your app is already reacting to the presence of user info, it should be rendering the **Logout** button already. Click it and observe the application reacting.

You should now be able to log a user in and out, with the app reacting to the changes in state.

API reference

This page lists the modules that the Advanced Identity Cloud/PingAM Login Widget provides for use in your apps.

Widget

This is a compiled Svelte class. This is what instantiates the component, mounts it to the DOM, and sets up all the event listeners.

```
import Widget from '@forgerock/login-widget';

// Instantiate Widget
const widget = new Widget({
  target: widgetRootEl, // REQUIRED; Element mounted in DOM
  props: {
    type: 'modal', // OPTIONAL; "modal" or "inline"; "modal" is default
  },
});

// OPTIONAL; Remove widget from DOM and destroy component listeners
widget.$destroy();
```

Important

Call `$destroy()` if you instantiate the Advanced Identity Cloud/PingAM Login Widget within a part of your application frequently created and destroyed.
We strongly encourage you to instantiate the modal form factor of the Advanced Identity Cloud/PingAM Login Widget high up in your application code. Instantiate it close to the top-level file in a component that is created once and preserved.

Configuration

The Advanced Identity Cloud/PingAM Login Widget requires the URL of your server's `.well-known/openid-configuration` endpoint. If you use OAuth/OIDC tokens, user info, or logout, it also requires an OIDC client configuration.

For information on setting up your server for use with the Advanced Identity Cloud/PingAM Login Widget, refer to Prerequisites.

To provide these settings, import and `await` the async `configure()` function. Call it once at the top level of your application, before any other Widget API.

```
import { configure } from '@forgerock/login-widget';

// configure() is async, so await it before calling any other Widget API
await configure({
  // REQUIRED; the well-known URL, shared by the journey and OIDC clients
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  // REQUIRED if you use OAuth/OIDC tokens, user info, or logout
  oidcClient: {
    clientId: 'sdkPublicClient',
    redirectUri: `${window.location.origin}/callback`,
    // OPTIONAL; defaults to 'openid'
    scope: 'openid profile email address',
  },
  // OPTIONAL; see dedicated sections below
  captcha: {},
  content: {},
  journeys: {},
  links: {},
  style: {},
});
```

Important

`configure()` is asynchronous. Always `await` it, since the Advanced Identity Cloud/PingAM Login Widget constructs its internal clients during this call. Any Widget API called before `configure()` resolves throws an error.

Content configuration options

Use the `content` configuration element to pass custom text content to the Advanced Identity Cloud/PingAM Login Widget, replacing its default values.

You can use this method to localize user interface text for different regions. Learn more in Localizing the widget.

Example `content` configuration

```
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  content: {
    userName: 'Identifier',
    passwordCallback: 'Passphrase',
    nextButton: "Let's go!",
  },
});
```

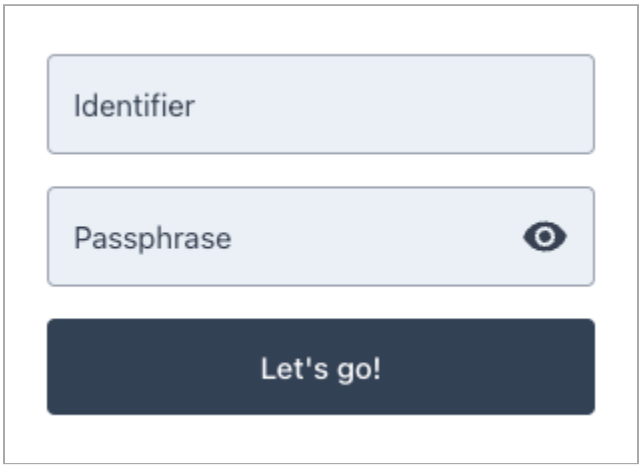


Figure 1. Result of example content configuration

For a list of the content you can override, refer to the en-us locale file in the Advanced Identity Cloud/PingAM Web Login Framework repository.

