ForgeOps Documentation

July 10, 2025

FORGEOPS

Version: 2025.1;latest

Table of Contents

Get started with ForgeOps

- Start here . . . 5
- Assess your skill level . . . 10
- Support for ForgeOps . . . 14
- Repositories . . . 17
- Third-party software . . . 24
- Release process . . . 26

Set up local environment and cluster

- Setup overview . . . 29
- Google Cloud . . . 29
- AWS . . . 36
- Azure . . . 43
- Minikube . . . 49

Deploy ForgeOps

- Deployment overview . . . 55
- ForgeOps architecture . . . 56
- ForgeOps deployment . . . 61
  - Use Helm on GKE, EKS, or AKS . . . 62
  - Use Helm on Minikube . . . 66
  - Use Kustomize on GKE, EKS, or AKS . . . 69
  - Use Kustomize on Minikube . . . 74
- UI and API access . . . 77
- Next steps . . . 83
- Remove a ForgeOps deployment . . . 84

Customize your deployment

- Customization overview . . . 87
- Additional setup . . . 87
- About custom images . . . 90
- dsimage . . . 91
- amandidmimages . . . 94
  - Property value substitution . . . 96
  - amimage . . . 99
  - idmimage . . . 105
- Customized Docker images . . . 111

Prepare to deploy in production

- Overview . . . 114
- Identity Gateway . . . 115
- Monitoring . . . 115

Security . . . 116

Back up and restore data

Overview . . . 118
Volume snapshots . . . 118
dsbackuputility . . . 125

Upgrade Overview . . . 132

Migrate your Kustomize overlays . . . 134
Migrate from ForgeOps 7.4 or 7.5 branch . . . 136
Upgrade all Docker images to a new major or minor version . . . 140
Update a Docker image to a newer patch release . . . 143
Upgrade Helm charts . . . 146

Troubleshoot a deployment

Overview . . . 147
Kubernetes logs and other diagnostics . . . 148
DS diagnostic tools . . . 150
Troubleshooting theamsterpod . . . 152
Theno-upgradeoption ofconfig export . . . 154
Staged installation . . . 154
Ingress issues . . . 158
Third-party software versions . . . 158
Kustomize . . . 159
Minikube . . . 160
Shell autocompletion . . . 161

Reference material

Base Docker images . . . 162
Theforgeopscommand . . . 178
forgeopscommand reference . . . 182
Secrets Reference . . . 192
Benchmarks . . . 200
About ForgeOps benchmarking . . . 200
Third-party software . . . 201
Test user generation . . . 202
Authentication rate . . . 204
OAuth 2.0 authorization code flow . . . 205
Ingress . . . 207
Glossary . . . 208
Beyond the docs . . . 217

ForgeOps2025.1release notes . . . 218

Get started with ForgeOps

Start here

Ping Identity provides several resources to help you get started in the cloud. These resources demonstrate how to deploy the Ping Identity Platform on Kubernetes. Before you proceed, review the following precautions:

- Deploying Ping Identity Platform software in a containerized environment requires advanced proficiency in many technologies. Learn more about the required skills in Assess Your Skill Level.
- If you don't have experience with complex Kubernetes deployments, then either engage a certified Ping Identity Platform consulting partner or deploy the platform on traditional architecture.
- Don't deploy Ping Identity Platform software in Kubernetes in production until you have successfully deployed and tested the software in a non-production Kubernetes environment.

Learn more about getting support for Ping Identity Platform software in Support for ForgeOps.

> **Important**
>
> Ping Identity only offers its software or services to legal entities that have entered into a binding license agreement with Ping Identity. When you install Docker images provided by ForgeOps, you agree either that: 1) you are an authorized user of a Ping Identity Platform customer that has entered into a license agreement with Ping Identity governing your use of the Ping Identity software; or 2) your use of the Ping Identity Platform software is subject to the Ping Identity Subscription Agreements.

Introducing ForgeOps deployments

The forgeops repository and ForgeOps documentation address a range of typical business needs of our customers. The repository contains artifacts that let you get a sample Ping Identity Platform deployment up and running quickly. After you get the out-of-the-box deployment running, you can tailor it to explore how you might configure your Kubernetes cluster before you deploy the platform in production.

ForgeOps deployments have the following characteristics:

- Fully integrated AM, IDM, and DS installations
- Randomly generated secrets
- Multi-zone high availability[1]
- Replicated directory services[1]
- Ingress configuration[2]
- Certificate management

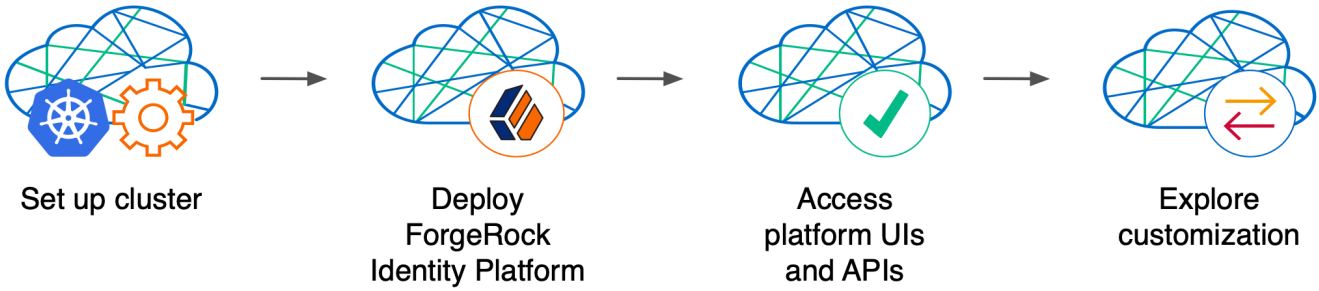
- Prometheus monitoring, Grafana reporting, and alert management[1]

The ForgeOps documentation helps you work with ForgeOps deployments:

- Tells you how you can quickly create a Kubernetes cluster on Google Cloud, Amazon Web Services (AWS), or Microsoft Azure, deploy the Ping Identity Platform, and and access components in the deployment.
- Contains how-tos for preparing for production deployments by customizing monitoring, setting alerts, backing up and restoring directory data, modifying the default security configuration, and running lightweight benchmarks to test DS, AM, and IDM performance.
- Tells you how to modify the AM and IDM configurations in ForgeOps deployments and create customized Docker images for the Ping Identity Platform.
- Keeps you up-to-date with the latest changes to the `forgeops` repository.

Try an out-of-the-box ForgeOps deployment

Before you start planning a production deployment, perform a ForgeOps deployment without any customizations. If you're new to Kubernetes, or new to the Ping Identity Platform, it's a great way to learn, and you'll have a sandbox suitable for exploring the Ping Identity Platform in a cloud environment.



Perform a ForgeOps deployment on Google Cloud, AWS, or Microsoft Azure to quickly spin up the platform for demonstration purposes. You'll get a feel for what it's like to deploy the platform on a Kubernetes cluster in the cloud. When you're done, you'll have a robust starter deployment that you can use to test deployment customizations that you'll need for your production environment. Examples of deployment customizations include, but are not limited to:

- Running lightweight benchmark tests
- Making backups of data and restoring the data
- Securing TLS with a certificate that's dynamically obtained from Let's Encrypt
- Using an ingress controller other than the Ingress-NGINX controller
- Resizing the cluster to meet your business requirements
- Configuring Alert Manager to issue alerts when usage thresholds have been reached

Prerequisite technologies and skills:

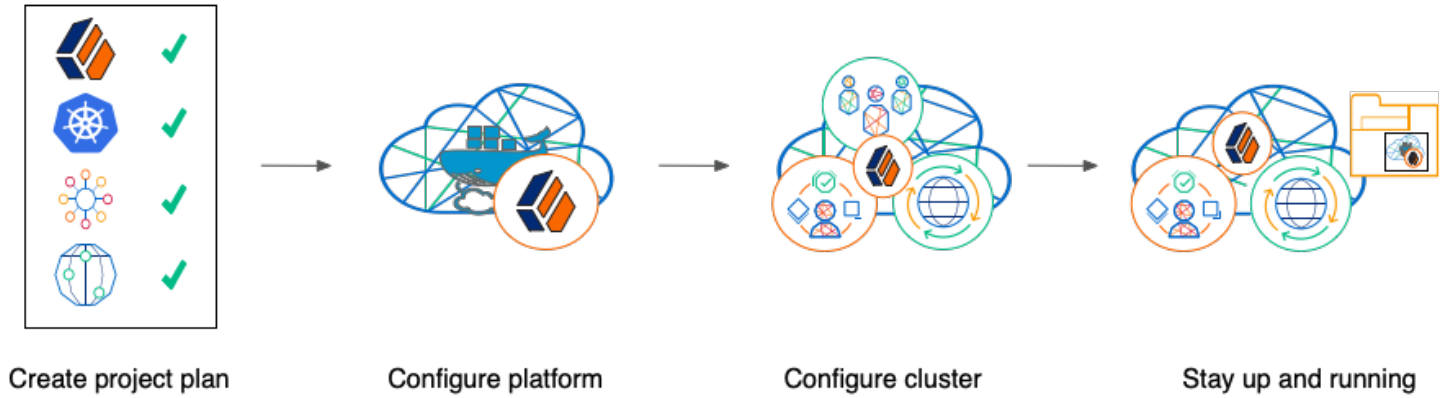
- Git
- Google Cloud, AWS, or Azure

- Kubernetes, running on Google Cloud, AWS, or Azure

More information:

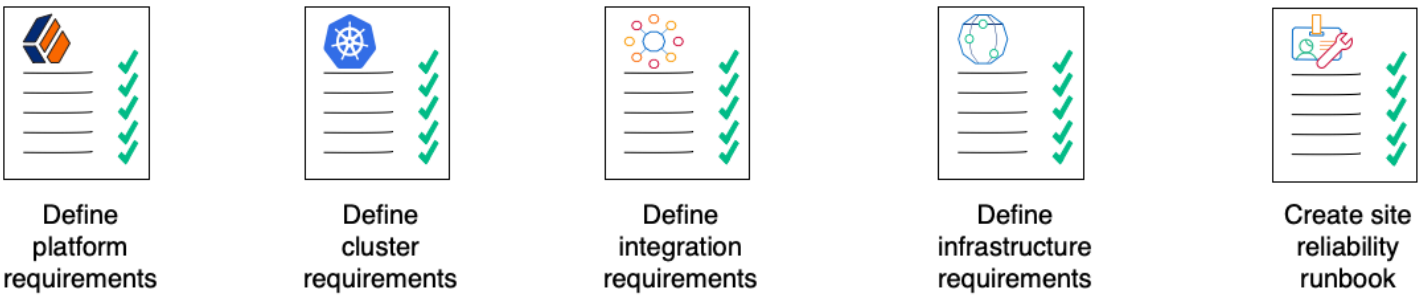
- Setup overview

Build your own service



Perform the following activities to customize, deploy, and maintain a production Ping Identity Platform implementation in the cloud:

Create a project plan



After you've spent some time exploring a ForgeOps deployment, you're ready to define requirements for your production deployment. *Remember, an out-of-the-box ForgeOps deployment is not a production deployment*. Use out-of-the-box ForgeOps deployments to explore deployment customizations. Then, incorporate the lessons you've learned as you build your own production service.

Analyze your business requirements and define how the Ping Identity Platform needs to be configured to meet your needs. Identify systems to be integrated with the platform, such as identity databases and applications, and plan to perform those integrations. Assess and specify your deployment infrastructure requirements, such as backup, system monitoring, Git repository management, CI/CD, quality assurance, security, and load testing.

Be sure to do the following when you transition to a production environment:

- Obtain and use certificates from an established certificate authority.
- Create and test your backup plan.
- Use a working production-ready FQDN.
- Implement monitoring and alerting utilities.

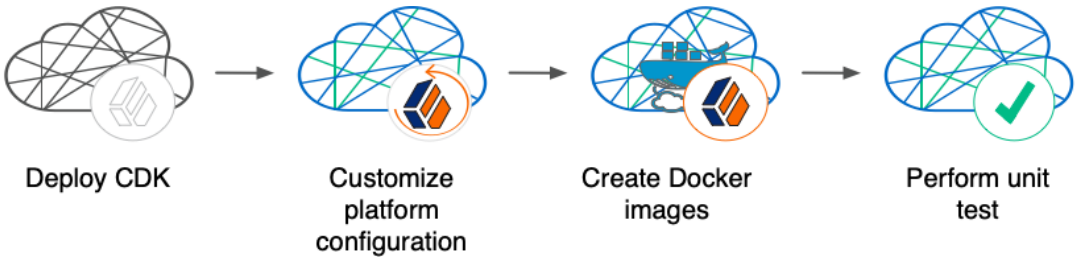
Prerequisite technologies and skills:

- Project planning and management
- Git
- Docker
- Google Cloud, AWS, or Azure
- Kubernetes, running on Google Cloud, AWS, or Azure
- Ping Identity Platform
- Applications and databases that you plan to integrate with Ping Identity Platform
- CI/CD for a production deployment in the cloud
- Integration testing
- Deployment hardening and security
- Benchmarking and load testing
- Site reliability

More information:

- All the ForgeOps documentation

Configure the platform



With your project plan defined, you're ready to configure the Ping Identity Platform to meet the plan's requirements. Install single-instance ForgeOps deployments on your developers' computers. Configure AM and IDM. If needed, include integrations with external applications in the configuration. Iteratively unit test your configuration as you modify it. Build customized Docker images that contain the configuration.

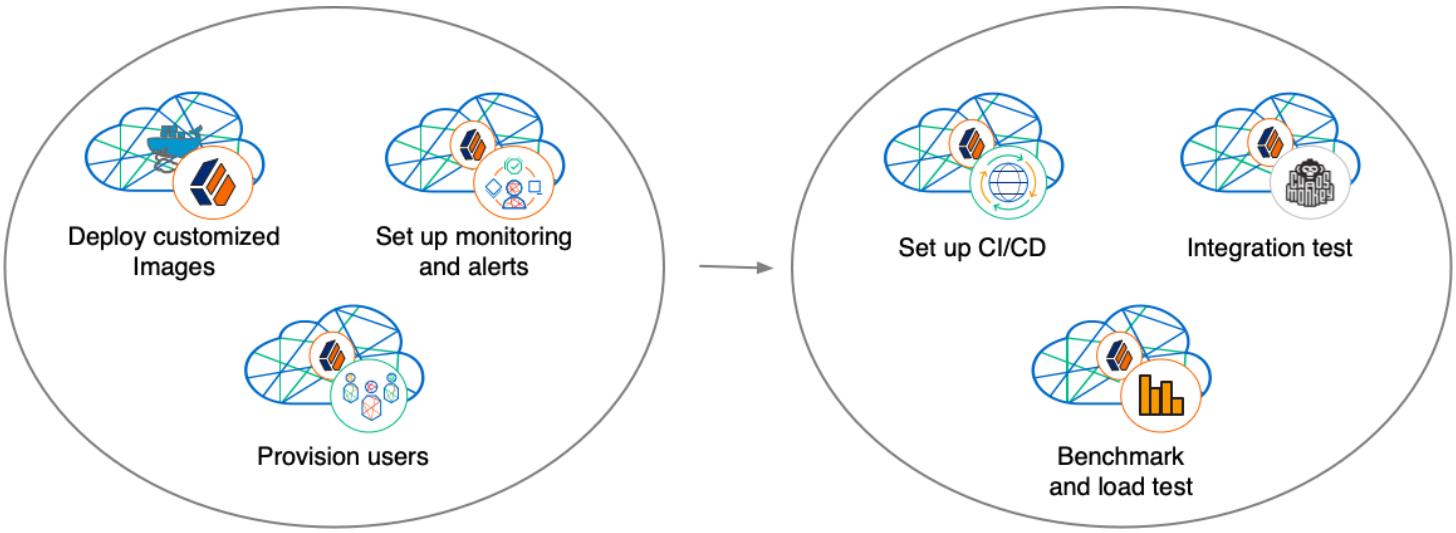
Prerequisite technologies and skills:

- Ping Identity Platform
- Git
- Kubernetes, running on Google Cloud, AWS, or Azure
- Docker

More information:

- Customization overview

Configure your cluster



With your project plan defined, you're ready to configure a Kubernetes cluster that meets the requirements defined in the plan. Install the platform using the customized Docker images developed in Configure the platform. Provision the identity repository with users, groups, and other identity data. Load test your deployment, and then size your cluster to meet service level agreements. Perform integration tests. Harden your deployment. Set up CI/CD for your deployment. Create monitoring alerts so that your site reliability engineers are notified when the system reaches thresholds that affect your SLAs. Implement database backup and test database restore. Simulate failures while under load to make sure your deployment can handle them.

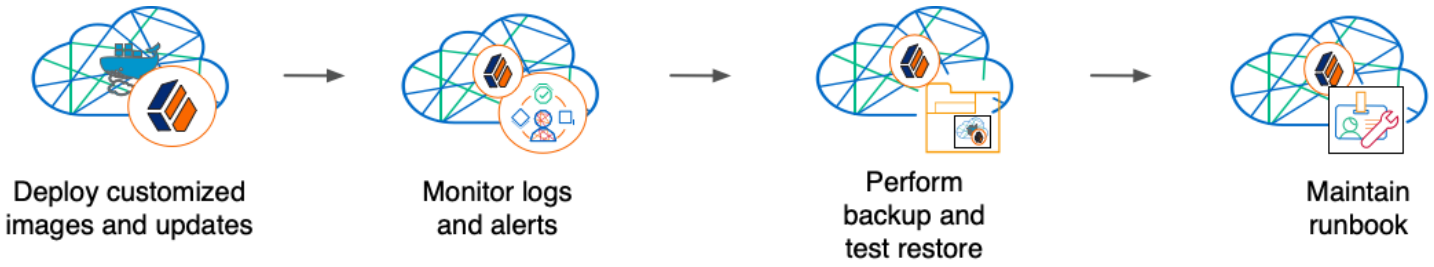
Prerequisite technologies and skills:

- Google Cloud, AWS, or Azure
- Git
- Kubernetes, running on Google Cloud, AWS, or Azure
- Ping Identity Platform
- CI/CD for a production deployment in the cloud
- Integration testing
- Deployment hardening and security
- Kubernetes backup and restore
- Benchmarking and load testing
- Site reliability

More information:

- Prepare to deploy in production
- Setup overview

Stay up and running



By now, you've configured the platform, configured a Kubernetes cluster, and deployed the platform with your customized configuration. Run your Ping Identity Platform deployment in your cluster, continually monitoring it for performance and reliability. Take backups as needed.

Prerequisite technologies and skills:

- Git
- Google Cloud, AWS, or Azure
- Kubernetes, running on Google Cloud, AWS, or Azure
- Ping Identity Platform
- CI/CD for a production deployment in the cloud
- Kubernetes backup and restore
- Site reliability

More information:

- Prepare to deploy in production

1. Not available on single-instance ForgeOps deployments.
2. Not available on ForgeOps deployments on Minikube.

Assess your skill level

Benchmarking and load testing

I can:

- Write performance tests, using tools such as Gatling and Apache JMeter, to ensure that the system meets required performance thresholds and service level agreements (SLAs).
- Resize a Kubernetes cluster, taking into account performance test results, thresholds, and SLAs.
- Run Linux performance monitoring utilities, such as top.

CI/CD for cloud deployments

I have experience:

- Designing and implementing a CI/CD process for a cloud-based deployment running in production.
- Using a cloud CI/CD tool, such as Tekton, Google Cloud Build, Codefresh, AWS CloudFormation, or Jenkins, to implement a CI/CD process for a cloud-based deployment running in production.
- Integrating GitOps into a CI/CD process.

Docker

I know how to:

- Write Dockerfiles.
- Create Docker images, and push them to a private Docker registry.
- Pull and run images from a private Docker registry.

I understand:

- The concepts of Docker layers, and building images based on other Docker images using the FROM instruction.
- The difference between the COPY and ADD instructions in a Dockerfile.

Git

I know how to:

- Use a Git repository collaboration framework, such as GitHub, GitLab, or Bitbucket Server.
- Perform common Git operations, such as cloning and forking repositories, branching, committing changes, submitting pull requests, merging, viewing logs, and so forth.

External application and database integration

I have expertise in:

- AM policy agents.
- Configuring AM policies.
- Synchronizing and reconciling identity data using IDM.
- Managing cloud databases.
- Connecting Ping Identity Platform components to cloud databases.

Ping Identity Platform

I have:

- Attended Ping Identity University training courses.
- Deployed the Ping Identity Platform in production, and kept the deployment highly available.
- Configured DS replication.
- Passed the Certified Access and Identity Management exams from Ping Identity (highly recommended).

Google Cloud, AWS, or Azure (basic)

I can:

- Use the graphical user interface for Google Cloud, AWS, or Azure to navigate, browse, create, and remove Kubernetes clusters.
- Use the cloud provider's tools to monitor a Kubernetes cluster.
- Use the command user interface for Google Cloud, AWS, or Azure.
- Administer cloud storage.

Google Cloud, AWS, or Azure (expert)

In addition to the basic skills for Google Cloud, AWS, or Azure, I can

- Review Terraform artifacts in the `forgeops-extras` repository to see how clusters that support ForgeOps deployments are configured.
- Create and manage a Kubernetes cluster using an infrastructure-as-code tool such as Terraform, AWS CloudFormation, or Pulumi.
- Configure multi-zone and multi-region Kubernetes clusters.
- Configure cloud-provider identity and access management (IAM).
- Configure virtual private clouds (VPCs) and VPC networking.
- Manage keys in the cloud using a service such as Google Key Management Service (KMS), Amazon KMS, or Azure Key Vault.
- Configure and manage DNS domains on Google Cloud, AWS, or Azure.
- Troubleshoot a deployment running in the cloud using the cloud provider's tools, such as Google Stackdriver, Amazon CloudWatch, or Azure Monitor.
- Integrate a deployment with certificate management tools, such as cert-manager and Let's Encrypt.
- Integrate a deployment with monitoring and alerting tools, such as Prometheus and Alertmanager.

I have obtained one of the following certifications (highly recommended):

- Google Certified Associate Cloud Engineer Certification.

- AWS professional-level or associate-level certifications (multiple).
- Azure Administrator.

Integration testing

I can:

- Automate QA testing using a test automation framework.
- Design a chaos engineering test for a cloud-based deployment running in production.
- Use chaos engineering testing tools, such as Chaos Monkey.

Kubernetes (basic)

I've gone through the tutorials at kubernetes.io, and am able to:

- Use the kubectl command to determine the status of all the pods in a namespace, and to determine whether pods are operational.
- Use the kubectl describe pod command to perform basic troubleshooting on pods that are not operational.
- Use the kubectl command to obtain information about namespaces, secrets, deployments, and stateful sets.
- Use the kubectl command to manage persistent volumes and persistent volume claims.

Kubernetes (expert)

In addition to the basic skills for Kubernetes, I have:

- Configured role-based access to cloud resources.
- Configured Kubernetes objects, such as deployments and stateful sets.
- Configured Kubernetes ingresses.
- Configured Kubernetes resources using Kustomize.
- Passed the Cloud Native Certified Kubernetes Administrator exam (highly recommended).

Kubernetes backup and restore

I know how to:

- Schedule backups of Kubernetes persistent volumes on volume snapshots.
- Restore Kubernetes persistent volumes from volume snapshots.

I have experience with one or more of the following:

- Volume snapshots on Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), or Azure Kubernetes Service (AKS)

- A third-party Kubernetes backup and restore product, such as Velero, Kasten K10, TrilioVault, Commvault, or Portworx PX-Backup.

Project planning and management for cloud deployments

I have planned and managed:

- A production deployment in the cloud.
- A production deployment of Ping Identity Platform.

Security and hardening for cloud deployments

I can:

- Harden a Ping Identity Platform deployment.
- Configure TLS, including mutual TLS, for a multi-tiered cloud deployment.
- Configure cloud identity and access management and role-based access control for a production deployment.
- Configure encryption for a cloud deployment.
- Configure Kubernetes network security policies.
- Configure private Kubernetes networks, deploying bastion servers as needed.
- Undertake threat modeling exercises.
- Scan Docker images to ensure container security.
- Configure and use private Docker container registries.

Site reliability engineering for cloud deployments

I can:

- Manage multi-zone and multi-region deployments.
- Implement DS backup and restore in order to recover from a database failure.
- Manage cloud disk availability issues.
- Analyze monitoring output and alerts, and respond should a failure occur.
- Obtain logs from all the software components in my deployment.
- Follow the cloud provider's recommendations for patching and upgrading software in my deployment.
- Implement an upgrade scheme, such as blue/green or rolling upgrades, in my deployment.
- Create a Site Reliability Runbook for the deployment, documenting all the procedures to be followed and other relevant information.
- Follow all the procedures in the project's Site Reliability Runbook, and revise the runbook if it becomes out-of-date.

Support for ForgeOps

This appendix contains information about support options for ForgeOps deployments and the Ping Identity Platform.

ForgeOps support

The Ping Identity ForgeOps team has developed artifacts in the forgeops and forgeops-extras Git repositories for deploying the Ping Identity Platform in the cloud. The companion ForgeOps documentation provides examples to help you get started.

These artifacts and documentation are provided on an as-is basis. Ping Identity doesn't guarantee the individual success developers may have in implementing the code on their development platforms or in production configurations.

ForgeOps product support lifecycle policy is described here.

Licensing

Ping Identity only offers its software or services to legal entities that have entered into a binding license agreement with Ping Identity. When you install Docker images provided by ForgeOps, you agree either that: 1) you are an authorized user of a Ping Identity Platform customer that has entered into a license agreement with Ping Identity governing your use of the Ping Identity software; or 2) your use of the Ping Identity Platform software is subject to the Ping Identity Subscription Agreements.

Support

Ping Identity provides support for the following resources:

- Docker images provided by the ForgeOps team.
- Artifacts in the forgeops Git repository:
  - Files used to build Docker images for the Ping Identity Platform:
    - Dockerfiles
    - Scripts and configuration files incorporated into the Docker images provided by ForgeOps
    - Canonical configuration profiles for the platform
  - Helm charts
  - Kustomize bases and overlays
- ForgeOps Documentation

For more information about support for specific directories and files in the `forgeops` repository, refer to the `forgeops` repository reference.

