Ping Advanced Identity Software

August 4, 2025

PING ADVANCED IDENTITY SOFTWARE

Version: 8

Copyright

Trademark

Disclaimer

The information provided in Ping Identity product documentation is provided "as is" without warranty of any kind. Ping Identity disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall Ping Identity or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if Ping Identity or its suppliers have been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages so the foregoing limitation may not apply.

Table of Contents

Release notes 3

Overview 6
PingAM 13
PingIDM 22
PingDS. 28
Edge Security 32
HSM support 36

Overview 43
Choose your sample. 45
Prepare the servers 50
Separate identity stores 55
- Set up PingAM 62
- Set up PingIDM 77

Shared identity store 81
- Set up PingAM 88
- Set up PingIDM 104

Protect the deployment. 109
Set up the Platform UIs 123
Test your deployment. 132
Upgrade. 135
Migration and customization. 138
- UI customization 141
- Hosted pages 144
- Localize end-user and login UIs 147
- Customize end-user and login UI themes. 153
- Custom attributes 156

Groups 160
- Manage groups. 169

Release notes

Ping Advanced Identity Software is a comprehensive security solution providing a unified platform for access management, identity management, user-managed access, directory services, and an identity gateway. Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments.

> **Note**
>
> Ping Advanced Identity Software was formerly known as the *Ping Identity Platform*.

These release notes provide information on self-managed deployments of the following products:

- PingAM
- PingIDM
- PingDS
- PingGateway

If you don't need maximum extensibility and flexibility, there are simpler alternatives to a self-managed solution:

- To consume Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity Support.

Refer to the component product release notes for details:

Component	*What's new*	*Fixes*	*Deprecation*
PingAM 8	New PingAM features	PingAM fixes	Deprecated PingAM features
PingDS 8	New PingDS features	PingDS fixes	Deprecated PingDS features
PingIDM 8	New PingIDM features	PingIDM fixes	Deprecated PingIDM features
PingGateway 2025.3	New PingGateway features	PingGateway fixes	Deprecated PingGateway features

When you follow the links, you find the release notes for each component product include information about:

- Deployment requirements and prerequisites
- New features and improvements
- Fixes, limitations, and known issues
- Compatibility changes, deprecation notices, and discontinued features

> **Important**
>
> Before you deploy a component product in production, make sure you review its release notes.

Overview

Ping Advanced Identity Software is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

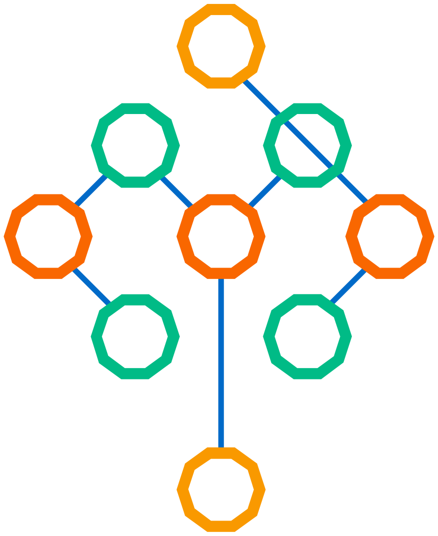
About this documentation

This documentation includes general statements of functionality for the latest versions of the following software:

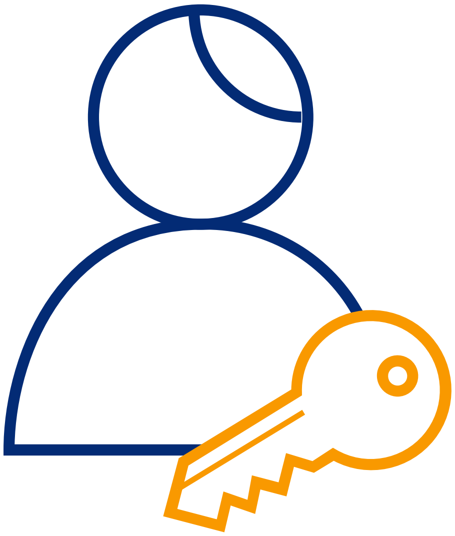
- PingAM, with Web Agents and Java Agents
- PingIDM
- PingDS
- Edge Security (PingGateway)

This documentation describes in general terms the modules that compose the Ping Identity Platform, and indicates where to find the documentation corresponding to each module. This documentation is not meant to serve as a statement of functional specifications. Software functionality may evolve in incompatible ways in major and minor releases, and occasionally in maintenance (patch) releases. Release notes cover many incompatible changes. If you see an incompatible change for a stable interface that is not mentioned in the release notes, please report an issue with the product documentation for that release.

PingAM modules



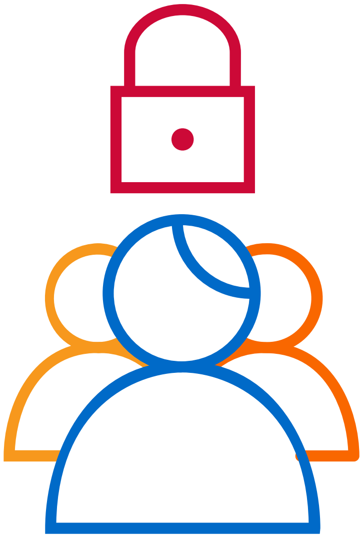
Intelligent Access



Authorization

Federation

Self-Managed Strong Authentication



User-Managed Access

PingIDM modules

Identity Synchronization

Workflow

Identity Lifecycle and Relationship

PingDS modules

Directory Server

Directory Proxy Server

PingGateway module

Edge Security (PingGateway)

Deployment enhancements

In addition to the modules listed in the preceding section, you can use the following software to enhance Advanced Identity Software deployments.

Run Advanced Identity Software in containers on Kubernetes

Ping Advanced Identity Software, (PingAM, PingIDM, PingDS, PingGateway, and the Platform UI) is supported when running in containers on Kubernetes platforms, including Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (Amazon EKS), Microsoft Azure Kubernetes Service (AKS), and IBM RedHat OpenShift. It's recommended that you have a support contract in place with your Kubernetes platform vendor or partner to resolve any infrastructure or Kubernetes platform-related issues. Ping Identity supports Ping Advanced Identity Software while the Kubernetes vendor or partner provides support for their platform.

You are responsible for building images and running containers of the Ping Identity software components using a supported operating system and all required software dependencies.

Kubernetes deployment tools from Ping Identity

Ping Identity provides a reference toolset in the forgeops and forgeops-extras Git repositories for automating the deployment of Advanced Identity Software in Kubernetes. These reference tools are provided for use with Google Kubernetes Engine, Amazon Elastic Kubernetes Service, and Microsoft Azure Kubernetes Service. (Ping Identity supports running Advanced Identity Software on IBM RedHat OpenShift but does not provide the reference tools for IBM RedHat OpenShift.)

Ping Identity also publishes reference Docker images for testing and development, but these images should *not* be used in production. For production deployments, it is recommended that customers build and run containers using a supported operating system and all required software dependencies. Additionally, to help ensure interoperability across container images and the ForgeOps tools, Docker images must be built using the Dockerfile templates as described in the ForgeOps documentation.

Partner offerings

Ping Identity’s partner, Midships Limited, offers a Kubernetes deployment accelerator (supported by Midships) for Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (Amazon EKS), Microsoft Azure Kubernetes Service (AKS), and IBM RedHat OpenShift.

ForgeRock Authenticator application

This app allows end users to perform multi-factor authentication and transactional authorization from a registered Android or iOS device. It is designed for use in both multi-factor and passwordless authentication scenarios. It is associated with a Push Authentication Simple Notification Service module that depends on the module described in Intelligent Access modules.

See MFA: push authentication and Transactional authorization.

PingAM

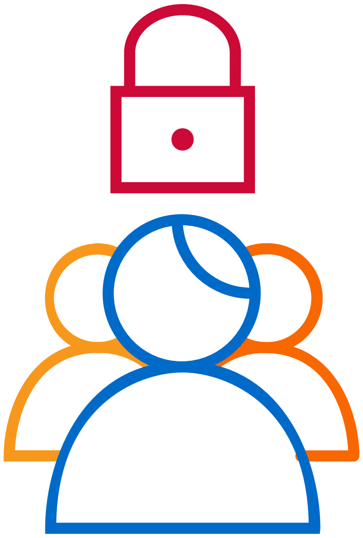
PingAM modules:

Intelligent Access

Authorization

Federation

Self-Managed Strong Authentication



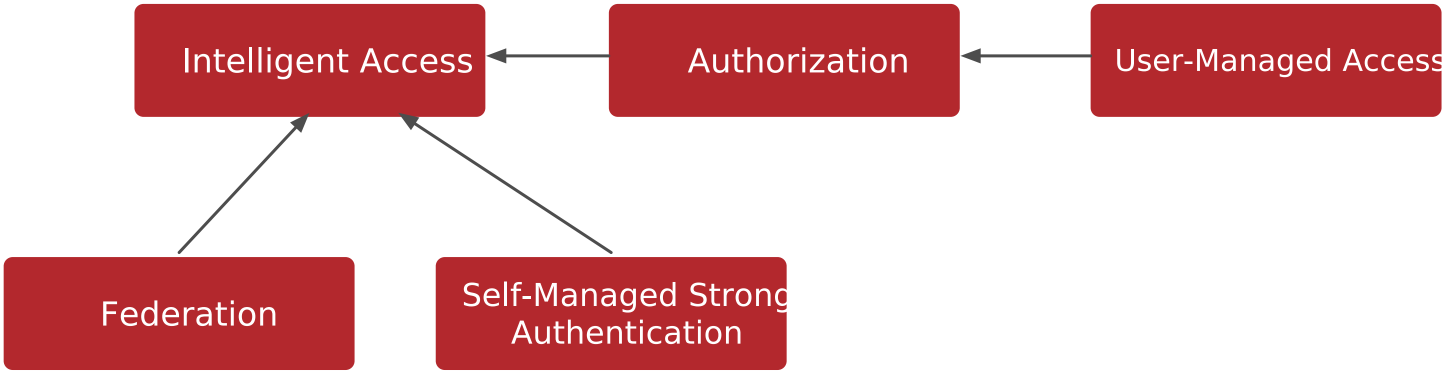
User-Managed Access

Overview of capabilities

- Intelligent access
- Mobile authentication
- Push authentication
- Adaptive risk authentication
- Authorization policies and enforcement
- Federation
- Single sign-on (SSO)
- User self-services and social sign-on
- High-availability and scalability
- Adaptable monitoring and auditing services
- Developer-friendly, rich standards support

Dependencies

Several Access Management modules require other modules. For example, the Federation module requires the Intelligent Access module. The following diagram summarizes Access Management module dependencies:



Intelligent Access modules

This module helps you build secure, robust, centrally managed single sign-on services. The user, application, or device signs on once and is granted appropriate access everywhere. Authentication management integrates delegated authentication with many authentication methods supported by default. Authentication journeys store authentication sessions in the client as a cookie, or in the CTS store. If the PingAM server goes down or the user is redirected to another PingAM while authenticating, the new PingAM server can grab the authentication session and continue the flow. All authentication-related events are logged for auditing and reporting purposes.

Required modules: none.

Feature	Description	Documentation
Authentication trees and nodes	Authentication trees provide fine-grained authentication, social authentication, and multi-factor authentication. Trees are made up of authentication nodes. Authentication nodes allow multiple paths and decision points throughout the authentication flow, enabling PingAM to handle different modes of authenticating users.	Authentication nodes and trees
Session high availability	Persistent access management sessions, authenticating the user until the session expires.	Session high availability is enabled by default with no setup required.
Multi-factor and strong authentication	Capability to challenge for additional credentials when authentication takes place under centrally-defined risky or suspicious conditions.	Multi-factor authentication
External configuration store	Configuration storage in PingDS for high-availability.	Prepare configuration stores
Security token service	Bridges identities across web and enterprise identity access management (IAM) systems through a token transformation process, securely providing cross-system access to service resources by authenticated requesting applications.	STS overview
Web and Java agents for SSO	Intercept requests to access protected resources and redirect for appropriate authentication.	Web policy agents and Java policy agents
User login analytics	Measure authentication flows using counters and start/stop timers to monitor performance.	Timer Start node, Timer Stop node, Meter node, and Monitoring metric types

Authorization module

This module will help you create powerful, context-based policies with a GUI-based policy editor and with REST APIs to control access to online resources. Resources can be URLs, external services, or devices and things. Authorization management lets you manage policies centrally and enforce them locally through installable agents, or through REST, C, and Java applications. Authorization management is extensible, making it possible to define external subjects, complex conditions, and custom access decisions.

Required module: Intelligent Access.

Feature	Description	Documentation
Entitlement policies	Modern web-based policy editor for building policies, making it possible to add and update policies as needed without touching the underlying applications.	Authorization and policy decisions
Web and Java agents for enforcement	Access enforcement for online resources with the capability to require higher levels of authentication and session upgrade when accessing sensitive resources.	Web policy agents and Java policy agents
Transactional authorization	Requires a user to perform additional actions such as reauthenticating to a module or node, or responding to a push notification, to gain access to a protected resource.	Transactional authorization
OAuth 2.0 dynamic scopes	A single OAuth 2.0 client configured for a comprehensive list of scopes can serve different scope subsets to resource owners based on policy conditions.	Dynamic OAuth 2.0 authorization

Federation module

This module will help you extend SSO capabilities across organization boundaries based on standards-based interoperability.

Required module: Intelligent Access.

Feature	Description	Documentation
SAML 2.0 IDP and SP	Identity federation with SaaS applications, such as Salesforce.com, Google Apps, WebEx, and many more.	Configure IdPs, SPs, and COTs
SAML 2.0 SSO and SLO	Web Single Sign-On and Single Logout profile support.	Implement SSO and SLO
ADFS	Federation with Active Directory Federation Services.	Introduction to SAML v2.0
SAML 2.0 Attribute and Advanced Profiles	Support for transmitting only attributes used by targeted applications.	SAML v2.0

Feature	Description	Documentation
OpenID Connect	OpenID Connect 1.0 compliance for running an OpenID Provider, including advanced profiles, such as Mobile Connect.	OpenID Connect 1.0
OAuth 2.0	OAuth 2.0 compliance for running an authorization server.	OAuth 2.0
Social login	For acting as an OAuth 2.0 client of social identity providers, such as Facebook, Google, and Microsoft.	Social authentication
OAuth 2.0 dynamic scopes	A single OAuth 2.0 client configured for a comprehensive list of scopes can serve different scope subsets to resource owners based on policy conditions.	Dynamic OAuth 2.0 authorization

Self-Managed Strong Authentication module

This module gives end users additional authentication capabilities through their own devices. Multi-faceted device management enhances security and simplifies the authentication experience for end users.

Required modules: Intelligent Access

Feature	Description	Documentation
Geolocation	Manages risk by assessing whether the authenticating user's device is located within range of configured, trusted locations, or within range of somewhere the user has authenticated from, and saved, previously. PingAM compares collected device location metadata with trusted locations in the authentication configuration or with device locations stored in the user's profile.	Device Geofencing node and Device Profile Location Match node
Device tampering	Provides a configurable threshold for assessing the risk of a user's device during authentication based on a risk score. The higher the score returned from the device, the more likely the device is jailbroken, rooted, or is a potential security risk.	Device Tampering Verification node

Feature	Description	Documentation
Device binding and trust	During the Device Binding process, secure cryptography keys are generated, which are used to perform various functions, such as encryption and decryption of user data. These keys are unique to the device and can't be transferred to another device, ensuring that the user's account can only be accessed from authorized devices. This helps to protect against unauthorized access. One user can have multiple associated devices and multiple users can associate with one device. Devices are bound using authentication journeys. PingAM supports a number of different authentication types for device binding, including biometric,biometric with fallback, application PIN and silent authentication.	Device Binding node↗, Device Binding Storage node↗, and Device Signing Verifier node↗
Application MFA	Open AuTHentication (OATH) OATH-related nodes supports the following MFA methods: • Time-based one-time passwords (TOTP) • HMAC-based one-time passwords (HOTP) • Push notifications These nodes can integrate with the ForgeRock Authenticator app for Android and iOS and with third-party authenticator apps that support the HOTP and TOTP standards.	OATH Device Storage node↗, OATH Registration node↗, OATH Token Verifier node↗, and ForgeRock Authenticator↗
Transaction signing	Digital signing helps ensure the authenticity and integrity of customer data in a variety of use cases, including financial transactions, end user agreements, changes in account details, and so on.	Device Signing Verifier node↗
WebAuthN passwordless authentication	Passwordless authentication (WebAuthn) is better than traditional password-based authentication because it reduces the risk of password-based attacks like phishing and brute-force attacks. Passwordless authentication also creates a better, more seamless experience for end users. It's compatible across most devices and systems and end users don't have to remember a number of passwords across systems. Passwordless authentication uses other forms of identity verification instead of a standard password login. These methods can include face recognition, fingerprints, software or hardware token devices, and so on. End users authenticate with an authenticator device, such as the fingerprint scanner on their laptop or phone.	MFA: Web authentication (WebAuthn)↗

User-Managed Access module

This module consists of a consumer-facing implementation of the User-Managed Access (UMA) 2.0 standard. The standard defines an OAuth 2.0-based protocol designed to give individuals a unified control point for authorizing who and what can access their digital data, content, and services. For example, you can use this module to build a solution where end users can delegate access through a share button, and then monitor and change sharing preferences through a central dashboard.

Required modules: Authorization, Intelligent Access.

Feature	Description	Documentation
UMA standard conformance	Conformance to the UMA 2.0 standard for interoperability with organizational and partner systems, including federated authorization and customer-centric use cases.	User-Managed Access (UMA) 2.0
UMA authorization server	Authorization server with dynamic resource set registration, end-user control of resource sharing, responses to access requests, and full audit history.	PingAM as UMA authorization server
UMA protector	PingGateway protection for resources and services with the UMA 2.0 standard.	UMA support

PingIDM

PingIDM 8 brings together multiple sources of identity for policy and workflow-based management that puts you in control of the data. Build a solution to consume, transform, and feed data to external sources to help you maintain control over identities of users, devices, and things. Identity governance features in PingIDM let you gain visibility into employee provisioning, and help you proactively take action in managing employee access to external systems.

PingIDM modules:

Identity Synchronization

Workflow

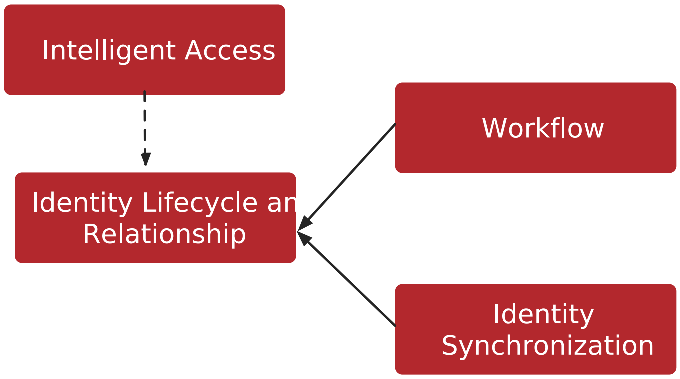
Identity Lifecycle and Relationship

Overview of capabilities

- Provisioning
- Synchronization and reconciliation
- Adaptable monitoring and auditing services
- Flexible developer access
- Password synchronization
- Identity data visualization
- Delegated administration
- Privacy and consent
- Progressive profile completion
- Workflow engine
- OpenICF connector framework to external systems

Dependencies

Several Identity Management modules require other modules. For example, the Synchronization module requires the Identity Lifecycle and Relationship module. The following diagram summarizes Identity Management module dependencies:



Identity Synchronization module

This module can serve as the foundation for provisioning and identity data reconciliation. Synchronization capabilities are available as a service and hrough REST APIs to be used directly by external applications. Activities occurring in the system can be configured to log and audit events for reporting purposes.

Required module: Identity Lifecycle and Relationship.

Feature	Description	Documentation
Discovery and synchronization	Synchronization of identity data across managed data stores.	Synchronization types
Reconciliation	Alignment between accounts across managed data stores.	Synchronization types
Password synchronization	Near real-time password synchronization across managed data stores.	Password synchronization plugins
PingDS and Active Directory plugins	Native password synchronization plugins for PingDS and Microsoft Active Directory.	Synchronize passwords with DS, Synchronize passwords with Active Directory
Delegated administration	Grant role-based, limited access to perform fine-grained administrative tasks on managed objects.	Delegated administration

Feature	Description	Documentation
All connectors	Extensible interoperability for identity, compliance, and risk management across a variety of specific applications and services.	Connector reference

Workflow module

This module can be used to visually organize identity synchronization, reconciliation, and provisioning into repeatable processes with logging and auditing for reporting purposes.

Required modules: Identity Lifecycle and Relationship.

Feature	Description	Documentation
BPMN 2.0 support	Standards-based Business Process Model and Notation 2.0 support.	BPMN 2.0 and workflow tools
Flowable process engine	Lightweight workflow and business process management platform.	Enable workflows
Workflow-driven provisioning	Define provisioning workflows for sunrise and sunset processes, approvals, escalations, and maintenance.	Create workflows, Invoke workflows

Identity Lifecycle and Relationship module

This module can help you to provision user identities into PingIDM, and includes the capability to manage roles, relationships between identities, and entitlements.

Required modules: none.

Feature	Description	Documentation
Inbound provisioning engine	Provisioning engine to import data from an external resource into PingIDM.	Synchronization
Data modeling	Ability to map PingIDM objects to tables in a JDBC database or to organizational units in a PingDS repository.	Object mappings
Identity lifecycle management	An extensible object model that enables you to manage the complete lifecycle of identity objects.	Managed objects

Feature	Description	Documentation
Identity relationship lifecycle management	Ability to create and track relationship references between objects.	Relationships between objects
Role lifecycle management	Provisioning roles to control how objects are exported to external systems and authorization roles to control authorization within PingIDM.	Roles
Entitlement lifecycle management	Entitlements to provision attributes or sets of attributes, based on role membership.	Use assignments to provision users

PingDS

PingDS 8 serves as a foundation for LDAPv3 and RESTful directories.

PingDS modules:

Directory Server

Directory Proxy Server

Overview of capabilities

- Large-scale, distributed read and write performance
- Flexible key-value data model for storing users, devices, and things
- Data storage with confidentiality, integrity, and security
- High-availability through data replication and proxy services
- Single logical entry point for use in protecting LDAPv3 directory services
- Load balancing and failover for LDAPv3 directory services
- Maximum interoperability and pass-through delegated authentication
- Adaptable monitoring and auditing services
- Easy installation, configuration, and management
- Developer-friendly, rich standards support
- REST API to access LDAP native capabilities over HTTP

Dependencies

Neither of the Directory Services modules are dependent upon other modules.

Directory Server module

The Directory Server module helps you store identities for users, devices, and things in a highly available and secure way. This module provides data replication to help you build highly available directory services. It also offers fine-grained access control, password digests, encryption schemes, and customizable password policies to allow you to build very secure directory services. Data may be accessed using LDAP or REST with the same level of security constraints and access control.

Required modules: none.

Feature	Description	Documentation
LDAPv3	Compliance with the latest LDAP protocol standards.	About directories
HDAP	Access LDAP data over HTTP using Directory Access Protocol (HDAP) APIs that transform HTTP operations into LDAP operations.	Learn HDAP
High-availability multi-master replication	Data replication for always-on services, enabling failover and disaster recovery.	Replication
User/object store	Flexible key-value data model for storing users, devices, and things.	Use LDAP
Passwords and data security	Password digests, encryption schemes, and customizable rules for password policy compliance to help protect data on disk and shared infrastructure.	Data encryption, Passwords
REST APIs and REST to LDAP gateway (deprecated)	HTTP-based RESTful access to user data.	Use REST/HTTP
DSMLv2 gateway (deprecated)	HTTP-based SOAP access to LDAP operations for web services.	Install a DSML gateway

Directory Proxy Server module

The ForgeRock Directory Proxy Server module helps you increase the availability of a Directory Service deployment, providing a single point of access to a large-scale distributed data store. The module offers a choice of strategies for request load balancing and failover. Data may be accessed using LDAP or REST with the same level of security constraints and access control.

Required modules: none.

Feature	Description	Documentation
Single point of access	Uniform view of underlying LDAPv3 directory services for client applications.	Single point of access
High service availability	LDAP services with reliable crossover and DN-based routing.	High availability
Load balancing and failover	Configurable load balancing across directory servers with redundancy, and capabilities to handle referrals, connection failures, and network partitions.	Load balancing
Protection for PingDS	Secure incoming and outgoing connections, and provide coarse-grained access control.	Secure connections and Proxy global policies
Scaling out using data distribution	Distribute data across multiple shards.	Data distribution
LDAPv3	Compliance with the latest LDAP protocol standards.	Supported standards
REST APIs	HTTP-based RESTful access to user data.	Use HDAP

Edge Security

Use the Edge Security module to integrate web applications, APIs, microservices, Internet of Things devices, and cloud-based services with the Ping Advanced Identity Software.

Edge Security module:

Edge Security (PingGateway)

Dependencies

The Edge Security module is not dependent upon any other modules.

Edge Security (PingGateway) module

PingGateway helps you integrate web applications, APIs, and microservices with Advanced Identity Software, without modifying the application or the container where it runs. Based on reverse proxy architecture, it enforces security and access control in conjunction with the PingAM modules.

PingGateway software provides the following capabilities:

- Protection for IoT services, microservices, and APIs
- Policy enforcement
- Adaptable throttling, monitoring, and auditing
- Secure token transformation
- Support for identity standards such as OAuth 2.0, OpenID Connect, SAML 2.0, and UMA 2.0
- Password capture and replay
- Rapid prototyping

Required modules: none.

Feature	Description	Documentation
Studio	User interface for rapid development and prototyping.	Studio guide
Single sign-on	Single sign-on in a single domain and across domains.	Single sign-on and Cross-domain single sign-on
Password replay	Secure replay of credentials to legacy applications or APIs.	Password replay from AM, Password replay from a database, and Password replay from a file
Policy enforcement	Enforcement of centralized authorization policies for applications requiring PingAM.	Enforce policy decisions from AM and Harden authorization with advice from AM
Federation	OpenID Connect 1.0.	OpenID Connect
	OAuth 2.0.	IG as an OAuth 2.0 client and IG as an OAuth 2.0 resource server
	SAML 2.0.	SAML
	SAML resources for mobile applications.	Transform OpenID Connect ID tokens into SAML assertions
Finance APIs	Support for OAuth 2.0 Mutual TLS and Financial-Grade APIs.	Validate certificate-bound access tokens and FapiInteractionIdFilter
WebSocket protocol	Detection of requests to upgrade from HTTPS to the WebSocket protocol, and creation of a secure, dedicated tunnel to send and receive WebSocket traffic.	WebSocket traffic
Throttling	Throttling to limit access to protected applications.	Throttling
UMA resource server	Protection for resources and services according to the UMA 2.0 standard.	UMA support
DevOps tooling	Deployment of basic and customized configurations through Docker.	Deployment guide
Integration with Advanced Identity Cloud	Protection and integration of APIs and applications with PingOne Advanced Identity Cloud for authentication and authorization.	Identity Cloud guide

Feature	Description	Documentation
Microgateway	PingGateway standalone deployed as a microgateway, securing microservices with OAuth 2.0.	PingGateway as a microgateway

HSM support

This page covers how you can use a Hardware Security Module (HSM) to protect Ping Advanced Identity Software software private and secret keys.

On protecting secrets

You must protect private keys and secret keys that servers use for cryptographic operations:

Operating system protections

In many deployments, operating system protections are sufficient. Operating systems are always sufficient for public keys and keystores that do not require protection.

Isolate the server account on the operating system where it runs, and allow only that account to access the keys. If you store the keys with version control software, also control access to that.

This deployment option generally offers a better performance/cost ratio. You must take more care to prevent private and secret keys from being exposed, however.

HSM

For stronger protection, you can use an HSM.

An HSM is a secure device that holds the keys, and protects them from unauthorized access. With an HSM, no one gets direct access to the keys. If an attacker does manage to break into an HSM and theoretically get access to the keys, HSM are designed to make the tampering evident, so you can take action.

To put a private or secret key on an HSM, you have the HSM generate the key. The key never leaves the HSM. Instead, if you need the result of a cryptographic operation involving the key, you authenticate to the HSM and request the operation. The HSM performs the operation without ever exposing any private or secret keys. It only returns the result.

An HSM can be certified to comply with international standards, including FIPS-140 and Common Criteria. An HSM that is certified to comply with these standards can be part of your supported Ping Identity software solution.

Ping Identity software uses the HSM through standard PKCS#11 interfaces, and supports the use of compliant cryptographic algorithms.

An HSM generally offers higher security, but with a significant cost and impact on performance. Good HSMs and standards compliance come with their own monetary costs.

In terms of performance, each cryptographic operation incurs at minimum a round trip on a secure connection, compared with an operation in local memory for keys on the system. Even if the deployment may guarantee the same throughput, take the latency into account when deciding to deploy with HSMs.

Key management services

Key management services, such as Google Cloud KMS and others, offer similar advantages and tradeoffs as HSMs.

Latency for online key management services can be very high.

Performance

Throughput, the rate of operations completed, might or might not be impacted by your choice of protecting secrets.

Latency, how long individual operations take, will be impacted by your choice of protecting secrets.

Performance Example

For example, suppose you want a system that signs one million JWTs per second. As long as you can distribute the signing key across enough hardware, your system can sign one million JWTs a second. This is true even if each signing operation takes ten seconds. You simply need ten million systems, and the network hardware to use them in parallel. Throughput therefore depends mainly on how much you can spend.

Suppose, however, that each signing operation must take no longer than ten milliseconds. Furthermore, suppose you have an HSM that can perform a signing operation in one millisecond, but that the network round trip from the system to the HSM takes an average of eleven milliseconds. In this case, you cannot fix performance by buying a faster HSM, or by somehow speeding up the software. You must first reduce the network time if you want to meet your latency requirements.

Perhaps the same signing operation with a key stored on the operating system takes two milliseconds. Consider the options based on your cost and security requirements.

Performance also depends on the following:

- The impact of latency is greater when performing symmetric cryptographic operations (AES, HMAC) compared to public key cryptography (RSA, EC).
- For public key cryptography, only operations that use the private key must contact the HSM (signing, decryption).

  Operations using the public key (verifying a signature, encryption) retrieve the public key from the HSM once, and then use it locally.
- Most public key cryptography uses hybrid encryption, where only a small operation must be done on the HSM, and the rest can be done locally.