Links configuration options

Use the `links` configuration element to set the full canonical URL to your terms and conditions page.

This should be a page hosted on your website or elsewhere within your app. Users are sent to this URL if they click to view the terms and conditions in the Advanced Identity Cloud/PingAM Login Widget.

This supports the `TermsAndConditionsCallback` often used in registration journeys.

Example `links` configuration

```
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-configuration',
  links: {
    termsAndConditions: 'https://example.com/terms',
  },
});
```

Style configuration options

Use the `style` configuration element to configure the look and feel of the Advanced Identity Cloud/PingAM Login Widget. This allows you to choose the type of labels used or provide a logo for the modal.

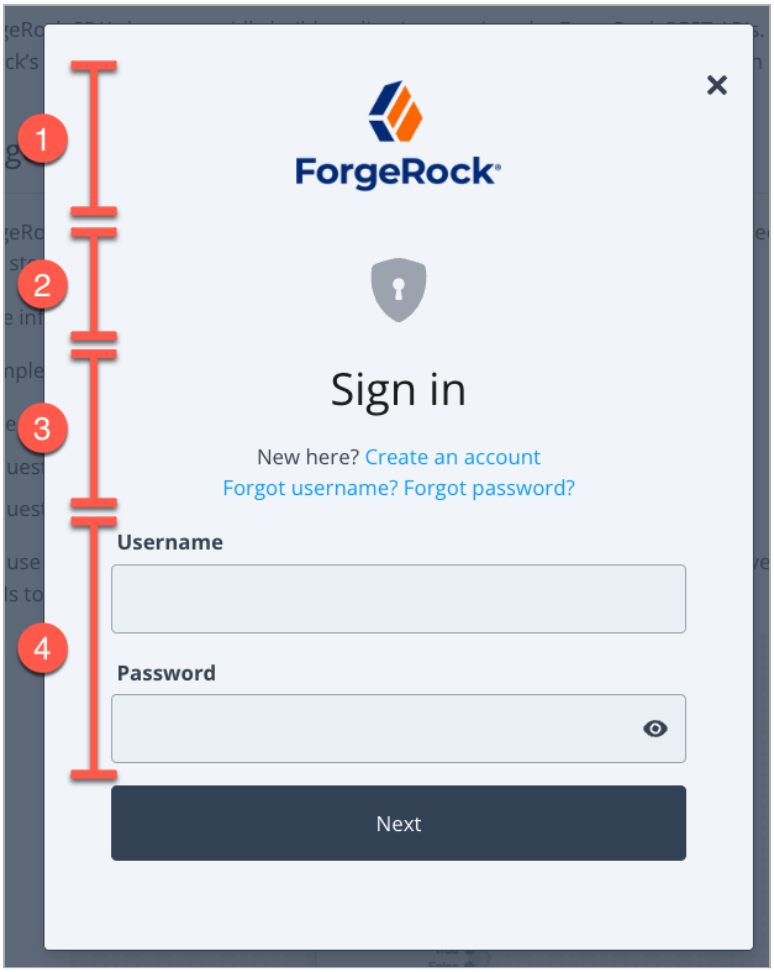


Figure 2. Use the `style` property to control aspects of the display

Key:

1. Use `style/logo` to add images for use in dark or light modes
2. Set `style/stage/icon` to `true` to render UI specific to certain **stage** parameter values. Supported **stage** values are:
    - `OneTimePassword` - enable the Advanced Identity Cloud/PingAM Login Widget to display one-time password entry forms correctly.
    - `DefaultRegistration` - adds UI elements to the display most suitable for user self-registration forms.
    - `DefaultLogin` - adds UI elements to the display most suitable for user log in forms.
3. A section that displays the **Page Header** and **Page Description** fields from the page node configuration
4. To float labels above their respective fields, set `style/labels` to `floating`
5. Use `style/showPassword` to control how the password visibility toggle is rendered