Ping Identity provides support for the Ping Identity Platform. For supported components, containers, and Java versions, refer to the following:

- PingAM Release Notes
- PingIDM Release Notes
- PingDS Release Notes

- PingGateway Release Notes

Support limitations

Ping Identity provides no support for the following:

- Artifacts in the forgeops-extras repository. For more information about support for specific directories and files in the `forgeops-extras` repository, refer to the `forgeops-extras` repository reference.
- Artifacts other than Dockerfiles, Helm charts, Kustomize bases, and Kustomize overlays in the forgeops Git repository. Examples include scripts, example configurations, and so forth.
- Infrastructure outside Ping Identity. Examples include Docker, Kubernetes, Google Cloud Platform, Amazon Web Services, Microsoft Azure, and so forth.
- Software outside Ping Identity. Examples include Java, Apache Tomcat, NGINX, Apache HTTP Server, Certificate Manager, Prometheus, and so forth.
- Deployments that deviate from the published ForgeOps architecture. Deployments that do not include the following architectural features are not supported:
  - PingAM and PingIDM are integrated and deployed together in a Kubernetes cluster.
  - PingIDM login is integrated with PingAM.
  - PingAM uses PingDS as its data repository.
  - PingIDM uses PingDS as its repository.
- Ping Identity publishes Docker images for testing and development. For production deployments, it is recommended that customers build and run containers using a supported operating system, required software dependencies, and their customized platform component configurations.

Third-party Kubernetes services

The ForgeOps reference tools are provided for use with Google Kubernetes Engine, Amazon Elastic Kubernetes Service, and Microsoft Azure Kubernetes Service.

Ping Identity supports running the platform on other Kubernetes platforms such as IBM RedHat OpenShift. However, ForgeOps reference tools are not provided on these platforms, and customers must build, maintain, and support their own tools and configurations.

Ping Identity doesn't support Kubernetes itself. Customers must have a support contract in place with their Kubernetes vendor to resolve infrastructure issues. To avoid any misunderstandings, it must be clear that Ping Identity cannot troubleshoot underlying Kubernetes issues.

Modifications to ForgeOps deployment assets may be required to adapt the platform to the customer's Kubernetes implementation. For example, ingress routes, storage classes, NAT gateways, etc., might need to be modified. Making the modifications requires competency in Kubernetes and familiarity with their chosen distribution.

Documentation access

Ping Identity publishes comprehensive documentation online:

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

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

- The developer documentation, such as this site, aims to be technically accurate with respect to the sample that is documented. It is visible to everyone.

Problem reports and information requests

If you are a named customer Support Contact, contact Ping Identity using the Customer Support Portal to request information or report a problem with Dockerfiles, Helm charts, Kustomize bases, or Kustomize overlays in the `forgeops` repository.

When requesting help with a problem, include the following information:

- Description of the problem, including when the problem occurs and its impact on your operation.
- Steps to reproduce the problem.

  If the problem occurs on a Kubernetes system other than Minikube, GKE, EKS, or AKS, we might ask you to reproduce the problem on one of those.

- HTML output from the debug-logs command. For more information, refer to Kubernetes logs and other diagnostics.

Suggestions for fixes and enhancements to artifacts

ForgeOps greatly appreciates suggestions for fixes and enhancements to ForgeOps-provided artifacts in the forgeops and forgeops-extras repositories.

If you would like to report a problem with or make an enhancement request for an artifact in either repository, create a GitHub issue in the repository.

Contact information

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

Ping Identity has staff members around the globe who support our international customers and partners. Learn more about Ping Identity's support offering, including support plans and service-level agreements (SLAs) in the Ping Identity Platform support page.

Repositories

The ForgeOps project provides two public GitHub repositories; the `forgeops` and `forgeops-extras` repositories.

This page provides a high-level overview of the two repositories.

`forgeops` repository

The `forgeops` repository contains files needed for customizing and deploying the Ping Identity Platform on a Kubernetes cluster:

- Files used to build Docker images for the Ping Identity Platform:
  - Dockerfiles
  - Scripts and configuration files incorporated into ForgeOps-provided Docker images
  - Canonical configuration profiles for the platform
- Helm charts
- Kustomize bases and overlays

In addition, the repository contains utility scripts and sample files. The scripts and samples are useful for:

- Performing ForgeOps deployments quickly and easily
- Exploring monitoring, alerts, and security customization

Learn more about the files in the repository, recommendations about how to work with them, and the support status for the files in the `forgeops` repository reference.

> **Note**
>
> Learn about how to configure GitHub notifications here so you can get notified on ForgeOps releases.

`forgeops` repository updates

New `forgeops` repository features become available in the `2025.1.2` tag of the `main` branch from time to time.

When you start working with the `forgeops` repository, clone the repository. Depending on your organization's setup, you'll clone the repository either from the public repository on GitHub, or from a fork. You can find more information in Git clone or Git fork?.

Then, check out the `2025.1.2` tag of the `main` branch and create a working branch. For example:

```
$ git checkout 2025.1.2
$ git checkout -b my-working-branch
```

The ForgeOps team recommends that you regularly incorporate updates to the `2025.1.2` tag into your working branch:

1. Get emails or subscribe to the ForgeOps RSS feed to be notified when there have been updates to ForgeOps 2025.1.2.
2. Pull new commits in the `2025.1.2` tag of the `main` branch into your clone's `2025.1.2` branch.
3. Rebase the commits from the new branch into your working branch in your `forgeops` repository clone.

It's important to understand the impact of rebasing changes from the `forgeops` repository into your branches. `forgeops` repository reference provides advice about which files in the `forgeops` repository to change, which files not to change, and what to look out for when you rebase. Follow the advice in `forgeops` repository reference to reduce merge conflicts, and to better understand how to resolve them when you rebase your working branch with updates that the ForgeOps team has made to the `2025.1.2` tag of the `main` branch.

`forgeops` repository reference

For more information about support for the `forgeops` repository, see Support for ForgeOps.

Directories

bin

Example scripts you can use or model for a variety of deployment tasks.

Recommendation: Don't modify the files in this directory. If you want to add your own scripts to the `forgeops` repository, create a subdirectory under bin, and store your scripts there.

Support Status: Sample files. Not supported by Ping Identity.

charts

Helm charts.

Recommendation: Don't modify the files in this directory. If you want to update a values.yaml file, create your deployment environment using the forgeops env command, and edit values.yaml files in the new environment you created. Learn more in forgeops:reference:forgeops-cmd-ref.adoc#_commandforgeops_env.

Support Status: Supported is available from Ping Identity.

cluster

Artifacts to configure third-party software, such as cert-manager, HAProxy, NGINX, Prometheus, and so on. It also contains storage class definition files.

Recommendation: Don't modify the files in this directory.

Support Status: Sample file. Not supported by Ping Identity.

docker

Contains three types of files needed to build Docker images for the Ping Identity Platform: Dockerfiles, support files that go into Docker images, and configuration profiles.

Dockerfile

Common deployment customizations require modifications to the Dockerfile in the docker directory.

Recommendation: Expect to encounter merge conflicts when you rebase changes from ForgeOps into your branches. Be sure to track changes you've made to Dockerfiles, so that you're prepared to resolve merge conflicts after a rebase.

Support Status: Dockerfiles. Support is available from Ping Identity.

Support Files Referenced by Dockerfiles

When customizing the default ForgeOps deployments, you might need to add files to the docker directory. For example, to customize the AM WAR file, you might need to add plugin JAR files, user interface customization files, or image files.

Recommendation: If you only add new files to the docker directory, you should not encounter merge conflicts when you rebase changes from ForgeOps into your branches. However, if you need to modify any files from ForgeOps, you might encounter merge conflicts. Be sure to track changes you've made to any files in the docker directory, so that you're prepared to resolve merge conflicts after a rebase.

Support Status:

Scripts and other files from ForgeOps that are incorporated into Docker images for the Ping Identity Platform: Support is available from Ping Identity.

User customizations that are incorporated into custom Docker images for the Ping Identity Platform: Support is not available from Ping Identity.

Configuration Profiles

The starter configuration profiles provided with ForgeOps. To create your own configuration profiles, use the forgeops config command in your ForgeOps deployment environment. Add your own configuration profiles to the docker directory using the export command. Don't modify the internal-use only `idm-only` and `ig-only` configuration profiles provided by ForgeOps.

Recommendation: You should not encounter merge conflicts when you rebase changes from ForgeOps into your branches.

Support Status: Configuration profiles. Support is available from Ping Identity.

etc

Files used to support ForgeOps deployments.

Recommendation: Don't modify the files in this directory (or its subdirectories).

Support Status: Sample files. Not supported by Ping Identity.

helm

Helm values files for each client environment (env) for use with Helm charts. The Helm values files are created and managed by the forgeops env command.

Files in each ForgeOps deployment environment

File	Description
env.log	Log of `forgeops env` runs.
values.yaml	Configuration of components in ForgeOps deployment using Helm.
values-images.yaml	Docker image used in ForgeOps deployment.
values-ingress.yaml	Ingress configuration, such as FQDN.
values-size.yaml	Component size information such as number of replicas, cpu, and memory

Support Status: Environment specific files. Support is available from ForgeRock.

how-tos

Description and usage of various utilities provided with ForgeOps.

Recommendation: Don't change these files.

Support Status: Description files. Support is available from ForgeRock.

intezer

For ForgeRock internal use only. Don't modify or use.

jenkins-scripts

For ForgeRock internal use only. Don't modify or use.

kustomize

Artifacts for orchestrating the Ping Identity Platform using Kustomize.

Recommendation: Common deployment customizations, such as changing the deployment namespace and providing a customized FQDN, require modifications to files in the kustomize/overlay directory. Be sure to track changes you've made to the files in the kustomize directory, so that you're prepared to resolve merge conflicts after a rebase.

Support Status: Kustomize bases and overlays. Support is available from Ping Identity.

legacy-docs

Documentation for performing ForgeOps deployments using older versions. Includes documentation for supported and deprecated versions of the `forgeops` repository.

Recommendation: Don't modify the files in this directory.

Support Status:

Documentation for supported versions of the `forgeops` repository: Support is available from Ping Identity.

Documentation for deprecated versions of the `forgeops` repository: Not supported by Ping Identity.

lib

Python and shell library files used internally. Don't modify.

releases

For ForgeRock internal use only. Don't modify or use.

Files in the top-level directory

.gcloudignore, .gitchangelog.rc, .gitignore, forgeops.conf.example

For ForgeOps internal use only. Don't modify.

LICENSE

Software license for artifacts in the `forgeops` repository. Don't modify.

Makefile

For ForgeOps internal use only. Don't modify.

notifications.json

For ForgeOps internal use only. Don't modify.

README.md

The top-level `forgeops` repository README file. Don't modify.

`forgeops-extras` repository

Use the forgeops-extras repository to create sample Kubernetes clusters in which you can deploy the Ping Identity Platform.

`forgeops-extras` repository reference

For more information about support for the `forgeops-extras` repository, see Support for ForgeOps.

Directories

terraform

Example Terraform artifacts that automate cluster creation and deletion.

Recommendation: Don't modify the files in this directory. If you want to add your own cluster creation support files to the `forgeops` repository, copy the terraform.tfvars file to a new file and make changes there.

Support Status: Sample files. Not supported by Ping Identity.

Git clone or Git fork?

For the simplest use cases—a single user in an organization performing a ForgeOps deployment for a proof of concept, or exploration of the platform—cloning the ForgeOps public repositories from GitHub provides a quick and adequate way to access the repositories.

If, however, your use case is more complex, you might want to fork the repositories, and use the forks as your common upstream repositories. For example:

- Multiple users in your organization need to access a common version of the repository and share changes made by other users.
- Your organization plans to incorporate `forgeops` and `forgeops-extras` repository changes from ForgeOps.
- Your organization wants to use pull requests when making repository updates.

If you've forked the `forgeops` and `forgeops-extras` repositories:

- You'll need to synchronize your forks with ForgeOps repositories on GitHub when ForgeOps releases new branches.
- Your users will need to clone your forks before they start working instead of cloning the public repositories from GitHub. Because procedures in the documentation tell users to clone the public repositories, you'll need to make sure your users follow different procedures to clone the forks instead.
- The steps to initially get and update your repository clones will differ from the steps provided in the documentation. You'll need to let users know how to work with the forks as the upstream repositories instead of following the steps in the documentation.

Third-party software

Before performing a ForgeOps deployment, install the requisite third-party software on your local computer.

The ForgeOps team recommends you install third-party software using Homebrew on macOS and Linux[1].

Required third-party software

Software	Version	Homebrew package
On all platforms		
• Python 3	3.13.2	`python@3.13`
• Bash	5.2.37	`bash`
• Docker client	28.0.1	`docker`

Software	Version	Homebrew package
• Kubernetes client (kubectl)	1.32.3	`kubernetes-cli`
• Kubernetes context switcher (kubectx)	0.9.5	`kubectx`
• Kustomize	5.6.0	`kustomize`
• Helm	3.17.2	`helm`
• JSON processor jq	1.7.1	`jq`
• Six (Python compatibility library)	1.17.0	`six`
• Setup tools (Python)	75.6.0	`python-setuptools`
• Terraform	1.5.7	`terraform`
Additionally on Google GKE		
• Google Cloud SDK	451.0.1	`google-cloud-sdk` (cask)[1]
Additionally on Amazon EKS		
• Amazon AWS Command Line Interface	2.25.12	`awscli`
• AWS IAM Authenticator for Kubernetes	0.6.28	`aws-iam-authenticator`
Additionally on Azure AKS		
• Azure Command Line Interface	2.71.0	`azure-cli`

Software	Version	Homebrew package
Additionally on Minikube		
• Minikube	1.35.0	`minikube`
• PyYaml	6.0.1	`pyyaml`

Python `venv`

The new `forgeops` utility is built on Python3. Some of the Python3 packages used by `forgeops` have to be installed using `pip`. To separate such Python3 specific packages, Python recommends the use of the `venv` Python virtual environment. Learn more about Python `venv` in venv - Virtual environments.

1. Create a `venv` for using the `forgeops` utility.

```
$ python3 -m venv .venv
```

2. Set up Python3 dependencies for `forgeops` utility.

```
$ source .venv/bin/activate
$ /path/to/forgeops/bin/forgeops configure
```

1. The Linux version of Homebrew doesn't support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew to install software in several cases. You'll need to refer to the software's documentation for information about how to install the software on a Linux system.

ForgeOps release process

ForgeOps release process is aimed at simplifying the management of multiple ForgeOps versions in a single branch. This process:

- Removes the need to create and maintain multiple release branches for every product release.
- Enables easier upgrading and switching between multiple supported versions.
- Supports delivery of regularly promoted secure Docker images.
- Unifies documentation to easily consume product information.

Key features

Single `main` branch for all new releases

- The `main` branch doesn't replace the already released forgeops branches. The currently released branches continue to be supported and updated with new minor and patch releases of the products.

- Deployments use the latest available image for the current product version by default. Customers can select to use a specific supported image as per their needs.

Product versions are separated from the `forgeops` repository

- Product image tags are now maintained in tag files at http://releases.forgeops.com.
- The refactored `forgeops` tool retrieves the requested tags and updates Dockerfiles/Helm/Kustomize as required.
- All new major releases will be released using the new release process only.
- The new process also supports 7.5 and 7.4 product image tags. Customers can continue to use the release/7.5-* and release/7.4-* branches until they are ready to migrate to the new process.
- Customers can select the latest early-adapter images of the products to test the latest product features (similar to the dev image tag on the `master` branch previously).

Supported Docker images

- The new process is required to handle regular delivery of updated secure product images.
  - Docker images are routinely scanned for OS-level vulnerabilities and addressed by the ForgeOps team where OS patches are available.
  - The Ping Identity platform teams provide the platform-level patches.
- The delivery process is automated, and the newly promoted tags will be available at http://releases.forgeops.com.
- New `dev` branch for latest forgeops features. Equivalent to the erstwhile `master` branch which is no longer available.
- Updated release information on https://github.com/ForgeRock/forgeops. Customers can get release updates by setting up their GitHub notifications.
- Single set of documentation that is updated for ForgeOps releases and updates to the `main` branch.
- Best efforts to document the dev branch updates similar to the early-adapter documentation.

Release support matrix

The below table shows the release process for each supported product version:

Product version	New branch for each existing release(`release/7.*-<YYYYMMDD>`)	The `main` branch with a new tag (`YYYY-MM`) for each new release
7.2	☑	
7.3	☑	
7.4	☑	☑
7.5	☑	☑

Product version	New branch for each existing release(`release/7.*-<YYYYMMDD>`)	The `main` branch with a new tag (`YYYY-MM`) for each new release
New releases such as **7.6** or **8.0**		☑

Set up local environment and cluster

Setup overview

Before performing a ForgeOps deployment, you must perform some setup tasks in your local computer, create a Kubernetes cluster (or have access to an existing cluster), and configure your local machine to access the cluster.

The specific tasks you'll need to do vary depending on the platform on which you run Kubernetes:

Google Cloud

Set up your local computer to deploy ForgeOps on Google Cloud.

AWS

Set up your local computer to deploy ForgeOps on AWS.

Azure

Set up your local computer to deploy ForgeOps on Azure.

Minikube

Set up your local computer to deploy ForgeOps on Minikube.

Google Cloud

Before you can perform a ForgeOps deployment on a Kubernetes cluster running on Google Cloud, you must complete these prerequisite tasks:

- Clone the `forgeops` and `forgeops-extras` repositories

- Install third-party software on your local computer
- Start a virtual machine that runs Docker engine on your local computer
- Set up a Google Cloud project that meets the requirements for ForgeOps deployments
- Create a Kubernetes cluster in the project
- Set up your local computer to access the cluster's ingress controller

`forgeops` and `forgeops-extras` repositories

> **Note**
>
> Learn about how to configure GitHub notifications here so you can get notified on ForgeOps releases.

Get the `forgeops` and `forgeops-extras` repositories:

1. Clone the repositories. For example:

```
$ git clone https://github.com/ForgeRock/forgeops.git
$ git clone https://github.com/ForgeRock/forgeops-extras.git
```

   Both repositories are public; you do not need credentials to clone them.

2. Check out the `forgeops` repository's `2025.1.2` tag:

```
$ cd /path/to/forgeops
$ git checkout 2025.1.2
```

   Depending on your organization's repository strategy, you might need to clone the repository from a fork. You might also need to create a working branch from the `2025.1.2` tag of your fork. Learn more about Repository Updates here.

3. Check out the `forgeops-extras` repository's `main` branch:

```
$ cd /path/to/forgeops-extras
$ git checkout main
```

Third-party software

Before performing a ForgeOps deployment, obtain third-party software and install it on your local computer.

ForgeOps team recommends that you install third-party software using Homebrew on macOS and Linux'[1]' .

The versions listed in the following table have been validated for ForgeOps deployments on Google Cloud. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
On all platforms		
• Python 3	3.13.2	`python@3.13`
• Bash	5.2.37	`bash`
• Docker client	28.0.1	`docker`
• Kubernetes client (kubectl)	1.32.3	`kubernetes-cli`
• Kubernetes context switcher (kubectx)	0.9.5	`kubectx`
• Kustomize	5.6.0	`kustomize`
• Helm	3.17.2	`helm`
• JSON processor jq	1.7.1	`jq`
• Six (Python compatibility library)	1.17.0	`six`
• Setup tools (Python)	75.6.0	`python-setuptools`
• Terraform	1.5.7	`terraform`
Additionally on Google GKE		
• Google Cloud SDK	451.0.1	`google-cloud-sdk` (cask)[1]

Python venv

The new `forgeops` utility is built on Python3. Some of the Python3 packages used by `forgeops` have to be installed using `pip`. To separate such Python3 specific packages, Python recommends the use of the `venv` Python virtual environment. Learn more about Python `venv` in venv - Virtual environments.

1. Create a `venv` for using the `forgeops` utility.

```
$ python3 -m venv .venv
```

2. Set up Python3 dependencies for `forgeops` utility.

```
$ source .venv/bin/activate
$ /path/to/forgeops/bin/forgeops configure
```

Docker engine

In addition to the software listed in the preceding table, you'll need to start a virtual machine that runs Docker engine.

- On macOS systems, use Docker Desktop or an alternative, such as Colima.
- On Linux systems, use Docker Desktop for Linux, install Docker machine from your Linux distribution, or use an alternative, such as Colima.

For more information about using Colima when performing ForgeOps deployments, refer to this article.

The default configuration for a Docker virtual machine provides adequate resources for a ForgeOps deployment.

For users running Microsoft Windows

ForgeOps deployments are supported on macOS and Linux. If you have a Windows computer, you'll need to create a Linux VM. We tested the following configurations:

- Hypervisor: Hyper-V, VMWare Player, or VMWare Workstation
- Guest OS: Current Ubuntu LTS release with 12 GB memory and 60 GB disk space
- Nested virtualization enabled in the Linux VM.

Perform all the procedures in this documentation within the Linux VM. In this documentation, the local computer refers to the Linux VM for Windows users.

> **Important**
>
> The Minikube implementation on Windows Subsystem for Linux (WSL2) has networking issues. As a result, consistent access to the ingress controller or the apps deployed on Minikube is not possible. This issue is tracked here. Do not attempt to perform ForgeOps deployments on WSL2 until this issue is resolved.

Google Cloud project setup

Perform these steps to set up a Google Cloud project that meets the requirements for ForgeOps deployments:

1. Log in to the Google Cloud Console and create a new project.
2. Authenticate to the Google Cloud SDK to obtain the permissions you'll need to create a cluster:
    1. Configure the gcloud CLI to use your Google account. Run the following command:

        ```
        $ gcloud auth application-default login
        ```

    2. A browser window appears, prompting you to select a Google account. Select the account you want to use for cluster creation.

        A second screen requests several permissions. Select **Allow**.

        A third screen should appear with the heading, **You are now authenticated with the gcloud CLI!**
3. Assign the following roles to users who will be creating Kubernetes clusters and performing ForgeOps deployments:
    - Editor
    - Kubernetes Engine Admin
    - Kubernetes Engine Cluster Admin
    - Project IAM Admin

    Remember, a ForgeOps deployment is a reference implementation, and is not for production use. The roles you assign in this step are suitable for ForgeOps deployments. When you create a project plan, you'll need to determine which Google Cloud roles are required.

Kubernetes cluster creation

ForgeOps provides Terraform artifacts for GKE cluster creation. Use them to create a cluster that supports ForgeOps deployments. After performing a ForgeOps deployment, you can use your cluster as a sandbox to explore Ping Identity Platform customization.

When you create a project plan, you'll need to identify your organization's preferred infrastructure-as-code solution, and, if necessary, create your own cluster creation automation scripts.

Here are the steps the ForgeOps team follows to create a Kubernetes cluster on GKE:

1. Copy the file that contains default Terraform variables to a new file:
    1. Change to the /path/to/forgeops-extras/terraform directory.
    2. Copy the terraform.tfvars file to override.auto.tfvars [2].

    Copying the terraform.tfvars file to a new file preserves the original content in the file.
2. Determine the deployment size: small, medium, or large.

3. Define your cluster's configuration:

    1. Open the override.auto.tfvars file.
    2. Determine the location of your cluster's configuration in the override.auto.tfvars file:

Cluster size	Section containing the cluster configuration
Small	`cluster.tf_cluster_gke_small`
Medium	`cluster.tf_cluster_gke_medium`
Large	`cluster.tf_cluster_gke_large`

    3. Modify your cluster's configuration by setting values in the section listed in the table:
        1. Set the value of the `enabled` variable to `true`.
        2. Set the value of the `auth.project_id` variable to your new Google Cloud project. Specify the project ID, not the project name.
        3. Set the value of the `meta.cluster_name` variable to the name of the GKE cluster you'll create.
        4. Set the values of the `location.region` and `location.zones` variables to the region and zones where perform your ForgeOps deployment.

            Before continuing, go to Google's Regions and Zones page and verify that the zones you have specified are available in your region you specified.
    4. Save and close the override.auto.tfvars file.

4. Ensure your region has an adequate CPU quota for a ForgeOps deployment.

    Locate these two variables in your cluster's configuration in the override.auto.tfvars file:

    - `node_pool.type`: the machine type to be used in your cluster
    - `node_pool.max_count`: the maximum number of machines to be used in your cluster

    Your quotas must be large enough to let you allocate the maximum number of machines in your region. If your quotas are too low, request and wait for a quota increase from Google Cloud before attempting to create your cluster.

5. Create a cluster using Terraform artifacts in the `forgeops-extras` repository:

    1. Change to the directory that contains Terraform artifacts:

```
$ cd /path/to/forgeops-extras/terraform
```

    2. Run the tf-apply script to create your cluster:

```
$ ./tf-apply
```

Respond `yes` to the `Do you want to perform these actions?` prompt.

When the tf-apply script finishes, it issues a message that provides the path to a kubeconfig file for the cluster.

The script creates:

- The GKE cluster
- The `fast` storage class
- The `ds-snapshot-class` volume snapshot class

The script deploys:

- An ingress controller
- Certificate manager

6. Set your Kubernetes context to reference the new cluster by setting the `KUBECONFIG` environment variable as shown in the message from the tf-apply command's output.
7. To verify that the tf-apply script created the cluster, log in to the Google Cloud console. Select the Kubernetes Engine option. The new cluster should appear in the list of Kubernetes clusters.

Hostname resolution

Set up hostname resolution for the Ping Identity Platform servers you'll deploy in your namespace:

1. Get the ingress controller's external IP address:

```
$ kubectl get services --namespace ingress-nginx
NAME                                 TYPE           CLUSTER-IP   EXTERNAL-IP
PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.4.6.154   35.203.145.112   80:30300/TCP,
443:30638/TCP   58s
ingress-nginx-controller-admission   ClusterIP      10.4.4.9     <none>           443/
TCP                      58s
```

   The ingress controller's IP address should appear in the `EXTERNAL-IP` column. There can be a short delay while the ingress starts before the IP address appears in the `kubectl get services` command's output; you might need to run the command several times.

2. Configure hostname resolution for the ingress controller:
   1. Choose an FQDN (referred to as the *deployment FQDN*) that you'll use when you deploy the Ping Identity Platform, and when you access its GUIs and REST APIs.

      Some examples in this documentation use `forgeops.example.com` as the deployment FQDN. You are not required to use `forgeops.example.com`; you can specify any FQDN you like.

   2. If DNS doesn't resolve your deployment FQDN, add an entry to the /etc/hosts file that maps the ingress controller's external IP address to the deployment FQDN. For example:

```
35.203.145.112 forgeops.example.com
```

1. The Linux version of Homebrew doesn't support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew to install software in several cases. You'll need to refer to the software's documentation for information about how to install the software on a Linux system.
2. The Terraform configuration contains a set of variables under `forgerock` that adds labels required for clusters created by Ping Identity employees. If you're a Ping Identity employee creating a cluster, set values for these variables.