  For example, when decrypting a large message, typically, the HSM is only used to decrypt a per-message AES key that is then used to decrypt the rest of the message locally.

  Similarly, for signing, the message to sign is first hashed in-memory using a secure hash algorithm, and only the short hash value is sent to the HSM to be signed.

  In contrast, when using symmetric key algorithms directly with an HSM, the entire message must be streamed to and from the HSM.

How Ping Advanced Identity Software interacts with HSMs

Services built with Java interact with HSMs through Java PKCS#11 providers.

The PKCS#11 standard defines a cryptographic token interface, a platform-independent API for accessing an HSM, for example. Ping Advanced Identity Software supports the use of the Sun PKCS11 provider.

The Sun PKCS11 provider doesn't implement every operation and algorithm supported by every HSM. For details on the Sun PKCS11 provider's capabilities, refer to the JDK Providers Documentation, and the JDK PKCS#11 Reference Guide.

How you configure the JVM to use your HSM depends on the Java environment and on your HSM. Advanced Identity Software components don't implement or manage this configuration, but they do depend on it. Before configuring Advanced Identity Software to use your HSM, you must therefore configure the JVM's Sun PKCS11 provider to access your HSM. For details on how to configure the Java Sun PKCS11 provider with your HSM, refer to the documentation for your HSM and the JDK PKCS#11 Reference Guide.

Sun PKCS11 provider hints

The provider configuration depends on your provider.

Compare the following hints for Advanced Identity Software software with the documentation for your HSM:

`name = FooHsm`

The name used to produce the Sun PKCS11 provider instance name for your HSM.

`FooHsm` is just an example.

`CKA_TOKEN = false`

Keys generated using the Java `keytool` command effectively have this set to `true`. They are permanent keys, to be shared across the deployment.

Set this to `false` to prevent temporary keys for RSA hybrid encryption used by some Advanced Identity Software components from exhausting HSM storage.

`CKA_PRIVATE = true`

The key can only be accessed after authenticating to the HSM.

`CKA_EXTRACTABLE = false`
`CKA_SENSITIVE = true`

`CKA_EXTRACTABLE = false` means the key cannot be extracted *unless it is encrypted*.

`CKA_SENSITIVE = true` means do not let the secret key be extracted out of the HSM. Set this to `true` for long-term keys.

Only change these settings to back up keys to a different HSM, when you cannot use a proprietary solution. For example, you might change these settings if the HSMs are from different vendors.

`CKA_ENCRYPT = true`

The key can be used to encrypt data.

`CKA_DECRYPT = true`

The key can be used to decrypt data.

`CKA_SIGN = true`

The key can be used to sign data.

`CKA_VERIFY = true`

The key can be used to verify signatures.

`CKA_WRAP = true`

The key can be used to wrap another key.

`CKA_UNWRAP = true`

The key can be used to unwrap another key.

When using keytool

When using the Java `keytool` command to generate or access keys on the HSM, set the following options appropriately:

`-alias` *alias*

If key pair generation fails, use a new *alias* for the next try. This prevents the conflicts where the previous attempt to create a key was only a partial failure, leaving part of key pair using the previous alias.

`-keystore none`

Required setting.

`-providername` *providerName*

Required setting.

Prefix `SunPKCS11-` to the name in the configuration. If, in the Sun PKCS11 configuration, `name = FooHSM`, then the *providerName* is `SunPKCS11-FooHSM`.

`-storetype pkcs11`

Required setting.

If necessary, add the following settings:

`-providerClass sun.security.pkcs11.SunPKCS11`

Use this to install the provider dynamically.

`-providerArg` */path/to/config/file.conf*

Use this to install the provider dynamically.

The exact configuration depends on your HSM.

When Java cannot find keys

When keys generated with one Java PKCS#11 provider are later accessed using the Sun PKCS11 provider, the providers may have different naming conventions.

Java's KeyStore abstraction requires that all private key objects have a corresponding Certificate object. SecretKey objects, such as AES or HMAC keys, are excluded from this requirement.

The Sun PKCS11 KeyStore provider loops through all defined private key entries in the HSM (class = private key), and tries to match them up with a corresponding certificate entry (class = certificate) by comparing the `CKA_ID` of the certificate entry to the `CKA_ID` of the private key entry. There may be multiple certificates using the same private key pair The matching process can take several seconds at startup time if you have many keys.

If keys are not found, it is likely that the private key entry `CKA_ID` does not match the certificate entry `CKA_ID`.

HSM features and Ping Advanced Identity Software

This part outlines how some Ping Advanced Identity Software features support HSMs.

JWT encryption algorithms

JWT Encryption Algorithm	Java Algorithm Equivalent	Supported by Sun PKCS11 Provider?
RSA1_5	RSA/ECB/PKCS1Padding[1]	Yes
RSA-OAEP	RSA/ECB/OAEPWithSHA-1AndMGF1Padding	No
RSA-OAEP-256	RSA/ECB/OAEPWithSHA-256AndMGF1Padding	No
ECDH-ES (+A128KW, and so on)[2]	ECDH (KeyAgreement vs. Cipher)	Yes
dir with A128GCM, A192GCM, A256GCM	AES/GCM/NoPadding	Yes
dir with A128CBC-HS256, and so on	AES/CBC/PKCS5Padding + HmacSHA256, and so on	No
A128KW, A192KW, A256KW	AESWrap	No

[1] PKCS#1 version 1.5 padding has known vulnerabilities and should be avoided.

[2] The Sun PKCS11 implementation of ECDH KeyAgreement requires that the derived key be extractable. This is not a security issue, as the derived key is unique to each message, and therefore no more sensitive than the message it is protecting, due to the use of fresh ephemeral keys for each message (ECIES). To ensure that the derived keys are extractable, add the following to the PKCS11 configuration file:

```
attributes(*,CKO_PRIVATE_KEY,CKK_EC) = {
  CKA_SIGN = true
  CKA_DERIVE = true
  CKA_TOKEN = true
}

attributes(generate,CKO_SECRET_KEY,CKK_GENERIC_SECRET) = {
  CKA_SENSITIVE = false
  CKA_EXTRACTABLE = true
  CKA_TOKEN = false
}
```

This also ensures that EC keys generated with the `keytool` command are marked as allowing ECDH key derivation. The derived keys are also marked as `CKA_TOKEN = false`, which ensures that the derived keys are only created as session keys, and automatically deleted when the session ends to prevent filling up the HSM with temporary keys.

JWT signing algorithms

JWT Signing Algorithm	Java Algorithm Equivalent	Supported by Sun PKCS11 Provider?
HS256, HS384, HS512	HmacSHA256, HmacSHA384, HmacSHA512	Yes
RS256, RS384, RS512	SHA256WithRSA, SHA384WithRSA, SHA512WithRSA	Yes
PS256, PS384, PS512	RSASSA-PSS or SHA256WithRSAAndMGF1, and so on	No
ES256, ES384, ES512	SHA256WithECDSA, and so on	Yes
EdDSA	Under development	No

Configure Ping Advanced Identity Software to use an HSM

Once you have configured the Java environment to use your HSM through the Sun PKCS11 Provider, you can configure Ping Advanced Identity Software to use the HSM:

- PingAM: Refer to Secrets, certificates, and keys.
- PingDS: Refer to PKCS#11 hardware security module.
- PingIDM: Refer to Hardware secret stores .
- PingGateway: Refer to Keys and secrets, and HsmSecretStore.

Overview

Ping Advanced Identity Software is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

Advanced Identity Software serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about Advanced Identity Software, visit https://www.pingidentity.com.

The Common REST API works across Advanced Identity Software deployments to provide common ways to access web resources and collections of resources.

> **Important**
>
> This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
> Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:
>
> - To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
> - To deploy in Kubernetes, start with the ForgeOps reference implementation.
>
> For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

Advanced Identity Software setup checklist

These topics show you how to set up Advanced Identity Software components in a self-managed deployment without using PingOne Advanced Identity Cloud or ForgeOps.

- □ Choose your sample
- □ Prepare the servers

  Separate identity stores

  - □ Set up PingDS
  - □ Set up PingAM
  - □ Set up PingIDM

  Shared identity store

  - □ Set up PingDS
  - □ Set up PingAM
  - □ Set up PingIDM
- □ Protect the deployment
- □ Set up the Platform UIs
- □ Test your deployment

Choose your sample

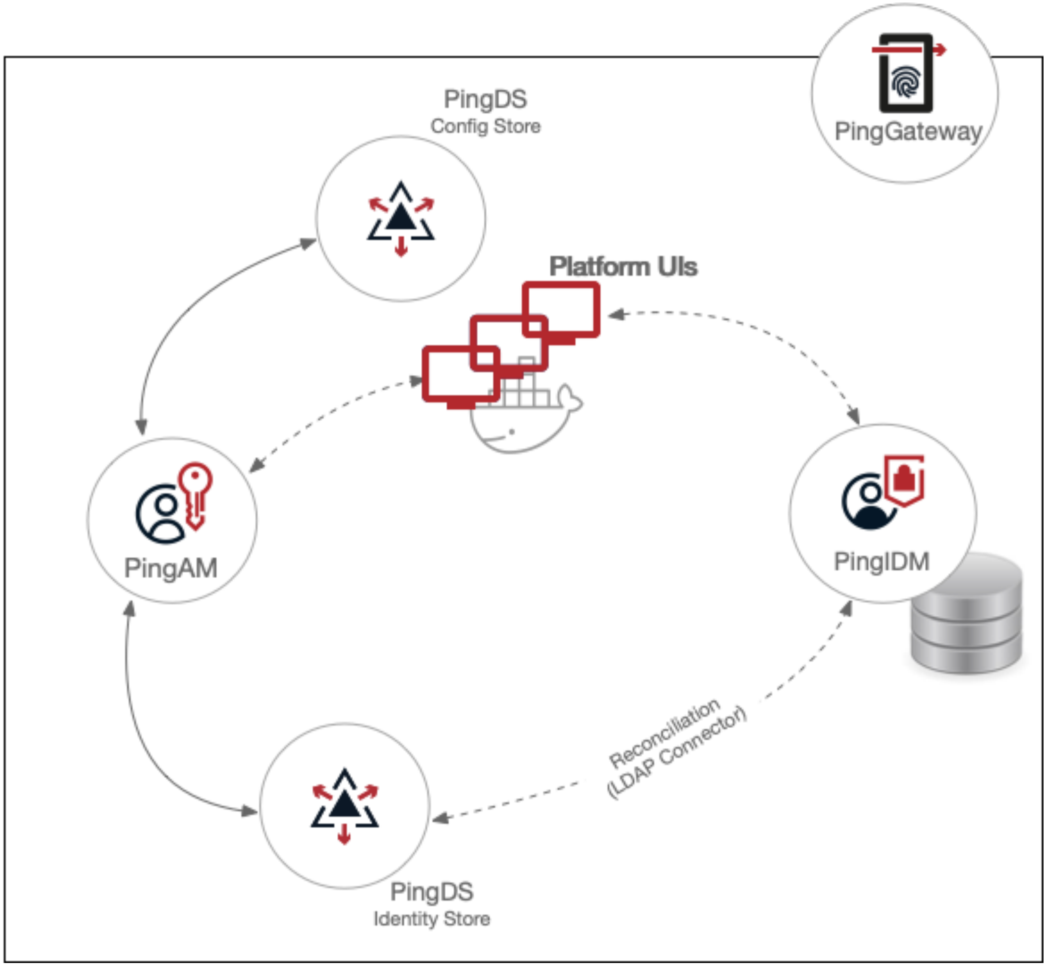
This page describes the sample deployment alternatives and how the Advanced Identity Software components interact.

> **Important**
>
> This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
> Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:
>
> - To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
> - To deploy in Kubernetes, start with the ForgeOps reference implementation.
>
> For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

Sample: separate identity stores

This sample deployment has an external PingDS server configured as the PingAM configuration store and PingAM identity store (shown separately in the illustration). The PingIDM repository is an external JDBC database. The sample was tested with MySQL. The deployment uses an LDAP connector to synchronize the identities between PingIDM and PingAM.

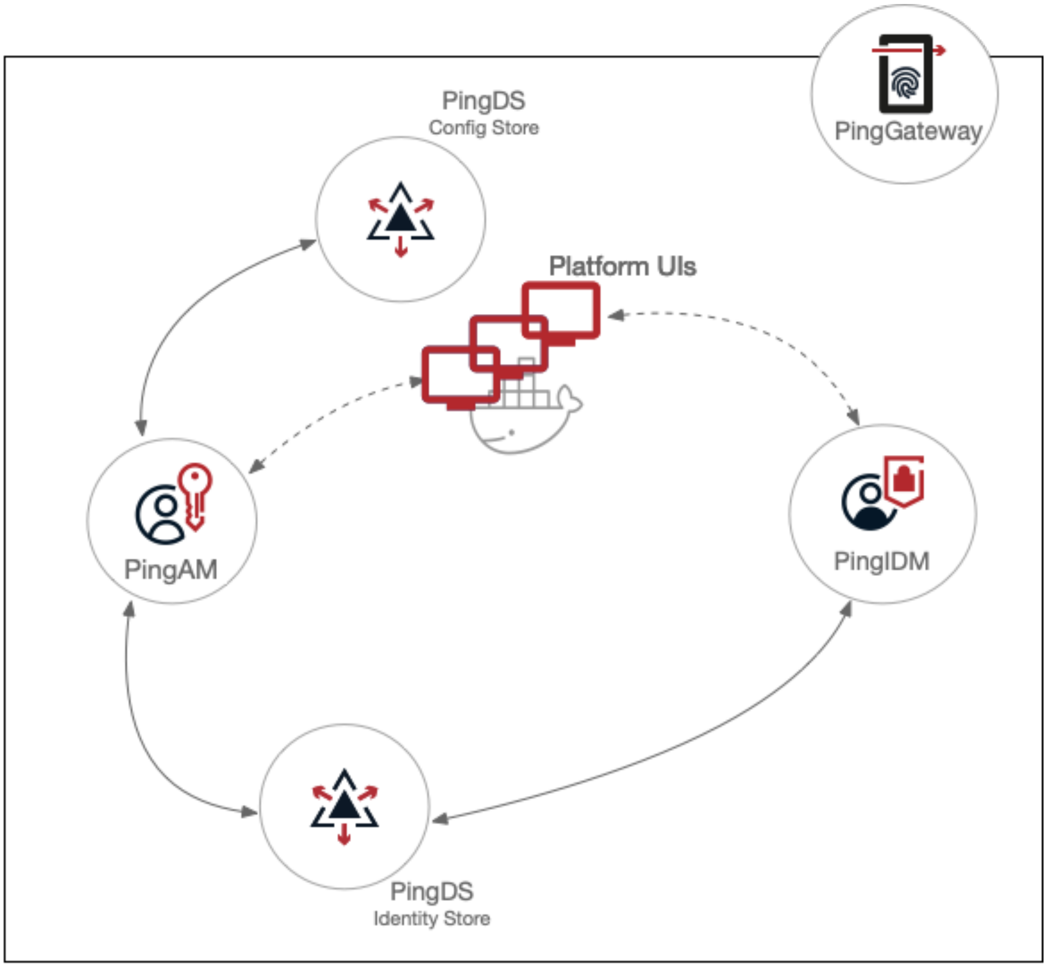
PingGateway serves as a single-point of entry for Platform UI access:



Sample: shared identity store

This sample deployment has an external PingDS server configured as the PingAM configuration store and shared by the PingAM and PingIDM servers share an external PingDS server as the identity store (shown separately in the illustration). No synchronization configuration is required.

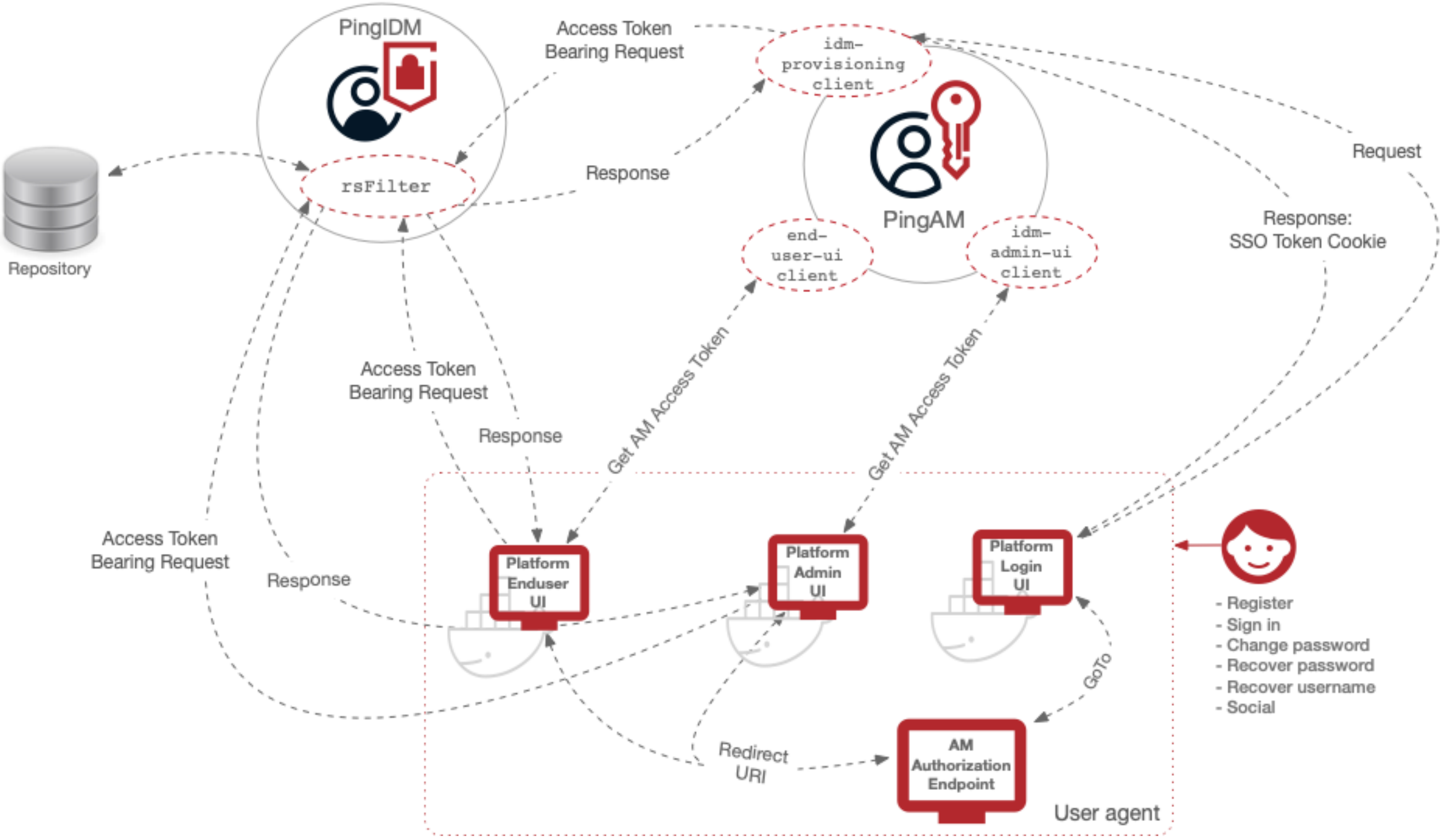
PingGateway serves as a single-point of entry for Platform UI access:



> **Important**
>
> In both sample deployments, the Platform UIs can run in separate Docker containers. If you want to run the Platform UIs in containers, get Docker before you start.

Component interaction

An Advanced Identity Software configuration relies on multiple components working together. The following image shows how the PingAM OAuth 2 clients interact with the PingIDM resource server filter (`rsFilter`) to grant access through the Platform UIs:



1. The Platform UIs send a request to the PingAM Authorization Endpoint.

2. If the end user is authenticated, the user agent is redirected back to the UI, according to the Redirection URI request parameter.

3. If the end user is not authenticated, the PingAM Authorization Endpoint redirects the user agent to the Advanced Identity Software Login UI.

4. After successful authentication, the Advanced Identity Software Login UI redirects the user agent back to the PingAM Authorization Endpoint, according to the GoTo request parameter.

Important

- Don't use the PingIDM stand-alone End User UI if you're running PingAM and PingIDM in an Advanced Identity Software deployment. This UI is delivered with PingIDM under `ui/enduser`. It is *not* the same as the `platform-enduser` UI and won't work in an Advanced Identity Software deployment.
- By default, the IDM admin UI only supports users from the PingAM root realm. If you need to support users from other realms, adjust the `/oauth2/` references in the `/path/to/openidm/ui/admin/default/index.html` file:

```
commonSettings.authorizationEndpoint = calculatedAMUriLink.href + '/oauth2/authorize';

AppAuthHelper.init({
    clientId: commonSettings.clientId,
    authorizationEndpoint: commonSettings.authorizationEndpoint,
    tokenEndpoint: calculatedAMUriLink.href + '/oauth2/access_token',
    revocationEndpoint: calculatedAMUriLink.href + '/oauth2/token/revoke',
    endSessionEndpoint: calculatedAMUriLink.href + '/oauth2/connect/endSession',
```

For example, if your realm is named `alpha`, replace `/oauth2/` with `/oauth2/realms/root/realms/alpha/`.

Next step

Choose your sample

□ Prepare the servers

Separate identity stores

□ Set up PingDS

□ Set up PingAM

□ Set up PingIDM

Shared identity store

□ Set up PingDS

□ Set up PingAM

□ Set up PingIDM

□ Protect the deployment

□ Set up the Platform UIs

□ Test your deployment

Prepare the servers

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

Downloads

The sample deployments use software from the Download Center and other locations.

Download the required files for your sample deployment:

Component	Downloads

PingAM	`AM-8.0.1.zip`
	`Amster-8.0.1.zip`
	`apache-tomcat-10.1.36.zip` [1]

PingDS	`DS-8.0.0.zip`
PingIDM	`IDM-8.0.0.zip` `mysql-8.0.29-macos12-x86_64.tar.gz` [2] `mysql-connector-java-8.0.29.zip` [2]
PingGateway	`PingGateway-2025.3.0.zip`
Platform UI[3]	`PlatformUI-8.*.zip`

[1] Download this from the Apache site; Ping Identity doesn't ship Apache software.

[2] Separate identity stores deployment only. Download these from the MySQL site; Ping Identity doesn't ship MySQL software.

[3] (Optional) The alternative is to pull the Platform UI Docker images.

Basic requirements

These basic evaluation requirements concern the sample deployments.

Before deploying the platform in production, *make sure the deployment meets all the requirements for each server component product*. Find more information in the release notes for each product.

Memory and disk space

These are basic sizing requirements for each sample deployment:

- Reserve 1 GB RAM for each server product you install.
- Reserve at least 10 GB disk space for the files.
- For PingDS, reserve 5% + 1 GB of the filesystem size as free disk space.

Note

Production deployments may have very different requirements.

Operating systems

Common supported operating systems for the server component products include:

- Microsoft Windows Server
- Red Hat Enterprise Linux and Rocky Linux
- Ubuntu Linux

For details, read the release notes for each product.

Tip

Although it is not a supported operating system for the platform, you can try the sample deployments on Apple macOS.

Java

Supported Java versions for the sample deployments include:

- Java 21

Before installing the servers, set the `JAVA_HOME` environment variable.

Keep your Java installation up-to-date with the latest security fixes. Some security features may require later updates.

For details, read the release notes for each product.

Hosts and ports

The sample deployments use these hostnames and port numbers:

Component	Hostname	Port numbers
PingAM	`am.example.com`	HTTP: `8081`

Component	Hostname	Port numbers
PingDS	`directory.example.com`	Admin: `4444` LDAP: `1389` LDAPS: `1636`
PingIDM	`openidm.example.com`	HTTP: `8080` HTTPS: `8443`
PingGateway	`platform.example.com`	HTTP: `9080` HTTPS: `9443`
Platform Admin UI	`admin.example.com`	HTTP: `8082`
Platform End User UI	`enduser.example.com`	HTTP: `8888`
Platform Login UI	`login.example.com`	HTTP: `8083`

Sample deployment on a single computer

The sample deployments assume all servers are deployed on their own hosts.

> **Tip**
>
> To try a sample deployment on a single computer, the recommended alternative is to use an out-of-the-box ForgeOps deployment.
> If you nevertheless choose to demonstrate these sample deployments on your computer, add aliases for the fully qualified domain names used for the servers and Platform UIs to your `/etc/hosts` file:

```
127.0.0.1    admin.example.com
127.0.0.1    am.example.com
127.0.0.1    directory.example.com
127.0.0.1    enduser.example.com
127.0.0.1    login.example.com
127.0.0.1    openidm.example.com
127.0.0.1    platform.example.com
```

Next step

Choose your sample

Prepare the servers

Separate identity stores

- □ Set up PingDS
- □ Set up PingAM
- □ Set up PingIDM

Shared identity store

- □ Set up PingDS
- □ Set up PingAM
- □ Set up PingIDM

□ Protect the deployment

□ Set up the Platform UIs

□ Test your deployment

Separate identity stores

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.

Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

This sample deployment uses the following data stores:

- An external PingDS server as the PingAM configuration store and the PingAM identity store.
- A MySQL repository as the PingIDM data store.

Note

The PingIDM End User UI is not supported in an Advanced Identity Software deployment, as it does not support authentication through PingAM.

You can use the Set up the Platform UIs with this deployment, or create your own UIs that support authentication through PingAM.

Download PingDS

Follow the instructions in the PingDS documentation to download PingDS, and prepare for installation.

The instructions that follow assume you download the cross-platform .zip distribution.

Set up PingDS

1. Unpack the PingDS files you downloaded.
2. Generate and save a unique PingDS deployment ID:

   ```
   /path/to/opendj/bin/dskeymgr create-deployment-id --deploymentIdPassword password
   ```

   You will need the deployment ID and password to install PingDS, and to export the server certificate.

   Set the deployment ID in your environment:

   ```
   export DEPLOYMENT_ID=deployment-id
   ```

3. Install a PingDS server with the necessary setup profiles:

   - `am-config`
   - `am-cts`
   - `am-identity-store`

   For more information about PingDS setup profiles, refer to setup profiles in the PingDS documentation.

```
/path/to/opendj/setup \
--deploymentId $DEPLOYMENT_ID \
--deploymentIdPassword password \
--rootUserDN uid=admin \
--rootUserPassword str0ngAdm1nPa55word \
--monitorUserPassword str0ngMon1torPa55word \
--hostname directory.example.com \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--profile am-config \
--set am-config/amConfigAdminPassword:5up35tr0ng \
--profile am-cts \
--set am-cts/amCtsAdminPassword:5up35tr0ng \
--set am-cts/tokenExpirationPolicy:am-sessions-only \
--profile am-identity-store \
--set am-identity-store/amIdentityStoreAdminPassword:5up35tr0ng \
--acceptLicense
```

> **Note**
>
> For simplicity, this sample deployment uses a standalone directory server that:
>
> - Does not replicate directory data (no `--replicationPort` or `--bootstrapReplicationServer` options).
>   In production deployments, always replicate directory data for availability and resilience.
> - Consolidates all directory data in the same replicas.
>   In very high-volume production deployments, test whether this meets your performance requirements and adjust your directory deployment if necessary.
> - Keeps PingAM identity data and PingIDM repository data under distinct base DNs.
>   Both PingAM and PingIDM expect exclusive access to their data. *Keep their data separate with distinct base DNs and domains in your setup profiles.* Don't accidentally mix their data by choosing a base DN under the other base DN.
>
> For details, refer to the PingDS installation documentation.

4. Start the PingDS server:

```
/path/to/opendj/bin/start-ds
```

Set up a container

Install a Java container to deploy PingAM.

These deployment examples assume you're using Apache Tomcat:

1. Follow the instructions in the PingAM documentation to prepare your environment.
2. Use a supported version of Apache Tomcat as the web application container:
    1. Configure Tomcat to listen on port `8081`.

       This non-default port requires that you update Tomcat's `conf/server.xml` file. Instead of the default line, `<Connector port="8080" protocol="HTTP/1.1">`, use:

       ```
       <Connector port="8081" protocol="HTTP/1.1">
       ```

    2. Create a Tomcat `bin/setenv.sh` or `bin\setenv.bat` file to hold your environment variables.
    3. Follow the instructions in the PingAM documentation to prepare Tomcat as the web application container.

You can find complete instructions on setting up Tomcat in the PingAM documentation.

Secure connections

> **Important**
>
> From PingDS 7 onwards, you *must* secure connections to PingDS servers.