Adding logos and enabling icons

```
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  style: {
    checksAndRadios: 'animated', // OPTIONAL; choices are 'animated' or 'standard'
    labels: 'floating', // OPTIONAL; choices are 'floating' or 'stacked'
    showPassword: 'button', // OPTIONAL; choices are 'none', 'button', or 'checkbox'
    logo: {
      // OPTIONAL; only used with modal form factor
      dark: 'https://example.com/img/white-logo.png', // OPTIONAL; used if theme has a dark variant
      light: 'https://example.com/img/black-logo.png', // REQUIRED if logo property is provided; full URL
      height: 300, // OPTIONAL; number of pixels for providing additional controls to logo display
      width: 400, // OPTIONAL; number of pixels for providing additional controls to logo display
    },
    sections: {
      // OPTIONAL; only used with modal form factor
      header: false, // OPTIONAL; separate the logo section from the rest of the modal
    },
    stage: {
      icon: true, // OPTIONAL; displays generic icons for the provided stages
    },
  },
});
```

Note

The `logo` and `sections` properties only apply to the "modal" form factor and not the "inline".

`showPassword` *values*

Value	Description
`none`	The Advanced Identity Cloud/PingAM Login Widget does not render a way to reveal the password. This is the previous, pre-2.0 behavior.
`button`	Renders a button inside the password field that toggles visibility when clicked. This is the default.
`checkbox`	Renders a checkbox beneath the password field that toggles visibility when checked.

Add a header section

Enabling the header section separates the logo or branding from the journey form.

If you set `header: true` within the `style/sections` property, the modal uses a section with a separating line, and extra space:

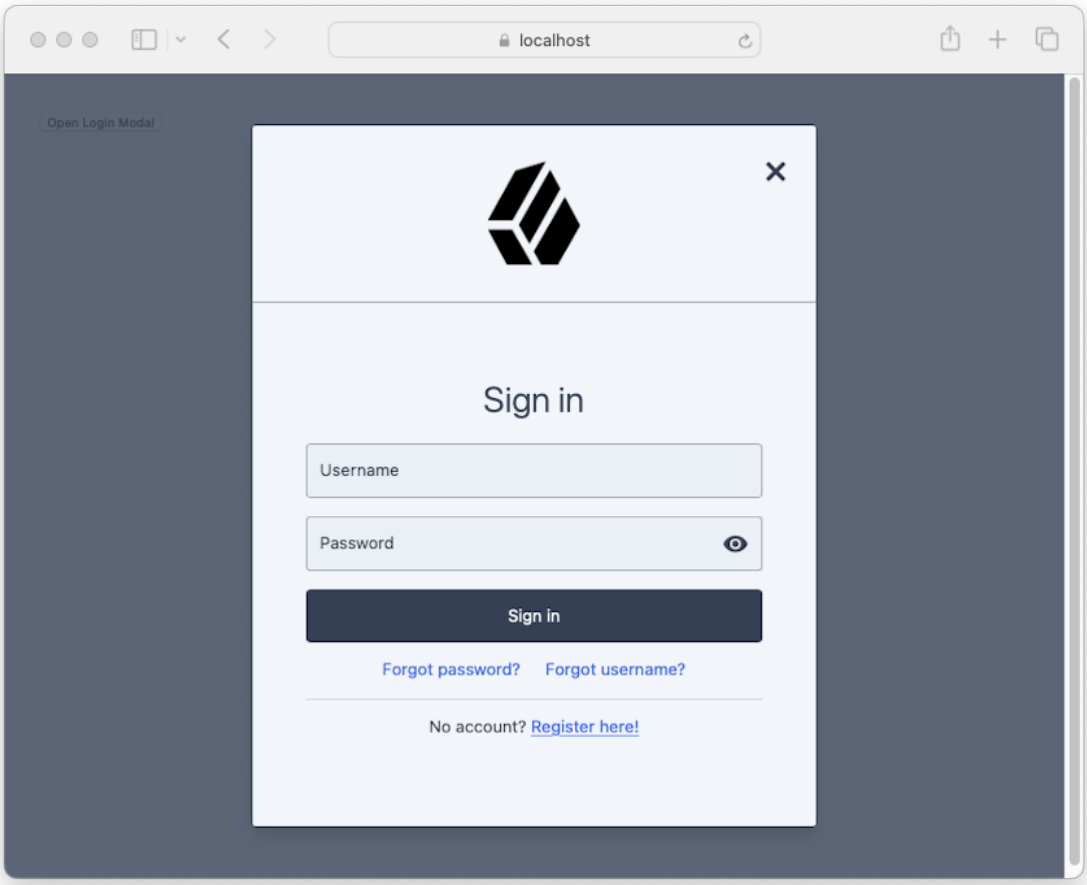


Figure 3. Modal form factor with header enabled

By default, the separating section is not enabled:

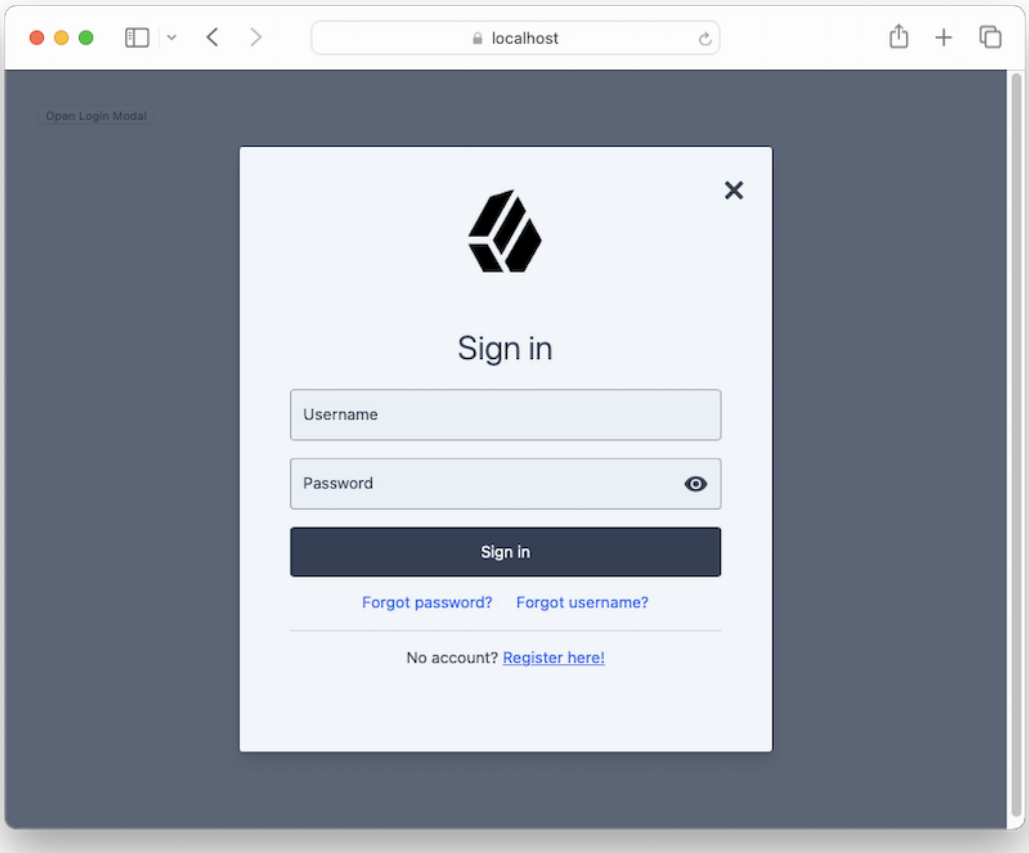


Figure 4. Default modal form factor with header disabled

Set theme colors and fonts

In addition to rebuilding the Advanced Identity Cloud/PingAM Login Widget with a customized Tailwind configuration (refer to Theming the widget), you can override individual theme values at runtime using the `style/theme` configuration element. This is the quickest way to apply brand colors and fonts without cloning and rebuilding the widget.