AWS

Before you can perform a ForgeOps deployment on a Kubernetes cluster running on AWS, you must complete these prerequisite tasks:

- Clone the `forgeops` and `forgeops-extras` repositories
- Install third-party software on your local computer
- Start a virtual machine that runs Docker engine on your local computer
- Set up your AWS environment to meet the requirements for ForgeOps deployments
- Create a Kubernetes cluster in AWS
- Set up your local computer to access the cluster's ingress controller

`forgeops` and `forgeops-extras` repositories

> **Note**
>
> Learn about how to configure GitHub notifications here so you can get notified on ForgeOps releases.

Get the `forgeops` and `forgeops-extras` repositories:

1. Clone the repositories. For example:

   ```
   $ git clone https://github.com/ForgeRock/forgeops.git
   $ git clone https://github.com/ForgeRock/forgeops-extras.git
   ```

   Both repositories are public; you do not need credentials to clone them.

2. Check out the `forgeops` repository's `2025.1.2` tag:

   ```
   $ cd /path/to/forgeops
   $ git checkout 2025.1.2
   ```

   Depending on your organization's repository strategy, you might need to clone the repository from a fork. You might also need to create a working branch from the `2025.1.2` tag of your fork. Learn more about Repository Updates here.

3. Check out the `forgeops-extras` repository's `main` branch:

```
$ cd /path/to/forgeops-extras
$ git checkout main
```

Third-party software

Before performing a ForgeOps deployment, obtain third-party software and install it on your local computer.

ForgeOps team recommends that you install third-party software using Homebrew on macOS and Linux[1].

The versions listed in the following table have been validated for ForgeOps deployments on Amazon Web Services. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
On all platforms		
• Python 3	3.13.2	`python@3.13`
• Bash	5.2.37	`bash`
• Docker client	28.0.1	`docker`
• Kubernetes client (kubectl)	1.32.3	`kubernetes-cli`
• Kubernetes context switcher (kubectx)	0.9.5	`kubectx`
• Kustomize	5.6.0	`kustomize`
• Helm	3.17.2	`helm`
• JSON processor jq	1.7.1	`jq`
• Six (Python compatibility library)	1.17.0	`six`

Software	Version	Homebrew package
• Setup tools (Python)	75.6.0	`python-setuptools`
• Terraform	1.5.7	`terraform`
Additionally on Amazon EKS		
• Amazon AWS Command Line Interface	2.25.12	`awscli`
• AWS IAM Authenticator for Kubernetes	0.6.28	`aws-iam-authenticator`

Python venv

The new `forgeops` utility is built on Python3. Some of the Python3 packages used by `forgeops` have to be installed using `pip`. To separate such Python3 specific packages, Python recommends the use of the `venv` Python virtual environment. Learn more about Python `venv` in venv - Virtual environments.

1. Create a `venv` for using the `forgeops` utility.

```
$ python3 -m venv .venv
```

2. Set up Python3 dependencies for `forgeops` utility.

```
$ source .venv/bin/activate
$ /path/to/forgeops/bin/forgeops configure
```

Docker engine

In addition to the software listed in the preceding table, you'll need to start a virtual machine that runs Docker engine.

- On macOS systems, use Docker Desktop or an alternative, such as Colima.
- On Linux systems, use Docker Desktop for Linux, install Docker machine from your Linux distribution, or use an alternative, such as Colima.

For more information about using Colima when performing ForgeOps deployments, refer to this article.

The default configuration for a Docker virtual machine provides adequate resources for a ForgeOps deployment.

For users running Microsoft Windows

ForgeOps deployments are supported on macOS and Linux. If you have a Windows computer, you'll need to create a Linux VM. We tested the following configurations:

- Hypervisor: Hyper-V, VMWare Player, or VMWare Workstation
- Guest OS: Current Ubuntu LTS release with 12 GB memory and 60 GB disk space
- Nested virtualization enabled in the Linux VM.

Perform all the procedures in this documentation within the Linux VM. In this documentation, the local computer refers to the Linux VM for Windows users.

> **Important**
>
> The Minikube implementation on Windows Subsystem for Linux (WSL2) has networking issues. As a result, consistent access to the ingress controller or the apps deployed on Minikube is not possible. This issue is tracked here. Do not attempt to perform ForgeOps deployments on WSL2 until this issue is resolved.

Setup for AWS

Perform these steps to set up an AWS environment that meets the requirements for ForgeOps deployments:

1. Create and configure an IAM group:
    1. Create a group with the name `forgeops-users`.
    2. Attach the following AWS preconfigured policies to the `forgeops-users` group:
        - `IAMUserChangePassword`
        - `IAMReadOnlyAccess`
        - `AmazonEC2FullAccess`
        - `AmazonEC2ContainerRegistryFullAccess`
        - `AWSCloudFormationFullAccess`
    3. Create two policies in the IAM service of your AWS account:
        1. Create the `EksAllAccess` policy using the `eks-all-access.json` file in the /path/to/forgeops/etc/aws-example-iam-policies directory.
        2. Create the `IamLimitedAccess` policy using the `iam-limited-access.json` file in the /path/to/forgeops/etc/aws-example-iam-policies directory.
    4. Attach the policies you created to the `forgeops-users` group.

        Remember, a ForgeOps deployment is a reference implementation, and is not for production use. The policies you create in this procedure are suitable for ForgeOps deployments. When you create a project plan, you'll need to determine how to configure AWS permissions.
    5. Assign one or more AWS users who will perform ForgeOps deployments to the `forgeops-users` group.

2. If you haven't already done so, set up your aws command-line interface environment using the aws configure command.
3. Verify that your AWS user is a member of the `forgeops-users` group:

```
$ aws iam list-groups-for-user --user-name my-user-name --output json
{
    "Groups": [
        {
            "Path": "/",
            "GroupName": "forgeops-users",
            "GroupId": "ABCDEFGHIJKLMNOPQRST",
            "Arn": "arn:aws:iam::048497731163:group/forgeops-users",
            "CreateDate": "2020-03-11T21:03:17+00:00"
        }
    ]
}
```

4. Verify that you are using the correct user profile:

```
$ aws iam get-user
{
    "User": {
        "Path": "/",
        "UserName": "my-user-name",
        "UserId": "...",
        "Arn": "arn:aws:iam::01...3:user/my-user-name",
        "CreateDate": "2020-09-17T16:01:46+00:00",
        "PasswordLastUsed": "2021-05-10T17:07:53+00:00"
    }
}
```

Kubernetes cluster creation

ForgeOps provides Terraform artifacts for Amazon EKS cluster creation. Use them to create a cluster that supports ForgeOps deployments. After performing a ForgeOps deployment, you can use your cluster as a sandbox to explore Ping Identity Platform customization.

When you create a project plan, you'll need to identify your organization's preferred infrastructure-as-code solution, and, if necessary, create your own cluster creation automation scripts.

Here are the steps the ForgeOps team follows to create a Kubernetes cluster on Amazon EKS:

1. Copy the file that contains default Terraform variables to a new file:
    1. Change to the /path/to/forgeops-extras/terraform directory.
    2. Copy the terraform.tfvars file to override.auto.tfvars [2].

    Copying the terraform.tfvars file to a new file preserves the original content in the file.
2. Determine the cluster size: small, medium, or large.

3. Define your cluster's configuration:
    1. Open the override.auto.tfvars file.
    2. Determine the location of your cluster's configuration in the override.auto.tfvars file:

Cluster size	Section containing the cluster configuration
Small	`cluster.tf_cluster_eks_small`
Medium	`cluster.tf_cluster_eks_medium`
Large	`cluster.tf_cluster_eks_large`

    3. Modify your cluster's configuration by setting values in the section listed in the table:
        1. Modify your cluster's configuration by setting values in the section listed in the table:
        2. Set the value of the `enabled` variable to `true`.
        3. Set the value of the `meta.cluster_name` variable to the name of the Amazon EKS cluster you'll create.
        4. Set the values of the `location.region` and `location.zones` variables to the region and zones where you'll perform the ForgeOps deployment.

           Before continuing:

           - Go to the Amazon Elastic Kubernetes Service endpoints and quotas page and verify the region you're specifying supports Amazon EKS.
           - Run the aws ec2 describe-availability-zones --region region-name command to identify three availability zones in your AWS region.
    4. Save and close the override.auto.tfvars file.
4. Ensure your region has an adequate CPU quota for a ForgeOps deployment.

   Locate these two variables in your cluster's configuration in the override.auto.tfvars file:

   - `node_pool.type`: the machine type to be used in your cluster
   - `node_pool.max_count`: the maximum number of machines to be used in your cluster

   Your quotas must be large enough to let you allocate the maximum number of machines in your region. If your quotas are too low, request and wait for a quota increase from Amazon Web Services before attempting to create your cluster.
5. Create a cluster using Terraform artifacts in the `forgeops-extras` repository:
    1. Change to the directory that contains Terraform artifacts:

```
$ cd /path/to/forgeops-extras/terraform
```

    2. Run the tf-apply script to create your cluster:

```
$ ./tf-apply
```

Respond `yes` to the `Do you want to perform these actions?` prompt.

When the tf-apply script finishes, it issues a message that provides the path to a kubeconfig file for the cluster.

The script creates:

- The EKS cluster
- The `fast` storage class
- The `ds-snapshot-class` volume snapshot class

The script deploys:

- An ingress controller
- Certificate manager

6. Set your Kubernetes context to reference the new cluster by setting the `KUBECONFIG` environment variable as shown in the message from the tf-apply command's output.
7. To verify the tf-apply script created the cluster, log in to the AWS console. Access the console panel for the Amazon Elastic Kubernetes Service, and then list the EKS clusters. The new cluster should appear in the list of Kubernetes clusters.

Hostname resolution

Set up hostname resolution for the Ping Identity Platform servers you'll deploy in your namespace:

1. Get the ingress controller's FQDN from the `EXTERNAL-IP` column of the kubectl get services command output:

```
$ kubectl get services --namespace ingress-nginx
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-
IP                                    PORT(S)                       AGE
ingress-nginx-controller             LoadBalancer   10.100.43.88   k8s-ingress ...elb.us-
east-1.amazonaws.com   80:30005/TCP,443:30770/TCP   62s
ingress-nginx-controller-admission   ClusterIP      10.100.2.215
<none>                                          443/TCP                      62s
```

2. Run the host command to get the ingress controller's external IP addresses. For example:

```
$ host k8s-ingress ...elb.us-east-1.amazonaws.com
k8s-ingress ...elb.us-east-1.amazonaws.com has address 3.210.123.210
k8s-ingress ...elb.us-east-1.amazonaws.com has address 3.208.207.77
k8s-ingress ...elb.us-east-1.amazonaws.com has address 44.197.104.140
```

Depending on the state of the cluster, between one and three IP addresses appear in the host command's output.

3. Configure hostname resolution for the ingress controller:

    1. Choose an FQDN (referred to as the *deployment FQDN*) that you'll use when you deploy the Ping Identity Platform, and when you access its GUIs and REST APIs.

       Some examples in this documentation use `forgeops.example.com` as the deployment FQDN. You are not required to use `forgeops.example.com`; you can specify any FQDN you like.

    2. If DNS doesn't resolve your deployment FQDN, add an entry to the /etc/hosts file that maps the ingress controller's external IP address to the deployment FQDN. For example:

       ```
       3.210.123.210 forgeops.example.com
       ```

1. The Linux version of Homebrew doesn't support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew to install software in several cases. You'll need to refer to the software's documentation for information about how to install the software on a Linux system.
2. The Terraform configuration contains a set of variables under `forgerock` that adds labels required for clusters created by Ping Identity employees. If you're a Ping Identity employee creating a cluster, set values for these variables.

Azure

Before you can perform a ForgeOps deployment on a Kubernetes cluster running on Azure], you must complete these prerequisite tasks:

- Clone the `forgeops` and `forgeops-extras` repositories
- Install third-party software on your local computer
- Start a virtual machine that runs Docker engine on your local computer
- Set up an Azure subscription that meets the requirements for ForgeOps deployments
- Create a Kubernetes cluster in the subscription
- Set up your local computer to access the cluster's ingress controller

`forgeops` and `forgeops-extras` repositories

> **Note**
>
> Learn about how to configure GitHub notifications here so you can get notified on ForgeOps releases.

Get the `forgeops` and `forgeops-extras` repositories:

1. Clone the repositories. For example:

   ```
   $ git clone https://github.com/ForgeRock/forgeops.git
   $ git clone https://github.com/ForgeRock/forgeops-extras.git
   ```

   Both repositories are public; you do not need credentials to clone them.

2. Check out the `forgeops` repository's `2025.1.2` tag:

```
$ cd /path/to/forgeops
$ git checkout 2025.1.2
```

   Depending on your organization's repository strategy, you might need to clone the repository from a fork. You might also need to create a working branch from the `2025.1.2` tag of your fork. Learn more about Repository Updates here.

3. Check out the `forgeops-extras` repository's `main` branch:

```
$ cd /path/to/forgeops-extras
$ git checkout main
```

Third-party software

Before performing a ForgeOps deployment, obtain third-party software and install it on your local computer.

ForgeOps team recommends that you install third-party software using Homebrew on macOS and Linux'[1]' .

The versions listed in the following table have been validated for ForgeOps deployments on Microsoft Azure. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
On all platforms		
• Python 3	3.13.2	`python@3.13`
• Bash	5.2.37	`bash`
• Docker client	28.0.1	`docker`
• Kubernetes client (kubectl)	1.32.3	`kubernetes-cli`
• Kubernetes context switcher (kubectx)	0.9.5	`kubectx`
• Kustomize	5.6.0	`kustomize`

Software	Version	Homebrew package
• Helm	3.17.2	`helm`
• JSON processor jq	1.7.1	`jq`
• Six (Python compatibility library)	1.17.0	`six`
• Setup tools (Python)	75.6.0	`python-setuptools`
• Terraform	1.5.7	`terraform`
Additionally on Azure AKS		
• Azure Command Line Interface	2.71.0	`azure-cli`

Python `venv`

The new `forgeops` utility is built on Python3. Some of the Python3 packages used by `forgeops` have to be installed using `pip`. To separate such Python3 specific packages, Python recommends the use of the `venv` Python virtual environment. Learn more about Python `venv` in venv - Virtual environments.

1. Create a `venv` for using the `forgeops` utility.

```
$ python3 -m venv .venv
```

2. Set up Python3 dependencies for `forgeops` utility.

```
$ source .venv/bin/activate
$ /path/to/forgeops/bin/forgeops configure
```

Docker engine

In addition to the software listed in the preceding table, you'll need to start a virtual machine that runs Docker engine.

- On macOS systems, use Docker Desktop or an alternative, such as Colima.
- On Linux systems, use Docker Desktop for Linux, install Docker machine from your Linux distribution, or use an alternative, such as Colima.

For more information about using Colima when performing ForgeOps deployments, refer to this article.

The default configuration for a Docker virtual machine provides adequate resources for a ForgeOps deployment.

For users running Microsoft Windows

ForgeOps deployments are supported on macOS and Linux. If you have a Windows computer, you'll need to create a Linux VM. We tested the following configurations:

- Hypervisor: Hyper-V, VMWare Player, or VMWare Workstation
- Guest OS: Current Ubuntu LTS release with 12 GB memory and 60 GB disk space
- Nested virtualization enabled in the Linux VM.

Perform all the procedures in this documentation within the Linux VM. In this documentation, the local computer refers to the Linux VM for Windows users.

> **Important**
>
> The Minikube implementation on Windows Subsystem for Linux (WSL2) has networking issues. As a result, consistent access to the ingress controller or the apps deployed on Minikube is not possible. This issue is tracked here. Do not attempt to perform ForgeOps deployments on WSL2 until this issue is resolved.

Azure subscription setup

Perform these steps to set up an Azure subscription that meets the requirements for ForgeOps deployments:

1. Assign the following roles to users who will perform ForgeOps deployments:
   - Azure Kubernetes Service Cluster Admin Role
   - Azure Kubernetes Service Cluster User Role
   - Contributor
   - User Access Administrator

   Remember, a ForgeOps deployment is a reference implementation, and is not for production use. The roles you assign in this step are suitable for ForgeOps deployments. When you create a project plan, you'll need to determine which Azure roles are required.

2. Log in to Azure services as a user with the roles you assigned in the previous step:

   ```
   $ az login --username my-user-name
   ```

3. View your current subscription ID:

   ```
   $ az account show
   ```

4. If necessary, set the current subscription ID to the one you will use to perform the ForgeOps deployment:

```
$ az account set --subscription my-subscription-id
```

Kubernetes cluster creation

ForgeOps team provides Terraform artifacts for AKS cluster creation. Use them to create a cluster that supports ForgeOps deployments. After performing a ForgeOps deployment, you can use your cluster as a sandbox to explore Ping Identity Platform customization.

When you create a project plan, you'll need to identify your organization's preferred infrastructure-as-code solution, and, if necessary, create your own cluster creation automation scripts.

Here are the steps the ForgeOps team follows to create a Kubernetes cluster on AKS:

1. Copy the file that contains default Terraform variables to a new file:
    1. Change to the /path/to/forgeops-extras/terraform directory.
    2. Copy the terraform.tfvars file to override.auto.tfvars [2].

    Copying the terraform.tfvars file to a new file preserves the original content in the file.
2. Determine the cluster size: small, medium, or large.
3. Define your cluster's configuration:
    1. Open the override.auto.tfvars file.
    2. Determine the location of your cluster's configuration in the override.auto.tfvars file:

Cluster size	Section containing the cluster configuration
Small	`cluster.tf_cluster_aks_small`
Medium	`cluster.tf_cluster_aks_medium`
Large	`cluster.tf_cluster_aks_large`

    3. Modify your cluster's configuration by setting values in the section listed in the table:
        1. Set the value of the `enabled` variable to `true`.
        2. Set the value of the `meta.cluster_name` variable to the name of the AKS cluster you'll create.
        3. Set the values of the `location.region` and `location.zones` variables to the region and zones where you'll perform the ForgeOps deployment.

            Before continuing, go to Microsoft's Products available by region page and verify that Azure Kubernetes Service is available in the region you specified.
    4. Save and close the override.auto.tfvars file.
4. Ensure your region has an adequate CPU quota for a ForgeOps deployment.

Locate these two variables in your cluster's configuration in the override.auto.tfvars file:

- `node_pool.type`: the machine type to be used in your cluster
- `node_pool.max_count`: the maximum number of machines to be used in your cluster

Your quotas must be large enough to let you allocate the maximum number of machines in your region. If your quotas are too low, request and wait for a quota increase from Microsoft Azure before attempting to create your cluster.

5. Create a cluster using Terraform artifacts in the `forgeops-extras` repository:
    1. Change to the directory that contains Terraform artifacts:

        ```
        $ cd /path/to/forgeops-extras/terraform
        ```

    2. Run the tf-apply script to create your cluster:

        ```
        $ ./tf-apply
        ```

        Respond `yes` to the `Do you want to perform these actions?` prompt.

        When the tf-apply script finishes, it issues a message that provides the path to a kubeconfig file for the cluster.

        The script creates:

        - The AKS cluster
        - The `fast` storage class
        - The `ds-snapshot-class` volume snapshot class

        The script deploys:

        - An ingress controller
        - Certificate manager

6. Set your Kubernetes context to reference the new cluster by setting the `KUBECONFIG` environment variable as shown in the message from the tf-apply command's output.
7. To verify that the tf-apply script created the cluster, log in to the Azure portal. Search for Kubernetes services and access the Kubernetes services page. The new cluster should appear in the list of Kubernetes clusters.

Hostname resolution

Set up hostname resolution for the Ping Identity Platform servers you'll deploy in your namespace:

1. Get the ingress controller's external IP address:

```
$ kubectl get services --namespace ingress-nginx
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP
PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.0.166.247   20.168.193.68   80:31377/TCP,
443:31099/TCP   74m
ingress-nginx-controller-admission   ClusterIP      10.0.40.40     <none>          443/
TCP                      74m
```

The ingress controller's IP address should appear in the `EXTERNAL-IP` column. There can be a short delay while the ingress starts before the IP address appears in the `kubectl get services` command's output; you might need to run the command several times.

2. Configure hostname resolution for the ingress controller:

    1. Choose an FQDN (referred to as the *deployment FQDN*) that you'll use when you deploy the Ping Identity Platform, and when you access its GUIs and REST APIs.

        Some examples in this documentation use `forgeops.example.com` as the deployment FQDN. You are not required to use `forgeops.example.com`; you can specify any FQDN you like.

    2. If DNS doesn't resolve your deployment FQDN, add an entry to the /etc/hosts file that maps the ingress controller's external IP address to the deployment FQDN. For example:

        ```
        20.168.193.68 forgeops.example.com
        ```

1. The Linux version of Homebrew doesn't support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew to install software in several cases. You'll need to refer to the software's documentation for information about how to install the software on a Linux system.
2. The Terraform configuration contains a set of variables under `forgerock` that adds labels required for clusters created by Ping Identity employees. If you're a Ping Identity employee creating a cluster, set values for these variables.

Minikube

Before you can perform a ForgeOps deployment on a Kubernetes cluster running on Minikube, you must complete these prerequisite tasks:

- Clone the `forgeops` repository
- Install third-party software on your local computer
- Start a virtual machine that runs Docker engine on your local computer
- Create a Kubernetes cluster on Minikube
- Set up your local computer to access the cluster's ingress controller

`forgeops` repository

> **Note**
>
> Learn about how to configure GitHub notifications here so you can get notified on ForgeOps releases.

Before you can perform a ForgeOps deployment, you must first get the `forgeops` repository and check out the `2025.1.2` tag you want to use:

1. Clone the `forgeops` repository. For example:

   ```
   $ git clone https://github.com/ForgeRock/forgeops.git
   ```

   The `forgeops` repository is a public Git repository. You do not need credentials to clone it.

2. Check out the `2025.1.2` tag:

   ```
   $ cd forgeops
   $ git checkout 2025.1.2
   ```

Depending on your organization's repository strategy, you might need to clone the repository from a fork. You might also need to create a working branch from the `2025.1.2` tag. Learn more in Repository Updates.

Third-party software

Before performing a ForgeOps deployment, obtain third-party software and install it on your local computer.

ForgeOps team recommends that you install third-party software using Homebrew on macOS and Linux[1].

The versions listed in this section have been validated for ForgeOps deployments on Minikube. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Software	Version	Homebrew package
On all platforms		
• Python 3	3.13.2	`python@3.13`
• Bash	5.2.37	`bash`
• Docker client	28.0.1	`docker`
• Kubernetes client (kubectl)	1.32.3	`kubernetes-cli`

Software	Version	Homebrew package
• Kubernetes context switcher (kubectx)	0.9.5	`kubectx`
• Kustomize	5.6.0	`kustomize`
• Helm	3.17.2	`helm`
• JSON processor jq	1.7.1	`jq`
• Six (Python compatibility library)	1.17.0	`six`
• Setup tools (Python)	75.6.0	`python-setuptools`
Additionally on Minikube		
• Minikube	1.35.0	`minikube`
• PyYaml	6.0.1	`pyyaml`

Python venv

The new `forgeops` utility is built on Python3. Some of the Python3 packages used by `forgeops` have to be installed using `pip`. To separate such Python3 specific packages, Python recommends the use of the `venv` Python virtual environment. Learn more about Python `venv` in venv - Virtual environments.

1. Create a `venv` for using the `forgeops` utility.

```
$ python3 -m venv .venv
```

2. Set up Python3 dependencies for `forgeops` utility.

```
$ source .venv/bin/activate
$ /path/to/forgeops/bin/forgeops configure
```

Docker engine

In addition to the software listed in the preceding table, you'll need to start a virtual machine that runs Docker engine.

- On macOS systems, use Docker Desktop or an alternative, such as Colima.
- On Linux systems, use Docker Desktop for Linux, install Docker machine from your Linux distribution, or use an alternative, such as Colima.

For more information about using Colima when performing ForgeOps deployments, refer to this article.

Minimum requirements for the virtual machine:

- 4 CPUs
- 10 GB RAM
- 60 GB disk space

For users running Microsoft Windows

ForgeOps deployments are supported on macOS and Linux. If you have a Windows computer, you'll need to create a Linux VM. We tested the following configurations:

- Hypervisor: Hyper-V, VMWare Player, or VMWare Workstation
- Guest OS: Current Ubuntu LTS release with 12 GB memory and 60 GB disk space
- Nested virtualization enabled in the Linux VM.

Perform all the procedures in this documentation within the Linux VM. In this documentation, the local computer refers to the Linux VM for Windows users.

> **Important**
>
> The Minikube implementation on Windows Subsystem for Linux (WSL2) has networking issues. As a result, consistent access to the ingress controller or the apps deployed on Minikube is not possible. This issue is tracked here. Do not attempt to perform ForgeOps deployments on WSL2 until this issue is resolved.

Minikube cluster

Minikube software runs a single-node Kubernetes cluster in a virtual machine.

The minikube start command example shown in the doc creates a Minikube cluster with a configuration that's adequate for a ForgeOps deployment.

The default driver option is fine for most users. For more information about Minikube virtual machine drivers, refer to Drivers in the Minikube documentation.

If you want to use a driver other than the default driver, specify the `--driver` option when you run the minikube start command in the next step.