1. Create a new directory that will house a dedicated truststore for PingAM:

   ```
   mkdir -p /path/to/openam-security/
   ```

2. Export the PingDS server certificate.

   You must run this command on `directory.example.com` in the terminal window where you set the `DEPLOYMENT_ID` variable:

   ```
   /path/to/opendj/bin/dskeymgr export-ca-cert \
   --deploymentId $DEPLOYMENT_ID \
   --deploymentIdPassword password \
   --outputFile ds-ca-cert.pem
   ```

3. Import the PingDS server certificate into the dedicated PingAM truststore.

   If you're not testing this example on a single host, you might need to copy each certificate file onto the PingAM host machine first:

```
keytool \
-importcert \
-trustcacerts \
-alias ds-ca-cert \
-file /path/to/ds-ca-cert.pem \
-keystore /path/to/openam-security/truststore \
-storepass changeit \
-storetype JKS
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

4. List the certificates in the new truststore and verify that the certificate you added is there:

   ```
   keytool \
   -list \
   -keystore /path/to/openam-security/truststore \
   -storepass changeit
   ```

5. Point Apache Tomcat to the path of the new truststore so that PingAM can access it.

   Append the truststore settings to the `CATALINA_OPTS` variable in the Tomcat `bin/setenv.sh` file; for example:

   ```
   CATALINA_OPTS="-Djavax.net.ssl.trustStore=/path/to/openam-security/truststore \
   -Djavax.net.ssl.trustStorePassword=changeit \
   -Djavax.net.ssl.trustStoreType=jks"
   ```

   Refer to your specific container's documentation for information on configuring truststores.

6. Verify secure authentication to the PingDS server with the dedicated PingAM accounts.

   If you deployed PingAM and PingDS on separate computers, first copy the PingAM truststore to `/path/to/openam-security/truststore` on the computer where PingDS runs. Use the PingDS `ldapsearch` command to connect to PingDS using the local copy of the PingAM truststore:

```
/path/to/opendj/bin/ldapsearch \
--hostname directory.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-config,ou=admins,ou=am-config \
--bindPassword 5up35tr0ng \
--baseDn ou=am-config \
"(&)" \
1.1
dn: ou=am-config

dn: ou=admins,ou=am-config

dn: uid=am-config,ou=admins,ou=am-config

/path/to/opendj/bin/ldapsearch \
--hostname directory.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-identity-bind-account,ou=admins,ou=identities \
--bindPassword 5up35tr0ng \
--baseDn ou=identities \
"(&)" \
1.1
dn: ou=identities

dn: ou=people,ou=identities

dn: ou=groups,ou=identities

dn: ou=admins,ou=identities

dn: uid=am-identity-bind-account,ou=admins,ou=identities
```

Next step

Choose your sample

Prepare the servers

Separate identity stores

Set up PingDS

□ Set up PingAM

□ Set up PingIDM

□ Protect the deployment

□ Set up the Platform UIs

□ Test your deployment

Set up PingAM

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

When your external data stores are configured, follow these procedures to configure PingAM in an Advanced Identity Software deployment:

Install PingAM

1. Follow the instructions in the PingAM documentation to download PingAM. Make sure you download the `.zip` file, not just the `.war` file.
2. Extract the `.zip` file, locate the PingAM `.war` file, and copy the `.war` file to deploy in Apache Tomcat as `am.war`:

```
cp AM-8.0.1.war /path/to/tomcat/webapps/am.war
```

3. Start Tomcat if it's not already running.

   It can take Apache Tomcat several seconds to deploy PingAM.

4. Go to the deployed PingAM application; for example, `http://am.example.com:8081/am/`.
5. On the initial configuration page, click **Create New Configuration**.
6. Review the software license agreement. If you agree to the license, click **I accept the license agreement**, and click **Continue**.
7. Set a password for the default user, `amAdmin`.

   These instructions assume that the `amAdmin` password is `Passw0rd`.

8. On the **Server Settings** screen, enter the following details and click **Next**:

   Server URL

   `http://am.example.com:8081`

   Cookie Domain

   `example.com`

Platform Locale

`en_US`

Configuration Directory

`/path/to/am`

9. On the **Configuration Data Store Settings** page, enter the following details and click **Next**:

SSL/TLS Enabled

`Enabled` (PingDS requires TLS.)

Host Name

`ds.example.com`

Port

`1636`

Encryption Key

Keep the randomly generated key.

Root Suffix

`ou=am-config`

Login ID

`uid=am-config,ou=admins,ou=am-config`

Password

`5up35tr0ng`

Server configuration

Leave the `New deployment` option selected.

10. On the **User Data Store** page, enter the following details and click **Next**:

> **Note**
>
> The PingAM setup process requires that you configure an identity store.
> You will use this store later in an `alpha` realm for non-administrative identities.

This list reflects the PingDS identity store installed with the listed server settings.

User Data Store Type

Leave the `ForgeRock Directory Services (DS)` option selected.

SSL/TLS Enabled

`Enabled` (PingDS requires TLS.)

Directory Name

`directory.example.com`

Port

`1636`

Root Suffix

`ou=identities`

Login ID

`uid=am-identity-bind-account,ou=admins,ou=identities`

Password

`5up35tr0ng`

11. On the **Site Configuration** page, leave the `No` option selected and click **Next**.

12. Review the **Configurator Summary Details**, then click **Create Configuration**.

Use the external CTS store

Update the PingAM configuration to use PingDS as an external token store:

1. Log in to the AM admin UI as the `amAdmin` user.

2. Browse to **Configure** > **Server Defaults** > **CTS**.

3. Make the following changes, saving your work before switching tabs:

Setting	Choice
CTS Token Store > **Store Mode**	**External Token Store**
CTS Token Store > **Root Suffix**	`ou=famrecords,ou=openam-session,ou=tokens`
External Store Configuration > **SSL/TLS Enabled**	Selected
External Store Configuration > **Connection String(s)**	`localhost:1636`
External Store Configuration > **Login Id**	`uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`
External Store Configuration > **Password**	`5up35tr0ng`

Keep the defaults for all other settings.

Create a realm

The PingAM Top Level Realm should serve administration purposes only.

These steps create an `alpha` realm for non-administrative identities:

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. Click **+ New Realm**, and create the new realm with the following settings:
   - **Name**: `alpha`
   - Keep the defaults for all other settings.

For background information, see Realms in the PingAM documentation.

Configure OAuth clients

The deployment depends on the following OAuth 2.0 clients:

Client ID	Description	In Top Level Realm?	In subrealms?
`idm-resource-server`	PingIDM uses this confidential client to introspect access tokens through the `/oauth2/introspect` endpoint to obtain information about users. This client is not used for OAuth 2.0 flows, only to introspect tokens. It is not a *real* OAuth 2.0 client, in the traditional sense. Rather, it is an OAuth 2.0 Resource Server account, used exclusively for token introspection. As such, you do not need to specify redirect URIs or grant types. When you configure the client, PingAM adds the `Authorization Code` grant type to the client profile by default. This client does not use it, but there's no requirement to remove it from the client profile.	✓	✕
`idm-provisioning`	PingAM nodes in authentication journeys (trees) use this confidential client to authenticate through PingAM and provision identities through PingIDM. This client uses the client credentials flow, and does not authenticate resource owners other than itself.	✓	✓

Client ID	Description	In Top Level Realm?	In subrealms?
`idm-admin-ui`	The Advanced Identity Software Admin UI uses this public client to access Advanced Identity Software configuration features, such as identity management and journey (tree) editing. The client uses the implicit flow, and authenticates administrator users who are authorized to perform administrative operations.	✓	✓
`end-user-ui`	The End User UI uses this public client to access and present end-user profile data. The client uses the implicit flow, and authenticates end users who are authorized to view and edit their profiles.	✓	✓

When configuring a client in more than one realm, make sure the client configurations are identical.

Follow these steps to configure the OAuth 2.0 clients:

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. In **the Top Level Realm**, configure an `idm-resource-server` client to introspect access tokens:
   - Select **Applications** > **OAuth 2.0** > **Clients**, and click **Add Client**.
   - Enter the following details:
     - **Client ID**: `idm-resource-server`
     - **Client secret**: `password`

       The value of this field must match the `clientSecret` that you will set in the `rsFilter` module in the PingIDM authentication configuration (`/path/to/openidm/conf/authentication.json`) during your PingIDM setup.
     - **Scopes**:
       `am-introspect-all-tokens`
       `am-introspect-all-tokens-any-realm`
   - Click **Create**.
3. In **the Top Level Realm and the** `alpha` **realm**, configure identical `idm-provisioning` clients to make calls to PingIDM:
   - Select **Applications** > **OAuth 2.0** > **Clients**, and click **Add Client**.
   - Enter the following details:
     - **Client ID**: `idm-provisioning`
     - **Client secret**: `openidm`
     - **Scopes**: `fr:idm:*`

- Click **Create**.
- On the **Advanced** tab:
  - **Response Types**: Check that `token` is present (it should be there by default).
  - **Grant Types**: Remove `Authorization Code` and add `Client Credentials`.
- Click **Save Changes**.

4. In **the Top Level Realm and the** `alpha` **realm**, configure identical `idm-admin-ui` clients that will be used by the Advanced Identity Software Admin UI:
   - Select **Applications > OAuth 2.0 > Clients**, and click **Add Client**.
   - Enter the following details:
     - **Client ID**: `idm-admin-ui`
     - **Client Secret**: (no client secret is required)
     - **Redirection URIs**:

       ```
       http://openidm.example.com:8080/platform/appAuthHelperRedirect.html
       http://openidm.example.com:8080/platform/sessionCheck.html
       http://openidm.example.com:8080/admin/appAuthHelperRedirect.html
       http://openidm.example.com:8080/admin/sessionCheck.html
       http://admin.example.com:8082/appAuthHelperRedirect.html
       http://admin.example.com:8082/sessionCheck.html
       ```

     - **Scopes**:

       ```
       openid
       fr:idm:*
       ```

       > **Note**
       >
       > At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your PingIDM setup.

   - Click **Create**.
   - On the **Core** tab:
     - **Client type**: Select `Public`.
     - Click **Save Changes**.
   - On the **Advanced** tab:
     - **Grant Types**: Add `Implicit`.
     - **Token Endpoint Authentication Method**: Select `none`.
     - **Implied consent**: Enable.

- Click **Save Changes**.

5. In **the Top Level Realm and the `alpha` realm**, configure identical `end-user-ui` clients that will be used by the Advanced Identity Software End User UI:

   - Select **Applications > OAuth 2.0 > Clients** , and click **Add Client**.

   - Enter the following details:

     - **Client ID**: `end-user-ui`

     - **Client Secret**: (no client secret is required)

     - **Redirection URIs**:

       `http://enduser.example.com:8888/appAuthHelperRedirect.html`
       `http://enduser.example.com:8888/sessionCheck.html`

     - **Scopes**:

       `openid`
       `fr:idm:*`

       > **Note**
       >
       > At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your PingIDM setup.

   - Click **Create**.

   - On the **Core** tab:

     - **Client type**: Select `Public` .

     - Click **Save Changes**.

   - On the **Advanced** tab:

     - **Grant Types**: Add `Implicit` .

     - **Token Endpoint Authentication Method**: Select `none` .

     - **Implied Consent**: Enable.

     - Click **Save Changes**.

Configure OAuth 2.0 providers

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.

2. Configure a provider for the Top Level Realm:

   1. In the Top Level Realm, select **Services**, and click **Add a Service**.

   2. Under **Choose a service type**, select **OAuth2 Provider**.

    3. For **Client Registration Scope Allowlist**, add the following scopes:
       `am-introspect-all-tokens`
       `am-introspect-all-tokens-any-realm`
       `fr:idm:*`
       `openid`
    4. Click **Create**.
    5. On the **Advanced** tab, make sure that **Response Type Plugins** includes the following values:
       `id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler`
       `code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler`
    6. Click **Save Changes**.
    7. On the **Consent** tab, enable **Allow Clients to Skip Consent**.
    8. Click **Save Changes**.

3. Configure a provider for the `alpha` realm:
    1. In the `alpha` realm, select **Services**, and click **Add a Service**.
    2. Under **Choose a service type**, select **OAuth2 Provider**.
    3. For **Client Registration Scope Allowlist**, add the following scopes:
       `fr:idm:*`
       `openid`
    4. Click **Create**.
    5. On the **Advanced** tab, make sure that **Response Type Plugins** includes the following values:
       `id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler`
       `code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler`
    6. Click **Save Changes**.
    7. On the **Consent** tab, enable **Allow Clients to Skip Consent**.
    8. Click **Save Changes**.

Configure a PingIDM provisioning service

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. From the top menu, select **Configure > Global Services > IDM Provisioning**.

   The AM admin UI doesn't let you configure a PingIDM provisioning service in a realm.
3. Set the following fields:
    - **Enabled**
    - **Deployment URL**: `http://openidm.example.com:8080`
    - **Deployment Path**: `openidm`

- **IDM Provisioning Client**: `idm-provisioning`

> **Caution**
>
> Do *not* enable the **useInternalOAuth2Provider** setting in an Advanced Identity Software environment. Doing so can cause other OAuth 2.0 client requests to fail with the following error:
>
> `No client profile or more than one profile found.`

4. Click **Save Changes**.

> **Tip**
>
> If some resource strings in the AM admin UI don't resolve properly at setup time, the UI labels mentioned will show internal keys instead of the labels shown in the steps above. Use the following table to check that you have the correct service and fields:
>
> | UI label | Internal label |
> |---|---|
> | **IDM Provisioning** | `IdmIntegrationService` |
> | **Enabled** | `enabled` |
> | **Deployment URL** | `idmDeploymentUrl` |
> | **Deployment Path** | `idmDeploymentPath` |
> | **IDM Provisioning Client** | `idmProvisioningClient` |

Configure a validation service

The Platform UIs need this validation service allow listing for `goto` redirection.

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. In the `alpha` realm, select **Services**, and click **Add a Service**.
3. Under **Choose a service type**, select **Validation Service**.
4. For **Valid goto URL Resources**, add the URLs for the Platform UI:

```
http://admin.example.com:8082/*
http://admin.example.com:8082/*?*
http://login.example.com:8083/*
http://login.example.com:8083/*?*
http://enduser.example.com:8888/*
http://enduser.example.com:8888/*?*
```

5. Click **Create**.

Enable CORS support

Cross-origin resource sharing (CORS) lets user agents make requests across domains.

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. From the top menu, select **Configure** > **Global Services** > **CORS Service**.
3. On the **Secondary Configurations** tab, click **Add a Secondary Configuration**.
4. On the **New Configuration** screen, enter the following values:
   - **Name** : `Cors Configuration`
   - **Accepted Origins** :

     `http://login.example.com:8083`
     `http://admin.example.com:8082`
     `http://enduser.example.com:8888`
     `http://openidm.example.com:8080`
     `https://openidm.example.com:8443`

     List only the origins that will be hosting OAuth 2.0 clients (such as the `platform-enduser` and IDM admin UIs).
   - **Accepted Methods**:

     `DELETE`
     `GET`
     `HEAD`
     `PATCH`
     `POST`
     `PUT`
   - **Accepted Headers**:

     `accept-api-version`
     `authorization`
     `cache-control`
     `content-type`
     `if-match`
     `if-none-match`
     `user-agent`
     `x-forgerock-transactionid`
     `x-openidm-nosession`
     `x-openidm-password`
     `x-openidm-username`
     `x-requested-with`
   - **Exposed Headers**: `WWW-Authenticate`
5. Click **Create**.
6. On the **Cors Configuration** screen, set the following values:
   - **Enable the CORS filter**: Enable

    - **Max Age**: `600`
    - **Allow Credentials**: Enable

7. Click **Save Changes**.

Configure authentication journeys

Advanced Identity Software deployment relies on three authentication trees to enable authentication through PingAM. When you extract the PingAM `.zip` file, you'll get a `sample-trees-8.zip` file that contains a number of authentication trees, in JSON files. Use the Amster command-line utility to import these authentication trees into your PingAM configuration:

1. Extract the `sample-trees-8.zip` file, and list the sample trees in the `root/AuthTree` directory:

   ```
   ls /path/to/openam-samples/root/AuthTree
   Agent.json                         PlatformForgottenUsername.json
   Example.json                       PlatformLogin.json
   Facebook-ProvisionIDMAccount.json  PlatformProgressiveProfile.json
   Google-AnonymousUser.json          PlatformRegistration.json
   Google-DynamicAccountCreation.json PlatformResetPassword.json
   HmacOneTimePassword.json           PlatformUpdatePassword.json
   PersistentCookie.json              RetryLimit.json
   ```

2. In all the sample tree files, replace `"realm" : "/"` with `"realm" : "/alpha"`.

3. Download and install Amster.

4. Start Amster, then connect to your PingAM instance:

   ```
   ./amster
   Amster OpenAM Shell (version build build, JVM: version)
   Type ':help' or ':h' for help.
   ---------------------------------------------------------------------------
   am> connect --interactive http://am.example.com:8081/am
   Sign in
   User Name: amAdmin
   Password: Passw0rd
   amster am.example.com:8081> :exit
   ```

5. Import the sample authentication trees and nodes:

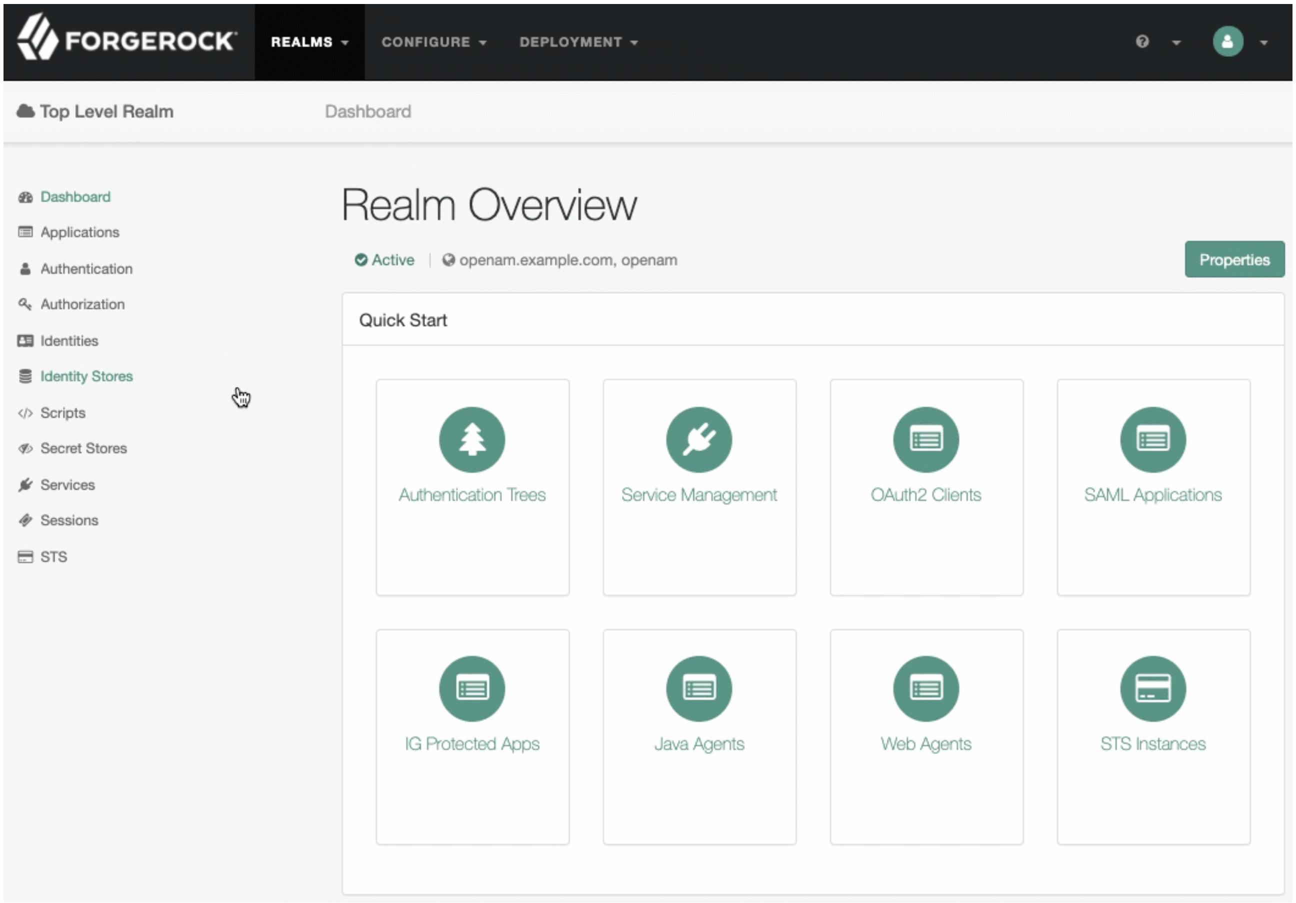
   ```
   amster am.example.com:8081>import-config --path /path/to/openam-samples/root
   Importing directory /path/to/openam-samples/root/AcceptTermsAndConditions
   ...
   Import completed successfully
   ```

6. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.

7. Configure the `PlatformRegistration` tree:

    - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformRegistration**.

- On the `PlatformRegistration` tree, add a `Success URL` node between `Increment Login Count` and `Success`, and set its value to `http://enduser.example.com:8888/?realm=alpha`.

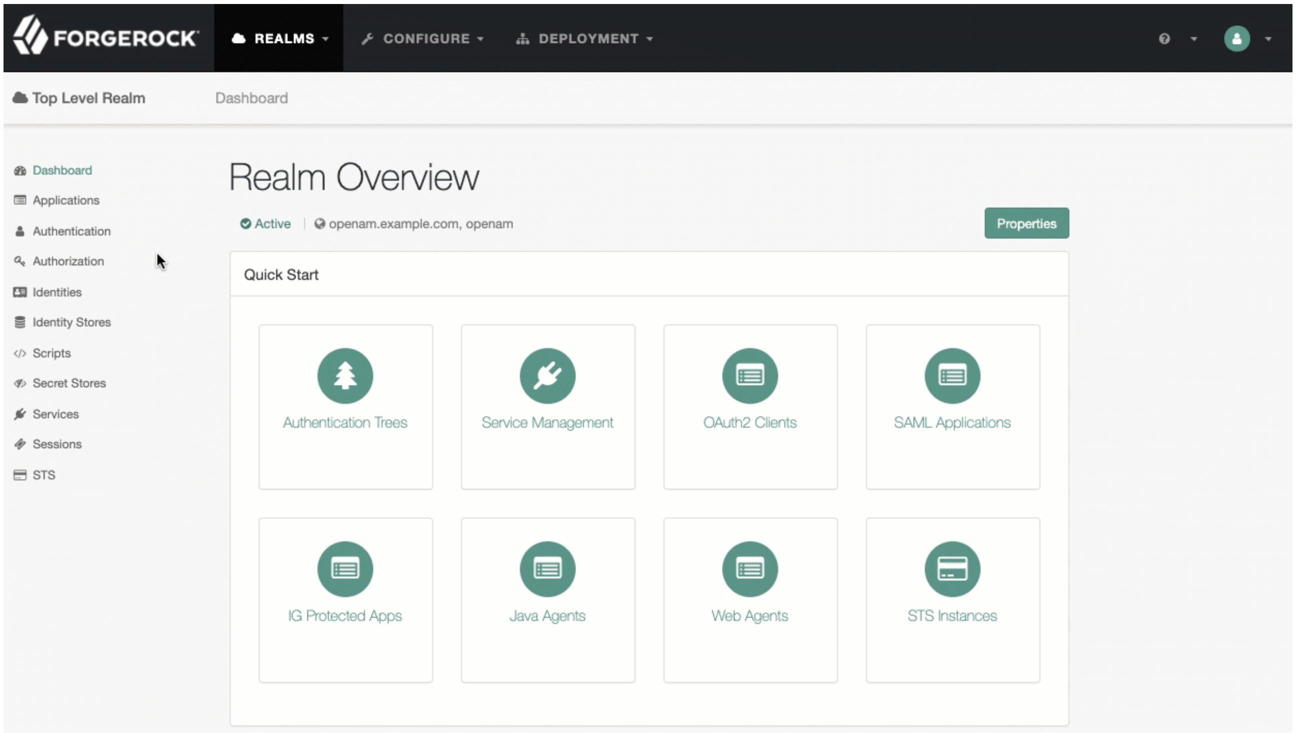


- Click **Save**.

8. Configure the `PlatformLogin` tree:

   - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformLogin**.

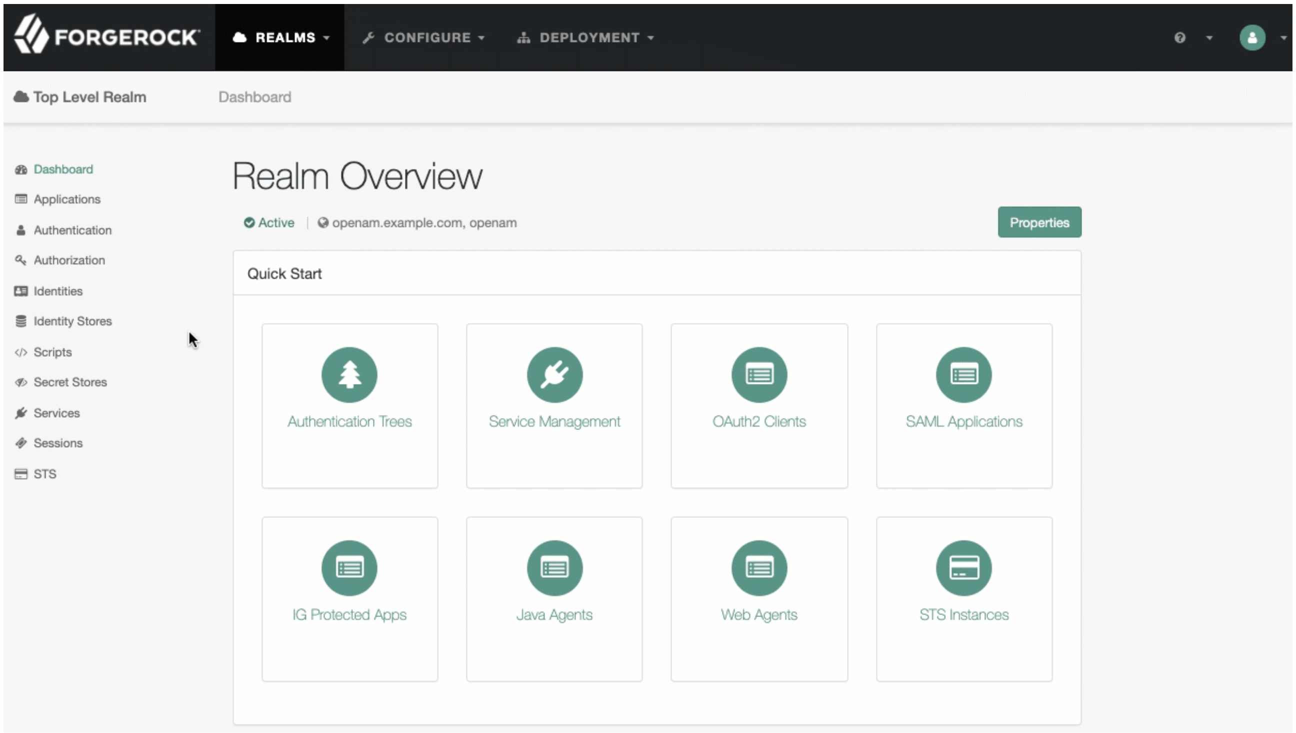
   - On the `PlatformLogin` tree, add a `Success URL` node between `Inner Tree Evaluator` and `Success`, and set its value to `http://enduser.example.com:8888/?realm=alpha`.



- Click **Save**.

9. Configure the `PlatformResetPassword` tree:

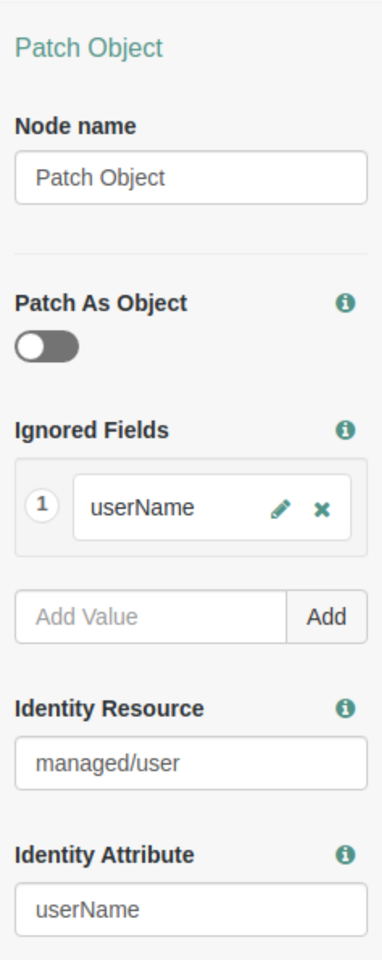
   - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformResetPassword**.
   - On the `PlatformResetPassword` tree, add a `Success URL` node between `Patch Object` and `Success`, and set its value to `http://enduser.example.com:8888/?realm=alpha`.



- Click **Save**.

10. Configure the `PlatformUpdatePassword` tree:

    - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformUpdatePassword**.
    - On the `PlatformUpdatePassword` tree, select the `Patch Object` node, and make sure `Patch As Object` is disabled.



Patch Object

Node name

Patch Object

Patch As Object

Ignored Fields

1 userName

Identity Resource

managed/user

Identity Attribute

userName

    - Click **Save**.

11. For the authentication trees that require email, set the **External Login Page URL**.

    - In the `alpha` realm, select **Authentication > Settings**, and click the **General** tab.
    - Set **External Login Page URL** to `http://login.example.com:8083` , then click **Save Changes**.

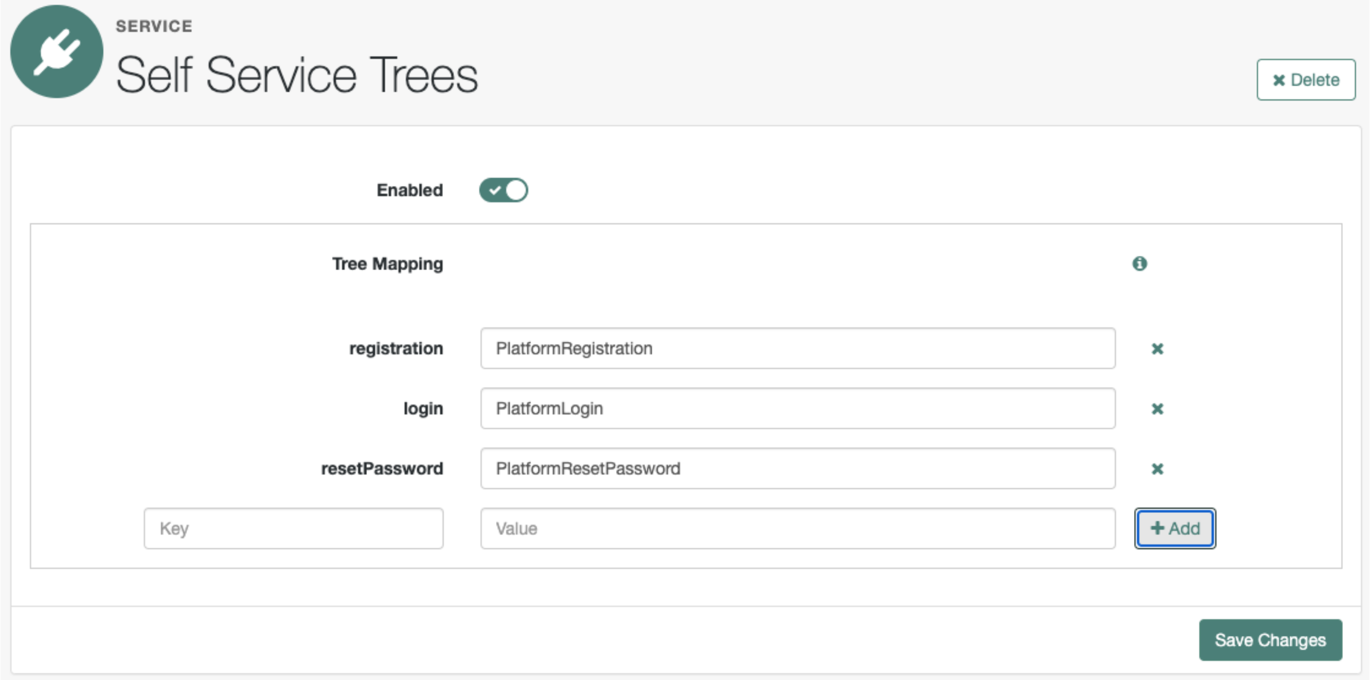
Map authentication trees

Map the Advanced Identity Software authentication trees to the corresponding Self-Service endpoints.

1. In the `alpha` realm, select **Services**, and click **Add a Service**.
2. Under **Choose a service type**, select **Self Service Trees**, and click **Create**.

3. Add the following tree mappings:

Key	Value
`registration`	`PlatformRegistration`
`login`	`PlatformLogin`
`resetPassword`	`PlatformResetPassword`



Enabled: ☑

Tree Mapping

registration: PlatformRegistration

login: PlatformLogin

resetPassword: PlatformResetPassword

4. Click **Save Changes**.

Additional PingAM configuration

The PingAM configuration shown is sufficient for this sample deployment. Read the PingAM documentation when using additional features.

For example, if PingAM runs behind a load balancer or a reverse proxy, configure a base URL source service. Adapt the PingAM configuration to use this service when you protect PingAM with PingGateway.

Restart Tomcat

Restart Tomcat to take all the configuration changes into account:

```
$ /path/to/tomcat/bin/shutdown.sh
# Wait a moment for Tomcat to shut down cleanly before starting it again.
$ /path/to/tomcat/bin/startup.sh
```

Next step

- Choose your sample
- Prepare the servers
- Separate identity stores
  - Set up PingDS
  - Set up PingAM
  - □ Set up PingIDM
- □ Protect the deployment
- □ Set up the Platform UIs
- □ Test your deployment

Set up PingIDM

> **Important**
>
> This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
> Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:
>
> - To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
> - To deploy in Kubernetes, start with the ForgeOps reference implementation.
>
> For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

This procedure sets up PingIDM with an external MySQL repository. The procedure reflects the listed server settings for installing PingIDM.

1. Follow the instructions in the PingIDM documentation to download, install, and run PingIDM.

   Before running PingIDM, make sure you set the `JAVA_HOME` environment variable.

2. Edit the `/path/to/openidm/resolver/boot.properties` file to set the hostname:

   ```
   openidm.host=openidm.example.com
   ```

3. Configure your PingIDM repository. This procedure was tested with a MySQL repository. Follow the instructions in the PingIDM documentation to set up a MySQL repository.

4. Configure social authentication.

In your project's `conf/managed.json` file:

- Add an `aliasList` property to the `user` object:

```
{
  "objects": [
    {
      "name": "user",
      ...
      "schema": {
        "properties": {
          ...
          "aliasList": {
            "title": "User Alias Names List",
            "description": "List of identity aliases used primarily to record social IdP subjects for this user",
            "type": "array",
            "items": {
              "type": "string",
              "title": "User Alias Names Items"
            },
            "viewable": false,
            "searchable": false,
            "userEditable": true,
            "returnByDefault": false,
            "isVirtual": false
          }
    ...
  ]
}
```

- Update the `password` property to ensure that users update their passwords through the self-service APIs, not directly:

```
"userEditable" : false
```

5. Change the authentication mechanism to `rsFilter` only:

   - Replace the default `conf/authentication.json` file with this authentication.json file.

   - Check that the `clientSecret` matches the `Client secret` that you set for the `idm-resource-server` client in PingAM (see Configure OAuth Clients).

   - Check that the `rsFilter` > `subjectMapping` > `propertyMapping` > `sub` property is correctly configured.

     The `authentication.json` file aligns with the default PingAM configuration for subject claim uniqueness. PingAM refers to the subject by its unique identifier, and so PingIDM does, too.

     If PingAM has its advanced server property, `org.forgerock.security.oauth2.enforce.sub.claim.uniqueness`, set to `false`, for example, because you upgraded from a previous release of PingAM, use this property mapping instead:

```
"propertyMapping": {
    "sub": "userName"
}
```

PingAM refers to the subject by its username in this case. For details, see the reference for the setting in the PingAM documentation.

For more information about authenticating using the `rsFilter`, see Authenticate through PingAM in the PingIDM documentation.

6. Edit the IDM admin UI configuration so that you can still authenticate through the IDM admin UI:

   - In your `conf/ui-configuration.json` file, insert a `platformSettings` object into the `configuration` object:

     ```
     {
         "configuration" : {
             "platformSettings" : {
                 "adminOauthClient" : "idm-admin-ui",
                 "adminOauthClientScopes" : "fr:idm:*",
                 "amUrl" : "http://am.example.com:8081/am",
                 "loginUrl" : ""
             }
         }
     }
     ```

     This object tells the IDM admin UI that it's operating in "Advanced Identity Software mode" (that is, as an OAuth 2.0 client of PingAM).

   - In your `conf/ui.context-admin.json` file, check that `X-Frame-Options` is set to `SAMEORIGIN`:

     ```
     {
       "enabled" : true,
       "cacheEnabled" : true,
       "urlContextRoot" : "/admin",
       "defaultDir" : "&{idm.install.dir}/ui/admin/default",
       "extensionDir" : "&{idm.install.dir}/ui/admin/extension",
       "responseHeaders" : {
           "X-Frame-Options" : "SAMEORIGIN"
       }
     }
     ```

   You should now be able to access the IDM admin UI at http://openidm.example.com:8080/admin. When you log in to the Admin UI, use the default PingAM administrative user (`amAdmin`), and not `openidm-admin`.

7. Configure the CORS servlet filter.

   Replace the default `conf/servletfilter-cors.json` file with this servletfilter-cors.json file.

8. Configure synchronization between the PingIDM repository and the PingAM identity store.

   - Add a configuration for the LDAP connector.

Create a configuration file named `provisioner.openicf-ldap.json` in the `/path/to/openidm/conf` directory. Use this provisioner.openicf-ldap.json file as a template.

Pay particular attention to the connection properties, `host`, `port`, `principal`, and `credentials`. These must match the configuration of the PingDS server that you set up as the identity store.

- Add a mapping between PingIDM managed user objects, and PingAM identities stored in PingDS.

  Create a mapping file named `sync.json` in the `/path/to/openidm/conf` directory. Use this sync.json file as a template.

9. Secure the connection to the PingDS server.

   This step assumes that you have set up PingDS and exported the PingDS CA certificate from `directory.example.com` (as shown in Step 4 of Secure connections).

   Import the PingDS CA certificate into the PingIDM truststore:

   ```
   keytool \
   -importcert \
   -alias ds-ca-cert \
   -file /path/to/ds-ca-cert.pem \
   -keystore /path/to/openidm/security/truststore \
   -storepass:file /path/to/openidm/security/storepass
   Owner: CN=Deployment key, O=ForgeRock.com
   Issuer: CN=Deployment key, O=ForgeRock.com
   ...
   Trust this certificate? [no]:  yes
   Certificate was added to keystore
   ```

10. Add the configuration to enable theming for hosted UI pages.

    1. Copy this ui-themerealm.json file to the `conf/` directory.

    2. In your `conf/access.json` file, insert a configuration object for the theme in the `configs` array:

       ```
       {
         "configs": [{
           "pattern": "config/ui/themerealm",
           "roles": "*",
           "methods": "read",
           "actions": "*"
         }]
       }
       ```

11. If you want to use the `PlatformForgottenUsername` or `PlatformResetPassword` trees, configure outbound email.

    > **Note**
    >
    > After you have installed the Platform UIs, you can configure email through the UI at `http://openidm.example.com:8080/admin`.

PingIDM is now configured for this deployment.

Next step

- Choose your sample
- Prepare the servers
- Separate identity stores
  - Set up PingDS
  - Set up PingAM
  - Set up PingIDM
- □ Protect the deployment
- □ Set up the Platform UIs
- □ Test your deployment

Shared identity store

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

This sample deployment uses a single external PingDS server as:

- The PingAM configuration store.
- The PingAM identity store.
- The PingIDM identity store.

Note

The PingIDM End User UI is not supported in an Advanced Identity Software deployment, as it does not support authentication through PingAM.
You can use the Set up the Platform UIs with this deployment, or create your own UIs that support authentication through PingAM.

Download PingDS

Follow the instructions in the PingDS documentation to download PingDS, and prepare for installation.

The instructions that follow assume you download the cross-platform .zip distribution.

Set up PingDS

1. Unpack the PingDS files you downloaded.
2. Generate and save a unique PingDS deployment ID:

   ```
   /path/to/opendj/bin/dskeymgr create-deployment-id --deploymentIdPassword password
   ```

   You will need the deployment ID and password to install PingDS, and to export the server certificate.

   Set the deployment ID in your environment:

   ```
   export DEPLOYMENT_ID=deployment-id
   ```

3. Install a PingDS server with the necessary setup profiles:
    - `am-config`
    - `am-cts`
    - `am-identity-store`
    - `idm-repo`

   For more information about PingDS setup profiles, refer to setup profiles in the PingDS documentation.

```
/path/to/opendj/setup \
--deploymentId $DEPLOYMENT_ID \
--deploymentIdPassword password \
--rootUserDN uid=admin \
--rootUserPassword str0ngAdm1nPa55word \
--monitorUserPassword str0ngMon1torPa55word \
--hostname directory.example.com \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--profile am-config \
--set am-config/amConfigAdminPassword:5up35tr0ng \
--profile am-cts \
--set am-cts/amCtsAdminPassword:5up35tr0ng \
--set am-cts/tokenExpirationPolicy:am-sessions-only \
--profile am-identity-store \
--set am-identity-store/amIdentityStoreAdminPassword:5up35tr0ng \
--profile idm-repo \
--set idm-repo/domain:forgerock.io \
--acceptLicense
```

> **Note**
>
> For simplicity, this sample deployment uses a standalone directory server that:
>
> - Does not replicate directory data (no `--replicationPort` or `--bootstrapReplicationServer` options).
>   In production deployments, always replicate directory data for availability and resilience.
> - Consolidates all directory data in the same replicas.
>   In very high-volume production deployments, test whether this meets your performance requirements and adjust your directory deployment if necessary.
> - Keeps PingAM identity data and PingIDM repository data under distinct base DNs.
>   Both PingAM and PingIDM expect exclusive access to their data. *Keep their data separate with distinct base DNs and domains in your setup profiles.* Don't accidentally mix their data by choosing a base DN under the other base DN.
>
> For details, refer to the PingDS installation documentation.

4. Start the PingDS server:

```
/path/to/opendj/bin/start-ds
```

Set up a container

Install a Java container to deploy PingAM.

These deployment examples assume you're using Apache Tomcat:

1. Follow the instructions in the PingAM documentation to prepare your environment.
2. Use a supported version of Apache Tomcat as the web application container:
    1. Configure Tomcat to listen on port `8081`.

        This non-default port requires that you update Tomcat's `conf/server.xml` file. Instead of the default line, `<Connector port="8080" protocol="HTTP/1.1">`, use:

        ```
        <Connector port="8081" protocol="HTTP/1.1">
        ```

    2. Create a Tomcat `bin/setenv.sh` or `bin\setenv.bat` file to hold your environment variables.
    3. Follow the instructions in the PingAM documentation to prepare Tomcat as the web application container.

You can find complete instructions on setting up Tomcat in the PingAM documentation.

Secure connections

> **Important**
>
> From PingDS 7 onwards, you *must* secure connections to PingDS servers.

1. Create a new directory that will house a dedicated truststore for PingAM:

    ```
    mkdir -p /path/to/openam-security/
    ```

2. Export the PingDS server certificate.

    You must run this command on `directory.example.com` in the terminal window where you set the `DEPLOYMENT_ID` variable:

    ```
    /path/to/opendj/bin/dskeymgr export-ca-cert \
    --deploymentId $DEPLOYMENT_ID \
    --deploymentIdPassword password \
    --outputFile ds-ca-cert.pem
    ```

3. Import the PingDS server certificate into the dedicated PingAM truststore.

    If you're not testing this example on a single host, you might need to copy each certificate file onto the PingAM host machine first:

```
keytool \
-importcert \
-trustcacerts \
-alias ds-ca-cert \
-file /path/to/ds-ca-cert.pem \
-keystore /path/to/openam-security/truststore \
-storepass changeit \
-storetype JKS
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

4. List the certificates in the new truststore and verify that the certificate you added is there:

```
keytool \
-list \
-keystore /path/to/openam-security/truststore \
-storepass changeit
```

5. Point Apache Tomcat to the path of the new truststore so that PingAM can access it.

   Append the truststore settings to the `CATALINA_OPTS` variable in the Tomcat `bin/setenv.sh` file; for example:

```
CATALINA_OPTS="-Djavax.net.ssl.trustStore=/path/to/openam-security/truststore \
-Djavax.net.ssl.trustStorePassword=changeit \
-Djavax.net.ssl.trustStoreType=jks"
```

   Refer to your specific container’s documentation for information on configuring truststores.

6. Verify secure authentication to the PingDS server with the dedicated PingAM accounts.

   If you deployed PingAM and PingDS on separate computers, first copy the PingAM truststore to `/path/to/openam-security/truststore` on the computer where PingDS runs. Use the PingDS `ldapsearch` command to connect to PingDS using the local copy of the PingAM truststore:

```
/path/to/opendj/bin/ldapsearch \
--hostname directory.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-config,ou=admins,ou=am-config \
--bindPassword 5up35tr0ng \
--baseDn ou=am-config \
"(&)" \
1.1
dn: ou=am-config

dn: ou=admins,ou=am-config

dn: uid=am-config,ou=admins,ou=am-config

/path/to/opendj/bin/ldapsearch \
--hostname directory.example.com \
--port 1636 \
--useSsl \
--useJavaTrustStore /path/to/openam-security/truststore \
--trustStorePassword changeit \
--bindDn uid=am-identity-bind-account,ou=admins,ou=identities \
--bindPassword 5up35tr0ng \
--baseDn ou=identities \
"(&)" \
1.1
dn: ou=identities

dn: ou=people,ou=identities

dn: ou=groups,ou=identities

dn: ou=admins,ou=identities

dn: uid=am-identity-bind-account,ou=admins,ou=identities
```

Next step

Choose your sample

Prepare the servers

Shared identity store

- Set up PingDS
- □ Set up PingAM
- □ Set up PingIDM

□ Protect the deployment

□ Set up the Platform UIs

□ Test your deployment

Set up PingAM

> **Important**
>
> This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
>
> Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:
>
> - To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
> - To deploy in Kubernetes, start with the ForgeOps reference implementation.
>
> For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

When your external data stores are configured, follow these procedures to configure PingAM in an Advanced Identity Software deployment:

Install PingAM

1. Follow the instructions in the PingAM documentation to download PingAM. Make sure you download the `.zip` file, not just the `.war` file.
2. Extract the `.zip` file, locate the PingAM `.war` file, and copy the `.war` file to deploy in Apache Tomcat as `am.war`:

   ```
   cp AM-8.0.1.war /path/to/tomcat/webapps/am.war
   ```

3. Start Tomcat if it's not already running.

   It can take Apache Tomcat several seconds to deploy PingAM.

4. Go to the deployed PingAM application; for example, `http://am.example.com:8081/am/`.
5. On the initial configuration page, click **Create New Configuration**.
6. Review the software license agreement. If you agree to the license, click **I accept the license agreement**, and click **Continue**.
7. Set a password for the default user, `amAdmin`.

   These instructions assume that the `amAdmin` password is `Passw0rd`.

8. On the **Server Settings** screen, enter the following details and click **Next**:

   Server URL

   `http://am.example.com:8081`

   Cookie Domain

   `example.com`

Platform Locale

`en_US`

Configuration Directory

`/path/to/am`

9. On the **Configuration Data Store Settings** page, enter the following details and click **Next**:

SSL/TLS Enabled

`Enabled` (PingDS requires TLS.)

Host Name

`ds.example.com`

Port

`1636`

Encryption Key

Keep the randomly generated key.

Root Suffix

`ou=am-config`

Login ID

`uid=am-config,ou=admins,ou=am-config`

Password

`5up35tr0ng`

Server configuration

Leave the `New deployment` option selected.

10. On the **User Data Store** page, enter the following details and click **Next**:

> ⓘ **Note**
>
> The PingAM setup process requires that you configure an identity store.
> You will use this store later in an `alpha` realm for non-administrative identities.

This list reflects the PingDS identity store installed with the listed server settings.

User Data Store Type

Leave the `ForgeRock Directory Services (DS)` option selected.

SSL/TLS Enabled

`Enabled` (PingDS requires TLS.)

Directory Name

`directory.example.com`

Port

`1636`

Root Suffix

`ou=identities`

Login ID

`uid=am-identity-bind-account,ou=admins,ou=identities`

Password

`5up35tr0ng`

11. On the **Site Configuration** page, leave the `No` option selected and click **Next**.
12. Review the **Configurator Summary Details**, then click **Create Configuration**.

Use the external CTS store

Update the PingAM configuration to use PingDS as an external token store:

1. Log in to the AM admin UI as the `amAdmin` user.
2. Browse to **Configure** > **Server Defaults** > **CTS**.
3. Make the following changes, saving your work before switching tabs:

Setting	Choice
CTS Token Store > **Store Mode**	**External Token Store**
CTS Token Store > **Root Suffix**	`ou=famrecords,ou=openam-session,ou=tokens`
External Store Configuration > **SSL/TLS Enabled**	Selected
External Store Configuration > **Connection String(s)**	`localhost:1636`
External Store Configuration > **Login Id**	`uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`
External Store Configuration > **Password**	`5up35tr0ng`

Keep the defaults for all other settings.

Create a realm

The PingAM Top Level Realm should serve administration purposes only.

These steps create an `alpha` realm for non-administrative identities:

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. Click **+ New Realm**, and create the new realm with the following settings:
   - **Name**: `alpha`
   - Keep the defaults for all other settings.

For background information, see Realms in the PingAM documentation.

Change the identity store configuration for PingIDM

1. In the `alpha` realm, select **Identity Stores** then click **OpenDJ**.
2. On the **Server Settings** tab, set **LDAPv3 Plug-in Search Scope** to `SCOPE_ONE`, then click **Save Changes**.
3. On the **User Configuration** tab, set **LDAP Users Search Attribute** to `fr-idm-uuid`, then click **Save Changes**.
4. On the **Authentication Configuration** tab, check that the **Authentication Naming Attribute** is set to `uid`, then click **Save Changes**.
5. In the Top Level Realm, under **Identity Stores**, delete the configuration for **OpenDJ**.

   The deployment will store non-administrative identities in the `alpha` realm.

Configure OAuth clients

The deployment depends on the following OAuth 2.0 clients:

Client ID	Description	In Top Level Realm?	In subrealms?
`idm-resource-server`	PingIDM uses this confidential client to introspect access tokens through the `/oauth2/introspect` endpoint to obtain information about users. This client is not used for OAuth 2.0 flows, only to introspect tokens. It is not a *real* OAuth 2.0 client, in the traditional sense. Rather, it is an OAuth 2.0 Resource Server account, used exclusively for token introspection. As such, you do not need to specify redirect URIs or grant types. When you configure the client, PingAM adds the `Authorization Code` grant type to the client profile by default. This client does not use it, but there's no requirement to remove it from the client profile.	✓	✗
`idm-provisioning`	PingAM nodes in authentication journeys (trees) use this confidential client to authenticate through PingAM and provision identities through PingIDM. This client uses the client credentials flow, and does not authenticate resource owners other than itself.	✓	✓
`idm-admin-ui`	The Advanced Identity Software Admin UI uses this public client to access Advanced Identity Software configuration features, such as identity management and journey (tree) editing. The client uses the implicit flow, and authenticates administrator users who are authorized to perform administrative operations.	✓	✓
`end-user-ui`	The End User UI uses this public client to access and present end-user profile data. The client uses the implicit flow, and authenticates end users who are authorized to view and edit their profiles.	✓	✓

When configuring a client in more than one realm, make sure the client configurations are identical.

Follow these steps to configure the OAuth 2.0 clients:

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. In **the Top Level Realm**, configure an `idm-resource-server` client to introspect access tokens:
   - Select **Applications** > **OAuth 2.0** > **Clients**, and click **Add Client**.
   - Enter the following details:
     - **Client ID**: `idm-resource-server`

        - **Client secret**: `password`

          The value of this field must match the `clientSecret` that you will set in the `rsFilter` module in the PingIDM authentication configuration (`/path/to/openidm/conf/authentication.json`) during your PingIDM setup.

        - **Scopes**:
          `am-introspect-all-tokens`
          `am-introspect-all-tokens-any-realm`

    - Click **Create**.

3. In **the Top Level Realm and the** `alpha` **realm**, configure identical `idm-provisioning` clients to make calls to PingIDM:

    - Select **Applications** > **OAuth 2.0** > **Clients**, and click **Add Client**.

    - Enter the following details:

        - **Client ID**: `idm-provisioning`

        - **Client secret**: `openidm`

        - **Scopes**: `fr:idm:*`

    - Click **Create**.

    - On the **Advanced** tab:

        - **Response Types**: Check that `token` is present (it should be there by default).

        - **Grant Types**: Remove `Authorization Code` and add `Client Credentials` .

    - Click **Save Changes**.

4. In **the Top Level Realm and the** `alpha` **realm**, configure identical `idm-admin-ui` clients that will be used by the Advanced Identity Software Admin UI:

    - Select **Applications** > **OAuth 2.0** > **Clients**, and click **Add Client**.

    - Enter the following details:

        - **Client ID**: `idm-admin-ui`

        - **Client Secret**: (no client secret is required)

        - **Redirection URIs**:

          `http://openidm.example.com:8080/platform/appAuthHelperRedirect.html`
          `http://openidm.example.com:8080/platform/sessionCheck.html`
          `http://openidm.example.com:8080/admin/appAuthHelperRedirect.html`
          `http://openidm.example.com:8080/admin/sessionCheck.html`
          `http://admin.example.com:8082/appAuthHelperRedirect.html`
          `http://admin.example.com:8082/sessionCheck.html`

        - **Scopes**:

```
openid
fr:idm:*
```

> **Note**
>
> At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your PingIDM setup.

- Click **Create**.
- On the **Core** tab:
  - **Client type**: Select `Public`.
  - Click **Save Changes**.
- On the **Advanced** tab:
  - **Grant Types**: Add `Implicit`.
  - **Token Endpoint Authentication Method**: Select `none`.
  - **Implied consent**: Enable.
  - Click **Save Changes**.

5. In **the Top Level Realm and the `alpha` realm**, configure identical `end-user-ui` clients that will be used by the Advanced Identity Software End User UI:
   - Select **Applications > OAuth 2.0 > Clients** , and click **Add Client**.
   - Enter the following details:
     - **Client ID**: `end-user-ui`
     - **Client Secret**: (no client secret is required)
     - **Redirection URIs**:

       ```
       http://enduser.example.com:8888/appAuthHelperRedirect.html
       http://enduser.example.com:8888/sessionCheck.html
       ```

     - **Scopes**:

       ```
       openid
       fr:idm:*
       ```

       > **Note**
       >
       > At a minimum, the scopes that you set here must include the scopes that you will set in the `rsFilter` authentication configuration (`/path/to/openidm/conf/authentication.json`) during your PingIDM setup.

   - Click **Create**.

- On the **Core** tab:
    - **Client type**: Select `Public`.
    - Click **Save Changes**.
- On the **Advanced** tab:
    - **Grant Types**: Add `Implicit`.
    - **Token Endpoint Authentication Method**: Select `none`.
    - **Implied Consent**: Enable.
    - Click **Save Changes**.

Configure OAuth 2.0 providers

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. Configure a provider for the Top Level Realm:
    1. In the Top Level Realm, select **Services**, and click **Add a Service**.
    2. Under **Choose a service type**, select **OAuth2 Provider**.
    3. For **Client Registration Scope Allowlist**, add the following scopes:
`am-introspect-all-tokens`
`am-introspect-all-tokens-any-realm`
`fr:idm:*`
`openid`
    4. Click **Create**.
    5. On the **Advanced** tab, make sure that **Response Type Plugins** includes the following values:
`id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler`
`code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler`
    6. Click **Save Changes**.
    7. On the **Consent** tab, enable **Allow Clients to Skip Consent**.
    8. Click **Save Changes**.
3. Configure a provider for the `alpha` realm:
    1. In the `alpha` realm, select **Services**, and click **Add a Service**.
    2. Under **Choose a service type**, select **OAuth2 Provider**.
    3. For **Client Registration Scope Allowlist**, add the following scopes:
`fr:idm:*`
`openid`
    4. Click **Create**.

5. On the **Advanced** tab, make sure that **Response Type Plugins** includes the following values:
   `id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler`
   `code|org.forgerock.oauth2.core.AuthorizationCodeResponseTypeHandler`

6. Click **Save Changes**.

7. On the **Consent** tab, enable **Allow Clients to Skip Consent**.

8. Click **Save Changes**.

Configure a PingIDM provisioning service

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.

2. From the top menu, select **Configure** > **Global Services** > **IDM Provisioning**.

   The AM admin UI doesn't let you configure a PingIDM provisioning service in a realm.

3. Set the following fields:

   - **Enabled**
   - **Deployment URL**: `http://openidm.example.com:8080`
   - **Deployment Path**: `openidm`
   - **IDM Provisioning Client**: `idm-provisioning`

   > **Caution**
   >
   > Do *not* enable the **useInternalOAuth2Provider** setting in an Advanced Identity Software environment. Doing so can cause other OAuth 2.0 client requests to fail with the following error:
   > `No client profile or more than one profile found.`

4. Click **Save Changes**.

Tip

If some resource strings in the AM admin UI don't resolve properly at setup time, the UI labels mentioned will show internal keys instead of the labels shown in the steps above. Use the following table to check that you have the correct service and fields:

UI label	Internal label
IDM Provisioning	`IdmIntegrationService`
Enabled	`enabled`
Deployment URL	`idmDeploymentUrl`
Deployment Path	`idmDeploymentPath`
IDM Provisioning Client	`idmProvisioningClient`

Configure a validation service

The Platform UIs need this validation service allow listing for `goto` redirection.

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. In the `alpha` realm, select **Services**, and click **Add a Service**.
3. Under **Choose a service type**, select **Validation Service**.
4. For **Valid goto URL Resources**, add the URLs for the Platform UI:

```
http://admin.example.com:8082/*
http://admin.example.com:8082/*?*
http://login.example.com:8083/*
http://login.example.com:8083/*?*
http://enduser.example.com:8888/*
http://enduser.example.com:8888/*?*
```

5. Click **Create**.

Enable CORS support

Cross-origin resource sharing (CORS) lets user agents make requests across domains.

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. From the top menu, select **Configure** > **Global Services** > **CORS Service**.
3. On the **Secondary Configurations** tab, click **Add a Secondary Configuration**.
4. On the **New Configuration** screen, enter the following values:
   - **Name** : `Cors Configuration`

- **Accepted Origins** :

  `http://login.example.com:8083`
  `http://admin.example.com:8082`
  `http://enduser.example.com:8888`
  `http://openidm.example.com:8080`
  `https://openidm.example.com:8443`

  List only the origins that will be hosting OAuth 2.0 clients (such as the `platform-enduser` and IDM admin UIs).

- **Accepted Methods**:

  `DELETE`
  `GET`
  `HEAD`
  `PATCH`
  `POST`
  `PUT`

- **Accepted Headers**:

  `accept-api-version`
  `authorization`
  `cache-control`
  `content-type`
  `if-match`
  `if-none-match`
  `user-agent`
  `x-forgerock-transactionid`
  `x-openidm-nosession`
  `x-openidm-password`
  `x-openidm-username`
  `x-requested-with`

- **Exposed Headers**: `WWW-Authenticate`

5. Click **Create**.

6. On the **Cors Configuration** screen, set the following values:

   - **Enable the CORS filter**: Enable

   - **Max Age**: `600`

   - **Allow Credentials**: Enable

7. Click **Save Changes**.

Configure authentication journeys

Advanced Identity Software deployment relies on three authentication trees to enable authentication through PingAM. When you extract the PingAM `.zip` file, you'll get a `sample-trees-8.zip` file that contains a number of authentication trees, in JSON files. Use the Amster command-line utility to import these authentication trees into your PingAM configuration:

1. Extract the `sample-trees-8.zip` file, and list the sample trees in the `root/AuthTree` directory:

   ```
   ls /path/to/openam-samples/root/AuthTree
   Agent.json                         PlatformForgottenUsername.json
   Example.json                       PlatformLogin.json
   Facebook-ProvisionIDMAccount.json  PlatformProgressiveProfile.json
   Google-AnonymousUser.json          PlatformRegistration.json
   Google-DynamicAccountCreation.json PlatformResetPassword.json
   HmacOneTimePassword.json           PlatformUpdatePassword.json
   PersistentCookie.json              RetryLimit.json
   ```

2. In all the sample tree files, replace `"realm" : "/"` with `"realm" : "/alpha"`.

3. Download and install Amster.

4. Start Amster, then connect to your PingAM instance:

   ```
   ./amster
   Amster OpenAM Shell (version build build, JVM: version)
   Type ':help' or ':h' for help.
   ---------------------------------------------------------------------------
   am> connect --interactive http://am.example.com:8081/am
   Sign in
   User Name: amAdmin
   Password: Passw0rd
   amster am.example.com:8081> :exit
   ```

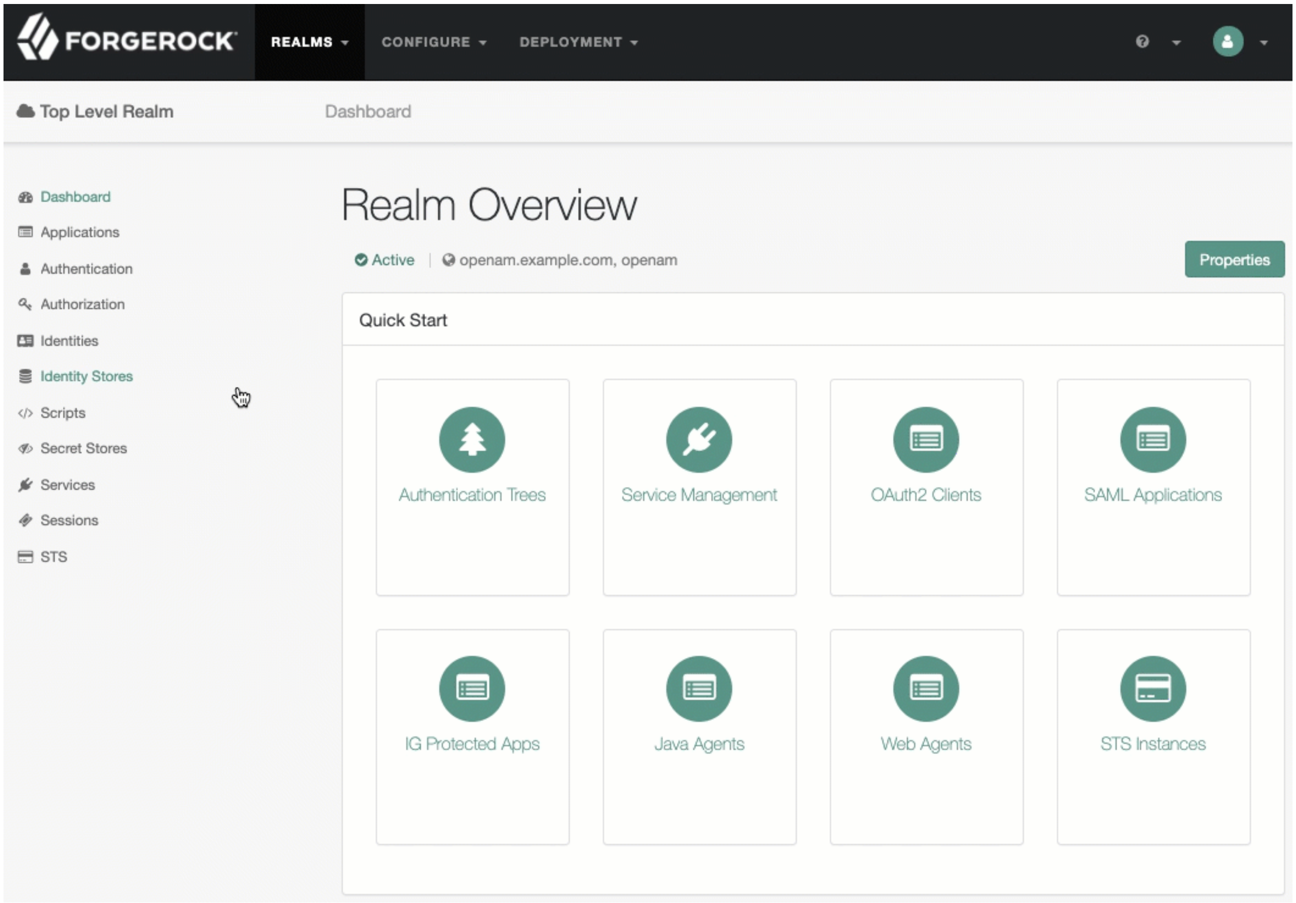
5. Import the sample authentication trees and nodes:

   ```
   amster am.example.com:8081>import-config --path /path/to/openam-samples/root
   Importing directory /path/to/openam-samples/root/AcceptTermsAndConditions
   ...
   Import completed successfully
   ```

6. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.