Example `theme` configuration

```
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-configuration',
  style: {
    theme: {
      primaryColor: '#f46200',
      secondaryColor: '#003049',
      backgroundColor: '#ffffff',
      linkColor: '#f46200',
      fontFamily: 'Inter, sans-serif',
      buttonBorderRadius: 8,
      cardBorderRadius: 12,
    },
  },
});
```

`theme` *properties*

Property	Description
`primaryColor`, `primaryOffColor`, `secondaryColor`	Hex color values (6 or 8 digit, for example `#f46200` or `#f46200ff`) for the widget's primary and secondary brand colors.
`backgroundColor`	Hex color for the page or modal background.
`linkColor`, `linkActiveColor`	Hex colors for links in their default and active states.
`logo`, `favicon`	Full URLs (or base64 data URIs) to override the logo and favicon images.
`logoHeight`	Number of pixels for the logo height.
`fontFamily`	A font family string, for example `Inter, sans-serif`.
`buttonBorderRadius`, `cardBorderRadius`	Numbers, in pixels, controlling the corner rounding of buttons and cards.
`cardBgColor`, `cardTextColor`, `bodyTextColor`	Hex colors for card backgrounds and body text.
`inputBgColor`, `inputBorderColor`, `inputLabelColor`, `inputFocusRingColor`, `inputTextColor`	Hex colors for form input styling.
`selectAccentColor`, `selectHoverBgColor`	Hex colors for `<select>` elements.

Property	Description
`buttonFocusRingColor`, `buttonTextColor`	Hex colors for button focus and text states.

> **Note**
>
> `theme.logo` accepts a single URL and is independent from `style.logo`, which accepts separate `dark` / `light` variants for the modal form factor. Use `style.logo` if you need different images per color scheme; use `theme.logo` for a single logo applied everywhere.

Journeys configuration options

Use the `journeys` configuration element to map HREF values rendered within the Advanced Identity Cloud/PingAM Login Widget to start a journey or authentication tree instead of visiting the URL.