1. Set up Minikube:

```
$ minikube start --cpus=3 --memory=9g --disk-size=40g --cni=true \
  --kubernetes-version=stable --addons=ingress,volumesnapshots,metrics-server \
  --driver=docker
😄  minikube v1.34.0 on Darwin 15.3.1
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.45 …
🔥  Creating docker container (CPUs=3, Memory=9216MB) …
   Preparing Kubernetes v1.31.0 on Docker 27.2.0 …
  ▪ Generating certificates and keys …
  ▪ Booting up control plane …
  ▪ Configuring RBAC rules …
🔗  Configuring CNI (Container Networking Interface) …
   Verifying Kubernetes components…
  ▪ Using image registry.k8s.io/metrics-server/metrics-server:v0.7.2
  ▪ Using image registry.k8s.io/sig-storage/snapshot-controller:v6.1.0
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
💡  After the addon is enabled, please run "minikube tunnel" and your ingress resources would be
available at "127.0.0.1"
  ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.3
  ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.3
  ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.11.2
   Verifying ingress addon…
   Enabled addons: storage-provisioner, default-storageclass, metrics-server, volumesnapshots,
ingress
   Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

> **Tip**
>
> If you are running Minikube on an ARM-based macOS system and the minikube output indicates that you are using the qemu driver, you probably did not start the virtual machine that runs your Docker engine.

2. Run the docker-env command to set up your local computer to use the Minikube's Docker engine:

```
$ eval $(minikube docker-env)
```

Hostname resolution

Set up hostname resolution for the Ping Identity Platform servers you'll deploy in your namespace:

1. Determine the Minikube ingress controller's IP address.
   - If Minikube is using the Docker driver on macOS system[2], use `127.0.0.1` as the ingress IP address.
   - If Minikube is running the Hyperkit driver on Intel-based macOS system or on a Linux system, get the IP address by running the minikube ip command:

     ```
     $ minikube ip
     ...
     ```

2. Choose an FQDN (referred to as the *deployment FQDN*) that you'll use when you deploy the Ping Identity Platform, and when you access its GUIs and REST APIs. Ensure that the FQDN is unique in the cluster you will be deploying the Ping Identity Platform.

   Some examples in this documentation use `forgeops.example.com` as the deployment FQDN. You are not required to use `forgeops.example.com`; you can specify any FQDN you like.

3. Add an entry to the /etc/hosts file to resolve the deployment FQDN:

   ```
   ingress-ip-address forgeops.example.com
   ```

   For `ingress-ip-address`, specify the IP address from step 1. For example:

   ```
   127.0.0.1 forgeops.example.com
   ```

1. The Linux version of Homebrew doesn't support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew to install software in several cases. You'll need to refer to the software's documentation for information about how to install the software on a Linux system.
2. For example, systems based on M1 or M2 chipsets.

Deploy ForgeOps

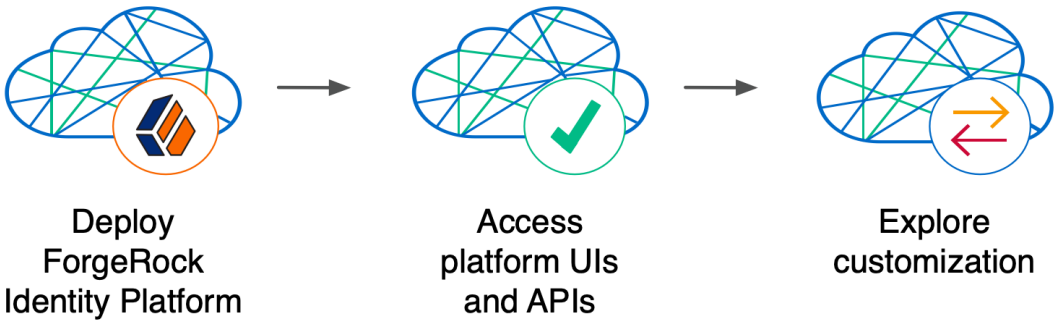
Deployment overview

A *ForgeOps deployment* is a deployment of the Ping Identity Platform on Kubernetes based on Docker images, Helm charts, Kustomize bases and overlays, utility programs, and other artifacts you can find in the `forgeops` repository on GitHub.

You can get a ForgeOps deployment up and running on Kubernetes quickly. After performing a ForgeOps deployment, you can use it to explore how you might configure a Kubernetes cluster before you deploy the platform in production.

A ForgeOps deployment is a robust sample deployment for demonstration and exploration purposes only. *It is not a production deployment*.

This section describes how to perform a ForgeOps deployment in a Kubernetes cluster and then access the platform's GUIs and REST APIs. When you're done, you can use ForgeOps deployment to explore deployment customizations.



Performing a ForgeOps deployment is a good learning and exploration exercise that helps prepare you to put together a project plan for deploying the platform in production. To better understand how this activity fits in to the overall deployment process, refer to Performing a ForgeOps deployment.

Using the ForgeOps artifacts and this documentation, you can quickly get the Ping Identity Platform running in a Kubernetes environment. You begin to familiarize yourself with some of the steps you'll need to perform when deploying the platform in the cloud for production use:

Standardizes the process—The ForgeOps team's mission is to standardize a process for deploying the Ping Identity Platform on Kubernetes. The team is made up of technical consultants and cloud software developers. We've had numerous interactions with our customers and discussed common deployment issues. Based on our interactions, we developed the ForgeOps artifacts to make deployment of the platform easier in the cloud.

Simplifies baseline deployment—We then developed artifacts: Dockerfiles, Kustomize bases and overlays, Helm charts, and utility programs to simplify the deployment process. We deployed small-sized, medium-sized, and large-sized production-quality Kubernetes clusters, and kept them up and running 24x7. We conducted continuous integration and continuous deployment as we added new capabilities and fixed problems in the system. We maintained, benchmarked, and tuned the system for optimized performance. Most importantly, we documented the process so you could replicate it.

Eliminates guesswork—If you use our ForgeOps artifacts and follow the instructions in this documentation without deviation, you can successfully deploy the Ping Identity Platform in the cloud. ForgeOps deployments take the guesswork out of setting up a cloud environment. They bypass the deploy-test-integrate-test-repeat cycle many customers struggle through when spinning up the Ping Identity Platform in the cloud for the first time.

Prepares you to deploy in production—After you've performed a ForgeOps deployment you'll be ready to start working with experts on deploying in production. We strongly recommend that you engage a Ping Identity technical consultant or partner to assist you with deploying the platform in production.

Next step

Become familiar with ForgeOps deployments

- ☐ Understand ForgeOps architecture
- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

ForgeOps architecture

After you perform a ForgeOps deployment, the Ping Identity Platform is fully operational in a Kubernetes cluster. `forgeops` artifacts provide preconfigured JVM settings, memory, CPU limits, and other configurations.

Here are some of the characteristics of ForgeOps deployments:

Cluster and deployment sizes

When you use the Terraform artifacts in the `forgeops-extras` repository to create a Kubernetes cluster on Google Cloud, AWS, or Azure, you specify one of three sizes:

- A small cluster with capacity to handle 1,000,000 test users
- A medium cluster with capacity to handle 10,000,000 test users
- A large cluster with capacity to handle 100,000,000 test users

When you use the minikube start command to create a Kubernetes cluster on Minikube, you don't specify a cluster size.

When you perform a ForgeOps deployment, you specify a deployment size. This deployment size should be the same as your cluster size, except when you perform *single-instance ForgeOps deployments*.

Single-instance deployments are special deployments that you use to configure AM and IDM and build custom Docker images for the Ping Identity Platform. They are called single-instance deployments because unlike small, medium, and large deployments, they have only single pods that run AM and IDM. They are only suitable for developing the AM and IDM configurations and must not be used for testing performance, monitoring, security, and backup requirements in production environments.

You can perform one or more single-instance deployments on small, medium, and large GKE, EKS, and AKS clusters. Each single-instance deployment resides in its own namespace.

You can perform one (and only one) single-instance deployment on a Minikube cluster.

Multi-zone Kubernetes cluster

In small, medium, and large ForgeOps deployments, Ping Identity Platform pods are distributed across three zones for high availability.

(In single-instance deployments, Ping Identity Platform pods reside in a single zone.)

Go here for a diagram that shows the organization of pods in zones and node pools in small, medium, and large ForgeOps deployments.

Third-party deployment and monitoring tools

- Ingress-NGINX Controller for Kubernetes ingress support.
- HAProxy Ingress Controller for Kubernetes ingress support.'[1]'
- Prometheus for monitoring and notifications.'[1]'
- Prometheus Alertmanager for setting and managing alerts.'[1]'
- Grafana for metrics visualization.'[1]'
- Certificate Manager for obtaining and installing security certificates.
- Helm for deploying Helm charts.
- Terraform for creating example clusters.'[1]'

Ready-to-use Ping Identity Platform components

- Multiple DS instances are deployed for higher availability. Separate instances are deployed for Core Token Service (CTS) tokens and identities. The instances for identities also contain AM and IDM run-time data.
- The AM configuration is file-based, stored at the path `/home/forgerock/openam/config` inside the AM Docker container (and in the AM pods).
- Multiple AM instances are deployed for higher availability.'[2]'
- AM instances are configured to access DS data stores.
- Multiple IDM instances are deployed for higher availability.'[2]'
- IDM instances are configured to access DS data stores.

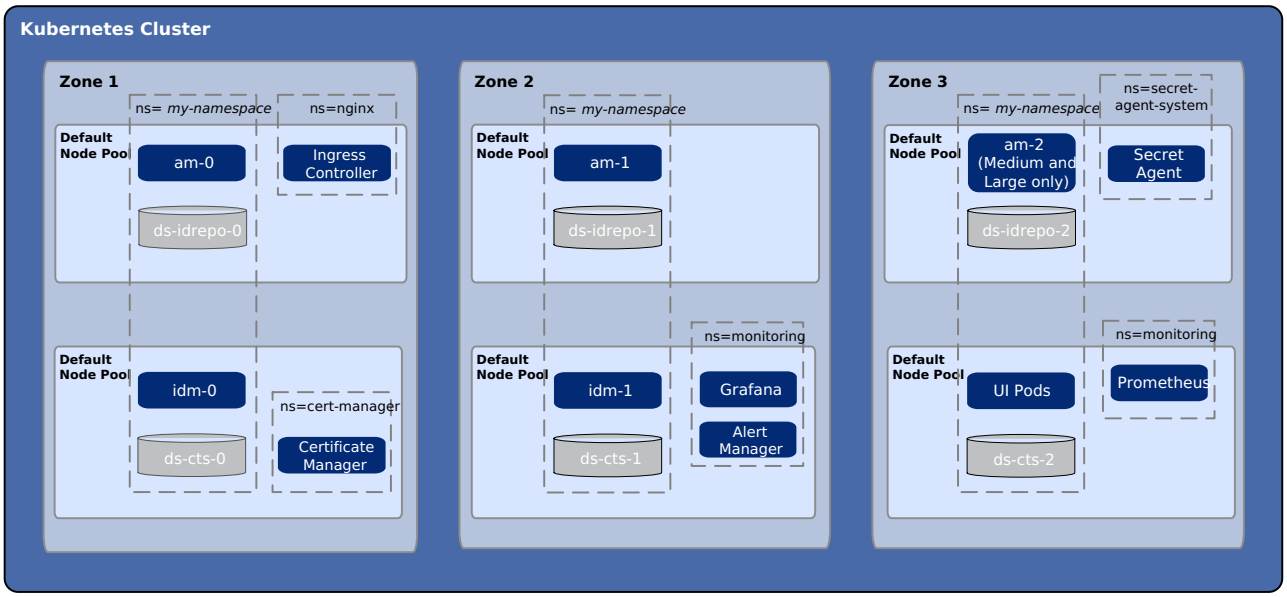
Highly available, distributed deployment[1], [2],

Deployment across three zones ensures that the ingress controller and all Ping Identity Platform components are highly available.

Pods that run DS are configured to use soft anti-affinity. Because of this, Kubernetes schedules DS pods to run on nodes that don't have any other DS pods whenever possible.

The exact placement of all other ForgeOps pods is delegated to Kubernetes.

Pods are organized across three zones in a single node pool with six nodes. Pod placement among the nodes might vary, but the DS pods should run on nodes without any other DS pods.



Ingress controller

The Ingress-NGINX Controller provides load balancing services for ForgeOps deployments. Ingress controller pods run in the `nginx` namespace. Implementation varies by cloud provider.

Optionally, you can deploy HAProxy Ingress as the ingress controller instead of Ingress-NGINX Controller.[1],

Secret generation and management

The open source Secret Agent operator generates Kubernetes secrets for Ping Identity Platform deployments. It also integrates with Google Cloud Secret Manager, AWS Secrets Manager, and Azure Key Vault, providing cloud backup and retrieval for secrets.

Secured communication

The ingress controller is TLS-enabled. TLS is terminated at the ingress controller. Incoming requests and outgoing responses are encrypted.

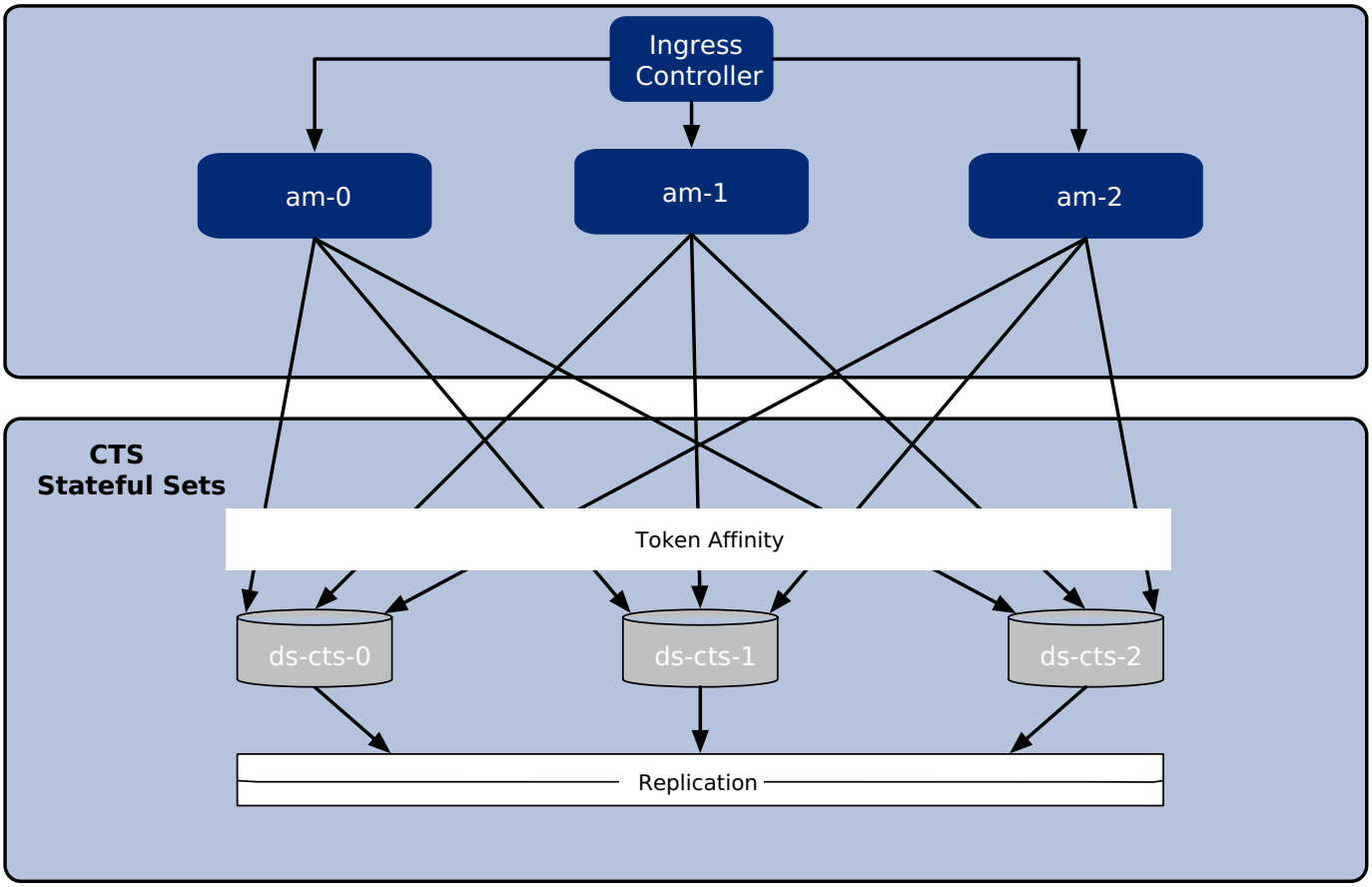
Inbound communication to DS instances occurs over secure LDAP (LDAPS).

For more information, refer to Secure HTTP.

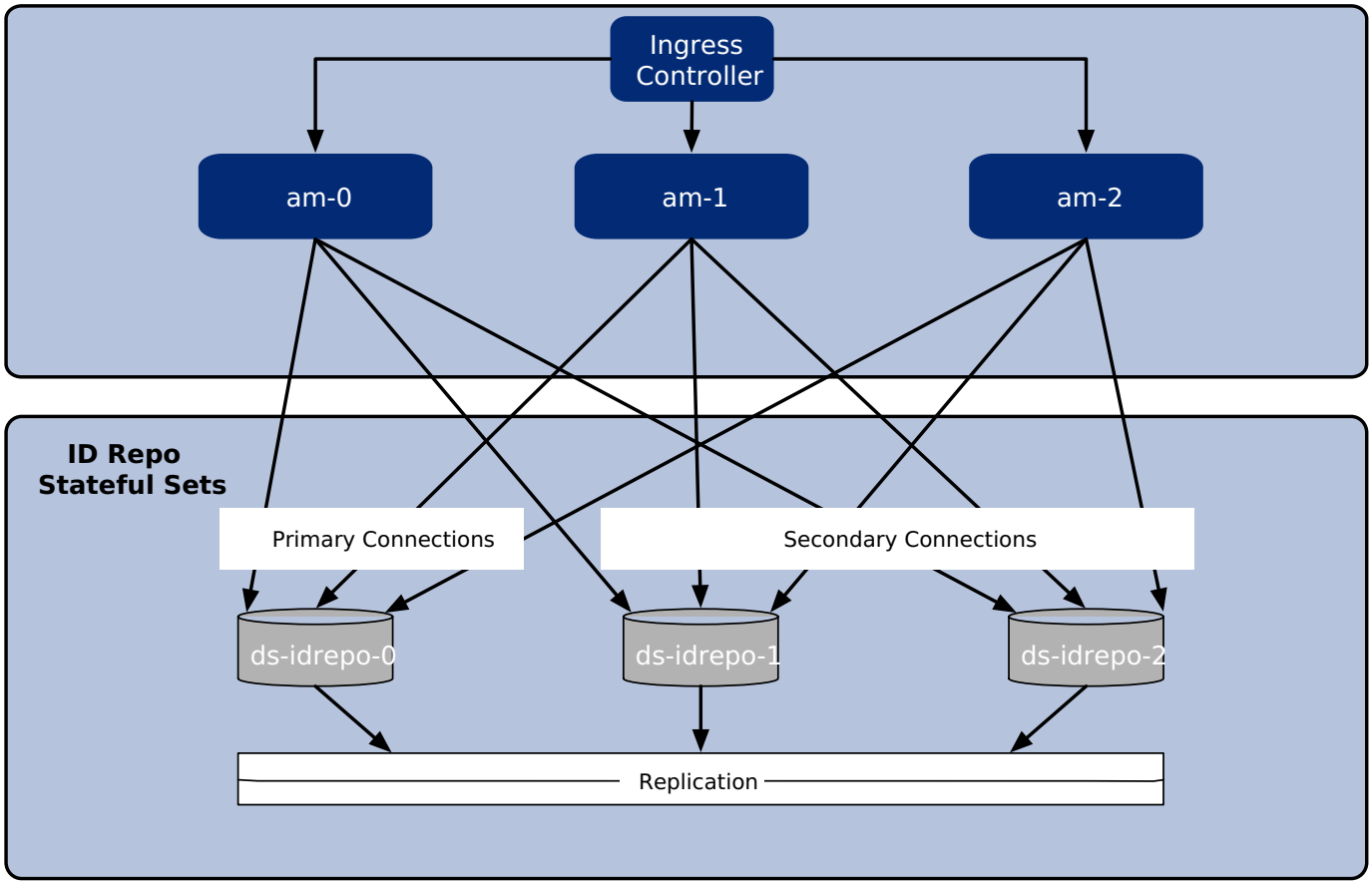
Stateful sets

ForgeOps deployments use Kubernetes stateful sets to manage the DS pods. Stateful sets protect against data loss if Kubernetes client containers fail.

On small-, medium- and large- deployments, CTS data stores are configured for affinity load balancing for optimal performance.



AM policies, application data, and identities reside in the `idrepo` directory service. Small-, medium- and large- deployments use a single `idrepo` master configured to fail over to one of two secondary directory services.



Authentication

IDM is configured to use AM for authentication.

DS replication[2],

All DS instances are configured for full replication of identities and session tokens.

Backup and restore[1],

Backup and restore can be performed using several techniques. You can:

- Use the volume snapshot capability in GKE, EKS, or AKS. The cluster where the ForgeOps deployment resides must be configured with a volume snapshot class before you can take volume snapshots, and persistent volume claims must use a CSI driver that supports volume snapshots.
- Use the ds-backup utility.
- Use a "last mile" backup archival solutions, such as Amazon S3, Google Cloud Storage, and Azure Cloud Storage that is specific to the cloud provider.
- Use a Kubernetes backup and restore product, such as Velero, Kasten K10, TrilioVault, Commvault, or Portworx PX-Backup.

For more information, refer to Backup and restore overview.

Initial data loading

After the first AM instance in a ForgeOps deployment has started, an `amster` job runs. This job loads application data, such as OAuth 2.0 client definitions, to the `idrepo` DS instance.

Next step

Become familiar with ForgeOps deployments

Understand ForgeOps architecture

- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

1. Not available on ForgeOps deployments on Minikube.
2. Not available on single-instance ForgeOps deployments.

ForgeOps deployment

After you set up your deployment environment and your Kubernetes cluster, you're ready to perform a ForgeOps deployment.

First, you'll need to choose a deployment technology.

Deployment technologies

You can perform ForgeOps deployments using either Kustomize or Helm.

The preferred deployment technology for ForgeOps deployments is Helm. If you are not familiar with either of these two technologies, choose Helm.

Choose Kustomize as your deployment technology when:

- You performed ForgeOps deployments before Helm charts were available in the `forgeops` repository, and you want to continue to use Kustomize-based deployments.
- You want to generate Kustomize manifests for the platform, including custom manifests, using the forgeops generate command.
- Kustomize is your organization's preferred deployment technology for Kubernetes.
- Kustomize offers needed features that are not available in Helm.

Deployment scenarios

Follow the steps in one of these scenarios to perform a ForgeOps deployment:

- Deploy using Helm on GKE, EKS, or AKS
- Deploy using Helm on Minikube
- Deploy using Kustomize on GKE, EKS, or AKS

- Deploy using Kustomize on Minikube

Deploy using Helm on GKE, EKS, or AKS

> **Important**
>
> In a development or demo environment, you can use the helm chart available locally in /path/to/forgeops/charts directory for performing ForgeOps deployment. In a production environment, it is highly recommended to use the Helm charts published on the registry.

1. Verify that you have set up your environment and created a Kubernetes cluster as documented in the setup section.
2. Enable the Python3 virtual environment:

```
$ source .venv/bin/activate
```

3. The configuration of a ForgeOps deployment is steered through the use of Kustomize overlays or Helm values. Use the forgeops env command to set up the Kustomize overlays and Helm values files to configure your ForgeOps deployment environment:
   - If you want to use the issuer provided with the platform for demo, then you can use default-issuer.
   - For a clusters on a cloud environment specify the --deployment-size as `--small`, `--medium`, or `--large`.
   - For a single-instance deployment, specify --deployment-size as `--single-instance`.

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn --cluster-issuer my-cluster-issuer --
deployment-size
```

   In the command above, replace my-fqdn, my-cluster-issuer, and --deployment-size with appropriate values from your environment.

   In a Minikube environment, use the single instance deployment. For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn \
  --cluster-issuer my-cluster-issuer --single-instance
```

   Learn more about deployment sizes in Cluster and deployment sizes and about single instances here.

4. (Optional) By default, the latest platform images are used for ForgeOps deployment. If you need a specific image version to be deployed, then ensure that the `image.repository` and `image.tag` settings for the platform components are correct in the /path/to/forgeops/helm/my-env/values.yaml Helm values file.
5. Set up your Kubernetes context:
   1. Set the `KUBECONFIG` environment variable so that your Kubernetes context references the cluster in which you'll perform the ForgeOps deployment.

2. Create a Kubernetes namespace in the cluster for the Ping Identity Platform pods:

```
$ kubectl create namespace my-namespace
```

3. Set the active namespace in your Kubernetes context to the Kubernetes namespace you just created:

```
$ kubens my-namespace
```

6. Set up the certificate management, secret agent, and NGINX:

> **Note**
>
> The `forgeops` repository contains the certmanager-deploy.sh to install `cert-manager` in your cluster. If you need to use a different certificate management utility, you refer to the corresponding documentation for installing that utility.

```
$ cd /path/to/forgeops/charts/scripts
$ ./install-prereqs
```

7. (Optional) If you've set up your Kubernetes cluster using ForgeOps-provided Terraform manifest, then you would've already created the required `fast` storage and volume snapshot classes. If you set up your Kubernetes cluster using your own scripts, then create these classes using the corresponding YAML scripts provided in the /path/to/forgeops/cluster/resources folder.

   For example, on GKE:

```
$ kubectl apply -f /path/to/forgeops/cluster/resources/gke-fast-storage-class.yaml
$ kubectl apply -f /path/to/forgeops/cluster/resources/gke-volume-snapshot-class.yaml
```

8. Run the helm upgrade command to perform a ForgeOps deployment:

```
$ helm upgrade --install identity-platform  identity-platform \
 --repo https://ForgeRock.github.io/forgeops/ \
 --version 2025.1.2 --namespace my-namespace \
 --values /path/to/forgeops/helm/my-env/values.yaml
```

   When deploying the platform with Docker images other than the ForgeOps-provided images, you'll also need to set additional Helm values such as `am.image.repository`, `am.image.tag`, `idm.image.repository`, and `idm.image.tag`. For an example, refer to Redeploy AM: Helm deployments.

> **Important**
>
> Ping Identity only offers its software or services to legal entities that have entered into a binding license agreement with Ping Identity. When you install Docker images provided by ForgeOps, you agree either that: 1) you are an authorized user of a Ping Identity Platform customer that has entered into a license agreement with Ping Identity governing your use of the Ping Identity software; or 2) your use of the Ping Identity Platform software is subject to the Ping Identity Subscription Agreements.

9. Check the status of the pods in the namespace in which you deployed the platform until all the pods are ready:
    1. Run the kubectl get pods command.
    2. Review the output. Deployment is complete when:
        - All entries in the `STATUS` column indicate `Running` or `Completed`.
        - The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
    3. If necessary, continue to query your deployment's status until all the pods are ready.
10. Back up and save the Kubernetes secrets that contain the master and TLS keys:
    1. To avoid accidentally putting the backups under version control, change to a directory that is outside your `forgeops` repository clone.
    2. The `ds-master-keypair` secret contains the DS master key. This key is required to decrypt data from a directory backup. *Failure to save this key could result in data loss.*

       Back up the Kubernetes secret that contains the DS master key:

       ```
       $ kubectl get secret ds-master-keypair -o yaml > master-key-pair.yaml
       ```
    3. The `ds-ssl-keypair` secret contains the DS TLS key. This key is needed for cross-environment replication topologies.

       Back up the Kubernetes secret that contains the DS TLS key pair:

       ```
       $ kubectl get secret ds-ssl-keypair -o yaml > tls-key-pair.yaml
       ```
    4. Save the two backup files.
11. (Optional) Deploy Prometheus, Grafana, and Alertmanager for monitoring and alerting[1]:
    1. Deploy Prometheus, Grafana, and Alertmanager pods in your ForgeOps deployment:

```
$ /path/to/forgeops/bin/prometheus-deploy.sh

**This script requires Helm version 3.04 or later due to changes in the behaviour of 'helm
repo add' command.**

namespace/monitoring created
"stable" has been added to your repositories
"prometheus-community" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "codecentric" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
Release "prometheus-operator" does not exist. Installing it now.
NAME: prometheus-operator
LAST DEPLOYED: ...
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus-operator"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create
& configure Alertmanager and Prometheus instances using the Operator.
...
Release "forgerock-metrics" does not exist. Installing it now.
NAME: forgerock-metrics
LAST DEPLOYED: ...
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

2. Check the status of the pods in the `monitoring` namespace until all the pods are ready:

```
$ kubectl get pods --namespace monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-operator-kube-p-alertmanager-0   2/2     Running   0          119s
prometheus-operator-grafana-95b8f5b7d-nn65h              3/3     Running   0          2m4s
prometheus-operator-kube-p-operator-7d54989595-pdj44     1/1     Running   0          2m4s
prometheus-operator-kube-state-metrics-d95996bc4-wcf7s   1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-67xq4       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-b4grn       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-cwhcn       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-h9brd       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-q8zrk       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-vqpt5       1/1     Running   0          2m4s
prometheus-prometheus-operator-kube-p-prometheus-0       2/2     Running   0          119s
```

12. (Optional) Install a TLS certificate instead of using the default self-signed certificate in your ForgeOps deployment. Refer to TLS certificate for details.