7. Configure the `PlatformRegistration` tree:

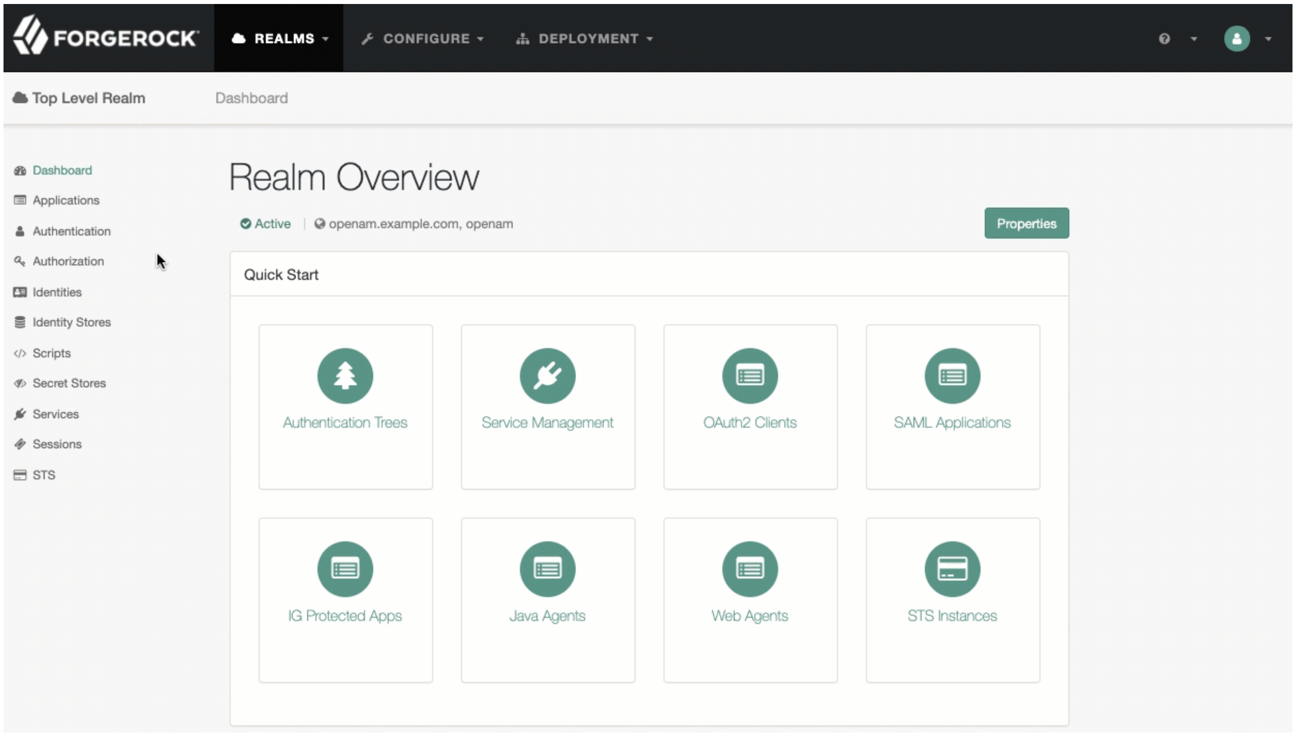
   - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformRegistration**.
   - On the `PlatformRegistration` tree, add a `Success URL` node between `Increment Login Count` and `Success`, and set its value to `http://enduser.example.com:8888/?realm=alpha`.



- Click **Save**.

8. Configure the `PlatformLogin` tree:

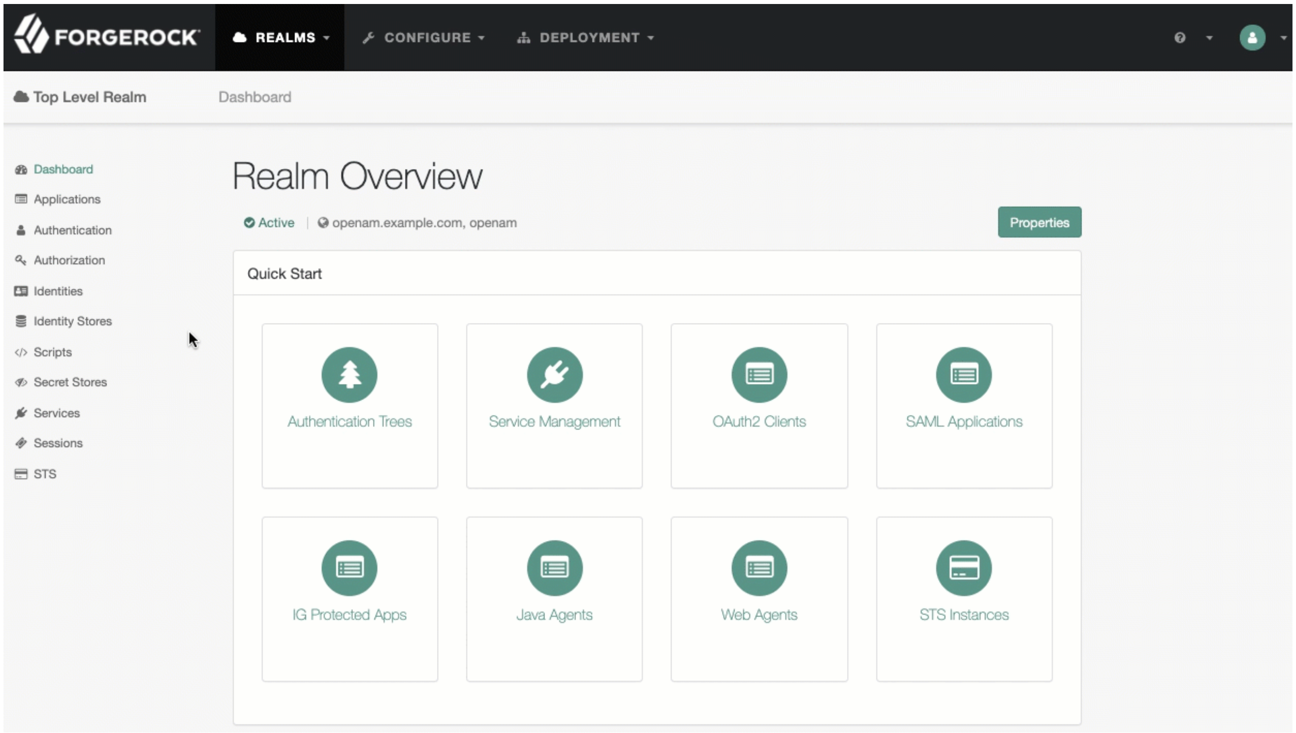
    - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformLogin**.
    - On the `PlatformLogin` tree, add a `Success URL` node between `Inner Tree Evaluator` and `Success`, and set its value to `http://enduser.example.com:8888/?realm=alpha`.



- Click **Save**.

9. Configure the `PlatformResetPassword` tree:

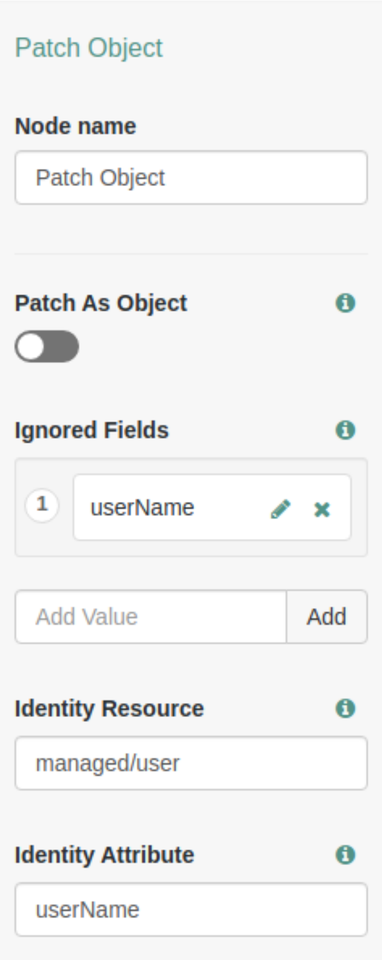
   - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformResetPassword**.
   - On the `PlatformResetPassword` tree, add a `Success URL` node between `Patch Object` and `Success`, and set its value to `http://enduser.example.com:8888/?realm=alpha`.



- Click **Save**.

10. Configure the `PlatformUpdatePassword` tree:

    - In the `alpha` realm, select **Authentication > Trees**, and click **PlatformUpdatePassword**.
    - On the `PlatformUpdatePassword` tree, select the `Patch Object` node, and make sure `Patch As Object` is disabled.



Patch Object

Node name

Patch Object

Patch As Object

Ignored Fields

1 userName

Identity Resource

managed/user

Identity Attribute

userName

- Click **Save**.

11. For the authentication trees that require email, set the **External Login Page URL**.

    - In the `alpha` realm, select **Authentication > Settings**, and click the **General** tab.
    - Set **External Login Page URL** to `http://login.example.com:8083`, then click **Save Changes**.

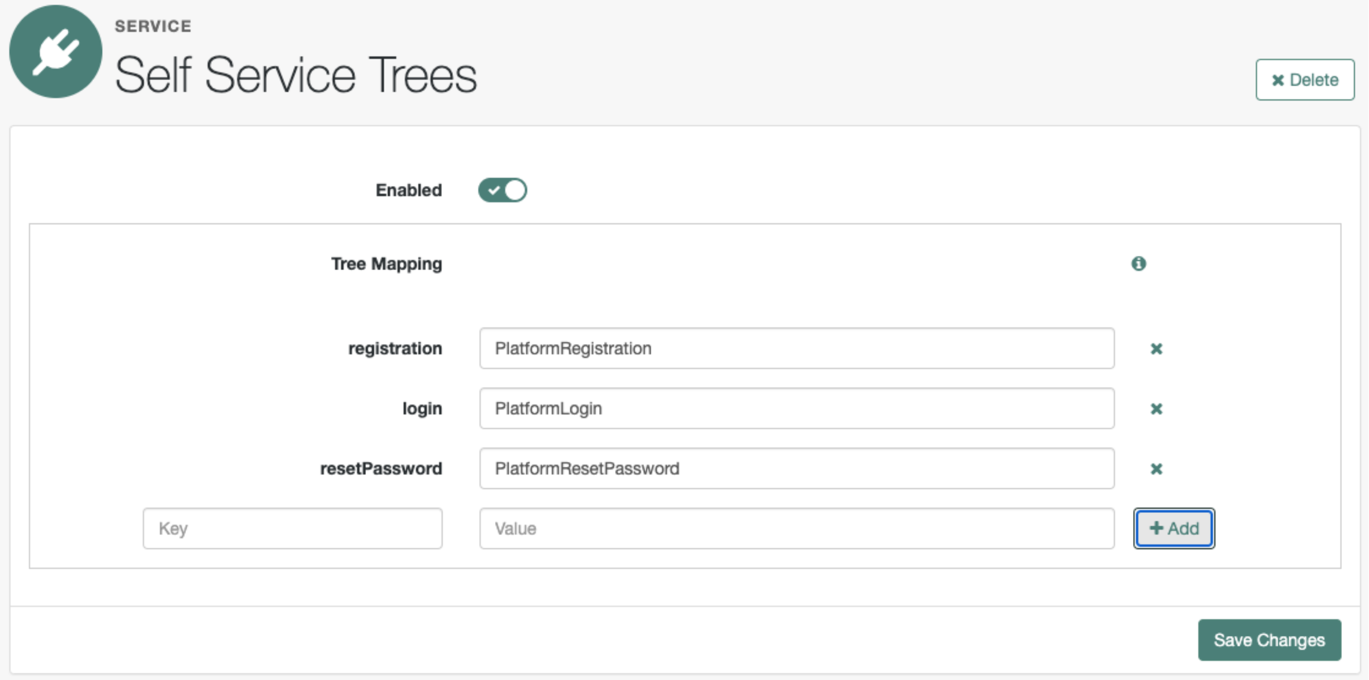
Map authentication trees

Map the Advanced Identity Software authentication trees to the corresponding Self-Service endpoints.

1. In the `alpha` realm, select **Services**, and click **Add a Service**.
2. Under **Choose a service type**, select **Self Service Trees**, and click **Create**.

3. Add the following tree mappings:

Key	Value
`registration`	`PlatformRegistration`
`login`	`PlatformLogin`
`resetPassword`	`PlatformResetPassword`



SERVICE

Self Service Trees

Enabled: on

Tree Mapping

registration	PlatformRegistration
login	PlatformLogin
resetPassword	PlatformResetPassword
Key	Value

4. Click **Save Changes**.

Additional PingAM configuration

The PingAM configuration shown is sufficient for this sample deployment. Read the PingAM documentation when using additional features.

For example, if PingAM runs behind a load balancer or a reverse proxy, configure a base URL source service. Adapt the PingAM configuration to use this service when you protect PingAM with PingGateway.

Restart Tomcat

Restart Tomcat to take all the configuration changes into account:

```
$ /path/to/tomcat/bin/shutdown.sh
# Wait a moment for Tomcat to shut down cleanly before starting it again.
$ /path/to/tomcat/bin/startup.sh
```

Next step

Choose your sample

Prepare the servers

Shared identity store

Set up PingDS

Set up PingAM

□ Set up PingIDM

□ Protect the deployment

□ Set up the Platform UIs

□ Test your deployment

Set up PingIDM

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

This procedure sets up PingIDM with an external PingDS instance as the repository. The PingDS instance is shared with PingAM as the identity store. The procedure reflects the listed server settings for installing PingIDM and PingDS.

1. Follow the instructions in the PingIDM documentation to download and install PingIDM.

2. Edit the `/path/to/openidm/resolver/boot.properties` file to set the hostname:

```
openidm.host=openidm.example.com
```

3. Configure the shared PingIDM repository.

   This step assumes that you have set up PingDS and exported the PingDS CA certificate from `directory.example.com` (as shown in Step 4 of Secure connections).

   - Import the PingDS CA certificate into the PingIDM truststore:

```
keytool \
-importcert \
-alias ds-ca-cert \
-file /path/to/ds-ca-cert.pem \
-keystore /path/to/openidm/security/truststore \
-storepass:file /path/to/openidm/security/storepass
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

- Replace the default `conf/repo.ds.json` file with this repo.ds.json file.

> **Important**
>
> Replace the values of `fileBasedTrustManagerFile` and `fileBasedTrustManagerPasswordFile` with the path to your PingIDM truststore and truststore password file.
> Check that the properties under `ldapConnectionFactories` match the PingDS server that you set up as the identity store.

4. It is worth starting PingIDM at this point, to make sure that it can connect to the external PingDS repository.

   1. Set the `JAVA_HOME` environment variable.

   2. Start PingIDM, and wait until you see `OpenIDM ready` in the output:

```
/path/to/openidm/startup.sh
-> OpenIDM version "8.0.0" <build-info>
OpenIDM ready
```

5. Update the `user` object in your managed object configuration by making the following changes to your `conf/managed.json` file.

   All changes are made to the object `user` > `schema` > `properties`:

```
{
    "objects": [
        {
            "name": "user",
            ...
            "schema": {
            "properties": {
                ...
            }
            ...
        }
    ]
}
```

- Remove encryption from the `password` property. Remove:

```
"encryption" : {
    "purpose": "idm.password.encryption"
}
```

Password *encryption* is not supported in a shared identity store setup. Instead, the shared identity store supports only *hashed* passwords, a format that both PingAM and PingIDM can use. This differs from Separate identity stores, where PingAM and PingIDM do not share the identity store, and so each service can store passwords in different formats.

- Update the `password` property to ensure that users update their passwords through the self-service APIs, not directly:

```
"userEditable" : false
```

- Add a `cn` property to the `user` object:

```
{
    "objects": [
        {
            "name": "user",
            ...
            "schema": {
                "properties" : {
                    "_id" : {
                        ...
                    },
                    "cn": {
                        "title": "Common Name",
                        "description": "Common Name",
                        "type": "string",
                        "viewable": false,
                        "searchable": false,
                        "userEditable": false,
                        "scope": "private",
                        "isPersonal": true,
                        "isVirtual": true,
                        "onStore": {
                            "type": "text/javascript",
                            "source": "object.cn || (object.givenName + ' ' + object.sn)"
                        }
                    },
                    ...
            }
            ...
        }
    ]
}
```

- Configure social authentication.

Add an `aliasList` property to the `user` object:

```
"aliasList": {
    "title": "User Alias Names List",
    "description": "List of identity aliases used primarily to record social IdP subjects for this
user",
    "type": "array",
    "items": {
        "type": "string",
        "title": "User Alias Names Items"
    },
    "viewable": false,
    "searchable": false,
    "userEditable": true,
    "returnByDefault": false,
    "isVirtual": false
},
```

6. Change the authentication mechanism to `rsFilter` only:

   - Replace the default `conf/authentication.json` file with this authentication.json file.
   - Check that the `clientSecret` matches the `Client secret` that you set for the `idm-resource-server` client in PingAM (see Configure OAuth Clients).
   - Check that the `rsFilter` > `subjectMapping` > `propertyMapping` > `sub` property is correctly configured.

     The `authentication.json` file aligns with the default PingAM configuration for subject claim uniqueness. PingAM refers to the subject by its unique identifier, and so PingIDM does, too.

     If PingAM has its advanced server property, `org.forgerock.security.oauth2.enforce.sub.claim.uniqueness`, set to `false`, for example, because you upgraded from a previous release of PingAM, use this property mapping instead:

     ```
     "propertyMapping": {
         "sub": "userName"
     }
     ```

     PingAM refers to the subject by its username in this case. For details, see the reference for the setting in the PingAM documentation.

   For more information about authenticating using the `rsFilter`, see Authenticate through PingAM in the PingIDM documentation.

7. Edit the IDM admin UI configuration so that you can still authenticate through the IDM admin UI:

   - In your `/path/to/openidm/conf/ui-configuration.json` file, insert a `platformSettings` object into the `configuration` object:

```
{
    "configuration" : {
        "platformSettings" : {
            "adminOauthClient" : "idm-admin-ui",
            "adminOauthClientScopes" : "fr:idm:*",
            "amUrl" : "http://am.example.com:8081/am",
            "loginUrl" : ""
        }
    }
}
```

This object tells the IDM admin UI that it is operating in "Advanced Identity Software mode" (that is, as an OAuth 2.0 client of PingAM).

- In your `conf/ui.context-admin.json` file, check that `X-Frame-Options` is set to `SAMEORIGIN`:

```
{
    "enabled" : true,
    "cacheEnabled": true,
    "urlContextRoot" : "/admin",
    "defaultDir" : "&{idm.install.dir}/ui/admin/default",
    "extensionDir" : "&{idm.install.dir}/ui/admin/extension",
    "responseHeaders" : {
        "X-Frame-Options" : "SAMEORIGIN"
    }
}
```

8. Configure the CORS servlet filter.

   Replace the default `conf/servletfilter-cors.json` file with this servletfilter-cors.json file.

9. Add the configuration to enable theming for hosted UI pages.

   1. Copy this ui-themerealm.json file to the `conf/` directory.

   2. In your `conf/access.json` file, insert a configuration object for the theme in the `configs` array:

```
{
  "configs": [{
    "pattern": "config/ui/themerealm",
    "roles": "*",
    "methods": "read",
    "actions": "*"
  }]
}
```

10. If you have not already started PingIDM, start it now, and test that you can access the IDM admin UI at http://openidm.example.com:8080/admin.

    When you log in to the IDM admin UI, use the default PingAM administrative user (`amAdmin`), and not `openidm-admin` .

11. If you want to use the `PlatformForgottenUsername` or `PlatformResetPassword` trees, configure outbound email.

> **Note**
>
> After you have installed the Platform UIs, you can configure email through the UI at `http://openidm.example.com:8080/admin`.

PingIDM is now configured for this deployment.

Next step

- Choose your sample
- Prepare the servers
- Shared identity store
  - Set up PingDS
  - Set up PingAM
  - Set up PingIDM
- □ Protect the deployment
- □ Set up the Platform UIs
- □ Test your deployment

Protect the deployment

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

Before you start

After you set up a sample deployment according to Separate identity stores or Shared identity store, you have a functionally complete Ping Advanced Identity Software installation.

Your deployment still lacks key features:

Security

Except when connecting to the directory service, connections use HTTP, not HTTPS. Users send their login credentials in unprotected cleartext.

Single entry point

The Platform UI component applications don't all run on the same host and port. This means cookies and iframes don't always share the same host and origin, a potential problem for modern browsers.

The instructions that follow show how to add PingGateway to your deployment, providing a single point of entry to Ping Advanced Identity Software, and securing communications from outside with HTTPS.

Tip

- The following examples deploy PingGateway on `platform.example.com` using port 9443 for HTTPS. Adapt the examples as necessary for your deployment.
  For example, if you're demonstrating the deployment on your computer, add an alias for the fully qualified domain name to your `/etc/hosts` file:

  ```
  127.0.0.1    platform.example.com
  ```

- Before using PingGateway to protect the deployment, make sure that PingAM uses `example.com` as the cookie domain.
  This setting ensures that your browser can share cookies across subdomains in `example.com`. If PingAM is using the default cookie domain, such as `am.example.com`, then users won't be able to sign in through PingGateway.
  You can check by logging in to the AM admin UI as `amAdmin`, browsing to **Global Services** > **Platform**, and verifying that **Cookie Domains** is set to `example.com`.

The instructions that follow are sufficient to add PingGateway to your sample deployment. Find in-depth documentation on using PingGateway in the PingGateway product documentation.

Prepare keys

HTTPS requires a server key pair for PingGateway.

In a public deployment where PingGateway is the entry point, get the PingGateway server certificate signed by a well-known CA. A server certificate signed by a well-known CA will be trusted by other systems and browsers, because the CA certificate is distributed with the system or the browser.

> **Note**
>
> The examples that follow do not use a well-known CA. Instead, they use an PingGateway server certificate generated with one of the PingDS deployment ID and password combinations that you used to set up the deployment.
>
> 1. You'll generate an PingGateway server key pair using the PingDS `dskeymgr` command.
> 2. You'll export and trust the deployment ID CA certificate to demonstrate and test the deployment.
>
> As long as no one else knows your deployment ID and password combination, this is safe, because only you have the secrets to generate this CA certificate, or to generate signed keys:

Generate the keys on a system where you installed PingDS:

1. Save a password file for the keystore:

   ```
   mkdir -p /path/to/security/
   touch /path/to/security/keystore.pin
   chmod 600 /path/to/security/keystore.pin
   echo -n password > /path/to/security/keystore.pin
   ```

   Be sure to use `echo -n`, as PingGateway can't use the secret if the file has a newline character at the end.

2. Set the deployment ID in an environment variable that you'll use in `dskeymgr` commands:

   ```
   export DEPLOYMENT_ID=deployment-id
   ```

3. Generate a server key pair for PingGateway to use for HTTPS:

   ```
   /path/to/opendj/bin/dskeymgr \
    create-tls-key-pair \
    --deploymentId $DEPLOYMENT_ID \
    --deploymentIdPassword password \
    --keyStoreFile /path/to/security/keystore \
    --keyStorePassword:file /path/to/security/keystore.pin \
    --hostname localhost \
    --hostname platform.example.com \
    --subjectDn CN=platform.example.com,O=ForgeRock
   ```

4. Inspect the contents of the keystore you just created to find the key alias is `ssl-key-pair`:

   ```
   keytool -list -keystore /path/to/security/keystore -storepass:file /path/to/security/keystore.pin
   Keystore type: PKCS12
   Keystore provider: SUN

   Your keystore contains 1 entry

   ssl-key-pair, <date>, PrivateKeyEntry,
   Certificate fingerprint (SHA-256): <fingerprint>
   ```

5. Copy the keystore and password to the system where PingGateway runs.

6. Export the deployment ID CA certificate to trust:

   ```
   /path/to/opendj/bin/dskeymgr \
   export-ca-cert \
   --deploymentId $DEPLOYMENT_ID \
   --deploymentIdPassword password \
   --outputFile ca-cert.pem
   ```

   Use this `ca-cert.pem` file when you need to trust the PingGateway server certificate. You can import it into your browser to trust the Advanced Identity Software URLs in this example.

Install PingGateway

This example uses PingGateway in standalone mode. In standalone mode, PingGateway runs in the web container provided in the .zip file.

1. Download PingGateway-2025.3.0.zip.

2. Unpack the files.

   This example shows the files unpacked in the `/path/to/identity-gateway-2025.3` directory.

3. Create a configuration directory for PingGateway:

   ```
   mkdir -p $HOME/.openig/config/
   ```

   The default PingGateway configuration directory is `$HOME/.openig/config` on Linux and UNIX, and `%appdata%\OpenIG\config` on Windows.

4. Add an admin.json configuration file for HTTPS support at the root of the PingGateway configuration directory.

   This example expects the keystore and `keystore.pin` file in the `/path/to/security/` directory:

```
{
    "heap": [
        {
            "name": "TlsConf",
            "type": "ServerTlsOptions",
            "config": {
                "keyManager": {
                    "type": "SecretsKeyManager",
                    "config": {
                        "signingSecretId": "key.manager.secret.id",
                        "secretsProvider": "ServerIdentityStore"
                    }
                }
            }
        },
        {
            "name": "SecretsPasswords",
            "type": "FileSystemSecretStore",
            "config": {
                "directory": "/path/to/security/",
                "format": "PLAIN"
            }
        },
        {
            "name": "ServerIdentityStore",
            "type": "KeyStoreSecretStore",
            "config": {
                "file": "/path/to/security/keystore",
                "storePasswordSecretId": "keystore.pin",
                "secretsProvider": "SecretsPasswords",
                "mappings": [
                    {
                        "secretId": "key.manager.secret.id",
                        "aliases": [
                            "ssl-key-pair"
                        ]
                    }
                ]
            }
        }
    ],
    "connectors": [
        {
            "port": 9080
        },
        {
            "port": 9443,
            "tls": "TlsConf"
        }
    ]
}
```

5. Start PingGateway:

```
/path/to/identity-gateway-2025.3bin/start.sh
...
[main] INFO ... started in XXXXms on ports : [9080, 9443]
```

6. Test that you can use the PingGateway "ping" endpoint over HTTP:

   ```
   curl -v http://platform.example.com:9080/openig/ping
   ```

   The output should include an HTTP `200` status code indicating success, and an empty reply from PingGateway.

7. When you can successfully ping PingGateway over HTTP, try HTTPS using your CA certificate file:

   ```
   curl -v --cacert ca-cert.pem https://platform.example.com:9443/openig/ping
   ```

   The `ca-cert.pem` file is the one you exported with the `dskeymgr` command. If it's not in the current directory, include the path to the file.

   As before, you should receive an empty success response. This demonstrates that you successfully connected to PingGateway over HTTPS.

   Now that you're sure the HTTPS connection works, you can edit `admin.json` to stop using port 9080 for HTTPS.

Trust the deployment ID certificate

> **Important**
>
> You generated a deployment ID CA certificate used in this example with the `dskeymgr` command. Your browser doesn't recognize this private CA until you trust it by importing the CA certificate.
> Make sure this certificate is trusted when negotiating SSL and HTTPS connections.
> If you don't do this, you'll see an error similar to the following when you attempt to access the Platform UI:
>
> ```
> Your connection is not private
> Attackers might be trying to steal your information from platform.example.com...
> net::ERR_CERT_AUTHORITY_INVALID
> ```

As long as no one else knows your deployment ID and password combination, this is safe, because only you have the secrets to generate this CA certificate, or to generate signed keys:

- Trust the deployment ID CA certificate by importing the `ca-cert.pem` file *as a certificate authority* trusted for SSL and HTTPS connections.

  Some browsers refer to "Authorities" in the browser settings. You can find more information in your browser help.

Configure PingGateway

PingGateway terminates HTTPS for your deployment. All access to your deployment goes through PingGateway.

Configure PingGateway to route traffic through to the components of the deployment and back to clients. The settings in the following steps depend on the settings you'll use to set up the Platform UI:

1. Add a config.json configuration file at the root of the PingGateway configuration directory.

   This example configures the capture decorator and a router with a default route:

```
{
    "heap": [
        {
            "name": "capture",
            "type": "CaptureDecorator"
        },
        {
            "name": "DefaultRoute",
            "type": "StaticResponseHandler",
            "config": {
                "status": 200,
                "headers": {
                    "Content-Type": [
                        "text/html"
                    ]
                },
                "entity": "<html><p><a href='/enduser-login/?realm=/alpha#/service/PlatformLogin'>End user access</a></p><p><a href='/platform-login/?realm=/#/'>Admin access</a></p></html>"
            }
        },
        {
            "type": "Router",
            "name": "_router",
            "config": {
                "defaultHandler": "DefaultRoute"
            }
        }
    ],
    "handler": "_router",
    "audit": "global"
}
```

   The default route in the sample file defines a simple static HTML page to present when a user accesses Advanced Identity Software before logging in, or after logging out.

   In this example, the page includes a link for each of the two user flows:

   1. An end user logs in to access their applications and profile page through the End User UI.

   2. An admin logs in to configure Advanced Identity Software through the Platform admin UI, AM admin UI, or IDM admin UI.

   You can configure PingGateway to redirect users to your site if desired. PingGateway can also serve static pages, so you can add an entry point that matches your site.

2. Create a `routes` directory under the PingGateway configuration file directory:

   ```
   mkdir $HOME/.openig/config/routes
   ```

3. Add a login.json route file in the `routes` directory.

   This route consumes the paths specified in the links on the default, static page to route users to the correct URL on the Login UI:

   ```
   {
       "name": "login",
       "condition": "${find(request.uri.path, '^/enduser-login|^/platform-login')}",
       "baseURI": "http://login.example.com:8083",
       "handler": {
           "type": "Chain",
           "config": {
               "filters": [
                   {
                       "type": "UriPathRewriteFilter",
                       "config": {
                           "mappings": {
                               "/enduser-login": "/",
                               "/platform-login": "/"
                           },
                           "failureHandler": "DefaultRoute"
                       }
                   }
               ],
               "handler": "ReverseProxyHandler"
           }
       }
   }
   ```

4. Add an `enduser-ui.json` route file in the `routes` directory.

   This route sends end users to the End User UI:

```
{
    "name": "enduser-ui",
    "condition": "${find(request.uri.path, '^/enduser-ui')}",
    "baseURI": "http://enduser.example.com:8888",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/enduser-ui": "/"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

```
{
    "name": "enduser-ui",
    "condition": "${find(request.uri.path, '^/enduser-ui')}",
    "baseURI": "http://am.example.com:8081",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/enduser-ui": "/enduser"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

5. Add a `platform-ui.json` route file in the `routes` directory.

   This route sends administrators to the Admin UI:

```
{
    "name": "platform-ui",
    "condition": "${find(request.uri.path, '^/platform-ui')}",
    "baseURI": "http://admin.example.com:8082",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/platform-ui": "/"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

```
{
    "name": "platform-ui",
    "condition": "${find(request.uri.path, '^/platform-ui')}",
    "baseURI": "http://am.example.com:8081",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "UriPathRewriteFilter",
                    "config": {
                        "mappings": {
                            "/platform-ui": "/platform"
                        },
                        "failureHandler": "DefaultRoute"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

6. Add an am.json route file in the `routes` directory.

   This route handles requests to PingAM, ensuring that the browser gets redirected through PingGateway:

```
{
    "name": "am",
    "baseURI": "http://am.example.com:8081",
    "condition": "${find(request.uri.path, '(?:^/am(?!/XUI))')}",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "comment": "Always redirect to IG rather than AM",
                    "type": "LocationHeaderFilter",
                    "config": {
                        "baseURI": "https://platform.example.com:9443/"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

7. Add an idm.json route file in the `routes` directory.

   This route handles requests to PingIDM, ensuring that the browser gets redirected through PingGateway:

```
{
    "name": "idm",
    "baseURI": "http://openidm.example.com:8080",
    "condition": "${find(request.uri.path, '(?:^/openidm)|(?:^/admin)|(?:^/upload)|(?:^/export)')}",
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "comment": "Always redirect to IG rather than IDM",
                    "type": "LocationHeaderFilter",
                    "config": {
                        "baseURI": "https://platform.example.com:9443/"
                    }
                }
            ],
            "handler": "ReverseProxyHandler"
        }
    }
}
```

8. Restart PingGateway to take all your changes into account.

   Verify in the server output that all your routes load successfully.