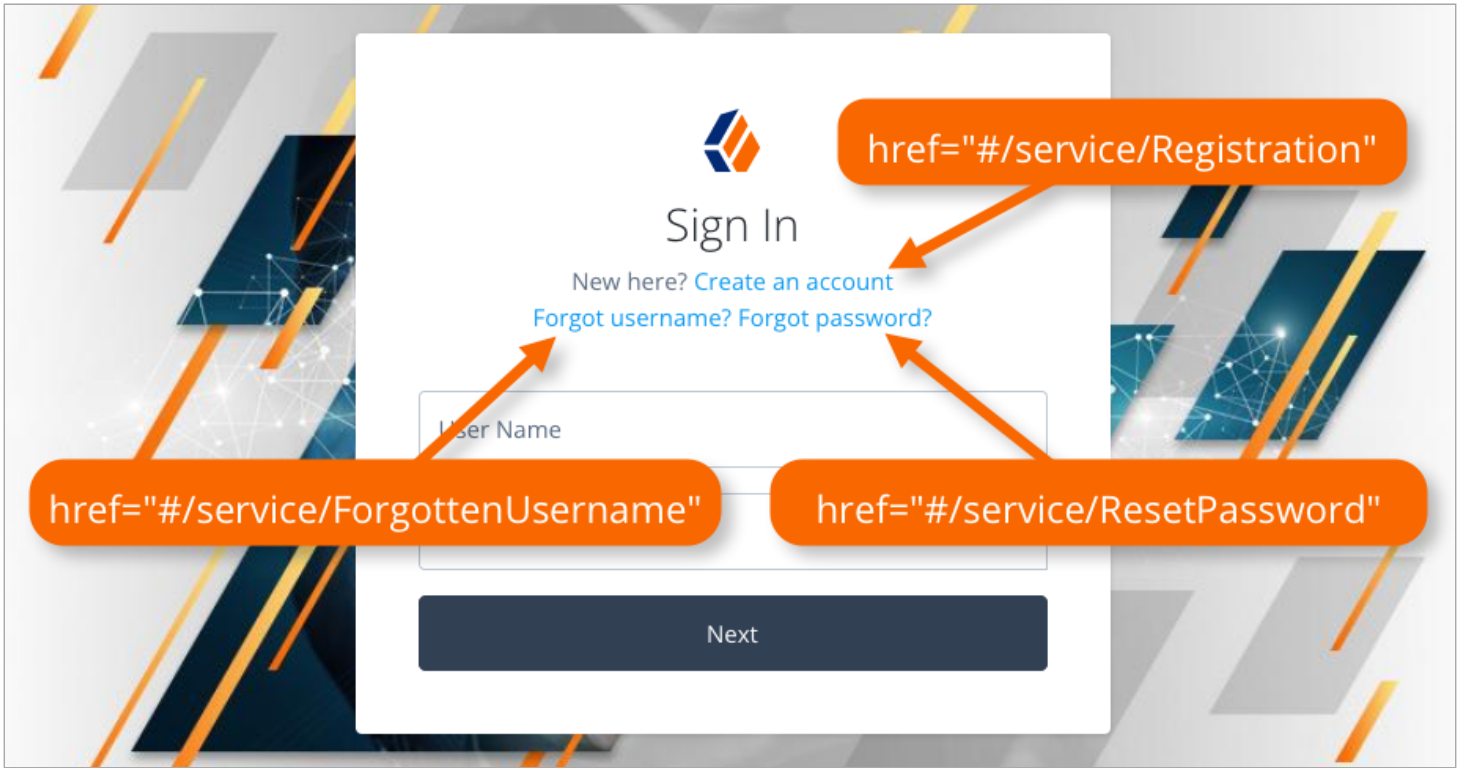


Figure 5. Example HREF values in a page node

The Advanced Identity Cloud/PingAM Login Widget listens for click events on elements rendered within its container and compares the HREF to the configured mappings. If there is a match, it prevents the default action of visiting the URL and starts the journey configured in the mapping.

Mapping HREF strings to a journey

```
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  journeys: {
    forgotPassword: {
      journey: 'ResetPassword', // Must match actual journey name in the server
      match: ['#/service/ResetPassword'], // Array of strings that match `HREF` values (case-sensitive)
    },
  },
});
```

The Advanced Identity Cloud/PingAM Login Widget has mappings configured internally to handle the links displayed in page nodes by default. These map the HREF values that are displayed by an out-of-the-box page node to corresponding journeys in an PingOne Advanced Identity Cloud tenant. You can override the mappings if required.

Default HREF strings to journey mappings

```
forgotPassword: {
    journey: 'ResetPassword',
    match: ['#/service/ResetPassword', '?journey=ResetPassword'],
},
forgotUsername: {
    journey: 'ForgottenUsername',
    match: ['#/service/ForgottenUsername', '?journey=ForgottenUsername'],
},
login: {
    journey: 'Login',
    match: ['#/service/Login', '?journey', '?journey=Login'],
},
register: {
    journey: 'Registration',
    match: ['#/service/Registration', '?journey=Registration'],
},
```

CAPTCHA configuration

AM does not signal invisible mode in the callback payload for either `ReCaptchaCallback` or `ReCaptchaEnterpriseCallback`. Use the `captcha` option to configure invisible rendering:

```
await configure({
  wellknown: 'https://openam-forgerock-sdks.forgeblocks.com/am/oauth2/realms/root/realms/alpha/.well-known/openid-
configuration',
  captcha: {
    mode: 'invisible', // 'visible' (default) | 'invisible'
  },
});
```

Refer to Implement a CAPTCHA for more information.