Next step

Become familiar with ForgeOps deployments

Understand ForgeOps architecture

Deploy the platform

☐ Access platform UIs and APIs

☐ Plan for production deployment

---

1. Installing Prometheus, Grafana, and Alertmanager technology in ForgeOps deployments provides an example of how you might set up monitoring and alerting in a Ping Identity Platform deployment in the cloud. Remember, ForgeOps deployments are reference implementations and not for production use. When you create a project plan, you'll need to determine how to monitor and send alerts in your production deployment.

Deploy using Helm on Minikube

> **Important**
>
> In a development or demo environment, you can use the helm chart available locally in /path/to/forgeops/charts directory for performing ForgeOps deployment. In a production environment, it is highly recommended to use the Helm charts published on the registry.

1. Verify that you have set up your environment and created a Kubernetes cluster as documented in the setup section.
2. Enable the Python3 virtual environment:

```
$ source .venv/bin/activate
```

3. The configuration of a ForgeOps deployment is steered through the use of Kustomize overlays or Helm values. Use the forgeops env command to set up the Kustomize overlays and Helm values files to configure your ForgeOps deployment environment:
   - If you want to use the issuer provided with the platform for demo, then you can use default-issuer.
   - For a clusters on a cloud environment specify the --deployment-size as `--small`, `--medium`, or `--large`.
   - For a single-instance deployment, specify --deployment-size as `--single-instance`.

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn --cluster-issuer my-cluster-issuer --
deployment-size
```

   In the command above, replace my-fqdn, my-cluster-issuer, and --deployment-size with appropriate values from your environment.

In a Minikube environment, use the single instance deployment. For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn \
  --cluster-issuer my-cluster-issuer --single-instance
```

Learn more about deployment sizes in Cluster and deployment sizes and about single instances here.

4. (Optional) By default, the latest platform images are used for ForgeOps deployment. If you need a specific image version to be deployed, then ensure that the `image.repository` and `image.tag` settings for the platform components are correct in the /path/to/forgeops/helm/my-env/values.yaml Helm values file.
5. Set up your Kubernetes context:
    1. Create a Kubernetes namespace in the cluster for the Ping Identity Platform pods:

    ```
    $ kubectl create namespace my-namespace
    ```

    2. Set the active namespace in your Kubernetes context to the Kubernetes namespace you just created:

    ```
    $ kubens my-namespace
    ```

6. Set up the certificate management and secret agent.

> **Note**
>
> 1. Since Minikube provides its own ingress controller, NGINX controller need not be installed.
> 2. The `forgeops` repository contains the certmanager-deploy.sh script to install `cert-manager` in your cluster. If you need to use a different certificate management utility, you refer to the corresponding documentation for installing that utility.

```
$ cd /path/to/forgeops/charts/scripts
$ ./install-prereqs cert-manager secrets
```

7. In a separate terminal tab or window, run the minikube tunnel command, and enter your system's superuser password when prompted:

```
$ minikube tunnel
☑  Tunnel successfully started

📌  NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be
accessible …

❗  The service/ingress forgerock requires privileged ports to be exposed: [80 443]
🔑  sudo permission will be asked for it.
❗  The service/ingress ig requires privileged ports to be exposed: [80 443]
🏃  Starting tunnel for service forgerock.
🔑  sudo permission will be asked for it.
🏃  Starting tunnel for service ig.
Password:
```

The tunnel creates networking that lets you access the Minikube cluster's ingress on the localhost IP address (127.0.0.1). Leave the tab or window that started the tunnel open for as long as you run the ForgeOps deployment.

Refer to this post for an explanation about why a Minikube tunnel is required to access ingress resources when running Minikube on an ARM-based macOS system.

8. Set up the `fast` storage class using the `minikube-fast-storage-class.yaml` file in the /path/to/forgeops/cluster/resources directory:

```
$ kubectl apply -f /path/to/forgeops/cluster/resources/minikube-fast-storage-class.yaml
```

9. Run the helm upgrade command to perform a ForgeOps deployment:

```
$ helm upgrade --install identity-platform identity-platform \
 --repo https://ForgeRock.github.io/forgeops/ \
 --version 2025.1.2 --namespace my-namespace \
 --values /path/to/forgeops/helm/my-env/values.yaml
```

The preceding command creates a single-instance ForgeOps deployment. Only single-instance deployments are supported on Minikube.

Learn more about single-instance deployments in Cluster and deployment sizes.

> **Important**
>
> Ping Identity only offers its software or services to legal entities that have entered into a binding license agreement with Ping Identity. When you install Docker images provided by ForgeOps, you agree either that: 1) you are an authorized user of a Ping Identity Platform customer that has entered into a license agreement with Ping Identity governing your use of the Ping Identity software; or 2) your use of the Ping Identity Platform software is subject to the Ping Identity Subscription Agreements.

10. Check the status of the pods in the namespace in which you deployed the platform until all the pods are ready:

    1. Run the kubectl get pods command.

2. Review the output. Deployment is complete when:
    - All entries in the `STATUS` column indicate `Running` or `Completed`.
    - The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
3. If necessary, continue to query your deployment's status until all the pods are ready.

11. (Optional) Install a TLS certificate instead of using the default self-signed certificate in your ForgeOps deployment. Refer to TLS certificate for details.

Next step

Become familiar with ForgeOps deployments

Understand ForgeOps architecture

Deploy the platform

☐ Access platform UIs and APIs

☐ Plan for production deployment

Deploy using Kustomize on GKE, EKS, or AKS

1. Verify that you have set up your environment and created a Kubernetes cluster as documented in the setup section.
2. Enable the Python3 virtual environment:

```
$ source .venv/bin/activate
```

3. The configuration of a ForgeOps deployment is steered through the use of Kustomize overlays or Helm values. Use the forgeops env command to set up the Kustomize overlays and Helm values files to configure your ForgeOps deployment environment:
    - If you want to use the issuer provided with the platform for demo, then you can use default-issuer.
    - For a clusters on a cloud environment specify the --deployment-size as `--small`, `--medium`, or `--large`.
    - For a single-instance deployment, specify --deployment-size as `--single-instance`.

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn --cluster-issuer my-cluster-issuer --
deployment-size
```

    In the command above, replace my-fqdn, my-cluster-issuer, and --deployment-size with appropriate values from your environment.

In a Minikube environment, use the single instance deployment. For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn \
  --cluster-issuer my-cluster-issuer --single-instance
```

Learn more about deployment sizes in Cluster and deployment sizes and about single instances here.

4. Identify Docker images to deploy:
   - If you want to use custom Docker images for the platform, update the image defaulter file with image names and tags generated by the forgeops build command. The image defaulter file is located in your environment (my-env) folder /path/to/forgeops/kustomize/overlay/my-env directory.

     You can get the image names and tags from the image defaulter file on the system on which the customized Docker images were developed.
   - If you want to use ForgeOps-provided Docker images for the platform, do not modify the image defaulter file.
   - Use the forgeops image command to set up the correct component images to be deployed. The following command sets up the latest ForgeOps-provided Docker image for deployment:

     ```
     $ cd /path/to/forgeops/bin
     $ ./forgeops image --env-name my-env --release 8.0.0 platform
     ```

     Note

     If you want to set up your deployment environment with your own image, then use the following example command:

     ```
     $ cd /path/to/forgeops/bin
     $ ./forgeops image --release my-image --release-name my-release --env-name my-env
     platform
     ```

5. Set up your Kubernetes context:
   1. Set the `KUBECONFIG` environment variable so that your Kubernetes context references the cluster in which you'll perform the ForgeOps deployment.
   2. Create a Kubernetes namespace in the cluster for the Ping Identity Platform pods:

      ```
      $ kubectl create namespace my-namespace
      ```

   3. Set the active namespace in your Kubernetes context to the Kubernetes namespace you just created:

      ```
      $ kubens my-namespace
      ```

6. (Optional) If you've set up your Kubernetes cluster using ForgeOps-provided Terraform manifest, then you would've already created the required `fast` storage and volume snapshot classes. If you set up your Kubernetes cluster using your own scripts, then create these classes using the corresponding YAML scripts provided in the /path/to/forgeops/cluster/resources folder.

   For example, on GKE:

   ```
   $ kubectl apply -f /path/to/forgeops/cluster/resources/gke-fast-storage-class.yaml
   $ kubectl apply -f /path/to/forgeops/cluster/resources/gke-volume-snapshot-class.yaml
   ```

7. Run the forgeops apply command to perform a ForgeOps deployment. Learn more in `forgeops apply` command reference.

   For example:

   ```
   $ cd /path/to/forgeops/bin
   $ ./forgeops apply --env-name my-env
   ```

   If you prefer not to deploy using a single forgeops apply command, you can find more information in Alternative deployment techniques when using Kustomize.

   > **Important**
   >
   > Ping Identity only offers its software or services to legal entities that have entered into a binding license agreement with Ping Identity. When you install Docker images provided by ForgeOps, you agree either that: 1) you are an authorized user of a Ping Identity Platform customer that has entered into a license agreement with Ping Identity governing your use of the Ping Identity software; or 2) your use of the Ping Identity Platform software is subject to the Ping Identity Subscription Agreements.

8. Check the status of the pods in the namespace in which you deployed the platform until all the pods are ready:

   1. Run the kubectl get pods command.
   2. Review the output. Deployment is complete when:
      - All entries in the `STATUS` column indicate `Running` or `Completed`.
      - The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
   3. If necessary, continue to query your deployment's status until all the pods are ready.

9. Back up and save the Kubernetes secrets that contain the master and TLS keys:

   1. To avoid accidentally putting the backups under version control, change to a directory that is outside your `forgeops` repository clone.
   2. The `ds-master-keypair` secret contains the DS master key. This key is required to decrypt data from a directory backup. *Failure to save this key could result in data loss.*

      Back up the Kubernetes secret that contains the DS master key:

```
$ kubectl get secret ds-master-keypair -o yaml > master-key-pair.yaml
```

3. The `ds-ssl-keypair` secret contains the DS TLS key. This key is needed for cross-environment replication topologies.

   Back up the Kubernetes secret that contains the DS TLS key pair:

   ```
   $ kubectl get secret ds-ssl-keypair -o yaml > tls-key-pair.yaml
   ```

4. Save the two backup files.

10. (Optional) Deploy Prometheus, Grafana, and Alertmanager for monitoring and alerting[1]:

    1. Deploy Prometheus, Grafana, and Alertmanager pods in your ForgeOps deployment:

       ```
       $ /path/to/forgeops/bin/prometheus-deploy.sh

       **This script requires Helm version 3.04 or later due to changes in the behaviour of 'helm
       repo add' command.**

       namespace/monitoring created
       "stable" has been added to your repositories
       "prometheus-community" has been added to your repositories
       Hang tight while we grab the latest from your chart repositories...
       ...Successfully got an update from the "ingress-nginx" chart repository
       ...Successfully got an update from the "codecentric" chart repository
       ...Successfully got an update from the "prometheus-community" chart repository
       ...Successfully got an update from the "stable" chart repository
       Update Complete. ⎈Happy Helming!⎈
       Release "prometheus-operator" does not exist. Installing it now.
       NAME: prometheus-operator
       LAST DEPLOYED: ...
       NAMESPACE: monitoring
       STATUS: deployed
       REVISION: 1
       NOTES:
       kube-prometheus-stack has been installed. Check its status by running:
         kubectl --namespace monitoring get pods -l "release=prometheus-operator"

       Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create
       & configure Alertmanager and Prometheus instances using the Operator.
       ...
       Release "forgerock-metrics" does not exist. Installing it now.
       NAME: forgerock-metrics
       LAST DEPLOYED: ...
       NAMESPACE: monitoring
       STATUS: deployed
       REVISION: 1
       TEST SUITE: None
       ```

    2. Check the status of the pods in the `monitoring` namespace until all the pods are ready:

```
$ kubectl get pods --namespace monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-operator-kube-p-alertmanager-0   2/2     Running   0          119s
prometheus-operator-grafana-95b8f5b7d-nn65h              3/3     Running   0          2m4s
prometheus-operator-kube-p-operator-7d54989595-pdj44     1/1     Running   0          2m4s
prometheus-operator-kube-state-metrics-d95996bc4-wcf7s   1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-67xq4       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-b4grn       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-cwhcn       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-h9brd       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-q8zrk       1/1     Running   0          2m4s
prometheus-operator-prometheus-node-exporter-vqpt5       1/1     Running   0          2m4s
prometheus-prometheus-operator-kube-p-prometheus-0       2/2     Running   0          119s
```

11. (Optional) Install a TLS certificate instead of using the default self-signed certificate in your ForgeOps deployment. Refer to TLS certificate for details.

Alternative deployment techniques when using Kustomize

Staged deployments

If you prefer not to perform a ForgeOps Kustomize deployment using a single forgeops apply command, you can deploy the platform in stages, component by component, instead of deploying with a single command. Staging deployments can be useful if you need to troubleshoot a deployment issue.

Generating Kustomize manifests and using `kubectl apply` commands

You can generate Kustomize manifests using the forgeops env command, and then deploy the platform using the kubectl apply -k command.

The forgeops env command generates Kustomize manifests for your ForgeOps deployment environment. The manifests are written to the /path/to/forgeops/kustomize/overlay/my-env directory of your `forgeops` repository clone. Advanced users who prefer to work directly with Kustomize manifests that describe their ForgeOps deployment can use the generated content in the kustomize/overlay/my-env directory as an alternative to using the forgeops command:

1. Generate an initial set of Kustomize manifests by running the forgeops env command.
2. Run kubectl apply -k commands to deploy and remove platform components. Specify a manifest in the kustomize/overlay/my-env directory as an argument when you run kubectl apply -k commands.
    1. Use GitOps to manage configuration changes to the kustomize/overlay/my-env directory.

Next step

Become familiar with ForgeOps deployments

Understand ForgeOps architecture

Deploy the platform

- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

1. Installing Prometheus, Grafana, and Alertmanager technology in ForgeOps deployments provides an example of how you might set up monitoring and alerting in a Ping Identity Platform deployment in the cloud. Remember, ForgeOps deployments are reference implementations and not for production use. When you create a project plan, you'll need to determine how to monitor and send alerts in your production deployment.

Deploy using Kustomize on Minikube

1. Verify that you have set up your environment and created a Kubernetes cluster as documented in the setup section.
2. Enable the Python3 virtual environment:

```
$ source .venv/bin/activate
```

3. The configuration of a ForgeOps deployment is steered through the use of Kustomize overlays or Helm values. Use the forgeops env command to set up the Kustomize overlays and Helm values files to configure your ForgeOps deployment environment:
   - If you want to use the issuer provided with the platform for demo, then you can use default-issuer.
   - For a clusters on a cloud environment specify the --deployment-size as `--small`, `--medium`, or `--large`.
   - For a single-instance deployment, specify --deployment-size as `--single-instance`.

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn --cluster-issuer my-cluster-issuer --
deployment-size
```

   In the command above, replace my-fqdn, my-cluster-issuer, and --deployment-size with appropriate values from your environment.

   In a Minikube environment, use the single instance deployment. For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn \
  --cluster-issuer my-cluster-issuer --single-instance
```

   Learn more about deployment sizes in Cluster and deployment sizes and about single instances here.

4. Identify Docker images to deploy:
   - If you want to use custom Docker images for the platform, update the image defaulter file with image names and tags generated by the forgeops build command. The image defaulter file is located in your environment (my-env) folder /path/to/forgeops/kustomize/overlay/my-env directory.

     You can get the image names and tags from the image defaulter file on the system on which the customized Docker images were developed.
   - If you want to use ForgeOps-provided Docker images for the platform, do not modify the image defaulter file.

- Use the forgeops image command to set up the correct component images to be deployed. The following command sets up the latest ForgeOps-provided Docker image for deployment:

```
$ cd /path/to/forgeops/bin
$ ./forgeops image --env-name my-env --release 8.0.0 platform
```

> **Note**
>
> If you want to set up your deployment environment with your own image, then use the following example command:
>
> ```
> $ cd /path/to/forgeops/bin
> $ ./forgeops image --release my-image --release-name my-release --env-name my-env
> platform
> ```

5. Set up your Kubernetes context:

    1. Create a Kubernetes namespace in the cluster for the Ping Identity Platform pods:

    ```
    $ kubectl create namespace my-namespace
    ```

    2. Set the active namespace in your Kubernetes context to the Kubernetes namespace you just created:

    ```
    $ kubens my-namespace
    ```

6. Set up the certificate management and secret agent.

> **Note**
>
> 1. Since Minikube provides its own ingress controller, NGINX controller need not be installed.
> 2. The `forgeops` repository contains the certmanager-deploy.sh script to install `cert-manager` in your cluster. If you need to use a different certificate management utility, you refer to the corresponding documentation for installing that utility.

```
$ cd /path/to/forgeops/charts/scripts
$ ./install-prereqs cert-manager secrets
```

7. In a separate terminal tab or window, run the minikube tunnel command, and enter your system's superuser password when prompted:

```
$ minikube tunnel
☑  Tunnel successfully started

📌  NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be
accessible …

❗  The service/ingress forgerock requires privileged ports to be exposed: [80 443]
🔑  sudo permission will be asked for it.
❗  The service/ingress ig requires privileged ports to be exposed: [80 443]
🏃  Starting tunnel for service forgerock.
🔑  sudo permission will be asked for it.
🏃  Starting tunnel for service ig.
Password:
```

The tunnel creates networking that lets you access the Minikube cluster's ingress on the localhost IP address (127.0.0.1). Leave the tab or window that started the tunnel open for as long as you run the ForgeOps deployment.

Refer to this post for an explanation about why a Minikube tunnel is required to access ingress resources when running Minikube on an ARM-based macOS system.

8. Set up the `fast` storage class using the `minikube-fast-storage-class.yaml` file in the /path/to/forgeops/cluster/resources directory:

```
$ kubectl apply -f /path/to/forgeops/cluster/resources/minikube-fast-storage-class.yaml
```

9. Run the forgeops apply command. Learn more in `forgeops apply` command reference.

   For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops apply --env-name my-env
```

   The preceding command creates a single-instance ForgeOps deployment. Only single-instance deployments are supported on Minikube.

   If you prefer not to deploy using a single forgeops apply command, you can find more information in Alternative deployment techniques when using Kustomize.

   > **Important**
   >
   > Ping Identity only offers its software or services to legal entities that have entered into a binding license agreement with Ping Identity. When you install Docker images provided by ForgeOps, you agree either that: 1) you are an authorized user of a Ping Identity Platform customer that has entered into a license agreement with Ping Identity governing your use of the Ping Identity software; or 2) your use of the Ping Identity Platform software is subject to the Ping Identity Subscription Agreements.

10. Check the status of the pods in the namespace in which you deployed the platform until all the pods are ready:

    1. Run the kubectl get pods command.

2. Review the output. Deployment is complete when:
    - All entries in the `STATUS` column indicate `Running` or `Completed`.
    - The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
3. If necessary, continue to query your deployment's status until all the pods are ready.

11. (Optional) Install a TLS certificate instead of using the default self-signed certificate in your ForgeOps deployment. Refer to TLS certificate for details.

Alternative deployment techniques when using Kustomize

Staged deployments

If you prefer not to perform a ForgeOps Kustomize deployment using a single forgeops apply command, you can deploy the platform in stages, component by component, instead of deploying with a single command. Staging deployments can be useful if you need to troubleshoot a deployment issue.

Generating Kustomize manifests and using `kubectl apply` commands

You can generate Kustomize manifests using the forgeops env command, and then deploy the platform using the kubectl apply -k command.

The forgeops env command generates Kustomize manifests for your ForgeOps deployment environment. The manifests are written to the /path/to/forgeops/kustomize/overlay/my-env directory of your `forgeops` repository clone. Advanced users who prefer to work directly with Kustomize manifests that describe their ForgeOps deployment can use the generated content in the kustomize/overlay/my-env directory as an alternative to using the forgeops command:

1. Generate an initial set of Kustomize manifests by running the forgeops env command.
2. Run kubectl apply -k commands to deploy and remove platform components. Specify a manifest in the kustomize/overlay/my-env directory as an argument when you run kubectl apply -k commands.
    1. Use GitOps to manage configuration changes to the kustomize/overlay/my-env directory.

Next step

Become familiar with ForgeOps deployments

Understand ForgeOps architecture

Deploy the platform

- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

UI and API access

This page shows you how to access and monitor the Ping Identity Platform components in a ForgeOps deployment.

AM and IDM are configured for access through the Kubernetes cluster's ingress controller. You can access these components using their admin UIs and REST APIs.

DS cannot be accessed through the ingress controller, but you can use Kubernetes methods to access the DS pods.

AM services

To access the AM admin UI:

1. Set the active namespace in your local Kubernetes context to the namespace in which you performed the ForgeOps deployment.
2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./forgeops info | grep amadmin
   vr58qt11ihoa31zfbjsdxxrqryfw0s31 (amadmin user)
   ```

3. Open a new window or tab in a web browser.
4. Go to https://my-fqdn/platform.

   The Kubernetes ingress controller handles the request, routing it to the `login-ui` pod.

   The login UI prompts you to log in.
5. Log in as the `amadmin` user.

   The Ping Identity Platform UI appears in the browser.
6. Select **Native Consoles > Access Management**.

   The AM admin UI appears in the browser.