When troubleshooting and developing routes, you can change an PingGateway route configuration while PingGateway is running, and PingGateway will reload changed routes every 10 seconds by default.

The PingGateway capture decorator is particularly useful when troubleshooting routes. To use it quickly, add `"capture": "all"` to an object in a route, and let PingGateway reload the route before you try it. It logs messages in the server output about incoming requests at any point until PingGateway sends the request, and outgoing responses at any point until PingGateway delivers the response.

For more information, see the PingGateway product documentation.

Adapt the PingAM configuration

A sample deployment configured according to Separate identity stores or Shared identity store does not route the traffic through PingGateway. Adapt the PingAM and OAuth 2.0 client configurations to use PingGateway.

The following minimal changes are sufficient to use the sample deployment. Add any additional changes necessary for your deployment:

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.

   The password used in the documentation to set up Advanced Identity Software is `Passw0rd`.

2. In **the Top Level Realm and the `alpha` realm**, add redirect URIs for the `end-user-ui` OAuth 2.0 client:

   ```
   https://platform.example.com:9443/enduser-ui/appAuthHelperRedirect.html
   https://platform.example.com:9443/enduser-ui/sessionCheck.html
   ```

3. In **the Top Level Realm and the `alpha` realm**, add redirect URIs for the `idm-admin-ui` OAuth 2.0 client:

   ```
   https://platform.example.com:9443/platform/appAuthHelperRedirect.html
   https://platform.example.com:9443/platform/sessionCheck.html
   https://platform.example.com:9443/admin/appAuthHelperRedirect.html
   https://platform.example.com:9443/admin/sessionCheck.html
   https://platform.example.com:9443/platform-ui/appAuthHelperRedirect.html
   https://platform.example.com:9443/platform-ui/sessionCheck.html
   ```

4. In the `alpha` realm, edit **Success URLs** for the following PingAM trees:

   PlatformLogin

   Success URL: `https://platform.example.com:9443/enduser-ui/?realm=/alpha`

   PlatformRegistration

   Success URL: `https://platform.example.com:9443/enduser-ui/?realm=/alpha`

   PlatformResetPassword

   Success URL: `https://platform.example.com:9443/enduser-ui/?realm=/alpha`

5. Update these settings:

   Top Level Realm > Authentication > Settings > General

   External Login Page URL: `https://platform.example.com:9443/platform-login`

Top Level Realm > *Services*

- **+ Add a Service** > **Validation Service**.
- In **Valid goto URL Resources**, set:

```
https://platform.example.com:9443/*
https://platform.example.com:9443/*?*
```

Top Level Realm > *Services*

- **+ Add a Service** > **Base URL Source**.
- In **Base URL Source**, choose `Fixed value`.
- In **Fixed value base URL**, set `https://platform.example.com:9443`.

`alpha` *Realm* > *Authentication* > *Settings* > *General*

External Login Page URL: `https://platform.example.com:9443/enduser-login`

`alpha` *Realm* > *Services* > *Validation Service*

To **Valid goto URL Resources**, add:

```
https://platform.example.com:9443/*
https://platform.example.com:9443/*?*
```

`alpha` *Realm* > *Services*

- **+ Add a Service** > **Base URL Source**.
- In **Base URL Source**, choose `Fixed value`.
- In **Fixed value base URL**, set `https://platform.example.com:9443`.

Configure > *Global Services* > *CORS Service* > *Secondary Configurations* > *Cors Configuration*

To **Accepted Origins**, add: `https://platform.example.com:9443`.

Adapt the PingIDM configuration

In the `/path/to/openidm/conf/ui-configuration.json` file, change `configuration > platformSettings > amUrl` to access the PingIDM UI through PingGateway:

```
{
    "configuration" : {
        "platformSettings" : {
            "adminOauthClient" : "idm-admin-ui",
            "adminOauthClientScopes" : "fr:idm:*",
            "amUrl" : "https://platform.example.com:9443/am",
            "loginUrl" : ""
        }
    }
}
```

Once you make this change, you can no longer access the IDM admin UI directly over HTTP. Use the link in the Platform admin UI to access the IDM admin UI.

Next step

- Choose your sample
- Prepare the servers
- Separate identity stores
  - Set up PingDS
  - Set up PingAM
  - Set up PingIDM
- Shared identity store
  - Set up PingDS
  - Set up PingAM
  - Set up PingIDM
- Protect the deployment
- □ Set up the Platform UIs
- □ Test your deployment

Set up the Platform UIs

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

This page describes two ways to set up the UIs for the sample deployments. Either deploy the Platform UI by running Docker images or by installing a `.zip` file. Do not attempt to deploy the UI both ways:

- Run Docker images
- Install a `.zip` file in the PingAM web container

The Platform UIs

Ping Advanced Identity Software includes three UIs:

Admin UI

Lets Advanced Identity Software administrators manage applications, identities, end-user journeys, and so on.

End User UI

An example UI that demonstrates profile maintenance, consent management, workflow, and delegated administration capabilities. This UI makes REST calls to PingIDM (using an OAuth 2 bearer token) and to PingAM (using an SSO cookie).

Login UI

A unified Login UI that lets both administrators and end users log in to Advanced Identity Software.

Run Docker images

Important

If you have already deployed the Platform UI by installing a `.zip` file, do not perform these steps.

In this sample deployment, the three UIs are deployed from three Docker images.

1. Download the Docker images:

   - Download the Admin UI:

```
docker pull gcr.io/forgerock-io/platform-admin-ui:8.0.1
```

   - Download the End User UI:

     ```
     docker pull gcr.io/forgerock-io/platform-enduser-ui:8.0.1
     ```

   - Download the Login UI:

     ```
     docker pull gcr.io/forgerock-io/platform-login-ui:8.0.1
     ```

2. Create an environment file named `platform_env` that will be passed to the `docker` command.

   The file should have the following contents:

   ```
   AM_URL=https://platform.example.com:9443/am
   AM_ADMIN_URL=https://platform.example.com:9443/am/ui-admin
   IDM_REST_URL=https://platform.example.com:9443/openidm
   IDM_ADMIN_URL=https://platform.example.com:9443/admin
   IDM_UPLOAD_URL=https://platform.example.com:9443/upload
   IDM_EXPORT_URL=https://platform.example.com:9443/export
   ENDUSER_UI_URL=https://platform.example.com:9443/enduser-ui/
   PLATFORM_ADMIN_URL=https://platform.example.com:9443/platform-ui/
   ENDUSER_CLIENT_ID=end-user-ui
   ADMIN_CLIENT_ID=idm-admin-ui
   THEME=default
   PLATFORM_UI_LOCALE=en
   ```

   > **Tip**
   >
   > When you deploy multiple Platform UI Docker containers with the same hostname, use the `SUBFOLDER` environment variable to prefix a base URL path for each UI.
   > For example, if you deploy all the UIs on `ui.example.com`, use the following settings to make the End User UI accessible under `/enduser`:
   > In the End User UI configuration, set:
   > `SUBFOLDER=enduser`
   > In all Platform UI environments, set:
   > `ENDUSER_UI_URL=http://ui.example.com:8888/enduser`
   > Adapt your configuration to account for the path changes.

3. Start each Docker container, specifying the environment file:

   - Change to the directory in which you saved the `platform_env` file, then start the Login UI:

```
docker run -t -i --rm -p 8083:8080 --env-file=platform_env \
gcr.io/forgerock-io/platform-login-ui:8.0.1
Replacing env vars in JS

Setting AM URL as https://platform.example.com:9443/am
Setting AM ADMIN URL as https://platform.example.com:9443/am/ui-admin
Setting IDM REST URL as https://platform.example.com:9443/openidm
Setting IDM ADMIN URL as https://platform.example.com:9443/admin
...
```

> **Note**
>
> The `--rm` option causes the `docker run` command to remove the container on exit. Remove this option if you want to keep a container when it exits.

- In a new terminal, change to the directory in which you saved the `platform_env` file, then start the Admin UI:

```
docker run -t -i --rm -p 8082:8080 --env-file=platform_env \
gcr.io/forgerock-io/platform-admin-ui:8.0.1
Replacing env vars in JS

Setting AM URL as https://platform.example.com:9443/am
Setting AM ADMIN URL as https://platform.example.com:9443/am/ui-admin
Setting IDM REST URL as https://platform.example.com:9443/openidm
Setting IDM ADMIN URL as https://platform.example.com:9443/admin
...
```

- In a new terminal, change to the directory in which you saved the `platform_env` file, then start the End User UI:

```
docker run -t -i --rm -p 8888:8080 --env-file=platform_env \
gcr.io/forgerock-io/platform-enduser-ui:8.0.1
Replacing env vars in JS

Setting AM URL as https://platform.example.com:9443/am
Setting AM ADMIN URL as https://platform.example.com:9443/am/ui-admin
Setting IDM REST URL as https://platform.example.com:9443/openidm
Setting IDM ADMIN URL as https://platform.example.com:9443/admin
...
```

Install a `.zip` file

> **Important**
>
> If you have already deployed the Platform UI by running Docker images, do not perform these steps.

In this sample deployment, you obtain the three UIs from a `.zip` file that you get from Ping Identity. Then, you install them in the PingAM web container. You also tweak the PingAM and PingIDM configurations so that they work with this UI installation.

1. Get the UI `.zip` file, and unzip it into the tmp directory:

    1. Create the /tmp/ui-zip directory.

    2. Download the most recent version of the 8.0.x UI `.zip` file from the Platform section of the downloads site.

    3. Copy the UI `.zip` file into the /tmp/ui-zip directory.

    4. Unzip the UI `.zip` file.

2. Replace URLs in the UI application with URLs for your Ping Advanced Identity Software deployment:

    1. Change to the /tmp/ui-zip/PlatformUI directory.

    2. Run the following commands:

        ```
        export AM_URL=https://platform.example.com:9443/am
        export AM_ADMIN_URL=https://platform.example.com:9443/am/ui-admin
        export IDM_REST_URL=https://platform.example.com:9443/openidm
        export IDM_ADMIN_URL=https://platform.example.com:9443/admin
        export IDM_UPLOAD_URL=https://platform.example.com:9443/upload
        export IDM_EXPORT_URL=https://platform.example.com:9443/export
        export ENDUSER_UI_URL=https://platform.example.com:9443/enduser-ui/
        export PLATFORM_ADMIN_URL=https://platform.example.com:9443/platform-ui/
        export ENDUSER_CLIENT_ID=end-user-ui
        export ADMIN_CLIENT_ID=idm-admin-ui
        export THEME=default
        export PLATFORM_UI_LOCALE=en
        ```

    3. Run the variable_replacement.sh script.

        This script replaces variables in the `.zip` file's UI web application with the values of the environment variables that you set in the previous step:

        ```
        ./variable_replacement.sh www/platform/js/*.js www/enduser/js/*.js www/login/js/*.js
        Replacing env vars in JS

        Setting AM URL as https://platform.example.com:9443/am
        Setting AM ADMIN URL as https://platform.example.com:9443/am/ui-admin
        Setting IDM REST URL as https://platform.example.com:9443/openidm
        Setting IDM ADMIN URL as https://platform.example.com:9443/admin
        ...
        ```

3. Install the UI:

    1. Install the `platform` and `enduser` web applications into PingAM's web container:

```
    cd /path/to/tomcat/webapps
    cp -r /tmp/ui-zip/PlatformUI/www/platform .
    cp -r /tmp/ui-zip/PlatformUI/www/enduser .
    ```

  2. Modify PingAM to use the new Login UI:

     ```
     cd /path/to/tomcat/webapps/am/XUI
     cp -r /tmp/ui-zip/PlatformUI/www/login/* .
     ```

4. Revise the PingAM and PingIDM configurations to work with the new UI installation:

   1. If you’re not currently logged in to the AM admin UI as the `amAdmin` user, log in.

   2. Revise the success URL for three authentication trees, so that they refer to the End User UI:

      1. Change the success URL for the `Platform Regstration` tree.

         In the `alpha` realm, select **Authentication > Trees**, and click on **PlatformRegistration**. Select the **Success URL** node. Change the **Success URL** value to **http://am.example.com:8081/enduser**, and then click **Save**.

      2. Use the same technique to change the **Success URL** value for the `PlatformLogin` tree to **http://am.example.com:8081/enduser**.

      3. Use the same technique to change the **Success URL** value for the `PlatformResetPassword` tree to **http://am.example.com:8081/enduser**.

   3. Clear the **External Login Page URL** value.

      In the `alpha` realm, select **Authentication > Settings**, and click the **General** tab. Set **External Login Page URL** to an empty string, and then click **Save Changes**.

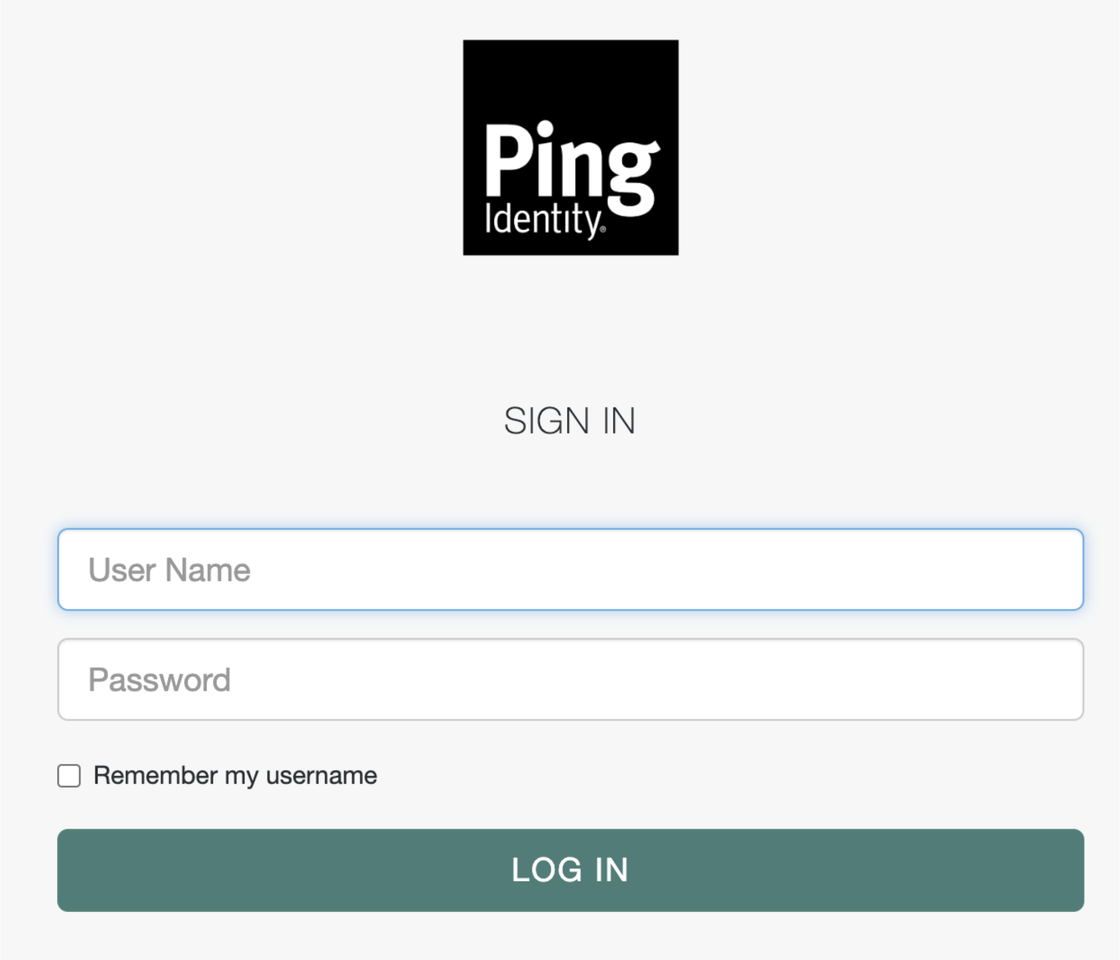
   4. Log out of the AM admin UI.

   5. If no `/path/to/openidm/conf/ui-themerealm.json` file exists, create it with the following contents:

      ```
      {
          "realm": {}
      }
      ```

Redirect on signout

When the test user signs out of the End User UI, the browser redirects to the PingAM XUI login page:



You can change this behavior to redirect to your application. Options for managing redirection on sign out include:

- (*Recommended*) Use PingGateway, as described in Protect the deployment, with a route to redirect the browser from the XUI to your application.

  If you choose this option, skip the steps that follow.

- Script a `post_logout_url` claim that PingAM returns to the End User UI through the user's ID token. When the user signs out, the End User UI redirects the browser to the URL specified in the claim.

Follow these steps to configure a `post_logout_url` claim:

1. If you're not currently logged in to the AM admin UI as the `amAdmin` user, log in.
2. Under **Scripts**, find the **OIDC Claims Script**, and click to view the script.

   This script lets you customize the OpenID Connect 1.0 claims in the user's ID token.
3. Add a `post_logout_url` claim method to the array of `claimAttributes`.

   The following Groovy example adds a final item to `claimAttributes` to return `https://pingidentity.com` as the `post_logout_url` claim. Adapt the method used to return the appropriate URL for your application:

```
claimAttributes = [
        //...,
        "post_logout_url": { claim, identity -> return [(claim.getName()): "https://pingidentity.com"] }
]
```

4. Add the claim for the `fr:idm:*` scope as a final item in the `scopeClaimsMap` :