Component

Use the `component` module for subscribing to modal and inline form factor events and for opening and controlling the modal form factor.

Call the `component()` method and assign the result to a variable to receive the observable. Subscribe to the observable to listen and react to the state of the Advanced Identity Cloud/PingAM Login Widget component.

```
import { component } from '@forgerock/login-widget';

// Initiate the component API
const componentEvents = component();

// Know when the component, both modal and inline has been mounted.
// When using the modal type, you will also receive open and close events.
// The property `reason` will be either "auto", "external", or "user"

const unsubComponentEvents = componentEvents.subscribe((event) => {
    /* Run anything you want */
});

// Open the modal
componentEvents.open();

// Close the modal
componentEvents.close();

// Recommended: call when your UI component is destroyed
unsubComponentEvents();
```

Schema for `component` events

The schema for `component` events is as follows:

Schema for `component` events

```
{
    lastAction: null, // null or the most recent action; one of `close`, `open`, or `mount`
    error: null, // null or object with `code` and `message` properties
    mounted: false, // boolean
    open: null, // boolean for the modal form factor, or null for inline form factor
    reason: null, // string to describe the reason for the event
}
```

Use the `reason` value to determine why the modal has closed.

The possible `reason` values are:

`user`

The user closed the dialog within the UI

`auto`

The modal was closed because the user successfully authenticated

`external`

The application called the `close()` function

Journey

Use the `journey` module to manage interaction with authentication and self-service journeys.

```
import { journey } from '@forgerock/login-widget';

// Call to start the journey
// Optional config can be passed in, see below for more details
const journeyEvents = journey({
  oauth: true, // OPTIONAL; defaults to true; uses OAuth flow for acquiring tokens
  user: true, // OPTIONAL; default to true; returns user information from `userinfo` endpoint
});

// Start a journey
journeyEvents.start({
  journey: 'Login', // OPTIONAL; specify the journey or tree you want to use
  query: {}, // OPTIONAL; additional query parameters to include when starting the journey
  resumeUrl: window.location.href, // OPTIONAL; the full URL for resuming a suspended journey
  recaptchaAction: 'myCaptchaTag', // OPTIONAL; tag reCAPTCHAs. Falls back to journey name.
});

// Change to a different journey
journeyEvents.change({
  journey: 'Registration',
});

// Subscribe to journey events
const unsubJourneyEvents = journeyEvents.subscribe((event) => {
  /* Run anything you want */
});

// Recommended: call when your UI component is destroyed
unsubJourneyEvents();
```

> **Note**
>
> PingOne Protect risk evaluations are initialized separately via `protect.start()`, not through the `journey()` API. Refer to Step 2. Configure the Advanced Identity Cloud/PingAM Login Widget for PingOne Protect.

Schema for `journey` events

The schema for `journey` events is as follows:

Schema for `journey` events

```
{
  journey: {
    completed: false, // boolean
    error: null, // null or object when the journey fails:
    // {
    //   code: null,         // number; HTTP status code, e.g. 401
    //   message: null,      // string; human-readable failure reason
    //   stage: null,        // string; page node stage value, if set
    //   troubleshoot: null, // always null; reserved for future use
    //   detail: null,       // object; extra detail from the server, e.g. { failureUrl: 'https://...' }
    // }
    loading: false, // boolean
    step: null, // null or object with the last step object from the server
    successful: false, // boolean
    response: null, // null or object if successful containing the success response from the server
  },
  oauth: {
    completed: false, // boolean
    error: null, // null or object with `code` (optional), `message`, and `troubleshoot` properties
    loading: false, // boolean
    successful: false, // boolean
    response: null, // null or object with OAuth/OIDC tokens
  },
  user: {
    completed: false, // boolean
    error: null, // null or object with `code` (optional), `message`, and `troubleshoot` properties
    loading: false, // boolean
    successful: false, // boolean
    response: null, // null or object with user information driven by OAuth scope config
  },
}
```