To access the AM REST APIs:

1. Start a terminal window session.
2. Run a curl command to verify that you can access the REST APIs through the ingress controller. For example:

```
$ curl \
 --insecure \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: amadmin" \
 --header "X-OpenAM-Password: vr58qt11ihoa31zfbjsdxxrqryfw0s31" \
 --header "Accept-API-Version: resource=2.0" \
 --data "{}" \
 "https://my-fqdn/am/json/realms/root/authenticate"

{
    "tokenId":"AQIC5wM2...",
    "successUrl":"/am/console",
    "realm":"/"
}
```

IDM services

To access the IDM admin UI:

1. Set the active namespace in your local Kubernetes context to the namespace in which you performed the ForgeOps deployment.
2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./forgeops info | grep amadmin
   vr58qt11ihoa31zfbjsdxxrqryfw0s31 (amadmin user)
   ```

3. Open a new window or tab in a web browser.
4. Go to https://my-fqdn/platform.

   The Kubernetes ingress controller handles the request, routing it to the `login-ui` pod.

   The login UI prompts you to log in.
5. Log in as the `amadmin` user.

   The Ping Identity Platform UI appears in the browser.
6. Select **Native Consoles > Identity Management**.

   The IDM admin UI appears in the browser.

To access the IDM REST APIs:

1. Start a terminal window session.
2. If you haven't already done so, get the `amadmin` user's password using the forgeops info command.

3. AM authorizes IDM REST API access using the OAuth 2.0 authorization code flow. ForgeOps deployments come with the `idm-admin-ui` client, which is configured to let you get a bearer token using this OAuth 2.0 flow. You'll use the bearer token in the next step to access the IDM REST API:

    1. Get a session token for the `amadmin` user:

    ```
    $ curl \
     --request POST \
     --insecure \
     --header "Content-Type: application/json" \
     --header "X-OpenAM-Username: amadmin" \
     --header "X-OpenAM-Password: vr58qt11ihoa31zfbjsdxxrqryfw0s31" \
     --header "Accept-API-Version: resource=2.0, protocol=1.0" \
     'https://my-fqdn/am/json/realms/root/authenticate'
    {
     "tokenId":"AQIC5wM...TU3OQ*",
     "successUrl":"/am/console",
     "realm":"/"}
    ```

    2. Get an authorization code. Specify the ID of the session token that you obtained in the previous step in the `--Cookie` parameter:

    ```
    $ curl \
     --dump-header - \
     --insecure \
     --request GET \
     --Cookie "iPlanetDirectoryPro=AQIC5wM...TU3OQ*" \
     "https://my-fqdn/am/oauth2/realms/root/authorize?redirect_uri=https://my-fqdn/platform/
    appAuthHelperRedirect.html&client_id=idm-admin-
    ui&scope=openid%20fr:idm:*&response_type=code&state=abc123"
    HTTP/2 302
    server: nginx/1.17.10
    date: Mon, 10 May 2021 16:54:20 GMT
    content-length: 0
    location: \https://my-fqdn/platform/appAuthHelperRedirect.html
     ?code=3cItL9G52DIiBdfXRngv2_dAaYM&iss=http://my-fqdn:80/am/oauth2&state=abc123
     &client_id=idm-admin-ui
    set-cookie: route=1595350461.029.542.7328; Path=/am; Secure; HttpOnly
    x-frame-options: SAMEORIGIN
    x-content-type-options: nosniff
    cache-control: no-store
    pragma: no-cache
    set-cookie: OAUTH_REQUEST_ATTRIBUTES=DELETED; Expires=Thu, 01 Jan 1970 00:00:00 GMT; Path=/;
    HttpOnly; SameSite=none
    strict-transport-security: max-age=15724800; includeSubDomains
    x-forgerock-transactionid: ee1f79612f96b84703095ce93f5a5e7b
    ```

    3. Exchange the authorization code for an access token. Specify the access code that you obtained in the previous step in the `code` URL parameter:

```
$ curl --request POST \
 --insecure \
 --data "grant_type=authorization_code" \
 --data "code=3cItL9G52DIiBdfXRngv2_dAaYM" \
 --data "client_id=idm-admin-ui" \
 --data "redirect_uri=https://my-fqdn/platform/appAuthHelperRedirect.html" \
 "https://my-fqdn/am/oauth2/realms/root/access_token"
{
 "access_token":"oPzGzGFY1SeP2RkI-ZqaRQC1cDg",
 "scope":"openid fr:idm:*",
 "id_token":"eyJ0eXAiOiJKV
  ...
  sO4HYqlQ",
 "token_type":"Bearer",
 "expires_in":239
}
```

4. Run a curl command to verify that you can access the `openidm/config` REST endpoint through the ingress controller. Use the access token returned in the previous step as the bearer token in the authorization header.

   The following example command provides information about the IDM configuration:

```
$ curl \
 --insecure \
 --request GET \
 --header "Authorization: Bearer oPzGzGFY1SeP2RkI-ZqaRQC1cDg" \
 --data "{}" \
 \https://my-fqdn/openidm/config
{
 "_id":"",
 "configurations":
  [
   {
    "_id":"ui.context/admin",
    "pid":"ui.context.4f0cb656-0b92-44e9-a48b-76baddda03ea",
    "factoryPid":"ui.context"
    },
    ...
   ]
}
```

DS command-line access

The DS pods in ForgeOps deployment are not exposed outside of the cluster. If you need to access one of the DS pods, use a standard Kubernetes method:

- Execute shell commands in DS pods using the kubectl exec command.
- Forward a DS pod's LDAPS port (1636) to your local computer. Then, you can run LDAP CLI commands, for example ldapsearch. You can also use an LDAP editor such as Apache Directory Studio to access the directory.

For all ForgeOps deployment directory pods, the directory superuser DN is `uid=admin`. Obtain this user's password by running the forgeops info command.

ForgeOps deployment monitoring

This section describes how to access Grafana dashboards and Prometheus UI'[1]' .

Grafana

To access Grafana dashboards:

1. Set up port forwarding on your local computer for port 3000:

```
$ /path/to/forgeops/bin/prometheus-connect.sh -G
Forwarding from 127.0.0.1:3000 → 3000
Forwarding from [::1]:3000 → 3000
```

2. In a web browser, navigate to http://localhost:3000 to access the Grafana dashboards.
3. Log in as the `admin` user with `password` as the password.

When you're done using the Grafana UI, stop Grafana port forwarding by entering Ctrl+c in the terminal window where you initiated port forwarding.

For information about Grafana, refer to the Grafana documentation.

Prometheus

To access the Prometheus UI:

1. Set up port forwarding on your local computer for port 9090:

```
$ /path/to/forgeops/bin/prometheus-connect.sh -P
Forwarding from 127.0.0.1:9090 → 9090
Forwarding from [::1]:9090 → 9090
```

2. In a web browser, navigate to http://localhost:9090 to access the Prometheus UI.

When you're done using the Prometheus UI, stop Prometheus port forwarding by entering Ctrl+c in the terminal window where you initiated port forwarding.

For information about Prometheus, refer to the Prometheus documentation.

For a description of ForgeOps monitoring architecture and information about how to customize ForgeOps monitoring, refer to ForgeOps deployment monitoring.

Next step

Become familiar with ForgeOps deployments

Understand ForgeOps architecture

Deploy the platform

Access platform UIs and APIs

☐ Plan for production deployment

1. Not available on ForgeOps deployments on Minikube.

Next steps

If you've followed the instructions for performing a ForgeOps deployment *without modifying configurations*, then the following indicates that you've been successful:

- The Kubernetes cluster and pods are up and running.
- DS, AM, and IDM are installed and running. You can access each ForgeOps component.
- DS replication and failover work as expected.[1]

When you're satisfied that all of these conditions are met, then you've successfully taken the first steps towards deploying the Ping Identity Platform on Kubernetes. Congratulations!

You can use the ForgeOps deployment to test deployment customizations—options that you might want to use in production but are not part of the base deployment. Examples[2] include, but are not limited to:

- Running lightweight benchmark tests
- Backing up and restoring your data
- Securing TLS with a certificate that's dynamically obtained from Let's Encrypt
- Using an ingress controller other than the Ingress-NGINX controller
- Resizing the cluster to meet your business requirements
- Configuring Alert Manager to issue alerts when usage thresholds have been reached

Now that you're familiar with ForgeOps deployments, you're ready to work with a project team to plan and configure your production deployment. You'll need a team with expertise in the Ping Identity Platform, in your cloud provider, and in Kubernetes on your cloud provider. We strongly recommend that you engage a Ping Identity technical consultant or partner to assist you with deploying the platform in production.

You'll perform these major activities:

Platform configuration—Ping Identity Platform experts configure AM and IDM using single-instance ForgeOps deployments and build custom Docker images for the Ping Identity Platform. The Customization overview provides information about platform configuration tasks.

Cluster configuration—Cloud technology experts configure the Kubernetes cluster that will host the Ping Identity Platform for optimal performance and reliability. Tasks include configuring your Kubernetes cluster to suit your business needs, setting up monitoring and alerts to track site health and performance, backing up configuration and user data for disaster preparedness, and securing your deployment. The Prepare to deploy in production and READMEs in the `forgeops` repository provide information about cluster configuration.

Site reliability engineering—Site reliability engineers monitor the Ping Identity Platform deployment and keep the deployment up and running based on your business requirements. These could include use cases, service-level agreements, thresholds, and load test profiles. The Prepare to deploy in production, and READMEs in the `forgeops` repository, provide information about site reliability.

1. Not available on single-instance ForgeOps deployments.
2. Not available on ForgeOps deployments on Minikube.

Remove a ForgeOps deployment

This page provides instructions for removing ForgeOps deployments for the following scenarios:

- Remove a Helm deployment on GKE, EKS, or AKS
- Remove a Helm deployment on Minikube
- Remove a Kustomize deployment on GKE, EKS, or AKS
- Remove a Kustomize deployment on Minikube

Remove a Helm deployment from GKE, EKS, or AKS

1. Set up your Kubernetes context:
    1. Set the `KUBECONFIG` environment variable so that your Kubernetes context references the cluster in which you deployed the platform.
    2. Set the active namespace in your Kubernetes context to the Kubernetes namespace in which you deployed the platform:

    ```
    $ kubens my-namespace
    ```

2. Remove the ForgeOps deployment:

    ```
    $ cd /path/to/forgeops/charts/identity-platform
    $ helm uninstall identity-platform
    ```

    Running helm uninstall identity-platform doesn't delete PVCs and the `amster` job from your namespace.

3. (Optional) To delete PVCs, use the kubectl command. For example, to delete `data-ds-idrepo-0` and `data-ds-cts-0`:

    ```
    $ kubectl delete pvc data-ds-idrepo-0 data-ds-cts-0
    ```

4. (Optional) To delete the `amster` job, use the kubectl command:

    ```
    $ kubectl delete job amster
    ```

5. (Optional) Delete your cluster:

    1. Change to the directory in your `forgeops-extras` repository clone that contains Terraform artifacts:

    ```
    $ cd /path/to/forgeops-extras/terraform
    ```

    2. Run the tf-destroy script to create your cluster:

    ```
    $ ./tf-destroy
    ```

    Respond `yes` to the `Do you really want to destroy all resources?` prompt.

Remove a Helm deployment from Minikube

1. Set the active namespace in your Kubernetes context to the Kubernetes namespace in which you deployed the platform:

```
$ kubens my-namespace
```

2. Remove the ForgeOps deployment:

```
$ cd /path/to/forgeops/charts/identity-platform
$ helm uninstall identity-platform
```

Running helm uninstall identity-platform doesn't delete PVCs and the `amster` job from your namespace.

3. (Optional) To delete PVCs, use the kubectl command. For example, to delete `data-ds-idrepo-0` and `data-ds-cts-0`:

```
$ kubectl delete pvc data-ds-idrepo-0 data-ds-cts-0
```

4. (Optional) To delete the `amster` job, use the kubectl command:

```
$ kubectl delete job amster
```

5. (Optional) Delete your cluster:

```
$ minikube stop
$ minikube delete
```

Remove a Kustomize deployment from GKE, EKS, or AKS

1. Set up your Kubernetes context:

    1. Set the `KUBECONFIG` environment variable so that your Kubernetes context references the cluster in which you deployed the platform.

2. Set the active namespace in your Kubernetes context to the Kubernetes namespace in which you deployed the platform:

```
$ kubens my-namespace
```

2. Remove the ForgeOps deployment:

```
$ cd /path/to/forgeops/bin
$ ./forgeops delete --env-name my-env
```

Respond `Y` to all the `OK to delete?` prompts.

3. (Optional) Delete your cluster:

1. Change to the directory in your `forgeops-extras` repository clone that contains Terraform artifacts:

```
$ cd /path/to/forgeops-extras/terraform
```

2. Run the tf-destroy script to create your cluster:

```
$ ./tf-destroy
```

Respond `yes` to the `Do you really want to destroy all resources?` prompt.

Remove a Kustomize deployment from Minikube

1. Set the active namespace in your Kubernetes context to the Kubernetes namespace in which you deployed the platform:

```
$ kubens my-namespace
```

2. Remove the ForgeOps deployment:

```
$ cd /path/to/forgeops/bin
$ ./forgeops delete --env-name my-env
```

Respond `Y` to all the `OK to delete?` prompts.

3. (Optional) Delete your cluster:

```
$ minikube stop
$ minikube delete
```

Customize your deployment

Customization overview

This section covers how developers build custom Docker images for the Ping Identity Platform. It also contains important conceptual material that you need to understand before you start creating Docker images.

Developer checklist

Setup:

- ☐ Perform additional setup
- ☐ Understand custom images

DS customization:

- ☐ Customize the DS image

AM and IDM customization:

- ☐ Understand AM and IDM configuration
- ☐ Understand property value substitution
- ☐ Customize the AM image
- ☐ Customize the IDM image

Additional setup

This page covers setup tasks that you'll need to perform before you can develop custom Docker images for the Ping Identity Platform. Complete all of the tasks on this page before proceeding.

Use a single-instance ForgeOps deployment

You must use a single-instance ForgeOps deployment to develop custom Docker images for the Ping Identity Platform.

Use the following links for information about how to create single-instance ForgeOps deployments:

- Deploy using Helm on GKE, EKS, or AKS
- Deploy using Helm on Minikube
- Deploy using Kustomize on GKE, EKS, or AKS

- Deploy using Kustomize on Minikube

Set up your environment to push to your Docker registry

ForgeOps deployments support any container registry that supports Docker containers. You'll need to set up your local environment to support your container registry. Here are setup steps for four commonly-used container registries:

Set up your local environment to execute docker commands on Minikube's Docker engine.

The ForgeOps team recommends using the built-in Docker engine when developing custom Docker images using Minikube. When you use Minikube's Docker engine, you don't have to build Docker images on a local engine and then push the images to a local or cloud-based Docker registry. Instead, you build images using the same Docker engine that Minikube uses. This streamlines development.

To set up your local computer to use Minikube's Docker engine, run the docker-env command in your shell:

```
$ eval $(minikube docker-env)
```

For more information about using Minikube's built-in Docker engine, refer to Use local images by re-using the Docker daemon in the Minikube documentation.

To set up your local computer to build and push Docker images:

1. If it's not already running, start a virtual machine that runs Docker engine. Refer to Docker engine for more information.
2. Set up a Docker credential helper:

   ```
   $ gcloud auth configure-docker
   ```

To set up your local computer to push Docker images:

1. If it's not already running, start a virtual machine that runs Docker engine. Refer to Docker engine for more information.
2. Log in to Amazon ECR:

   ```
   $ aws ecr get-login-password | \
    docker login --username AWS --password-stdin my-docker-registry
   Login Succeeded
   ```

   ECR login sessions expire after 12 hours. Because of this, you'll need to perform these steps again whenever your login session expires.[1]

To set up your local computer to push Docker images:

1. If it's not already running, start a virtual machine that runs Docker engine. Refer to Docker engine for more information.
2. Install the ACR Docker Credential Helper.

Identify the Docker repository to push to

When you execute the forgeops build command, you must specify the repository to push your Docker image to with the --push-to argument.

The forgeops build command appends a component name to the destination repository. For example, the command forgeops build am --push-to us-docker.pkg.dev/my-project pushes a Docker image to the `us-docker.pkg.dev/my-project/am` repository.

To determine how to specify the --push-to argument for four commonly-used container registries:

Specify --push-to none with the forgeops build command to push the Docker image to the Docker registry embedded in the Minikube cluster.

Obtain the --push-to location from your cluster administrator. After it builds the Docker image, the forgeops build command pushes the Docker image to this repository.

Obtain the --push-to location from your cluster administrator. After it builds the Docker image, the forgeops build command pushes the Docker image to this repository.

Obtain the --push-to location from your cluster administrator. After it builds the Docker image, the forgeops build command pushes the Docker image to this repository.

Initialize deployment environments

Deployment environments let you manage deployment manifests and image defaulters for multiple environments in a single `forgeops` repository clone.

By default, the forgeops build command updates the image defaulter in the kustomize/deploy directory.

When you specify a deployment environment, the forgeops build command updates the image defaulter in the kustomize/deploy-environment directory. For example, if you ran forgeops build --deploy-env production, the image defaulter in the kustomize/deploy-production/image-defaulter directory would be updated.

Before you can use a new deployment environment, you must initialize a directory based on the /path/to/forgeops/kustomize/deploy directory to support the deployment environment. Perform these steps to initialize a new deployment environment:

```
$ cd /path/to/forgeops/bin
$ ./forgeops clean
$ cd ../kustomize
$ cp -rp deploy deploy-my-environment
```

> **Note**
>
> If you need multiple deployment environments, you'll need to initialize each environment before you can start using it.

Next step

- Perform additional setup
- ☐ Understand custom images
- ☐ Customize the DS image

- ☐ Understand AM and IDM configuration
- ☐ Understand property value substitution
- ☐ Customize the AM image
- ☐ Customize the IDM image

1. You can automate logging into ECR every 12 hours by using the `cron` utility.

About custom images

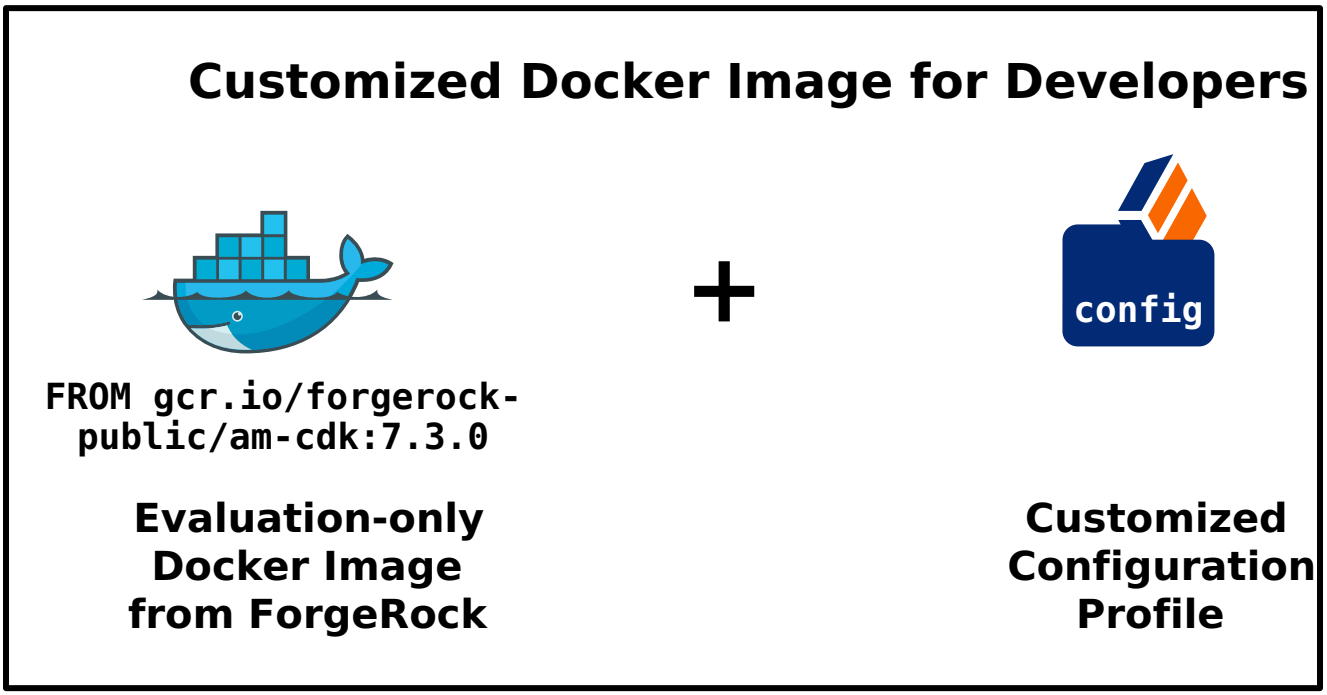
In development

To develop customized Docker images, start with ForgeOps-provided images. Then, build your configuration profile iteratively as you customize the platform to meet your needs. Building Docker images from time to time integrates your custom configuration profile into new Docker images.

To develop a customized DS Docker image, refer to `ds` image.

To develop a customized AM Docker image, refer to `am` image.

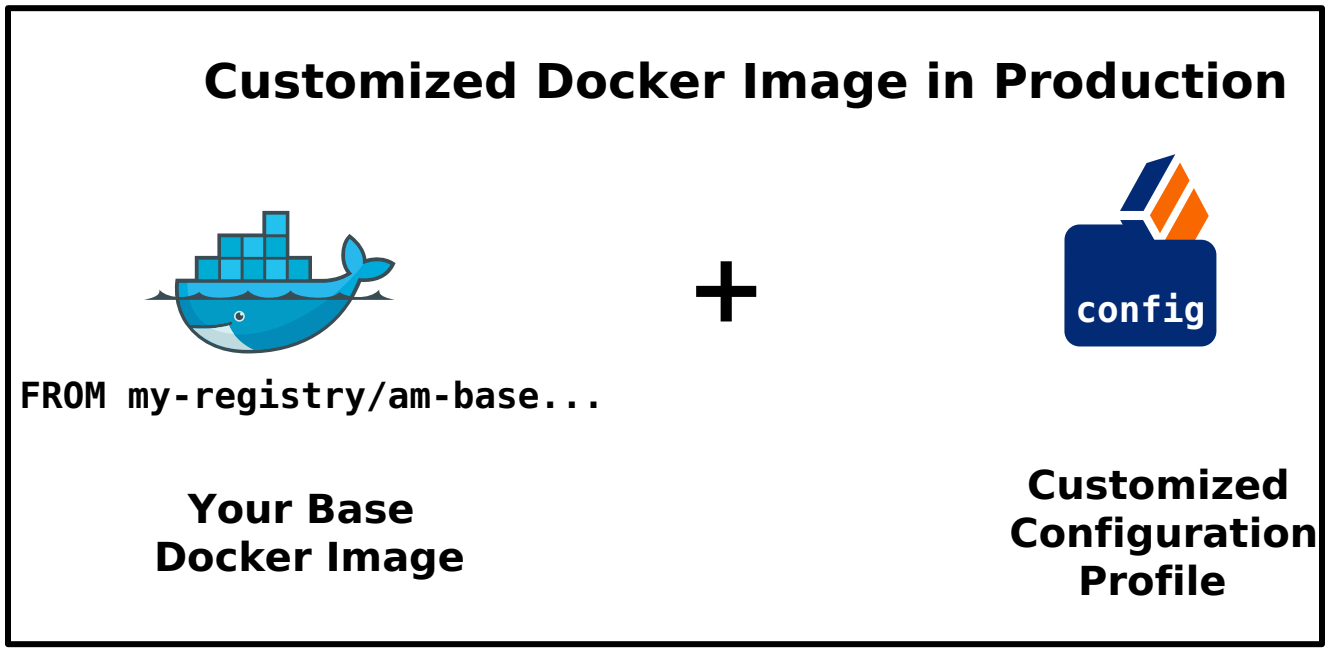
To develop a customized IDM Docker image, refer to `idm` image.



In production

Before you deploy the platform in production, you can build your own base images and integrate your configuration profiles into them.

Learn more about how to create Docker images for production deployment of the platform in Base Docker images.



Next step

Perform additional setup

Understand custom images

- ☐ Customize the DS image
- ☐ Understand AM and IDM configuration
- ☐ Understand property value substitution
- ☐ Customize the AM image
- ☐ Customize the IDM image

`ds` image

The `ds` Docker image contains the DS configuration. You can customize the DS image before deploying it in your production environment.

> **Note**
>
> The customization described here is for use in new Ping Identity Platform deployments.

This section covers:

- Customize LDAP configuration by including LDIF format LDAP configuration files in `ldif-ext` directory.
- Customize LDAP schema by including customized schema LDIF files in the `config` directory.
- Customize DS setup behavior by updating the setup and post-init runtime scripts in the `runtime-scripts` directory.
- Build an updated DS Docker image that contains the above-mentioned customizations.

- Redeploy DS.
- Verify the changes you've made to the DS configuration are in the new Docker image.

Detailed steps

1. Verify that:
    - You have access to a single-instance ForgeOps deployment.
    - The namespace where the platform is deployed is set in your Kubernetes context.
    - All required third-party software is installed in your local environment (Minikube | GKE | EKS | AKS).
    - You have set up your environment to push to your Docker registry.
2. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. (Optional) Run the git commit command to commit the changes.
3. Add your DS customizations:
    1. Learn more at custom LDAP configuration to add LDAP configuration.
    2. Learn more in custom LDAP schema to add LDAP schema.
    3. Customize DS's setup behavior in the /path/to/forgeops/docker/ds/ds-new directory:
        1. To set up profiles and indexes, edit the `runtime-scripts/setup` script. Learn more in `setup` script details.
        2. To add custom configurations after indexes have been rebuilt, edit the `runtime-scripts/post-init` script. Learn more in `post-init` script details.
        3. To prepare the DS docker image for setup, edit the `ds-setup.sh` script. Learn more in `ds-setup.sh` script details.
4. Identify the repository where you'll push the Docker image. You'll use this location to specify the --push-to argument value in the build ds image step.
5. Decide on the DS image tag for each build of the image. You'll use this tag to specify the `--tag` argument value in the build DS image step.
6. Build a new DS image that includes your customization:

```
$ cd /path/to/forgeops/bin
$ ./forgeops build ds --env-name my-env --config-profile my-profile --push-to my-repo --tag my-ds-tag
```

7. Redeploy DS using your new DS image:

Deploy using the forgeops command

The `forgeops build` command calls Docker to build a new `ds` Docker image and to push the image to your Docker repository. The new image includes your custom LDAP and schema files. It also updates the image defaulter file so that the next time you install DS, the deployed DS server includes your custom DS image.

Perform version control activities on your `forgeops` repository clone:

1. Run the git status command.

   Review the state of the kustomize/deploy/image-defaulter/kustomization.yaml file.

2. (Optional) Run the git commit command to commit changes to the image defaulter file.

3. Remove DS from your ForgeOps deployment:

```
$ ./forgeops delete ds --env-name my-env
...
deployment.apps "ds" deleted
```

4. Delete the PVCs attached to DS pods using the kubectl delete pvc command.

5. Redeploy DS using the new Docker image:

```
$ ./forgeops apply ds --env-name my-env --single-instance
Checking cert-manager and related CRDs: cert-manager CRD found in cluster.
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster
```

Deploy using Helm

1. Locate the repository and tag for the new DS Docker image from the forgeops build command output.

2. Delete the PVCs attached to DS pods using the kubectl delete pvc command.

   If the attached DS pod is running, the PVC is not deleted immediately. So you should stop the running DS pods.

   In another terminal window, stop the DS pods using the kubectl delete pods command. This deletes the pods and its attached PVC.

3. Redeploy DS using the new Docker image:

```
$ cd /path/to/forgeops/charts/identity-platform
$ helm upgrade identity-platform ./ \
 --namespace my-namespace \
 --set 'ds.image.repository=my-repository' \
 --set 'ds.image.tag=my-ds-tag'
```

Next step

Perform additional setup

Understand custom images

Customize the DS image

- ☐ Understand AM and IDM configuration
- ☐ Understand property value substitution
- ☐ Customize the AM image
- ☐ Customize the IDM image

`am` and `idm` images

AM and IDM use two types of configuration: static configuration and dynamic configuration.

Static configuration

Static configuration consists of properties and settings used by the Ping Identity Platform. Examples of static configuration include AM realms, AM authentication trees, IDM social identity provider definitions, and IDM data mapping models for reconciliation.

Static configuration is stored in JSON configuration files. Because of this, static configuration is also referred to as *file-based configuration*.

You build static configuration into the `am` and `idm` Docker images during development using the following general process:

1. Change the AM or IDM configuration in a single-instance ForgeOps deployment using the UIs and APIs.
2. Export the changes to your `forgeops` repository clone.
3. Build a new AM or IDM Docker image that contains the updated configuration.
4. Restart Ping Identity Platform services using the new Docker images.
5. Test your changes. Incorrect changes to static configuration might cause the platform to become inoperable.
6. Promote your changes to your test and production environments as desired.

Refer to `am` image and `idm` image for more detailed steps.

In Ping Identity Platform deployments, static configuration is *immutable*. Do not change static configuration in testing or production. Instead, if you need to change static configuration, return to the development phase, make your changes, and build new custom Docker images that include the changes. Then, promote the new images to your test and production environments.

Dynamic configuration

Dynamic configuration consists of access policies, applications, and data objects used by the Ping Identity Platform. Examples of dynamic configuration include AM access policies, AM agents, AM OAuth 2.0 client definitions, IDM identities, and IDM relationships.

Dynamic configuration can change at any time, including when the platform is running in production.

You'll need to devise a strategy for managing AM and IDM dynamic configuration, so that you can:

- Extract sample dynamic configuration for use by developers.
- Back up and restore dynamic configuration.

Tips for managing AM dynamic configuration

You can use one or both of the following techniques to manage AM dynamic configuration:

- Use the forgeops amster command to manage AM dynamic configuration. For example:

    1. Make modifications to AM dynamic configuration by using the AM admin UI.
    2. Export the AM dynamic configuration to your local file system by using the forgeops amster command. You might manage these files in a Git repository. For example:

```
$ cd /path/to/forgeops/bin
$ mkdir /tmp/amster
$ ./forgeops amster export --env-name my-env /tmp/amster
Cleaning up amster components
Packing and uploading configs
configmap/amster-files created
configmap/amster-export-type created
Deploying amster
job.batch/amster created