```
scopeClaimsMap = [
        //...,
        "fr:idm:*": [ "post_logout_url" ]
]
```

5. Click **Validate**, correct any errors accidentally introduced, and then click **Save Changes**.

6. Under **Services** > **OAuth2 Provider**, search for `Always Return Claims in ID Tokens`, then enable the option, and click **Save Changes**.

   PingAM can now return the custom claim in the user's ID token.

7. Test the changes you have made:

   1. Log out of the AM admin UI.

   2. Sign in as a test user at http://login.example.com:8083/?realm=/alpha#/service/PlatformLogin.

   3. Sign out.

      The End User UI redirects the browser to the URL you configured for the `post_logout_url` claim, in this case `https://pingidentity.com` .

Next step

Choose your sample

Prepare the servers

Separate identity stores

Set up PingDS

Set up PingAM

Set up PingIDM

Shared identity store

Set up PingDS

Set up PingAM

Set up PingIDM

Protect the deployment

Set up the Platform UIs

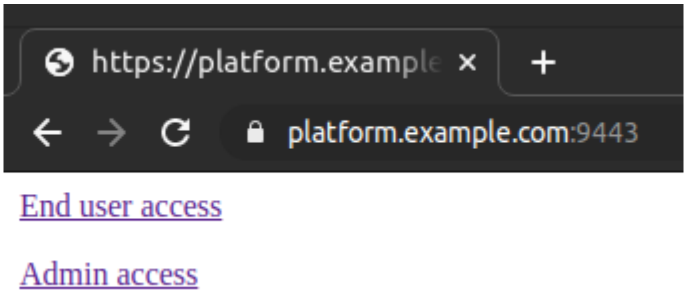
□ Test your deployment

Test your deployment

After configuring your sample deployment, test your work:

1. Log out of any open Advanced Identity Software applications.
2. Browse https://platform.example.com:9443/.

   Your browser displays the default route page:

   If a page warns the connection isn't private, the page poses a security risk, or your browser does not trust the PingGateway server certificate, read Trust the deployment ID certificate again.
3. Test the flow for an administrator:
   1. Click the **Admin access** link.
   2. Sign in as `amAdmin`.

      The password in the documentation to set up Advanced Identity Software is `Passw0rd`.

      Your browser redirects to the Platform admin UI page:

      ```
      https://platform.example.com:9443/platform-ui/?realm=root#/dashboard/overview
      ```
   3. Browse around the Platform admin UI, making sure to open the AM admin UI and IDM admin UI.

      The browser address shows a secure connection through `https://platform.example.com:9443/` for all pages.
   4. (*Optional*) Switch to the `alpha` realm and create a new end-user account under **Identities** > **Manage**.
   5. Sign out.

      Your browser redirects to the Advanced Identity Software default page.
4. Test the flow for an end user:
   1. Click the **End user access** link.
   2. Sign in as an end user.

      If you have not yet created an end-user account, follow the **Create an account** link, and register an end user with the platform.

      In the end, your browser redirects to the end-user page:

      ```
      https://platform.example.com:9443/enduser-ui/?realm=alpha#/dashboard
      ```
   3. Sign out.

Your browser redirects to the Advanced Identity Software default page.

Your sample deployment now routes traffic through PingGateway to the Advanced Identity Software. If you get the PingGateway server certificate signed by a well-known CA, instead of the private CA, other browsers and systems can connect without additional configuration.

Advanced Identity Software setup checklist

- Choose your sample
- Prepare the servers
- Separate identity stores
  - Set up PingDS
  - Set up PingAM
  - Set up PingIDM
- Shared identity store
  - Set up PingDS
  - Set up PingAM
  - Set up PingIDM
- Protect the deployment
- Set up the Platform UIs
- Test your deployment

Upgrade

Important

Advanced Identity Software upgrade complexity depends on the deployment. Upgrades for heavily customized deployments using many advanced features require far more care and planning than an upgrade for a sample evaluation deployment.
Make sure you plan and test appropriately before attempting to upgrade a production deployment.

Upgrade process

To upgrade your *sample deployment* from 7.5 to 8.0, follow these high-level steps:

1. Upgrade the Platform UIs.

   Use the new UIs described in Set up the Platform UIs.

2. Upgrade PingGateway.

   Find more information in Upgrading PingGateway.

3. Upgrade PingDS.

   Find more information in Upgrade in the PingDS documentation.

4. Upgrade PingAM.

   Find more information in Upgrade in the PingAM documentation.

5. Upgrade PingIDM.

   Find more information in Upgrade in the PingIDM documentation.

When upgrading from earlier versions of the sample deployment, read the earlier upgrade instructions and Migration and customization.

Changes from 7.5 to 8

- Use Java 21.
- Use Tomcat 10 for PingAM.
- PingIDM 8 uses Logback to generate its server logs. The default location for the server logging configuration file has changed from `conf/logging.properties` to `conf/logback.xml`. Find more information in Server logs in the PingIDM documentation.

Changes from 7.4 to 7.5

- Use Java 17.
- The sample deployments now require PingGateway.

  The Platform UI component applications don't all run on the same host and port. This means cookies and iframes don't always share the same host and origin, a potential problem for modern browsers.

PingGateway protects access to the Advanced Identity Software and ensures a single host and port for all browser-based interactions.

- The procedures to configure PingAM now demonstrate how to use PingDS as an external token store.
- The PingGateway config.json configuration file has changed. PingGateway 2025.3 requires you to declare objects directly in the heap before referencing them.

Changes from 7.3 to 7.4

- The Amster `.zip` file `Amster-8.0.1.zip` now includes a root folder named `amster`.
- The Platform UI Docker image version format has changed. For details, refer to Run Docker images.
- The procedures for configuring PingIDM now include:
  - A new ui-themerealm.json file to enable theming for hosted UI pages.

    This addition makes it possible to edit authentication journeys through the Advanced Identity Software Admin UI.
  - An update to the repo.ds.json file to support password policies with **Force Password Change** enabled.

    This update adds the following settings to `dsconfig/passwordPolicies` > `properties`:

    ```
    {
      "allowExpiredPasswordChanges": {
        "type": "simple",
        "ldapAttribute": "ds-cfg-allow-expired-password-changes"
      },
      "expirePasswordsWithoutWarning": {
        "type": "simple",
        "ldapAttribute": "ds-cfg-expire-passwords-without-warning"
      },
      "passwordExpirationInterval": {
        "type": "simple",
        "ldapAttribute": "ds-cfg-password-expiration-warning-interval"
      }
    }
    ```

    You can find the **Force Password Change** option in the Advanced Identity Software Admin UI under **Security > Password Policy**.

Migration and customization

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

Previous versions of Ping Advanced Identity Software provided a sample deployment called the *full stack sample*. Ping Identity no longer supports this kind of deployment for integrations between PingIDM and PingAM. Follow the steps outlined here to set up a new deployment. There's no automated migration facility for an existing deployment based on the old *full stack sample*, but you can follow the tips in this chapter to help you move to a new integrated deployment.

Note

This isn't an exhaustive list, and you'll have additional configuration changes to make, based on how you deployed the full stack sample.

Self-service configuration

All self-service configuration is now done in the PingAM UI, with the exception of the following elements:

- KBA
- Terms & Conditions
- Policies
- Privacy & Consent
- Email templates and services

You can therefore delete all the PingIDM `selfservice*` files from your configuration, apart from `selfservice.kba.json` and `selfservice.terms.json`. These two files are still used by the new PingAM authentication trees.

You will need to rebuild all existing self-service processes as PingAM authentication trees. Note that tree nodes are more modular than the legacy PingIDM self-service stages. It might take more than one node to replace a self-service stage.

In an Advanced Identity Software configuration, you can optionally customize the IDM admin UI to hide or remove all the self-service functions that are no longer configured through PingIDM (such as registration and password reset).

Authentication

Integrated deployments now use a single `rsFilter` authentication module that allows authentication using PingAM bearer tokens. You must replace your project's `conf/authentication.json` file with this authentication.json file and customize it, according to your setup.

Configure OAuth clients and the PingIDM provisioning service in the PingAM UI.

UI customization

The UI retrieves access tokens from PingAM.

Migration for shared identities

As shown in Shared identity store,PingAM and PingIDM can share a PingDS identity store. Since version 7, this is a well-supported and recommended deployment pattern.

Previously, as shown in Separate identity stores and Reconciliation and PingAM in the PingIDM 6.5 documentation, PingAM used a PingDS-based identity store, PingIDM used a relational database for its repository, and PingIDM synchronized identities between them.

When you migrate shared identities to a shared PingDS identity store, follow these high-level steps:

- Add any custom identity attributes to the shared identity store and Advanced Identity Software components.

  Many deployments define custom attributes that serve in profiles or security tokens, which are critical to authentication and authorization, such as OAuth 2.0 access tokens, OpenID Connect ID tokens, and SAML assertions.

  For detailed instructions, see Custom attributes.

- Use PingIDM to migrate identity data to the shared PingDS identity store.

  Make sure that any operations to modify managed objects, such as managed users, performed as part of the mapping (in `sync.json`), are reflected in the managed object schema (in `managed.json`).

  For more on the migration operation, see Migrate data in the PingIDM documentation.

UI customization

In version 8 of Ping Advanced Identity Software, the PingAM and PingIDM component products continue to provide their own, separate UIs, so you can deploy one independently of the other.

When you use PingAM and PingIDM together in an Advanced Identity Software configuration:

- PingAM centralizes authentication and authorization services for Advanced Identity Software deployments.

  PingIDM acts as an OAuth 2.0 client of PingAM. Even when you log in to administer PingIDM, you authenticate through PingAM.

  More generally, all new client applications are expected to obtain an access token from PingAM, and present it to PingIDM for authorization; for example, when calling PingIDM REST endpoints.

  This is why the authentication configuration for PingIDM includes an `rsFilter` (OAuth 2.0 ***r***esource ***s***erver) configuration.

- PingIDM centralizes identity management services.

  PingIDM relies on the Platform admin UI, which calls PingAM trees that integrate with PingIDM for user self-service operations.

New Platform admin UI components replace functions of the PingAM and PingIDM native UIs:

Platform admin UI UI	Characteristics
Admin UI	• Provides quick access to the configuration capabilities that administrators need every day. • Links to the AM admin UI and IDM admin UI for additional, less common configuration operations. • *Cannot be customized*, though you can write your own. • Runs as an administrator-facing service that you run in front of an Advanced Identity Software deployment. • Operates as an OAuth 2.0 client of PingAM. • Uses the self-service tree endpoints to configure authentication trees. You can find more information on these endpoints in the PingAM API explorer, which you can access through the AM admin UI. If you write your own admin UI, use these endpoints to manage tree configurations.
End User UI	• Provides users with personalized pages to access their applications and update their profiles. • Replaces the PingIDM end-user UI in an Advanced Identity Software deployment. • Can be customized for your deployment. • Runs separately as a single-page application. • Operates as an OAuth 2.0 client of PingAM.
Login UI	• Replaces the PingAM login UI (XUI) in an Advanced Identity Software deployment. • Can be customized for your deployment. • Runs separately as a single-page application. • Operates as a native PingAM client, not an OAuth 2.0 client.

High-level instructions for customizing the Advanced Identity Software Enduser and Login UIs:

1. Clone the Platform UI repository.

2. Carefully review the README.

3. Develop your customizations.

4. For each UI that you customize, adjust the variables found in the `.env.production` file to match your production deployment.

5. Build the customized UIs.

   You'll find the resulting single-page application files in the `dist/` folder of each UI.

6. Deploy the files in your environment.

   For details, keep reading.

End User UI customization

The Advanced Identity Software End User UI replaces the PingIDM end-user UI in an Advanced Identity Software deployment. PingIDM still includes the end-user UI files, as they are useful in standalone deployments.

Choose how to deploy your customized version of the End User UI:

- Deploy your customized End User UI in a separate web server.

  In the PingAM OAuth 2.0 client profile for the End User UI, set the redirection URIs to reflect the URL to your End User UI.

  This approach is reflected in the sample deployments in this documentation.

- Copy the contents of the `dist/*` folder of your customized end-user UI over the files in the expanded PingIDM `ui/enduser` folder, overwriting existing files.

  The End User UI is then in the same domain as PingIDM.

  If you use a path that is different from `ui/enduser` for the files, also update the `conf/ui.context-enduser.json` configuration to match.

Login UI customization

The Advanced Identity Software Login UI replaces the PingAM login UI (XUI) in an Advanced Identity Software deployment. PingAM still includes the login UI files, as they are useful in standalone deployments.

The Login UI operates as a native PingAM client, capable of working with authentication trees. It’s not an OAuth 2.0 client, but instead a component used in OAuth 2.0 flows. PingAM, acting as an OAuth 2.0 authorization server, relies on the Login UI for resource owner authentication operations.

The Login UI leverages the PingAM authentication trees that are compatible with Ping Advanced Identity Software.

The Login UI translates the PingAM `/json/authenticate` challenges into web pages for users, and translates user responses back into REST calls.

Choose how to deploy your customized version of the Advanced Identity Software Login UI:

If you…	How to deploy
Use only OAuth 2.0 clients of PingAM, not others such as SAML or same-domain policy agents.	Deploy your customized login UI in a separate web server. In AM admin UI, for each realm, browse to **Authentication** > **Settings**, and click the **General tab**. Set the **External Login Page URL** value to the URL of your customized Login UI. If necessary, update the PingAM **CORS** and **Validation Service** configurations. This approach is reflected in the sample deployments in this documentation.
Customize the PingAM .war file for your deployment.	Copy the contents of the `dist/*` folder of your customized Login UI over the files in the expanded PingAM `webapps/am/XUI` folder, overwriting XUI files.

If you…	How to deploy

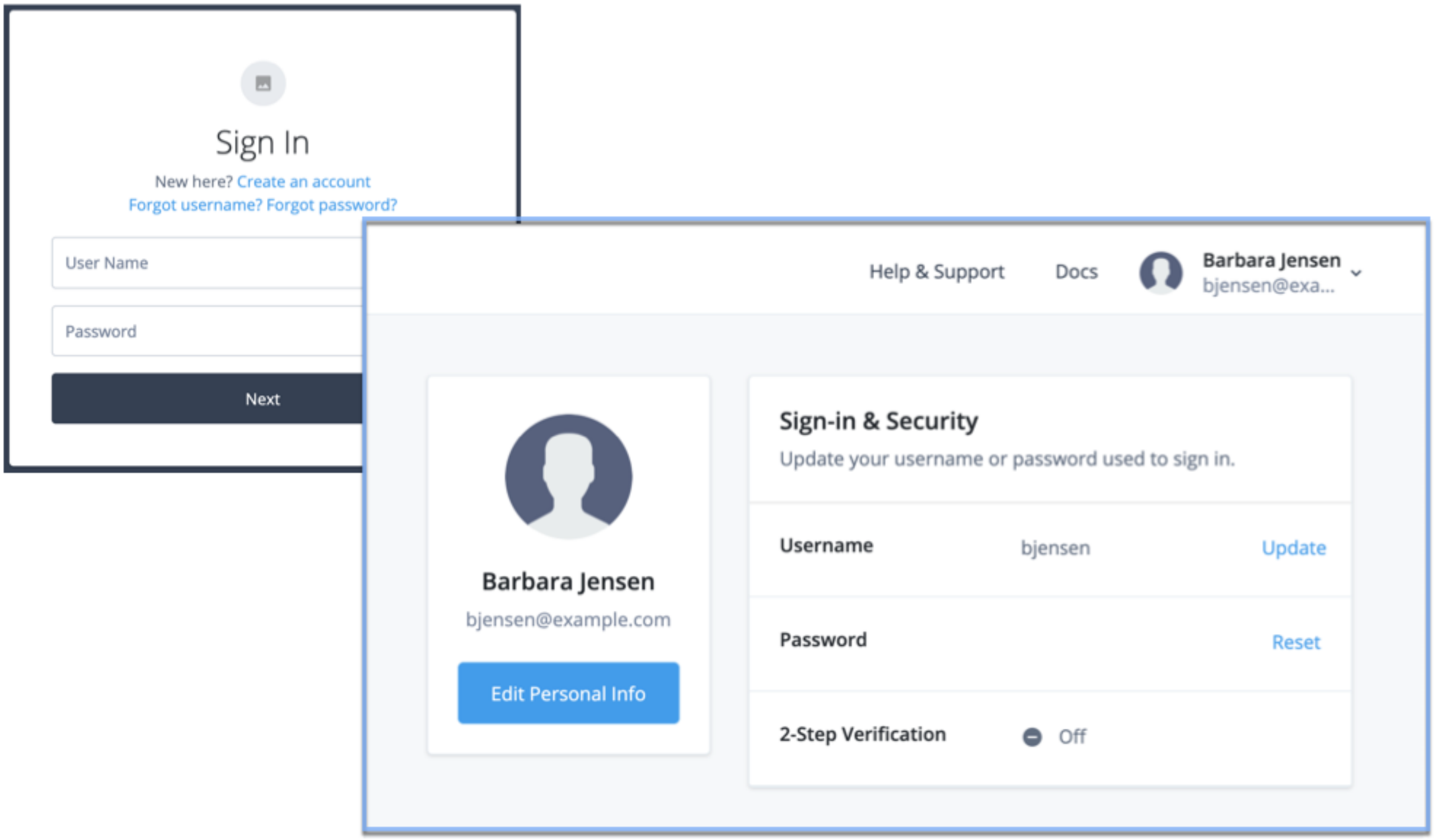
Run PingAM behind a reverse proxy.	Deploy your customized Login UI `dist/*` files in a separate web server, and use the reverse proxy to direct requests for PingAM XUI to your customized Login UI instead.
	A few of the JavaScript files from the PingAM XUI serve to properly render OAuth 2.0 UI screens. Therefore, in addition, customize the PingAM .war file to move PingAM XUI files, and specify that location when starting PingAM.
	1. Expand the PingAM .war.
	2. Move the XUI files:
	`mv am/XUI am/OAuth2_XUI`
	3. Update runtime options so that they specify the location of the XUI files when PingAM starts.
	For example, if you run in Tomcat as described above, add this option alongside the others in the `setenv.sh` script:
	`-Dorg.forgerock.am.oauth2.consent.xui_path=/OAuth2_XUI`

Hosted pages

Overview

PingOne Advanced Identity Cloud hosts default web pages, known as *hosted pages*, that you can use in end-user journeys. Hosted pages support localization, and have customizable themes. The pages are designed to help you quickly create and test common user self-service operations.

For example, the default login journey starts with a sign-in page for capturing the username and password. The journey ends with the end user’s profile page.



Sign In

New here? Create an account

Forgot username? Forgot password?

Barbara Jensen

bjensen@example.com

Sign-in & Security

Update your username or password used to sign in.

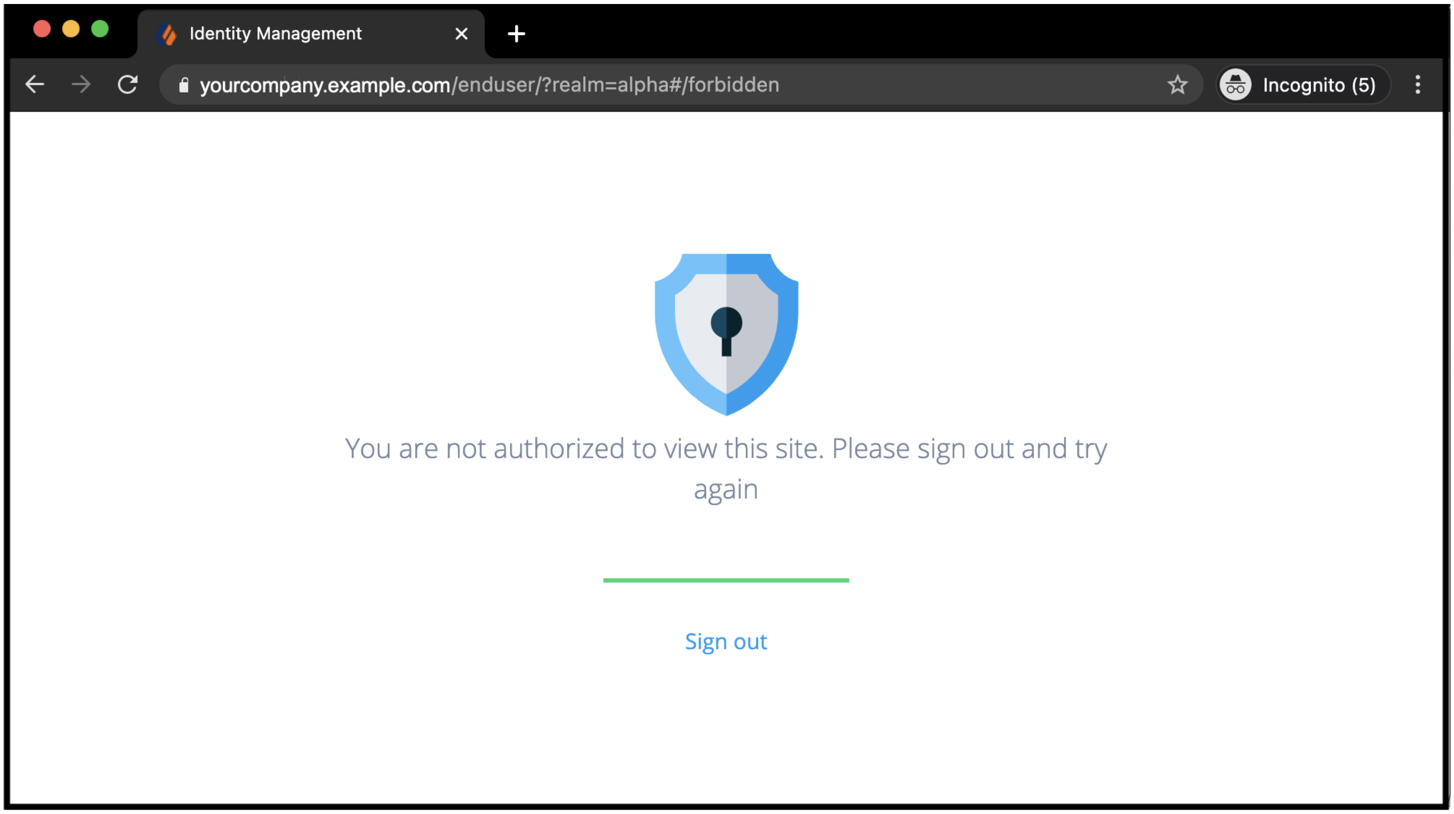
Username bjensen

Password

2-Step Verification ⊖ Off

If you don’t want to risk exposing information contained in the default end-user profile, deactivate its hosted page. You can use the Ping SDKs or your own APIs to create your own custom web pages.

When hosted pages are deactivated, this web page is displayed to unauthorized users:


Identity Management
yourcompany.example.com/enduser/?realm=alpha#/forbidden
Incognito (5)
You are not authorized to view this site. Please sign out and try again
Sign out

By deactivating the default end-user profile, you can still use the hosted end-user journey UI, while denying unauthorized access to end-user profiles. Your customers manage only their own profiles, or delegate administration, using *your* application.

When you deactivate hosted pages, all hosted pages are deactivated.

Activate or deactivate hosted pages

Manage the setting through the PingIDM UI configuration:

1. Get the current configuration for the Platform admin UI using an administrator's OAuth 2.0 access token.

   To get an access token, open your browser's developer tools before logging in as an administrator, and examine the response to a request to the `access_token` endpoint.

   ```
   curl \
    --header 'Authorization: Bearer <access-token>' \
    https://platform.example.com:9443/openidm/config/ui/configuration
   ```

   The endpoint returns the configuration as JSON.

2. Copy and edit the JSON to set the value of the boolean field, `configuration` > `platformSettings` > `hosted-pages`:

   Activate

   `"hostedPages": true`

Deactivate

`"hostedPages": false`

3. Replace the configuration with your updated copy:

```
curl \
 --request PUT \
 --header 'Authorization: Bearer <access-token>' \
 --data '<updated-configuration-json>'
 https://platform.example.com:9443/openidm/config/ui/configuration
```

The change takes effect immediately.

> **Important**
>
> When you deactivate hosted pages, all hosted pages are deactivated.

Localize end-user and login UIs

You can localize the End User and Login UIs to support the different languages of your end users. You do this with translation configuration, defining a locale-specific set of key/phrase translation pairs to override the default set of key/phrase pairs. You can override some, or all of the default keys, in as many locales as you need. The translation configuration has an effect in all realms.

The UIs try to find a translation configuration for the locale requested by the end user’s browser. If none is available, they default to the `en` locale.

To manage translation configuration, use the `/openidm/config/uilocale/*` endpoint in the REST API.

Translation configuration format

The translation configuration format for each locale is as follows:

```
{
    "enduser": { (1)
        "pages": {
            "dashboard": {
                "widgets": {
                    "welcome": {
                        "greeting": "Translation for predefined 'greeting' key", (2)
                    }
                }
            }
        }
    },
    "login": { (1)
        "login": {
            "next": "Translation for predefined 'next' key" (2)
        },
        "overrides": {
            "UserName": "Translation for literal phrase 'User Name'", (3)
            "Password": "Translation for literal phrase 'Password'" (3)
        }
    },
    "shared": { (1)
        "sideMenu": {
            "dashboard": "Translation for predefined 'dashboard' key" (2)
        }
    }
}
```

1 Top-level blocks

The `enduser`, `login`, and `shared` top-level blocks correspond to the names of the UI packages in GitHub:

- https://github.com/ForgeRock/platform-ui/tree/master/packages/platform-enduser
- https://github.com/ForgeRock/platform-ui/tree/master/packages/platform-login
- https://github.com/ForgeRock/platform-ui/tree/master/packages/platform-shared

2 Key/phrase translation pairs with *predefined* keys

Most of the key/phrase translation pairs are predefined in the `en` locale translation files for each package:

- https://github.com/ForgeRock/platform-ui/blob/master/packages/platform-enduser/src/locales/en.json
- https://github.com/ForgeRock/platform-ui/blob/master/packages/platform-login/src/locales/en.json
- https://github.com/ForgeRock/platform-ui/blob/master/packages/platform-shared/src/locales/en.json

You can translate some or all of the keys. To create different translations in the `enduser` and `login` blocks for a key from the `shared` block, you can copy the JSON structure for the `shared` key into each of the `enduser` and `login` blocks, where they override the key in the `shared` block.

3 Key/phrase translation pairs with *literal* keys

Key/phrase translation pairs defined within an `override` block are not predefined. Instead, the key is made from a literal phrase with all non-alphanumeric characters such as underscores stripped out.

These translation pairs with literal keys are designed as a catch-all solution for undefined UI phrases, and for any unlocalized phrases that come directly from the backend servers. The example shows two literal keys that translate the placeholder text from input fields in an authentication journey. Use this approach to translate server output from authentication messages and journey nodes.

> **Note**
>
> Translation pairs with literal keys are currently only available within the `login` top-level block.

Translation precedence and fall back

The UIs translate each key/phrase pair in a particular order. They determine a primary locale using the requested language from an end user's browser. If no translation is available for the primary locale, they fall back to the default `en` locale.

The translation precedence for an end user with a browser locale of `fr` (French) is as follows:

1. Attempt to use the primary `fr` locale:
    1. Look for the translation key in the configuration for the `fr` locale, `/openidm/config/uilocale/fr`.

       This returns HTTP 404 Not Found if the configuration is missing. Refer to Translation 404 responses.
    2. Look for the translation key in any translation files for the `fr` locale:
        - `platform-enduser/src/locales/fr.json`
        - `platform-login/src/locales/fr.json`
        - `platform-shared/src/locales/fr.json`
2. Fall back to the default `en` locale:
    1. Look for the translation key in the configuration for the `en` local, `/openidm/config/uilocale/en`.

       This returns HTTP 404 Not Found if the configuration is missing. Refer to Translation 404 responses.
    2. Look for the translation key in the translation files for the `en` locale :
        - platform-enduser/src/locales/**en.json**
        - platform-login/src/locales/**en.json**
        - platform-shared/src/locales/**en.json**

Translation 404 responses

One or more 404 responses may display in the browser console for the `/openidm/config/uilocale/*` endpoint. These are expected and do not indicate a UI error; the 404 responses mean that the UI cannot locate a translation configuration override, which is perfectly valid if you have not added one.

To suppress the 404 responses, create a translation configuration with an empty body for each locale reporting a 404 response.

REST API

Create or replace translation configuration

1. Get an administrator access token.

To get an access token, open your browser's developer tools before logging in as an administrator, and examine the response to a request to the `access_token` endpoint.

2. Create or replace the translation configuration for each locale:

```
curl \
--request PUT 'http://openidm.example.com:8080/openidm/config/uilocale/<locale>' \ (1)
--header 'Authorization: Bearer <access-token>' \ (2)
--header 'Content-Type: application/json' \
--data-raw '{ (3)
    "enduser": {
        "pages": {
            "dashboard": {
                "widgets": {
                    "welcome": {
                        "greeting": "Bonjour"
                    }
                }
            }
        }
    },
    "login": {
        "login": {
            "next": "Suivant"
        },
        "overrides": {
            "UserName": "Nom d'\''utilisateur",
            "Password": "Mot de passe"
        }
    },
    "shared": {
        "sideMenu": {
            "dashboard": "Tableau de bord"
        }
    }
}
'
```

1 Replace <locale> with a locale identifier. Some examples are:

- `en` (English)
- `es` (Spanish)
- `fr` (French)
- `en-us` (English - United States)
- `es-ar` (Spanish - Argentina)
- `fr-ca` (French - Canada)

2 Replace <access-token> with the access token.

3 Replace the example translation configuration with your own translation configuration.

```
{
    "_id": "uilocale/fr",
    "enduser": {
        "pages": {
            "dashboard": {
                "widgets": {
                    "welcome": {
                        "greeting": "Bonjour"
                    }
                }
            }
        }
    },
    "login": {
        "login": {
            "next": "Suivant"
        },
        "overrides": {
            "UserName": "Nom d'\''utilisateur",
            "Password": "Mot de passe"
        }
    },
    "shared": {
        "sideMenu": {
            "dashboard": "Tableau de bord"
        }
    }
}
```

View translation configuration

> **Note**
>
> An access token is not needed to view the translation configuration as it is publicly accessible.

1. View the translation configuration using a GET request:

```
curl \
--request GET 'http://openidm.example.com:8080/openidm/config/uilocale/<locale>' (1)
```

1 Replace <locale> with a locale identifier.

```
{
    "_id": "uilocale/fr",
    "enduser": {
        "pages": {
            "dashboard": {
                "widgets": {
                    "welcome": {
                        "greeting": "Bonjour"
                    }
                }
            }
        }
    },
    "login": {
        "login": {
            "next": "Suivant"
        },
        "overrides": {
            "UserName": "Nom d'\''utilisateur",
            "Password": "Mot de passe"
        }
    },
    "shared": {
        "sideMenu": {
            "dashboard": "Tableau de bord"
        }
    }
}
```

Delete translation configuration

1. Get an access token for the realm where the translations are applied.

   Open your browser's developer tools before logging in to the realm as an administrator, and examine the response to a request to the `access_token` endpoint.

2. Delete the translation configuration:

```
curl \
--request DELETE 'http://openidm.example.com:8080/openidm/config/uilocale/<locale>' \ (1)
--header 'Authorization: Bearer <access_token>' \ (2)
```

1 Replace <locale> with a locale identifier.
Replace <access-token> with the access token.

2

```
{
    "_id": "uilocale/fr",
    "enduser": {
        "pages": {
            "dashboard": {
                "widgets": {
                    "welcome": {
                        "greeting": "Bonjour"
                    }
                }
            }
        }
    },
    "login": {
        "login": {
            "next": "Suivant"
        },
        "overrides": {
            "UserName": "Nom d'\''utilisateur",
            "Password": "Mot de passe"
        }
    },
    "shared": {
        "sideMenu": {
            "dashboard": "Tableau de bord"
        }
    }
}
```

Customize end-user and login UI themes

Overview

Realms have a default *theme* that includes the colors of buttons and links, typefaces, and so on. This default theme applies to the End User and Login UIs. You can add custom themes so the screens displayed to the end users are *specific to their authentication journey*.

Custom themes let you create a different look and feel for each brand you support, including different profile page layouts, logos, headers, and footers.

A theme is *followed* throughout an authentication journey. This means that if a user logs in through the login UI with a specific theme, the remaining pages in the journey use the same theme.

Add a custom theme

In the Platform admin UI:

1. Select **Hosted Pages > + New Theme**.

   Duplicate an existing theme by clicking ••• next to the theme, then selecting **Duplicate**.

2. Enter a theme name that describes the theme's purpose; for example, the brand associated with an authentication journey.

3. Use the tabs and options to customize various aspects of the theme:

Tab	Option	What you can customize
Global	Styles	Colors of the text, links, menus, buttons, and background pages across all journey and user account pages
	Favicon	Favicon logo displayed for all journey and account pages
	Settings	Theme name and linked trees
Journey Pages	Styles	Colors of the text, links, menus, buttons, and background pages of authentication and registration journey pages
	Logo	Logo to display on sign-in and registration pages
	Layout	Layout of the authentication and registration journey pages, including custom headers and footers
Account Pages	Styles	Colors of the text, links, menus, buttons, and background pages of customer-facing pages, such as account profile and dashboard
	Logo	Logo to display on customer-facing pages
	Layout	Layout of customer-facing pages, including custom footers; categories of information that appear on the account profile page

Apply a custom theme to a journey

In the Platform admin UI:

1. Select **Journeys**, then select the journey for the custom theme, and click **Edit**.

2. Click **... > Edit Details** then select **Override theme**.

3. Select the custom theme to apply, then click **Save**.

Custom headers and footers

Each theme lets you configure localized custom headers and footers:

	Header	Footer
Journey pages	✓	✓
Account pages	n/a	✓

Headers and footers can take HTML or inline CSS to insert links, classes, and so on. Scripting is not currently supported in headers and footers.

The account footer is separate from the journey footer. This lets you set up different buttons, links, and so on, that are displayed to a user after they log in.

Enable headers and footers for a theme

1. In the Platform admin UI, go to **Hosted Pages**, then select a theme.
2. Select either **Journey Pages** or **Account Pages**.
3. In the panel on the right-hand side, click **Layout**.
    1. Find the **Header** section (journey pages only), then enable the switch.
    2. Find the **Footer** section, then enable the switch.

Edit headers and footers

1. Follow the steps in Enable headers and footers for a theme to find the appropriate **Header** or **Footer** section, then click the preview to open the editor.
2. If you do not need localized content, edit the HTML as appropriate, then go to step 4.
3. If you need localized content:
    1. Follow the steps in Localize headers and footers to add as many locales as you need.
    2. Use the locale selector to change locales, and edit the HTML in each locale as appropriate.
4. Click **Save**.

Localize headers and footers

1. Follow the steps in Enable headers and footers for a theme to find the appropriate **Header** or **Footer** section, then click the preview to open the editor.
2. To add an initial locale for the existing header or footer content:
    1. Click **+ Specify a Locale** to open a secondary modal.
    2. In the **Add a Locale** secondary modal, enter a locale identifier; for example, `fr` (French), or `fr-ca` (French - Canada).
    3. Click **Add** to add the locale and close the secondary modal.
    4. The **+ Specify a Locale** link is replaced by a locale selector, with the new locale preselected.
3. To add another locale:
    1. Click the locale selector, then click **+ Add Locale** to open a secondary modal.
    2. In the **Add a Locale** secondary modal, enter a locale identifier; for example, `es` (Spanish), or `es-ar` (Spanish - Argentina).
    3. Click **Add** to add the locale and close the secondary modal.

   4. The new locale is now available in the locale selector, and is preselected. The header or footer content for the new locale is a copy of the header or footer content from the initial locale.

   5. Translate the header or footer content for the new locale.

   Repeat the steps to add as many locales as you need.

4. Click **Save**.

Custom attributes

These sample deployments demonstrate using PingDS as a shared identity store for PingAM and PingIDM. The PingDS setup profile that configures PingDS as a shared identity store defines all the Advanced Identity Software attributes required by PingAM and PingIDM.

Many deployments use additional custom attributes in identity profiles. The following examples show how to add a custom attribute, and how to configure PingAM and PingIDM to use it.

> **Note**
>
> This example adds a custom attribute that Advanced Identity Software can retrieve with a user profile. This custom attribute is not searchable, and therefore not indexed.

Before you start, create a `demo` account for test purposes in your sample deployment:

1. Browse to the Advanced Identity Software End User UI page of the sample deployment, and click **Create an account**.

2. Create a user with user identifier `demo`, and whatever other attributes you like.

3. Find this user's entry in the PingDS shared identity repository:

```
/path/to/opendj/bin/ldapsearch \
--hostname directory.example.com \
--port 1636 \
--useSsl \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--bindDn uid=admin \
--bindPassword str0ngAdm1nPa55word \
--baseDn ou=identities \
"(uid=demo)"
```

   Notice that the user's entry is named for its `fr-idm-uuid` attribute.

Define the attribute in PingDS

You define the attribute in PingDS as an attribute type in the LDAP schema. In LDAP, an entry's object classes define which attributes it can have. You therefore also define an object class that lets the entry have the custom attribute.

The example custom attribute is a multi-valued directory string attribute named `customAttribute`. The auxiliary object class that lets the entry have the attribute is named `customAttributeOC`:

1. In PingDS, add LDAP schema for the new attribute and object class alongside other LDAP schema definitions:

   ```
   /path/to/opendj/bin/ldapmodify \
   --hostname directory.example.com \
   --port 1636 \
   --useSsl \
   --usePkcs12TrustStore /path/to/opendj/config/keystore \
   --trustStorePassword:file /path/to/opendj/config/keystore.pin \
   --bindDn uid=admin \
   --bindPassword str0ngAdm1nPa55word << EOF
   dn: cn=schema
   changetype: modify
   add: attributeTypes
   attributeTypes: ( customAttribute-oid
     NAME 'customAttribute'
     EQUALITY caseIgnoreMatch
     ORDERING caseIgnoreOrderingMatch
     SUBSTR caseIgnoreSubstringsMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     USAGE userApplications )
   -
   add: objectClasses
   objectClasses: ( customAttributeOC-oid
     NAME 'customAttributeOC'
     SUP top
     AUXILIARY
     MAY customAttribute )
   EOF
   ```

   By default, PingDS writes these definitions to the file `/path/to/opendj/db/schema/99-user.ldif`.

2. Test that you can add a custom attribute to the `demo` user entry.

   Use the `fr-idm-uuid` that you got when searching for `uid=demo` in `ou=identities`:

   ```
   /path/to/opendj/bin/ldapmodify \
   --hostname directory.example.com \
   --port 1636 \
   --useSsl \
   --usePkcs12TrustStore /path/to/opendj/config/keystore \
   --trustStorePassword:file /path/to/opendj/config/keystore.pin \
   --bindDn uid=admin \
   --bindPassword str0ngAdm1nPa55word << EOF
   dn: fr-idm-uuid=<fr-idm-uuid-for-demo-user>,ou=people,ou=identities
   changetype: modify
   add: objectClass
   objectClass: customAttributeOC
   -
   add: customAttribute
   customAttribute: Testing 1, 2...
   EOF
   ```

3. Read the `demo` user entry to check your work:

```
/path/to/opendj/bin/ldapsearch \
--hostname directory.example.com \
--port 1636 \
--useSsl \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--bindDn uid=admin \
--bindPassword str0ngAdm1nPa55word \
--baseDn ou=identities \
"(uid=demo)" \
customAttribute
dn: fr-idm-uuid=<fr-idm-uuid-for-demo-user>,ou=people,ou=identities
customAttribute: Testing 1, 2...
```

Notice that the value of `customAttribute` is set to `Testing 1, 2...`.

LDAP schema features are much richer than this simple example can demonstrate. For details about LDAP schema in PingDS, see LDAP schema.

Update PingAM to use the attribute

Update the PingAM configuration to make PingAM aware of the new object class and attribute:

1. Sign in to the Platform admin UI as `amAdmin`.

   The password used in the documentation to set up the platform is `Passw0rd`.

2. Under **Native Consoles**, select **Access Management** to open the PingAM admin console.

3. In the `alpha` realm, under **Identity Stores** > **OpenDJ** > **User Configuration**, update these settings:

   LDAP User Object Class

   Add `customAttributeOC`.

   LDAP User Attributes

   Add `customAttribute`.

4. Save your work.

For additional details, see Adding user profile attributes.

Update PingIDM to use the attribute

Update the PingIDM configuration to make PingIDM aware of the attribute:

1. In the `conf/managed.json` file, under `user` > `schema` > `order`, add the custom attribute to the list:

```
"customAttribute",
```

2. In the `conf/managed.json` file, under `user` > `schema` > `properties`, define a property corresponding to the custom attribute:

   ```
   "customAttribute" : {
    "title" : "Custom Attribute",
    "type" : "string",
    "viewable" : true,
    "searchable" : false,
    "userEditable" : true
   },
   ```

   Notice that this property is not searchable; meaning, it does not need to be indexed.

3. In the `conf/repo.ds.json` file, under `resourceMapping` > `explicitMapping` > `managed/user` > `objectClasses`, add the object class:

   ```
   "customAttributeOC",
   ```

4. In the `conf/repo.ds.json` file, under `resourceMapping` > `explicitMapping` > `managed/user` > `properties`, add a mapping for the attribute:

   ```
   "customAttribute" : {
    "type" : "simple",
    "ldapAttribute" : "customAttribute"
   },
   ```

5. Restart PingIDM to take the changes to the `conf/repo.ds.json` file into account.

For additional details, see Create and modify object types, and Explicit mappings (PingDS).

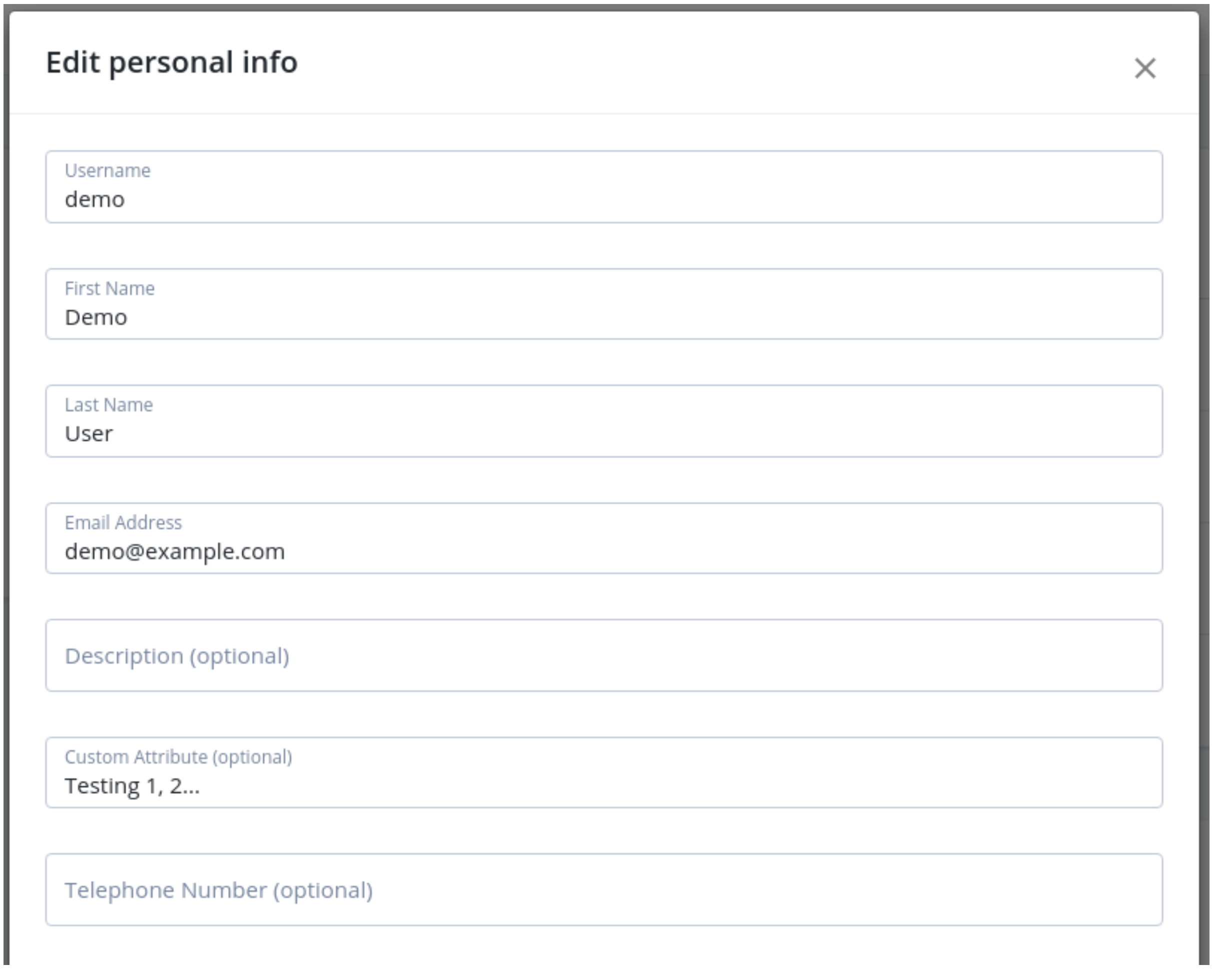
View the results

After configuring PingDS, PingAM, and PingIDM to use the custom attribute:

1. Browse to the Advanced Identity Software End User UI, and sign in as the `demo` user.

2. Click **Edit Your Profile**, and then click **Edit Personal Info**.

   Notice that your custom attribute is visible with the value you set:



Edit personal info

Username
demo

First Name
Demo

Last Name
User

Email Address
demo@example.com

Description (optional)

Custom Attribute (optional)
Testing 1, 2...

Telephone Number (optional)

Groups

Important

This is *not* a comprehensive Advanced Identity Software implementation guide. These sample setup instructions show a minimal integration of Advanced Identity Software components to get you started.
Ping Advanced Identity Software offers maximum extensibility and flexibility in self-managed deployments. Advanced Identity Software includes many features and options these sample setup instructions do not cover. If you don't need maximum extensibility and flexibility, there are simpler alternatives:

- To consume the Advanced Identity Software as a service, use PingOne Advanced Identity Cloud.
- To deploy in Kubernetes, start with the ForgeOps reference implementation.

For help with your deployment and to validate your plans before deploying in production, contact Ping Identity.

Groups are an important tool for identity management. They greatly simplify managing collections of users, applying permissions and authorizations to all members of a group, rather than to individual users. These groups might follow an organization structure, but might instead be based on the needs and privileges for an otherwise arbitrary set of users.

The managed *group* object is an optional managed object type and is defined defined in `managed.json` like any other managed object type. Managed groups simplify management by using common groups across the entire platform.

Users are made members of groups through the relationships mechanism. You should understand how relationships work before you read about PingIDM groups.

Add groups as a managed object

To add groups as a managed object type, update `managed.json` and `repo.ds.json` in your PingIDM `conf/` directory:

In `managed.json`:

1. Add `groups` to the `order` property in the `user` managed object:

```
...
"stateProvince",
"roles",
"groups",
"manager",
...
```

2. Add a `groups` property to the `user` managed object type:

```
"groups" : {
    "description" : "Groups",
    "title" : "Group",
    "id" : "urn:jsonschema:org:forgerock:openidm:managed:api:User:groups",
    "viewable" : true,
    "userEditable" : false,
    "returnByDefault" : false,
    "usageDescription" : "",
    "isPersonal" : false,
    "type" : "array",
    "relationshipGrantTemporalConstraintsEnforced" : false,
    "items" : {
        "type" : "relationship",
        "id" : "urn:jsonschema:org:forgerock:openidm:managed:api:User:groups:items",
        "title" : "Groups Items",
        "reverseRelationship" : true,
        "reversePropertyName" : "members",
        "notifySelf" : true,
        "validate" : true,
        "properties" : {
            "_ref" : {
                "description" : "References a relationship from a managed object",
                "type" : "string"
            },
            "_refProperties" : {
                "description" : "Supports metadata within the relationship",
                "type" : "object",
                "title" : "Groups Items _refProperties",
                "properties" : {
                    "_id" : {
                        "description" : "_refProperties object ID",
                        "type" : "string"
                    },
                    "_grantType" : {
                        "description" : "Grant Type",
                        "type" : "string",
                        "label" : "Grant Type"
                    }
                }
            }
        },
        "resourceCollection" : [
            {
                "path" : "managed/group",
                "label" : "Group",
                "conditionalAssociationField" : "condition",
                "query" : {
                    "queryFilter" : "true",
                    "fields" : [
                        "name"
                    ],
                    "sortKeys" : [
                        "name"
                    ]
                }
            }
        ]
    }
},
```

3. Add the `group` managed object type:

```
{
    "name" : "group",
    "schema" : {
        "id" : "urn:jsonschema:org:forgerock:openidm:managed:api:Group",
        "title" : "Group",
        "icon" : "fa-group",
        "mat-icon" : "group",
        "viewable" : true,
        "$schema" : "http://json-schema.org/draft-03/schema",
        "order" : [
            "_id",
            "name",
            "description",
            "condition",
            "members"
        ],
        "required" : [
            "name"
        ],
        "properties" : {
            "_id" : {
                "description" : "Group ID",
                "type" : "string",
                "viewable" : false,
                "searchable" : false,
                "userEditable" : false,
                "usageDescription" : "",
                "isPersonal" : false
            },
            "name" : {
                "title" : "Name",
                "description" : "Group Name",
                "type" : "string",
                "viewable" : true,
                "searchable" : true
            },
            "description" : {
                "title" : "Description",
                "description" : "Group Description",
                "type" : "string",
                "viewable" : true,
                "searchable" : true,
                "userEditable" : false
            },
            "condition" : {
                "description" : "A filter for conditionally assigned members",
                "title" : "Condition",
                "viewable" : false,
                "searchable" : false,
                "isConditional" : true,
                "type" : "string"
            },
            "members" : {
                "description" : "Group Members",
                "title" : "Members",
                "viewable" : true,
                "searchable" : false,
                "userEditable" : false,
                "policies" : [ ],
                "returnByDefault" : false,
                "type" : "array",
                "items" : {
                    "type" : "relationship",
                    "id" : "urn:jsonschema:org:forgerock:openidm:managed:api:Group:members:items",
                    "title" : "Group Members Items",
                    "reverseRelationship" : true,
                    "reversePropertyName" : "groups",
                    "validate" : true,
                    "properties" : {
                        "_ref" : {
                            "description" : "References a relationship from a managed object",
                            "type" : "string"
                        },
                        "_refProperties" : {
                            "description" : "Supports metadata within the relationship",
                            "type" : "object",
                            "title" : "Group Members Items _refProperties",
                            "properties" : {
                                "_id" : {
                                    "description" : "_refProperties object ID",
                                    "type" : "string"
                                },
                                "_grantType" : {
                                    "description" : "Grant Type",
                                    "type" : "string",
                                    "label" : "Grant Type"
                                }
                            }
                        }
                    },
                    "resourceCollection" : [
                        {
                            "notify" : true,
                            "conditionalAssociation" : true,
                            "path" : "managed/user",
                            "label" : "User",
                            "query" : {
                                "queryFilter" : "true",
                                "fields" : [
                                    "userName",
                                    "givenName",
                                    "sn"
                                ]
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

In `repo.ds.json`:

1. Add groups as a property for `managed/user`:

```
"managed/user" : {
    "dnTemplate" : "ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com",
    ...
    "properties" : {
        ...
        "groups" : {
            "type" : "reference",
            "ldapAttribute" : "fr-idm-managed-user-groups",
            "primaryKey" : "cn",
            "resourcePath" : "managed/group",
            "isMultiValued" : true
        },
...
```

2. Add the `managed/group` object type:

```
"managed/group" : {
    "dnTemplate" : "ou=groups,ou=identities",
    "namingStrategy" : {
        "type" : "clientDnNaming",
        "dnAttribute" : "cn"
    },
    "nativeId" : false,
    "objectClasses" : [
        "top",
        "groupOfURLs",
        "fr-idm-managed-group"
    ],
    "jsonAttribute" : "fr-idm-managed-group-json",
    "jsonQueryEqualityMatchingRule" : "caseIgnoreJsonQueryMatch",
    "properties" : {
        "_id" : {
            "primaryKey" : true,
            "type" : "simple",
            "ldapAttribute" : "cn",
            "writability" : "createOnly"
        },
        "_rev" : {
            "type" : "simple",
            "ldapAttribute" : "etag"
        },
        "description" : {
            "type" : "simple",
            "ldapAttribute" : "description"
        },
        "condition" : {
            "type" : "simple",
            "ldapAttribute" : "fr-idm-managed-group-condition"
        },
        "memberURL" : {
            "type" : "simple",
            "ldapAttribute" : "memberURL",
            "isMultiValued" : true,
            "writability" : "createOnly"
        },
        "members" : {
            "type" : "reverseReference",
            "resourcePath" : "managed/user",
            "propertyName" : "groups",
            "isMultiValued" : true
        }
    }
},
```

User group attributes in use

A group can be assigned to a user manually, as a static value of the user’s `groups` attribute, or dynamically, as a result of a condition or script. For example, a user might be assigned to a group such as `sales` dynamically, if that user is in the `sales` organization.

A user's `groups` attribute takes an array of *references* as a value, where the references point to the managed groups. For example, if user bjensen has been assigned to two groups (`employees` and `supervisors`), the value of bjensen's `groups` attribute would look something like the following:

```
"groups": [
    {
        "_ref": "managed/group/supervisors",
        "_refResourceCollection": "managed/group",
        "_refResourceId": "supervisors",
        "_refProperties": {
            "_id": "61315165-9269-4944-8db9-98f681c6b0a9",
            "_rev": "00000000586a94fd"
        }
    },
    {
        "_ref": "managed/group/employees",
        "_refResourceCollection": "managed/group",
        "_refResourceId": "employees",
        "_refProperties": {
            "_id": "2a965519-5788-428c-92d1-19fac497db8f",
            "_rev": "000000001e1793bc"
        }
    }
]
```

The `_refResourceCollection` is the container that holds the group. The `_refResourceId` is the ID of the group. The `_ref` property is a resource path that is derived from the `_refResourceCollection` and the URL-encoded `_refResourceId` . `_refProperties` provides more information about the relationship.

> **Note**
>
> In most cases, PingIDM uses UUIDs as the `_id` for managed objects. Managed groups are an exception: the `_id` and `name` properties should match.

While managed groups appear in the PingAM admin UI and can serve the same function as a static group created in PingAM, they are not the same. A managed group supports dynamic, conditional membership and can be leveraged in other parts of the platform. We recommend using managed objects for all data management in the platform.

Manage groups

The following sections show the REST calls to create, read, update, and delete groups, and to assign groups to users.

They are set up to use the default configuration for groups as a managed object type, discussed in Groups; if you made further customizations to your configuration, your calls may vary.

Authentication for REST

When you set up Advanced Identity Software as described in this documentation, you configure PingIDM to act as a resource server for PingAM OAuth 2.0 clients. The PingIDM OAuth 2.0 clients present a bearer access token to PingIDM when requesting an operation, and PingIDM makes the request to PingAM to introspect the token and reach an authorization decision.

To provision identities, use the `idm-provisioning` client, whose client secret shown in this documentation is `openidm`. The `idm-provisioning` client uses the client credentials grant to request an access token from PingAM:

```
curl \
--request POST \
--data "grant_type=client_credentials" \
--data "client_id=idm-provisioning" \
--data "client_secret=openidm" \
--data "scope=fr:idm:*" \
"http://am.example.com:8081/am/oauth2/realms/root/realms/alpha/access_token"
{
  "access_token": "<access_token>",
  "scope": "fr:idm:*",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

Use the bearer *access_token* to authorize REST calls to PingIDM to provision identity data, such as groups.

Create a group

Using the Admin UI

1. From the navigation bar, click **Identities** > **Manage** > **Groups**.
2. On the **Groups** page, click **New Group**.
3. On the **New Group** page, enter a name and description, and click **Save**.

Using REST

To create a group, send a PUT or POST request to the `/openidm/managed/group` context path.

The following example creates a group named `employees`, with ID `employees`:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "name": "employees",
  "description": "Group that includes temporary and permanent employees"
}' \
"http://openidm.example.com:8080/openidm/managed/group?_action=create"
{
  "_id": "employees",
  "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1028",
  "name": "employees",
  "condition": null,
  "description": "Group that includes temporary and permanent employees"
}
```

To get an *access_token*, see Authentication for REST.

You can also omit `?_action=create` and achieve the same result:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "name": "employees2",
  "description": "Second group that includes temporary and permanent employees"
}' \
"http://openidm.example.com:8080/openidm/managed/group"
{
  "_id": "employees2",
  "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1053",
  "name": "employees2",
  "condition": null,
  "description": "Second group that includes temporary and permanent employees"
}
```

List groups

To list groups over REST, query the `openidm/managed/group` endpoint. The following example shows the `employees` group that you created in the previous example:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://openidm.example.com:8080/openidm/managed/group?_queryFilter=true"
{
  "result": [
    {
      "_id": "employees",
      "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1028",
      "name": "employees",
      "condition": null,
      "description": "Group that includes temporary and permanent employees"
    },
    {
      "_id": "employees2",
      "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1053",
      "name": "employees2",
      "condition": null,
      "description": "Second group that includes temporary and permanent employees"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

To get an *access_token*, see Authentication for REST.

To list the managed groups in the Admin UI, select **Identities > Manage > Groups**.

If you have a large number of groups, use the search box to display only the groups you want.

Add users to a group

You add users to a group through the relationship mechanism. Relationships are essentially references from one managed object to another; in this case, from a user object to a group object. For more information about relationships, refer to Relationships between objects.

You can add group members *statically* or *dynamically*.

To add members statically, you must do one of the following:

- Update the value of the user's `groups` property to reference the group.
- Update the value of the group's `members` property to reference the user.

Dynamic groups use the result of a condition or script to update a user's list of groups.

Add group members statically

Add a user to a group statically, using the REST interface or the Admin UI as follows:

Using REST

Use one of these methods to add group members over REST.

To get an *access_token*, see Authentication for REST:

- Add the user as a group member.

  The following example adds the user with ID `808cca7b-6c7f-40e5-b890-92b7b6eda08c` as a member of the group you created, `employees`:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_ref":"managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_refProperties":{}
}' \
"http://openidm.example.com:8080/openidm/managed/group/employees/members?_action=create"
{
  "_id": "393916e8-e43b-4e83-b372-7816b5e18fac",
  "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4792",
  "_ref": "managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_refProperties": {
    "_id": "393916e8-e43b-4e83-b372-7816b5e18fac",
    "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4792"
  }
}
```

Note

This is the preferred method as it does not incur an unnecessary performance cost for groups with many members.

- Update the user’s `groups` property.

The following example adds the `employees2` group to the same user:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/groups/-",
    "value": {"_ref" : "managed/group/employees2"}
  }
]' \
"http://localhost:8080/openidm/managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c"
{
  "_id": "808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1569",
  "country": null,
  "telephoneNumber": null,
  "mail": "scarter@example.com",
  "memberOfOrgIDs": [],
  "city": null,
  "displayName": null,
  "assignedDashboard": [],
  "effectiveAssignments": [],
  "postalCode": null,
  "description": null,
  "profileImage": null,
  "expireAccount": null,
  "accountStatus": "active",
  "aliasList": [],
  "kbaInfo": [],
  "inactiveDate": null,
  "activeDate": null,
  "consentedMappings": [],
  "sn": "Carter",
  "effectiveGroups": [
    {
      "_refResourceCollection": "managed/group",
      "_refResourceId": "employees",
      "_ref": "managed/group/employees"
    },
    {
      "_refResourceCollection": "managed/group",
      "_refResourceId": "employees2",
      "_ref": "managed/group/employees2"
    }
  ],
  "preferences": null,
  "organizationName": null,
  "givenName": "Sam",
  "stateProvince": null,
  "userName": "scarter",
  "postalAddress": null,
  "effectiveRoles": [],
  "activateAccount": null
}
```

When you update a user's existing groups array, use the `-` special index to add the new value to the set. For more information, see *Set semantic arrays* in Patch operation: add.

- Update the group's `members` property to refer to the user.

  The following sample command makes the user a member of the group you created:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/members/-",
    "value": {"_ref" : "managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c"}
  }
]' \
"http://openidm.example.com:8080/openidm/managed/group/employees"
{
  "_id": "employees",
  "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1028",
  "name": "employees",
  "condition": null,
  "description": "Group that includes temporary and permanent employees"
}
```

  The `members` property of a group is not returned by default in the output. To show all members of a group, you must specifically request the relationship properties (`*_ref`) in your query. The following example lists the members of the group you created:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://openidm.example.com:8080/openidm/managed/group/employees?_fields=name,members"
{
  "_id": "employees",
  "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4160",
  "name": "employees",
  "members": [{
    "_ref": "managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c",
    "_refResourceCollection": "managed/user",
    "_refResourceId": "808cca7b-6c7f-40e5-b890-92b7b6eda08c",
    "_refProperties": {
      "_id": "245c9009-a812-4d54-ae6c-02756243f7cb",
      "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4898"
    }
  }]
}
```

Using the UI

Use one of the following UI methods to add members to a group:

- Update the user entry:
  1. Select **Identities** > **Manage** > **Users** and select the user that you want to add.
  2. Select the **Group** tab and click **Add Groups**.
  3. Select the group from the dropdown list and click **Add**.
- Update the group entry:
  1. Select **Identities** > **Manage** > **Groups** and select the group to which you want to add members.
  2. Select the **Members** tab and click **Add Members**.
  3. Select the user from the dropdown list and click **Add**.

Add group members dynamically

To add a member to a group *dynamically*, use a *condition*, expressed as a query filter, in the group definition. If the condition is `true` for a particular member, that member is added to the group.

A group whose membership is based on a defined condition is called a *conditional group*. To create a conditional group, include a query filter in the group definition.

> **Important**
>
> Properties that are used as the basis of a conditional group query *must* be configured as `searchable`, and must be indexed in the repository configuration. To configure a property as `searchable`, update its definition in your managed configuration.
> For more information, see Create and modify object types.

To add a condition to a group using the admin UI, select **Set up** on the group **Settings** tab, then define the query filter that will be used to assess the condition.

To create a conditional group over REST, include the query filter as a value of the `condition` property in the group definition. The following example creates a group whose members are only users who live in France, meaning their `country` property is set to `FR`:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "description": "Group for employees resident in France",
  "condition": "/country eq \"FR\""
}' \
"http://openidm.example.com:8080/openidm/managed/group?_action=create"
{
  "_id": "fr-employees",
  "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1898",
  "name": "fr-employees",
  "condition": "/country eq \"FR\"",
  "description": "Group for employees resident in France"
}
```

To get an *access_token*, see Authentication for REST.

When a conditional group is created or updated, PingIDM assesses all managed users, and recalculates the value of their `groups` property, if they are members of that group. When a condition is removed from a group, that is, when the group becomes an unconditional group, all members are removed from the group. So, users who became members based on the condition, have that group removed from their `groups` property.

> **Caution**
>
> When a conditional group is defined in an existing data set, every user entry (including the mapped entries on remote systems) must be updated with the relationships implied by that conditional group. The time that it takes to create a new conditional group is impacted by the number of managed users affected by the condition.
> In a data set with a very large number of users, creating a new conditional group can incur a significant performance cost when you create it. If possible, set up your conditional groups at the beginning of your deployment to avoid performance issues later.

Query a user’s group memberships

To list a user’s groups, query their `groups` property. The following example shows a user who is a member of two groups:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://openidm.example.com:8080/openidm/managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c/groups?
_queryFilter=true&_fields=_ref/*,name"
{
  "result": [
    {
      "_id": "38a23ddc-1345-48d6-b753-ad97f472a90e",
      "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1692",
      "_refResourceCollection": "managed/group",
      "_refResourceId": "employees",
      "_refResourceRev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1028",
      "name": "employees",
      "_ref": "managed/group/employees",
      "_refProperties": {
        "_id": "38a23ddc-1345-48d6-b753-ad97f472a90e",
        "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1692"
      }
    },
    {
      "_id": "0fabd212-f0c2-4d91-91f2-2b211bb58e89",
      "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1974",
      "_refResourceCollection": "managed/group",
      "_refResourceId": "supervisors",
      "_refResourceRev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1965",
      "name": "supervisors",
      "_ref": "managed/group/supervisors",
      "_refProperties": {
        "_id": "0fabd212-f0c2-4d91-91f2-2b211bb58e89",
        "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1974"
      }
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

To get an *access_token*, see Authentication for REST.

To view a user's group membership in the Admin UI:

1. Select **Identities > Manage > Users**, then select the user whose groups you want to see.
2. Select the **Groups** tab.

   If you have a large number of groups, use the search box to display only the groups you want.

Remove a member from a group

To remove a static group membership from a user entry, do one of the following:

- Update the value of the user's `groups` property to remove the reference to the role.

- Update the value of the group's `members` property to remove the reference to that user.

You can use both of these methods over REST, or use the Admin UI.

> **Important**
>
> A delegated administrator must use PATCH to add or remove relationships.
> Conditional group membership can only be removed when the condition is changed or removed, or when the group itself is deleted.

Using REST

Use one of the following methods to remove a member from a group.

To get an *access_token*, see Authentication for REST:

- DELETE the group from the user's `groups` property, including the reference ID (the ID of the relationship between the user and the group) in the DELETE request.

  The following example removes a group from a user. Note that ID required in the DELETE request is not the ID of the group but the reference `_id` of the relationship:

  ```
  curl \
  --header "Authorization: Bearer <access_token>" \
  --header "Accept-API-Version: resource=1.0" \
  --request DELETE \
  "http://openidm.example.com:8080/openidm/managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c/groups/e450a32c-
  e289-49e3-8de5-b0f84e07c740"
  {
    "_id": "e450a32c-e289-49e3-8de5-b0f84e07c740",
    "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-2122",
    "_ref": "managed/group/employees",
    "_refResourceCollection": "managed/group",
    "_refResourceId": "employees",
    "_refProperties": {
      "_id": "e450a32c-e289-49e3-8de5-b0f84e07c740",
      "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-2122"
    }
  }
  ```

- PATCH the user entry to remove the group from the array of groups, specifying the *value* of the group object in the JSON payload.

Caution

When you remove a group in this way, you must include the *entire object* in the value, as shown in the following example:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Content-type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation" : "remove",
    "field" : "/groups",
    "value" : {
      "_ref": "managed/group/0bf541d3-e7da-478a-abdd-41cdb74b2cbb",
      "_refResourceCollection": "managed/group",
      "_refResourceId": "0bf541d3-e7da-478a-abdd-41cdb74b2cbb",
      "_refProperties": {
        "_id": "8174332d-b448-4fe1-b507-b8e4751e08ba",
        "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-5015"
      }
    }
  }
]' \
"http://openidm.example.com:8080/openidm/managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c"
{
  "_id": "808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-5245",
  "userName": "demo",
  "customAttribute": "Testing 1, 2…",
  "accountStatus": "active",
  "givenName": "Demo",
  "sn": "User",
  "mail": "demo@example.com"
}
```

- DELETE the user from the group's `members` property, including the reference ID (the ID of the relationship between the user and the role) in the delete request.

  The following example first queries the members of the group to obtain the ID of the relationship, then removes the user's membership from that group:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://openidm.example.com:8080/openidm/managed/group/employees/members?_queryFilter=true&_fields=_ref/
*,name"
{
  "result": [{
    "_id": "ef3261cd-a66f-4d3e-aad8-c0850e0b4a0e",
    "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-2430"
    "_refResourceCollection": "managed/user",
    "_refResourceId": "808cca7b-6c7f-40e5-b890-92b7b6eda08c",
    "_refResourceRev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-5245",
    "_ref": "managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c",
    "_refProperties": {
      "_id": "245c9009-a812-4d54-ae6c-02756243f7cb",
      "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4898"
    }
  }],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}

curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://openidm.example.com:8080/openidm/managed/group/employees/members/245c9009-a812-4d54-ae6c-02756243f7cb"
{
  "_id": "245c9009-a812-4d54-ae6c-02756243f7cb",
  "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4898",
  "_ref": "managed/user/808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "808cca7b-6c7f-40e5-b890-92b7b6eda08c",
  "_refProperties": {
    "_id": "245c9009-a812-4d54-ae6c-02756243f7cb",
    "_rev": "ce85fb00-1c27-47da-9d7a-8b6d4fcedb16-4898"
  }
}
```

Using the UI

Use one of the following methods to remove a group member:

- Select **Identities** > **Manage** > **Users** and select the user whose group or groups you want to remove.

  Select the **Groups** tab, select the group that you want to remove, then select **Remove**.

- Select **Identities** > **Manage** > **Groups**, and select the group whose members you want to remove.

  Select the **Members** tab, select the member or members that you want to remove, then select **Remove**.

Delete a group

To delete a group over the REST interface, simply delete that managed object.

The following command deletes the group you created:

```
curl \
--header "Authorization: Bearer <access_token>" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://openidm.example.com:8080/openidm/managed/group/employees"
{
  "_id": "employees",
  "_rev": "ae6e63c4-94f5-463b-8bef-7a359b8e3004-1028",
  "name": "employees",
  "condition": null,
  "description": "Group that includes temporary and permanent employees"
}
```

To get an *access_token*, see Authentication for REST.

To delete a group using the Admin UI, select **Identities** > **Manage** > **Groups**, select the group you want to remove, then **Delete Group**.