User

Use the `user` module to access methods for managing users:

- `user.info`
- `user.tokens`
- `user.logout`

The `user.info` and `user.tokens` methods require use of OAuth 2.0, so you must configure `oidcClient` in your call to `configure()`.

The `user.info` method also requires a scope value of `openid`, which is the default.

You can use `user.logout` with both OAuth 2.0 and session-based authentication.

```
import { user } from '@forgerock/login-widget';

/**
 * User info API
 */
const userEvents = user.info();

// Subscribe to user info changes
const unsubUserEvents = userEvents.subscribe((event) => {
  // Return current, *local*, user info and future state changes
  console.log(event);
});

// Fetch/get fresh user info from the server
userEvents.get(); // New state is returned in your `userEvents.subscribe` callback function

/**
 * User tokens API
 */
const tokenEvents = user.tokens();

// Subscribe to user token changes
const unsubTokenEvents = tokenEvents.subscribe((event) => {
  // Return current, *local*, user tokens and future state changes
  console.log(event);
});

// Return existing user tokens if available and not expired or about to expire
// Otherwise obtain fresh ones from the server
tokenEvents.get(); // State is returned in your `tokenEvents.subscribe` callback function

/**
 * Logout
 * Log user out and clear user data (info and tokens)
 */
user.logout(); // Resets user and emits event to your info and tokens' `.subscribe` callback function

// Recommended: call when your UI component is destroyed
unsubUserEvents();
unsubTokenEvents();
```

To force the Advanced Identity Cloud/PingAM Login Widget to obtain fresh tokens from the server rather than returning cached tokens:

```
tokenEvents.get({ forceRenew: true });
```

Schema for `user.info` events

The schema for `user.info` events is as follows:

Schema for `user.info` events

```
{
    completed: false, // boolean
    error: null,  // null or object with `code` (optional), `message`, and `troubleshoot` properties
    loading: false, // boolean
    successful: false, // boolean
    response: null, // object returned from the `/userinfo` endpoint
}
```

Schema for `user.tokens` events

The schema for `user.tokens` events is as follows:

Schema for `user.tokens` events

```
{
    completed: false, // boolean
    error: null,  // null or object with `code` (optional), `message`, and `troubleshoot` properties
    loading: false, // boolean
    successful: false, // boolean
    response: null, // object returned from the `/access_token` endpoint
}
```

Protect

Use the `protect` module to manually control the PingOne Signals SDK: starting data collection, returning collected data, and pausing or resuming behavioral data capture.

Refer to Step 2. Configure the Advanced Identity Cloud/PingAM Login Widget for PingOne Protect for full details and code examples.

Calling protected resources

The Advanced Identity Cloud/PingAM Login Widget does not provide an HTTP client for calling protected resources.

To call a protected endpoint, get the current access token from `user.tokens().get()` and call `fetch()` yourself, adding the `Authorization` header:

```
import { user } from '@forgerock/login-widget';

const tokenEvents = user.tokens();
const { response: tokens } = await tokenEvents.get();

const response = await fetch('https://protected.resource.com', {
  method: 'GET',
  headers: {
    Authorization: `Bearer ${tokens.accessToken}`,
  },
});
```

Note

Earlier versions of the Advanced Identity Cloud/PingAM Login Widget exposed a `request` export that automatically refreshed tokens on a 401 response and parsed Identity Gateway policy advice. These behaviors are not provided by the Advanced Identity Cloud/PingAM Login Widget 2.0 and must be implemented by the consumer if needed. Refer to Breaking changes for more information.