Waiting for amster job to complete. This can take several minutes.
pod/amster-r99l9 condition met
tar: Removing leading `/' from member names
Updating amster config.
Updating amster config complete.
Cleaning up amster components
job.batch "amster" deleted
configmap "amster-files" deleted
configmap "amster-export-type" deleted
```

    3. If desired, import these files into another AM deployment by using the forgeops amster import command.

    Note that the forgeops amster command automatically converts passwords in AM dynamic configuration to configuration expressions. Because of this, passwords in AM configuration files will not appear in cleartext. For details about how to work with dynamic configuration that has passwords and other properties specified as configuration expressions, refer to Export Utilities and Configuration Expressions.

- Write REST API applications to import and export AM dynamic configuration. For more information, refer to Rest API in the AM documentation.

Tips for managing IDM dynamic configuration

You can use one or both of the following techniques to manage IDM dynamic configuration:

- Migrate dynamic configuration by using IDM's Data Migration Service. For more information, refer to Migrate Data in the IDM documentation.
- Write REST API applications to import and export IDM dynamic configuration. For more information, refer to the Rest API Reference in the IDM documentation.

Configuration profiles

A Ping Identity Platform *configuration profile* is a named set of configuration that describes the operational characteristics of a running ForgeOps deployment. A configuration profile consists of:

- AM static configuration
- IDM static configuration

Configuration profiles reside in the following paths in the `forgeops` repository:

- docker/am/config-profiles
- docker/idm/config-profiles

User-customized configuration profiles are stored in subdirectories of these paths. For example, a configuration profile named `my-profile` would be stored in the paths docker/am/config-profiles/my-profile and docker/idm/config-profiles/my-profile.

Use Git to manage the directories that contain configuration profiles.

Next step

Perform additional setup

Understand custom images

Customize the DS image

Understand AM and IDM configuration

- ☐ Understand property value substitution
- ☐ Customize the AM image
- ☐ Customize the IDM image

About property value substitution

Many property values in ForgeOps deployments' canonical configuration profile are specified as *configuration expressions* instead of as hard-coded values. Fully-qualified domain names (FQDNs), passwords, and several other properties are all specified as configuration expressions.

Configuration expressions are property values in the AM and IDM configurations that are set when AM and IDM start up. Instead of being set to fixed, hard-coded values in the AM and IDM configurations, their values vary, depending on conditions in the run-time environment.

Using configuration expressions lets you use a single configuration profile that takes different values at run-time depending on the deployment environment. For example, you can use a single configuration profile for development, test, and production deployments.

In the Ping Identity Platform, configuration expressions are preceded by an ampersand and enclosed in braces. For example, `&{am.encryption.key}`.

The statement, `am.encryption.pwd=&{am.encryption.key}` in the AM configuration indicates that the value of the property, `am.encryption.pwd`, is determined when AM starts up. Contrast this with a statement, `am.encryption.pwd=myPassw0rd`, which sets the property to a hard-coded value, `myPassw0rd`, regardless of the run-time environment.

How property value substitution works

This example shows how property value substitution works for a value specified as a configuration expression in the AM configuration:

1. Search the /path/to/forgeops/docker directory for the string `&{`.
2. Locate this line in your search results:

   ```
   "am.encryption.pwd=&{am.encryption.key}",
   ```

   Because the property `am.encryption.pwd` is being set to a configuration expression, its value will be determined when AM starts up.
3. Search the `forgeops` repository for the string `AM_ENCRYPTION_KEY`. You'll notice that the secret agent operator sets the environment variable, `AM_ENCRYPTION_KEY`. The property, `am.encryption.pwd`, will be set to the value of the environment variable, `AM_ENCRYPTION_KEY` when AM starts up.

Configuration expressions take their values from environment variables as follows:

- Uppercase characters replace lowercase characters in the configuration expression's name.
- Underscores replace periods in the configuration expression's name.

For more information about configuration expressions, refer to Property Value Substitution in the IDM documentation.

Export utilities and configuration expressions

This section covers differences in how `forgeops` repository utilities export configuration that contains configuration expressions from a running ForgeOps deployment.

In the IDM configuration

The IDM admin UI is aware of configuration expressions.

Passwords specified as configuration expressions in the IDM admin UI are stored in IDM's JSON-based configuration files as configuration expressions.

IDM static configuration export

The `forgeops` repository's bin/config export idm command exports IDM static configuration from running ForgeOps deployments to your `forgeops` repository clone. The config utility makes no changes to IDM static configuration; if properties are specified as configuration expressions, the configuration expressions are preserved in the IDM configuration.

In the AM configuration

The AM admin UI is *not* aware of configuration expressions.

Properties cannot be specified as configuration expressions in the AM admin UI; they must be specified as string values. The string values are preserved in the AM configuration.

AM supports specifying configuration expressions in both static and dynamic configuration.

AM static configuration export

The `forgeops` repository's bin/config export am command exports AM static configuration from running ForgeOps deployments to your `forgeops` repository clone. All AM static configuration properties, including passwords, have string values. However, after the config utility copies the AM static configuration from the `forgeops` repository, it calls the AM configuration upgrader. The upgrader transforms the AM configuration, following rules in the etc/am-upgrader-rules/placeholders.groovy file.

These rules tell the upgrader to convert a number of string values in AM static configuration to configuration expressions. For example, there are rules to convert all the passwords in AM static configuration to configuration expressions.

You'll need to modify the etc/am-upgrader-rules/placeholders.groovy file if:

- You add AM static configuration that contains new passwords.
- You want to change additional properties in AM static configuration to use configuration expressions.

> **Note**
>
> An alternative to modifying the etc/am-upgrader-rules/placeholders.groovy file is using the jq command to modify the output from the config utility.

AM dynamic configuration export

The `forgeops` repository's forgeops amster export command exports AM dynamic configuration from running ForgeOps deployments to your `forgeops` repository clone. When dynamic configuration is exported, it contains properties with string values. The forgeops amster export command transforms values for several types of properties to configuration expressions:

- Passwords
- Fully-qualified domain names

- The Amster version

The Secret Agent configuration computes and propagates passwords for AM dynamic configuration. You'll need to modify the `kustomize/base/secrets/secret_agent_config.yaml` file if:

- You add new AM dynamic configuration that contains passwords to be generated.
- You want to hard code a specific value for an existing password, instead of using a generated password.

Limitations on property value substitution in AM

AM doesn't support property value substitution for several types of configuration properties. Refer to Property value substitution in the AM documentation for more information.

Next step

Perform additional setup

Understand custom images

Customize the DS image

Understand AM and IDM configuration

Understand property value substitution

- ☐ Customize the AM image
- ☐ Customize the IDM image

`am` image

The `am` Docker image contains the AM configuration.

Customization overview

- Customize AM's configuration data by using the AM admin UI and REST APIs.
- Capture changes to the AM configuration by exporting them from the AM service running on Kubernetes to the staging area.
- Save the modified AM configuration to a configuration profile in your `forgeops` repository clone.
- Build an updated `am` Docker image that contains your customizations.
- Redeploy AM.
- Verify that changes you've made to the AM configuration are in the new Docker image.

Detailed steps

1. Verify that:
    - You have access to a single-instance ForgeOps deployment.
    - The namespace where the platform is deployed is set in your Kubernetes context.
    - All required third-party software is installed in your local environment (Minikube | GKE | EKS | AKS).
    - You have set up your environment to push to your Docker registry.
2. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. Review the state of the docker/am/config-profiles/my-profile directory.
    3. (Optional) Run the git commit command to commit changes to files that have been modified.
3. Modify the AM configuration using the AM admin UI or the REST APIs.

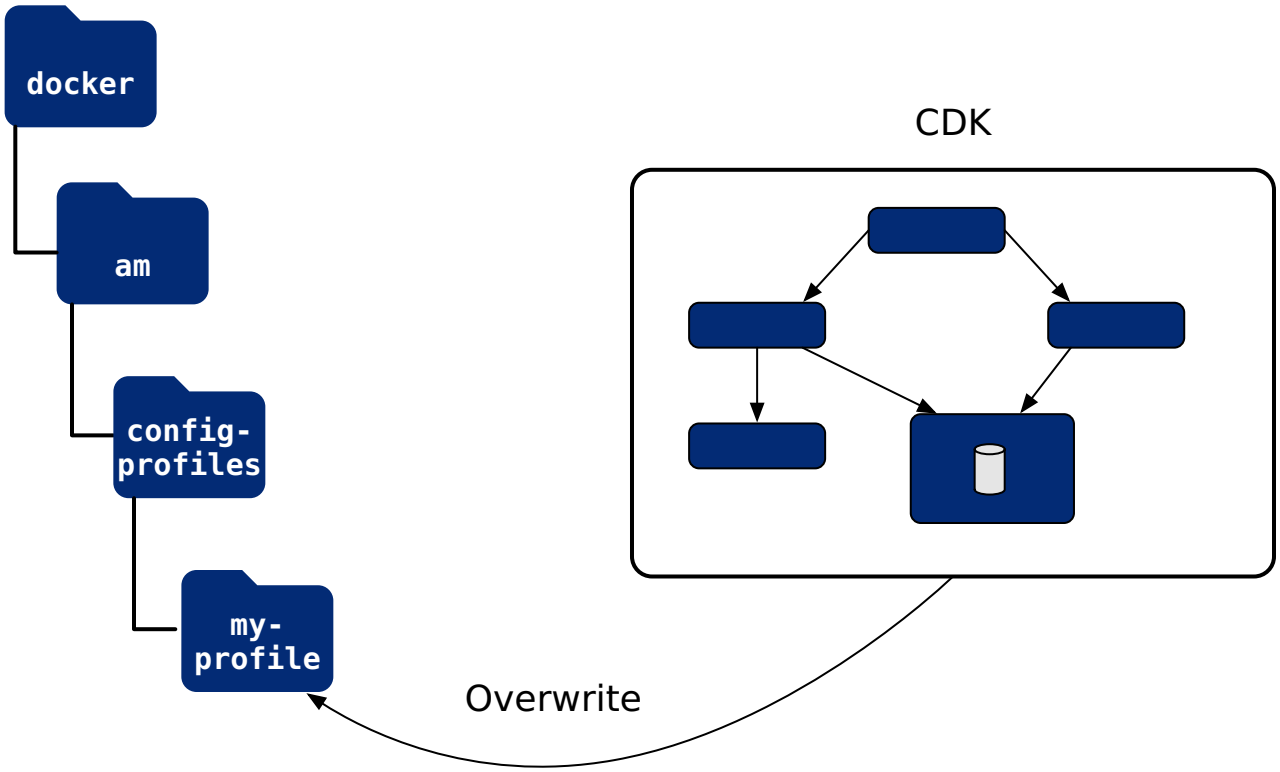
    You can find more information about how to access the AM admin UI or REST APIs in AM Services.

    You can find important information about configuring values that vary at run-time, such as passwords and host names in About property value substitution.
4. Export the changes you made to the AM configuration in the running ForgeOps deployment to a configuration profile:

```
$ cd /path/to/forgeops/bin
$ ./config export am my-profile --sort
[INFO] Running export for am in am-6fb64659f-bmdhh
[INFO] Updating existing profile: /path/to/forgeops/docker/am/config-profiles/my-profile
[INFO] Clean profile: /path/to/forgeops/docker/am/config-profiles/my-profile
[INFO] Exported AM config
[INFO] Running AM static config through the am-config-upgrader to upgrade to the current version of
forgeops.

+ docker run --rm --user 502:20 --volume /path/to/forgeops/docker/am/config-profiles/my-profile:/am-
config ...

Reading existing configuration from files in /am-config/config/services...
Modifying configuration based on rules in [/rules/latest.groovy]...
reading configuration from file-based config files
Writing configuration to new location at /am-config/config/services...
Upgrade Completed, modified configuration saved to /am-config/config/services
[INFO] Completed upgrading AM configuration
[INFO] Running AM static config through the am-config-upgrader to replace any missing default
placeholders.

+ docker run --rm --user 502:20 --volume /path/to/forgeops/docker/am/config-profiles/my-profile:/am-
config
...

Reading existing configuration from files in /am-config/config/services...
Modifying configuration based on rules in [/rules/placeholders.groovy]...
reading configuration from file-based config files
...
Writing configuration to new location at /am-config/config/services...
Upgrade Completed, modified configuration saved to /am-config/config/services
[INFO] Completed replacing AM placeholders
[INFO] Completed export
[INFO] Sorting configuration.
[INFO] Sorting completed.
```

If the configuration profile doesn't exist yet, the config export command creates it.

The config export am my-profile command copies AM static configuration from the ForgeOps deployment to the configuration profile:



5. Perform version control activities on your `forgeops` repository clone:

    1. Review the differences in the files you exported to the configuration profile. For example:

```
$ git diff
diff --git a/docker/am/config-profiles/my-profile/config/services/realm/root/selfservicetrees/
1.0/organizationconfig/default.json b/docker/am/config-profiles/my-profile/config/services/
realm/root/selfservicetrees/1.0/organizationconfig/default.json
index 970c5a257..19f4f17f0 100644
--- a/docker/am/config-profiles/my-profile/config/services/realm/root/selfservicetrees/1.0/
organizationconfig/default.json
+ b/docker/am/config-profiles/my-profile/config/services/realm/root/selfservicetrees/1.0/
organizationconfig/default.json
@@ -9,6 +9,7 @@
     "enabled": true,
     "treeMapping": {
       "Test": "Test",
+      "Test1": "Test1",
       "forgottenUsername": "ForgottenUsername",
       "registration": "Registration",
       "resetPassword": "ResetPassword",
```

    Note that if this is the first time that you have exported AM configuration changes to this configuration profile, the git diff command will not show any changes.

    2. Run the git status command.

    3. If you have new untracked files in your clone, run the git add command.

    4. Review the state of the docker/am/config-profiles/my-profile directory.

5. (Optional) Run the git commit command to commit changes to files that have been modified.

6. Identify the repository to which you'll push the Docker image. You'll use this location to specify the --push-to argument value in the build am image step.

7. Decide on the image tag name to tag each build of the image. You'll use this tag name to specify the --tag argument in the build am image step.

8. Build a new `am` image that includes your changes to AM static configuration:

> **Note**
>
> While the forgeops build command uses the Docker engine by default for ForgeOps deployments, it supports Podman as well. If you are using Podman engine instead of Docker in your environment, then set the `CONTAINER_ENGINE` environment variable to `podman` before running the forgeops build command, for example:
>
> ```
> $ export CONTAINER_ENGINE="podman"
> ```

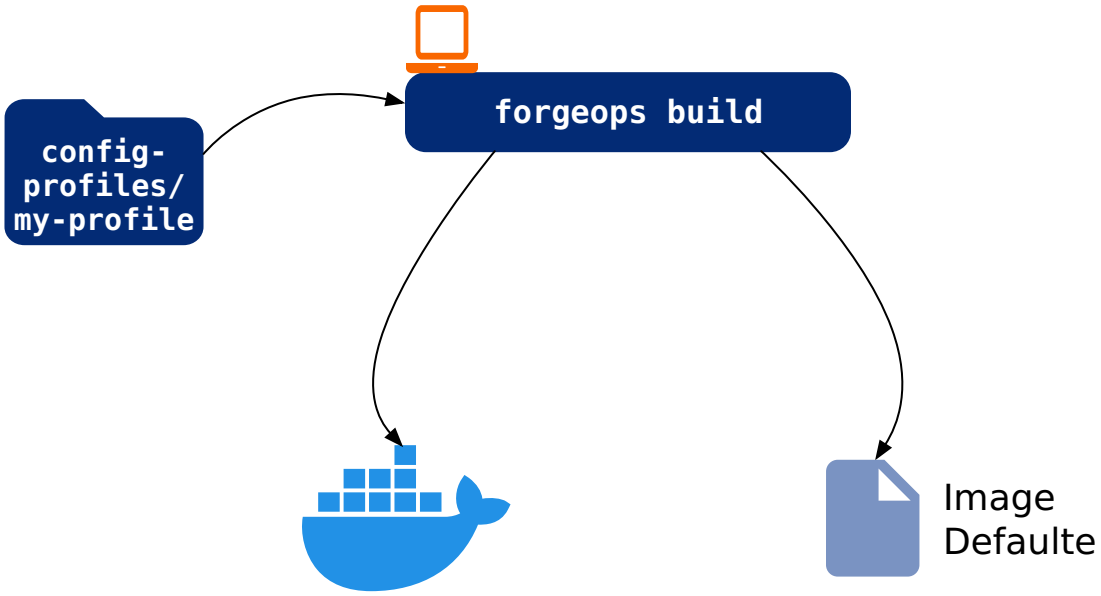
```
$ ./forgeops build am --env-name my-env --config-profile my-profile --push-to my-repo --tag my-am-tag
Flag --short has been deprecated, and will be removed in the future.
[+] Building 3.2s (10/10) FINISHED
...
 ⇒ [5/5] WORKDIR /home/forgerock
 ⇒ exporting to image
 ⇒ ⇒ exporting layers
 ⇒ ⇒ writing image sha256:...
 ⇒ ⇒ naming to docker.io/library/am

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
Updated the image_defaulter with your new image for am: "am".
```

9. Redeploy AM using your new AM image:

   - If you installed the platform using the forgeops command, follow the steps in Redeploy AM: Kustomize deployments.
   - If you installed the platform using Helm, follow the steps in Redeploy AM: Helm deployments.

Redeploy AM: Kustomize deployments

The forgeops build command calls Docker to build a new `am` Docker image and to push the image to your Docker repository. The new image includes your configuration profile. It also updates the image defaulter file so that the next time you install AM, the forgeops apply command gets AM static configuration from your new custom Docker image.



1. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. Review the state of the kustomize/overlay/default/image-defaulter/kustomization.yaml file.
    3. (Optional) Run the git commit command to commit changes to the image defaulter file.
2. Remove AM from your ForgeOps deployment:

```
$ ./forgeops delete am --env-name my-env
... platform detected in namespace: "my-namespace".
Uninstalling component(s): ['am'] from namespace: "my-namespace".
OK to delete components? [Y/N] Y
service "am" deleted
deployment.apps "am" deleted
```

3. Redeploy AM:

```
$ ./forgeops apply am --env-name my-env --single-instance
Checking cert-manager and related CRDs: cert-manager CRD found in cluster.
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster

Installing component(s): ['am'] ... from deployment manifests in ...

service/am created
deployment.apps/am created

Enjoy your deployment!
```

4. Validate that AM has the expected configuration:
    - Run the kubectl get pods command to monitor the status of the AM pod. Wait until the pod is ready before proceeding to the next step.
    - Describe the AM pod. Locate the tag of the Docker image that Kubernetes loaded, and verify that it's your new custom Docker image's tag.
    - Start the AM admin UI and verify that your configuration changes are present.

Redeploy AM: Helm deployments

1. Locate the `Successfully tagged` message in the forgeops build output, which contains the new AM Docker image's repository and tag.
2. Redeploy AM using the new AM Docker image:

```
$ cd /path/to/forgeops/charts/identity-platform
$ helm upgrade identity-platform ./ \
 --namespace my-namespace \
 --set 'am.image.repository=my-repository' \
 --set 'am.image.tag=my-am-tag' \
 --values /path/to/forgeops/helm/my-env/values.yaml
```

3. Validate that AM has the expected configuration:
    - Run the kubectl get pods command to monitor the status of the AM pod. Wait until the pod is ready before proceeding to the next step.
    - Describe the AM pod. Locate the tag of the Docker image that Kubernetes loaded, and verify that it's your new custom Docker image's tag.
    - Start the AM admin UI and verify that your configuration changes are present.

Next step

Perform additional setup

Understand custom images

Customize the DS image

Understand AM and IDM configuration

Understand property value substitution

Customize the AM image

☐ Customize the IDM image

`idm` image

The `idm` Docker image contains the IDM configuration.

Customization overview

- Customize IDM's configuration data by using the IDM admin UI and REST APIs.
- Capture changes to the IDM configuration by exporting them from the IDM service running on Kubernetes to the staging area.
- Save the modified IDM configuration to a configuration profile in your `forgeops` repository clone.
- Build an updated `idm` Docker image that contains your customizations.
- Redeploy IDM.
- Verify that changes you've made to the IDM configuration are in the new Docker image.

Detailed steps

1. Verify that:
    - You have access to a single-instance ForgeOps deployment.
    - The namespace where the platform is deployed is set in your Kubernetes context.
    - All required third-party software is installed in your local environment (Minikube | GKE | EKS | AKS).
    - You have set up your environment to push to your Docker registry.
2. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. Review the state of the docker/idm/config-profiles/my-profile directory.
    3. (Optional) Run the git commit command to commit changes to files that have been modified.
3. Modify the IDM configuration using the IDM admin UI or the REST APIs.

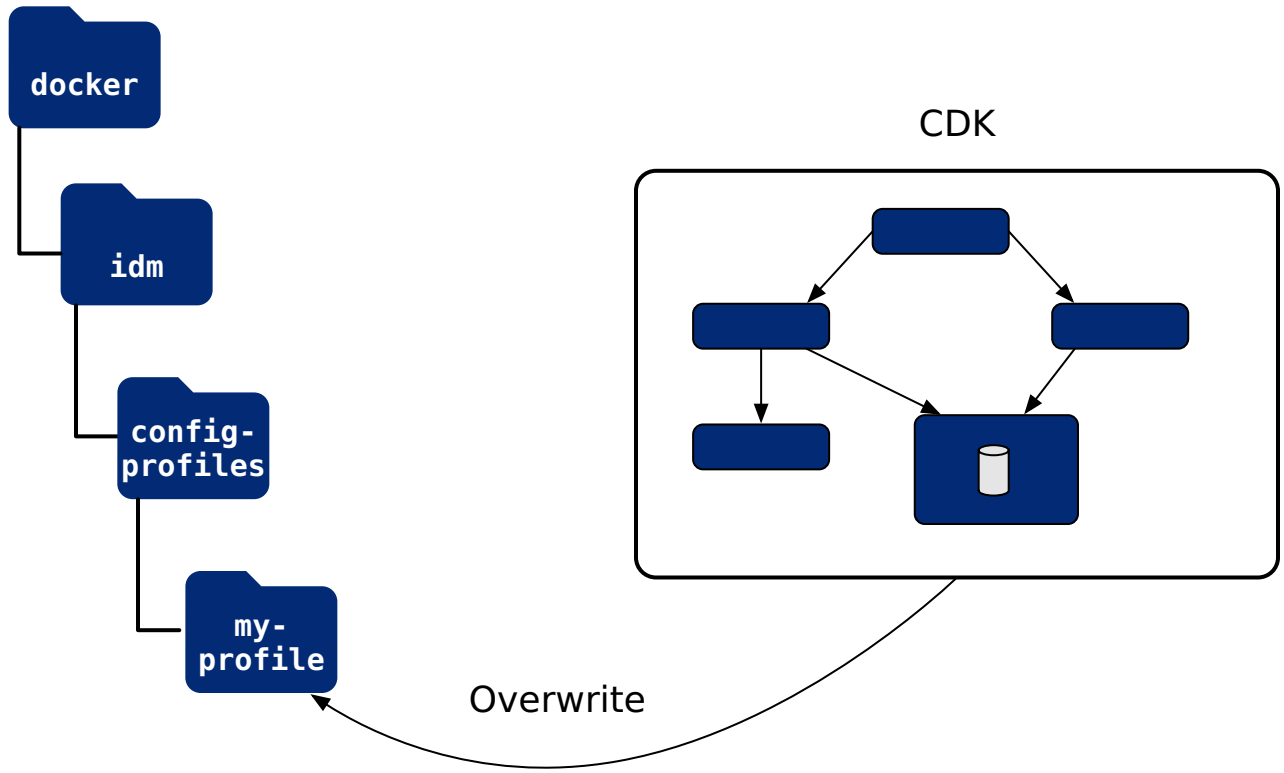
    For information about how to access the IDM admin UI or REST APIs, refer to IDM Services.

    Refer to About property value substitution for important information about configuring values that vary at run-time, such as passwords and host names.
4. Export the changes you made to the IDM configuration in the running ForgeOps deployment to a configuration profile:

```
$ cd /path/to/forgeops/bin
$ ./config export idm my-profile --sort
[.cyan][INFO] Running export for idm in idm-6b9db8cd7c-s7d46
[INFO] Updating existing profile: /path/to/forgeops/docker/idm/config-profiles/my-profile/conf
[INFO] Creating a new profile: /path/to/forgeops/docker/idm/config-profiles/my-profile/ui/admin/
default/config#
tar: Removing leading `/' from member names
[INFO] Completed export
[INFO] Sorting configuration.
[INFO] Sorting completed.
```

    If the configuration profile doesn't exist yet, the config export command creates it.

The config export idm my-profile command copies IDM static configuration from the ForgeOps deployment to the configuration profile:



5. Perform version control activities on your `forgeops` repository clone:

    1. Review the differences in the files you exported to the configuration profile. For example:

```
$ git diff
diff --git a/docker/idm/config-profiles/my-profile/conf/audit.json b/docker/idm/config-
profiles/my-profile/conf/audit.json
index 0b3dbeed6..1e5419eeb 100644
--- a/docker/idm/config-profiles/my-profile/conf/audit.json
+ b/docker/idm/config-profiles/my-profile/conf/audit.json
@@ -135,7 +135,9 @@
   },
   "exceptionFormatter": {
     "file": "bin/defaults/script/audit/stacktraceFormatter.js",
-    "globals": {},
+    "globals": {
+      "Test": "Test value"
+    },
     "type": "text/javascript"
   }
 }
```

    Note that if this is the first time that you have exported IDM configuration changes to this configuration profile, the git diff command will not show any changes.

    2. Run the git status command.

3. If you have new untracked files in your clone, run the git add command.
4. Review the state of the docker/idm/config-profiles/my-profile directory.
5. (Optional) Run the git commit command to commit changes to files that have been modified.

6. Identify the repository to which you'll push the Docker image. You'll use this location to specify the --push-to argument value in the build idm image step.
7. Decide on the image tag name so you can tag each build of the image. You'll use this tag name to specify the --tag argument value in the build idm image step.
8. Build a new `idm` image that includes your changes to IDM static configuration:

> **Note**
>
> While the forgeops build command uses the Docker engine by default for ForgeOps deployments, it supports Podman as well. If you are using Podman engine instead of Docker in your environment, then set the `CONTAINER_ENGINE` environment variable to `podman` before running the forgeops build command, for example:
>
> ```
> $ export CONTAINER_ENGINE="podman"
> ```

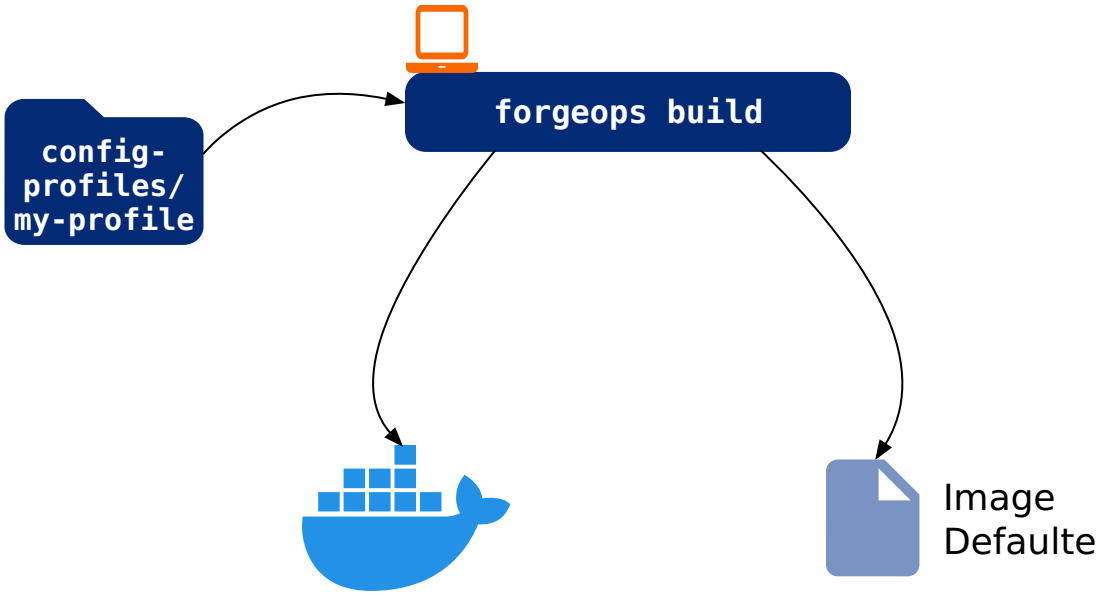
```
$ ./forgeops build idm --env-name my-env --config-profile my-profile --push-to my-repo --tag my-idm-tag

Flag --short has been deprecated, and will be removed in the future.

[+] Building 3.3s (12/12) FINISHED                              docker:default
 ⇒ [internal] load build definition from Dockerfile
⇒⇒ transferring dockerfile: 1.09kB
...
 ⇒ [internal] load metadata for us-docker.pkg.dev/forgeops-public/images-base/
idm:...                                                  2.0s
 ⇒ [internal] load build
context                                                                  0.1s
⇒⇒ transferring context: 563.76kB
 ...
 ⇒ [7/7] COPY --chown=forgerock:root  /opt/openidm
 ⇒ exporting to image
⇒⇒ exporting layers
⇒⇒ writing image
⇒⇒ naming to docker.io/library/idm

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
Updated the image_defaulter with your new image for idm: "idm".
```

9. Redeploy IDM using your new IDM image:
   - If you installed the platform using the forgeops command, follow the steps in Redeploy IDM: Kustomize deployments.

- If you installed the platform using Helm, follow the steps in Redeploy IDM: Helm deployments.

Redeploy IDM: Kustomize deployments

The forgeops build command calls Docker to build a new `idm` Docker image and to push the image to your Docker repository. The new image includes your configuration profile. It also updates the image defaulter file so that the next time you install IDM, the forgeops apply command gets IDM static configuration from your new custom Docker image.



1. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. Review the state of the kustomize/overlay/default/image-defaulter/kustomization.yaml file.
    3. (Optional) Run the git commit command to commit changes to the image defaulter file.
2. Remove IDM from your ForgeOps deployment:

```
$ ./forgeops delete idm --env-name my-env
"cdk" platform detected in namespace: "my-namespace".
Uninstalling component(s): ['idm'] from namespace: "my-namespace".
OK to delete components? [Y/N] Y
service "idm" deleted
deployment.apps "idm" deleted
```

3. Redeploy IDM:

```
$ ./forgeops apply idm --env-name my-env --single-instance
Checking cert-manager and related CRDs: cert-manager CRD found in cluster.
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster

Installing component(s): ['idm'] platform: ...

configmap/idm created
configmap/idm-logging-properties created
service/idm created
deployment.apps/idm created

Enjoy your deployment!
```

4. Validate that IDM has the expected configuration:
    - Run the kubectl get pods command to monitor the status of the IDM pod. Wait until the pod is ready before proceeding to the next step.
    - Describe the IDM pod. Locate the tag of the Docker image that Kubernetes loaded, and verify that it's your new custom Docker image's tag.
    - Start the IDM admin UI and verify that your configuration changes are present.

Redeploy IDM: Helm deployments

1. Locate the `Successfully tagged` message in the forgeops build output, which contains the new IDM Docker image's repository and tag.
2. Redeploy IDM using the new IDM Docker image:

```
$ cd /path/to/forgeops/charts/identity-platform
$ helm upgrade identity-platform ./ \
 --namespace my-namespace \
 --set 'idm.image.repository=my-repository' \
 --set 'idm.image.tag=my-idm-tag' \
 --values /path/to/forgeops/helm/my-env/values.yaml
```

3. Validate that IDM has the expected configuration:
    - Run the kubectl get pods command to monitor the status of the AM pod. Wait until the pod is ready before proceeding to the next step.
    - Describe the IDM pod. Locate the tag of the Docker image that Kubernetes loaded, and verify that it's your new custom Docker image's tag.
    - Start the IDM admin UI and verify that your configuration changes are present.

Next step

Perform additional setup

Understand custom images

Customize the DS image

Understand AM and IDM configuration

Understand property value substitution

Customize the AM image

Customize the IDM image

Customized Docker images

ForgeOps provides 12 Docker images for deploying the Ping Identity Platform:

- Eight component base images:
  - `amster`
  - `am-cdk`
  - `am-config-upgrader`
  - `ds`
  - `idm-cdk`
  - `ig`
  - `java-17` and `java-21`
- Four base images that implement the platform's user interface elements and ForgeOps operators:
  - `platform-admin-ui`
  - `platform-enduser-ui`
  - `platform-login-ui`
  - `secret-agent`

Note

Before you begin building custom images, ensure that you have installed Java version 21 or 17 on your computer. For example:

```
$ java --version
openjdk 21.0.5 2024-10-15 LTS
OpenJDK Runtime Environment Temurin-21.0.5+11 (build 21.0.5+11-LTS)
OpenJDK 64-Bit Server VM Temurin-21.0.5+11 (build 21.0.5+11-LTS, mixed mode)
```

```
$ java --version
openjdk 17.0.10 2024-01-16
OpenJDK Runtime Environment Temurin-17.0.10+7 (build 17.0.10+7)
OpenJDK 64-Bit Server VM Temurin-17.0.10+7 (build 17.0.10+7, mixed mode)
```

Building deployable ForgeOps Docker images

1. Set up your local ForgeOps deployment environment using the forgeops env command.

```
$ ./bin/forgeops env --env-name my-env
Updating existing overlay.
Helm environment dir exists, but has no values.yaml.

When creating a new environment, it's best practice to specify a HTTPS
certificate issuer (--issuer or --cluster-issuer).
You can also skip issuer creation with --skip-issuer.
For demos, you can use 'bin/certmanager-deploy.sh' to deploy cert-manager and
create a self-signed ClusterIssuer called 'default-issuer'.


Continue using a ClusterIssuer called "default-issuer"? [Y/N] y
Using ClusterIssuer: default-issuer
```

2. Select the ForgeOps image release you want to use for building your images.

   The following example uses the 8.0.0 image release from ForgeOps and names locally as my-8.0.0:

```
$ ./bin/forgeops image --release 8.0.0 --release-name my-8.0.0 platform
...
Updating release file(s) for docker builds [my-8.0.0]
```

3. Copy your customized AM and IDM configuration profiles to the docker/am/config-profiles and docker/idm/config-profiles directories respectively.

   If you don't have a ForgeOps deployment, you may not have customized configuration profiles. So you can ignore this step to create the first ForgeOps deployment.

4. Build your custom docker images. Use the `--push-to` option of the `forgeops build` command to push the customized images to your Docker repository.

```
$ ./bin/forgeops build --env-name my-env \
  --release-name my-8.0.0 \
  --config-profile my-profile --push-to my-repo platform
```

   If you don't have customized configuration profiles, then you don't specify the `--config-profile my-profile` option.

   You can use the `--dryrun` option to validate your `forgeops build` command before actual execution. For example:

```
$ ./bin/forgeops build --env-name my-env --release-name my-8.0.0 platform --dryrun
...
Component 'platform' given, setting components
docker build --build-arg REPO=us-docker.pkg.dev/forgeops-public/images-base/am --build-arg TAG=8.0.0
-t am -f .../forgeops/docker/am/Dockerfile .../forgeops/docker/am
.../forgeops/bin/commands/image -e my-env -k .../forgeops/kustomize -H .../forgeops/helm --image-
repo none -b .../forgeops/docker am
docker build --build-arg REPO=us-docker.pkg.dev/forgeops-public/images-base/idm --build-arg
TAG=8.0.0 -t idm -f .../forgeops/docker/idm/Dockerfile .../forgeops/docker/idm
.../forgeops/bin/commands/image -e my-env -k .../forgeops/kustomize -H .../forgeops/helm --image-
repo none -b .../forgeops/docker idm
docker build --build-arg REPO=us-docker.pkg.dev/forgeops-public/images-base/ds --build-arg TAG=8.0.0
-t ds -f .../forgeops/docker/ds/Dockerfile .../forgeops/docker/ds
.../forgeops/bin/commands/image -e my-env -k .../forgeops/kustomize -H .../forgeops/helm --image-
repo none -b .../forgeops/docker ds
docker build --build-arg REPO=us-docker.pkg.dev/forgeops-public/images-base/amster --build-arg
TAG=8.0.0 -t amster -f .../forgeops/docker/amster/Dockerfile .../forgeops/docker/amster
.../forgeops/bin/commands/image -e my-env -k .../forgeops/kustomize -H .../forgeops/helm --image-
repo none -b .../forgeops/docker amster
```

5. Perform a ForgeOps deployment using your customized Docker images.

> **Note**
>
> If you have performed the first ForgeOps deployment, then you need to customize your configuration profiles and redo the steps from the Copying configuration files step and redeploy the ForgeOps platform with your configuration.

Prepare to deploy in production

Prepare to deploy in production

After you get your ForgeOps deployment up and running, you can add deployment customizations—options that are not part of an out-of-the-box ForgeOps deployment, but which you may need when you deploy in production.

Production Deployment Overview

Customize, deploy, and maintain a production ForgeOps deployment.

Identity Gateway

Add PingGateway to your deployment.

Monitoring

Customize Prometheus monitoring and alerts.

Security

Customize the security features built into ForgeOps deployments.

Backup

Back up and restore data, such as identities and tokens.

Identity Gateway

IG Deployment

Add PingGateway to a ForgeOps deployment.

Custom IG Image

Build a custom PingGateway image and add it to a single-instance ForgeOps deployment.

ForgeOps deployment monitoring

ForgeOps deployments optionally use Prometheus to monitor Ping Identity Platform components and Kubernetes objects, Prometheus Alertmanager to send alert notifications, and Grafana to analyze metrics using dashboards.

This topic describes the use of monitoring tools in ForgeOps deployments:

Overview

Monitoring installation and architecture.

Monitoring Pods

Prometheus and Grafana pods that monitor ForgeOps deployments and provide reporting services.

Grafana Dashboards

Grafana dashboards for the platform that are available in ForgeOps deployments.

Prometheus Alerts

Prometheus alerts for the platform that are available in ForgeOps deployments.

Security

This topic describes several options for securing a ForgeOps deployment:

Secret Agent

Kubernetes operator that generates secrets and provides cloud secret management.

Secure Communications

Secure HTTP and certificate management.

IP Address Restriction

Access restriction by incoming IP address, enforced by the Ingress-NGINX controller.

Network Policies

Secure cross-pod communications, enforced by Kubernetes network policies.

Cluster Access on AWS

User entries in the Amazon EKS authorization configuration map.

Back up and restore data

Backup and restore overview

ForgeOps deployments include two directory services:

- The `ds-idrepo` service, which stores identities, application data, and AM policies
- The `ds-cts` service, which stores AM Core Token Service data

Before deploying the Ping Identity Platform in production, create and test a backup plan that lets you recover these two directory services should you experience data loss.

Choose a backup solution

There are numerous options to implement data backup. ForgeOps deployments provide two solutions:

- Kubernetes volume snapshots
- The dsbackup utility

You can also use backup products from third-party vendors. For example:

- Backup tooling from your cloud provider. For example, Google backup for GKE.
- Third-party utilities, such as Velero, Kasten K10, TrilioVault, Commvault, and Portworx Backup. These third-party products are cloud-platform agnostic, and can be used across cloud platforms.

Your organization might have specific needs for its backup solution. Some factors to consider include:

- Does your organization already have a backup strategy for Kubernetes deployments? If it does, you might want to use the same backup strategy for your Ping Identity Platform deployment.
- Do you plan to deploy the platform in a hybrid architecture, where part of your deployment is on-premises and another part of it is in the cloud? If you do, then you might want to employ a backup strategy that lets you move around DS data most easily.
- When considering how to store your backup data, is cost or convenience more important to you? If cost is more important, then you might need to take into account that archival storage in the cloud is much less expensive than snapshot storage—ten times less expensive, as of this writing.
- If you're thinking about using snapshots for backup, are there any limitations imposed by your cloud provider that are unacceptable to you? Historically, cloud providers have placed quotas on snapshots. Check your cloud provider's documentation for more information.

Backup and restore using volume snapshots

Kubernetes volume snapshots provide a standardized way to create copies of persistent volumes at a point in time without creating new volumes. Backing up your directory data with volume snapshots lets you perform rapid recovery from the last snapshot point. Volume snapshot backups also facilitate testing by letting you initialize DS with sample data.

In ForgeOps deployments, the DS data, changelog, and configuration are stored in the same persistent volume. This ensures the volume snapshot captures DS data and changelog together.

Backup

Set up backup

Kustomize overlays and Helm values necessary for configuring volume snapshots are already provided, but they have not been enabled to take backup. The default volume snapshot setup takes snapshots of the `data-ds-idrepo-0` and `data-ds-cts-0` PVCs once a day.

Enable volume snapshot before deployment

You can enable volume snapshot when you set up an environment before performing a ForgeOps deployment. For example, to enable snapshot for both `idrepo` and `cts`:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name  my-env --fqdn my-fqdn \
  --namespace my-namespace --cluster-issuer my_issuer \
  --idrepo-snap-enable --cts-snap-enable
```

Enable volume snapshot in a ForgeOps deployment

To enable volume snapshots of DS data where ForgeOps has been deployed in my-namespace namespace:

1. Revise the environment to enable snapshot:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name  my-env --idrepo-snap-enable --cts-snap-enable
```

> **Note**
>
> If you want to enable snapshot for `idrepo` alone, don't specify `--cts-snap-enable` in the forgeops env command.

2. Apply the changes to your ForgeOps deployment:

    1. In a Kustomize-based deployment:

```
$ cd /path/to/forgeops/bin
$ ./forgeops apply --env-name  my-env
```

    2. In a Helm-based deployment:

```
$ cd /path/to/forgeops/charts/identity-platform
$ helm upgrade --install identity-platform ./ \
 --namespace my-namespace --values /path/to/forgeops/helm/my-env/values.yaml
```

You can view the volume snapshots that are available for restore, using this command:

```
$ kubectl get volumesnapshots --namespace my-namespace

NAME                                READYTOUSE   SOURCEPVC          SOURCESNAPSHOTCONTENT   RESTORESIZE
SNAPSHOTCLASS       SNAPSHOTCONTENT
          CREATIONTIME   AGE
ds-idrepo-snapshot-20231117-1320   true         data-ds-idrepo-0                           100Gi
ds-snapshot-class   snapcontent-be3f4a44-cfb2-4f68-aa2b-60902
bb44192   3h29m          3h29m
ds-idrepo-snapshot-20231117-1330   true         data-ds-idrepo-0                           100Gi
ds-snapshot-class   snapcontent-7bcf6779-382d-40e3-9c9f-edf31
c54768e   3h19m          3h19m
ds-idrepo-snapshot-20231117-1340   true         data-ds-idrepo-0                           100Gi
ds-snapshot-class   snapcontent-c9c88332-ad05-4880-bda7-48616
ec13579   3h9m           3h9m
ds-idrepo-snapshot-20231117-1401   true         data-ds-idrepo-0                           100Gi
ds-snapshot-class   snapcontent-1f3f4ce9-0083-447f-9803-f6b45
e03ac27   167m           167m
ds-idrepo-snapshot-20231117-1412   true         data-ds-idrepo-0                           100Gi
ds-snapshot-class   snapcontent-4c39c095-0891-4da8-ae61-fac78
c7147ff   156m           156m
```

Customize the backup schedule

When enabled, volume snapshots are created once every day by default and purged after three days. You can customize the backup schedules as required in your environment.

In a Kustomize-based deployment

To modify the default schedule and purge delay for the `idrepo` repository[1]:

1. In a terminal window, change to the path/to/idrepo directory.
2. Copy the schedule.yaml file to a temporary location, so you can restore if needed.
3. Edit the schedule.yaml file and set the `schedule` and `purge-delay` parameters as needed.
4. Run the kubectl apply command.

Examples for scheduling snapshots

- To schedule snapshots twice a day, at noon and midnight:

```
...
  spec:
    schedule: "0 0/12 * * *"
...
```

- To schedule snapshots every 8 hours:

```
...
  spec:
    schedule: "0 */8 * * *"
...
```

Examples for purging schedule

- To schedule purge after 4 days:

```
...
          env:
            - name: PURGE_DELAY
              value: "-4 day"
```

- To schedule purge after a week:

```
...
          env:
            - name: PURGE_DELAY
              value: "-7 day"
```

In a Helm-based deployment

To modify the default schedule and purge delay:

1. In a terminal window, change to the /path/to/forgeops/helm/my-env directory:

```
$ cd /path/to/forgeops/helm/my-env
```

2. Copy the values.yaml file, so you can restore it if required:

```
$ cp values.yaml /tmp/values.yaml
```

3. Edit the values.yaml file and set the `schedule` and `purge-delay` parameters as needed.

4. Run the helm upgrade command.

Examples for scheduling snapshots for the `idrepo` repository[2]

- To schedule snapshots twice a day, at noon and midnight:

```
...
ds-idrepo:
  ...
  snapshot:
     ...
     schedule: "0 0/12 * * *"
...
```

- To schedule snapshots every 8 hours:

```
...
ds-idrepo:
  ...
  snapshot:
     ...
     schedule: "0 */8 * * *"
...
```

Examples for purging schedule for the `idrepo` repository[2]

- To schedule purge after 4 days:

```
...
ds-idrepo:
  ...
  snapshot:
     ...
     purgeDelay: "-4 day"
...
```

- To schedule purge after a week:

```
...
ds-idrepo:
  ...
  snapshot:
     ...
     purgeDelay: "-7 day"
...
```

Restore from volume snapshot

The snapshot-restore.sh script lets you restore DS instances in a ForgeOps deployment. By default, this script restores a DS instance from the latest available snapshot.

There are two options when using the snapshot-restore.sh script to restore a DS from a volume snapshot:

- Full—Use the full option to fully restore a DS instance from a volume snapshot. When you specify this option, the DS is scaled down to 0 pods before restoring data. The data is restored to an existing PVC from a snapshot. This operation requires downtime.
- Selective—Use the selective option to restore a portion of DS data from volume snapshot. The selective restore creates a new temporary DS instance with a new DS pod. You can selectively export from the temporary DS pod and import into your functional DS instance. After restoring data, you can clean up the temporary resources.

The snapshot-restore.sh command is available in the `bin` directory of the `forgeops` repository. To learn more about the snapshot-restore.sh command and its options, run snapshot-restore.sh --help.

Restore examples

Trial run without actually restoring DS data

1. In a terminal window, change to the /path/to/forgeops/bin directory.
2. Set your Kubernetes context to the correct cluster and namespace.

3. Run the snapshot-restore.sh command with the `--dryrun` option:

```
$ ./snapshot-restore.sh --dryrun --namespace my-namespace full idrepo

./snapshot-restore.sh --dryrun --namespace my-namespace full idrepo
/usr/local/bin/kubectl apply -f /tmp/snapshot-restore-idrepo.20231121T23:03:15Z/sts-
restore.json -n my-namespace
/usr/local/bin/kubectl delete pvc data-ds-idrepo-0 -n my-namespace
/usr/local/bin/kubectl apply -f /tmp/snapshot-restore-idrepo.20231121T23:03:15Z/data-ds-
idrepo-0.json -n my-namespace
/usr/local/bin/kubectl apply -f /tmp/snapshot-restore-idrepo.20231121T23:03:15Z/sts.json -n
my-namespace
```

Full restore of the `idrepo` instance from the latest available volume snapshot

1. In a terminal window, change to the /path/to/forgeops/bin directory.
2. Set your Kubernetes context to the correct cluster and namespace.
3. Get a list of available volume snapshots:

```
$ kubectl get volumesnapshots --namespace my-namespace
```

4. Restore the full DS instance:

```
$ ./snapshot-restore.sh --namespace my-namespace full idrepo
```

5. Verify that DS data has been restored.

Selective restore from a specific volume snapshot and storing data in a user-defined storage path

1. In a terminal window, change to the /path/to/forgeops/bin directory.
2. Set your Kubernetes context to the correct cluster and namespace.
3. Get a list of available volume snapshots:

```
$ kubectl get volumesnapshots --namespace my-namespace
```

4. Perform a selective restore trial run:

```
$ ./snapshot-restore.sh --dryrun --path /tmp/ds-restore --snapshot ds-idrepo-
snapshot-20231121-2250 --namespace my-namespace selective idrepo

VolumeSnapshot ds-idrepo-snapshot-20231121-2250 is ready to use
/usr/local/bin/kubectl apply -f /tmp/ds-rest/sts-restore.json -n my-namespace
/usr/local/bin/kubectl apply -f /tmp/ds-rest/svc.json -n my-namespace
```

5. Perform a selective restore using a specific snapshot:

```
$ ./snapshot-restore.sh --path /tmp/ds-restore --snapshot ds-idrepo-snapshot-20231121-2250 --
namespace my-namespace selective idrepo

statefulset.apps/ds-idrepo-restore created
service/ds-idrepo configured
```

6. Verify that a new `ds-idrepo-restore-0` pod was created:

```
$ kubectl get pods
NAME                          READY   STATUS      RESTARTS   AGE
admin-ui-656db67f54-2brbf     1/1     Running     0          3h17m
am-7fffff59fd-mkks5           1/1     Running     0          107m
amster-hgkv9                  0/1     Completed   0          3h18m
ds-idrepo-0                   1/1     Running     0          39m
ds-idrepo-restore-0           1/1     Running     0          2m40s
end-user-ui-df49f79d4-n4q54   1/1     Running     0          3h17m
idm-fc88578bf-lqcdj           1/1     Running     0          3h18m
login-ui-5945d48fc6-ljxw2     1/1     Running     0          3h17m
```

> **Note**
>
> The `ds-idrepo-restore-0` pod is temporary and not to be used as a complete DS instance. You can export required data from the temporary pod, and import data into your functional DS instance.

7. Clean up resources from the selective restore:

```
$ ./snapshot-restore.sh clean idrepo

statefulset.apps "ds-idrepo-restore" deleted
persistentvolumeclaim "data-ds-idrepo-restore-0" deleted
```

1. Use similar steps to modify the schedule and purge delay for the `cts` repository
2. Change the `ds-cts` parameters to modify the schedule and purge delay for the `cts` repository

dsbackup utility">

dsbackup utility

This page provides instructions for backing up and restoring DS data in a ForgeOps deployment using the dsbackup utility.

Back up using the dsbackup utility

Before you can back up DS data using the dsbackup utility, you must set up a cloud storage container in Google Cloud Storage, Amazon S3, or Azure Blob Storage and configure a Kubernetes secret with the container's credentials in your ForgeOps deployment. Then, you schedule backups by running the ds-backup.sh script.

Set up cloud storage

Cloud storage setup varies depending on your cloud provider. Expand one of the following sections for provider-specific setup instructions:

Set up a Google Cloud Storage (GCS) bucket for the DS data backup and configure the ForgeOps deployment with the credentials for the bucket:

1. Create a Google Cloud service account with required privileges to write objects in a GCS bucket. For example, Storage Object Creator.
2. Add a key to the service account, and then download the JSON file containing the new key.
3. Configure a multi-region GCS bucket for storing DS backups:
    1. Create a new bucket, or identify an existing bucket to use.
    2. Note the bucket's **Link for gsutil** value.
    3. Grant permissions on the bucket to the service account you created in step 1.
4. Make sure your current Kubernetes context references the cluster and namespace where the DS pods are running.
5. Create the `cloud-storage-credentials` secret that contains credentials to write to cloud storage. The DS pods use this secret when performing backups.

    For `my-sa-credential.json`, specify the JSON file containing the service account's key:

    ```
    $ kubectl create secret generic cloud-storage-credentials \
     --from-file=GOOGLE_CREDENTIALS_JSON=/path/to/my-sa-credential.json
    ```

6. Restart the pods that perform backups so that DS can get the credentials needed to write to the backup location:

    ```
    $ kubectl delete pods ds-cts-0
    $ kubectl delete pods ds-idrepo-0
    ```

After the pods have restarted, you can schedule backups.

Set up an S3 bucket for the DS data backup and configure the ForgeOps deployment with the credentials for the bucket:

1. Create or identify an existing S3 bucket for storing the DS data backup and note the S3 link of the bucket.
2. Make sure your current Kubernetes context references the cluster and namespace where the DS pods are running.
3. Create the `cloud-storage-credentials` secret that contains credentials to write to the cloud storage. The DS pods use this secret when performing backups:

    ```
    $ kubectl create secret generic cloud-storage-credentials \
     --from-literal=AWS_ACCESS_KEY_ID=my-access-key \
     --from-literal=AWS_SECRET_ACCESS_KEY=my-secret-access-key
    ```

4. Restart the pods that perform backups so that DS can get the credentials needed to write to the backup location:

```
$ kubectl delete pods ds-cts-0
$ kubectl delete pods ds-idrepo-0
```

After the pods have restarted, you can schedule backups.

Set up an Azure Blob Storage container for the DS data backup and configure the ForgeOps deployment with the credentials for the container:

1. Create or identify an existing Azure Blob Storage container for the DS data backup. Learn more about how to create and use Azure Blob Storage in Quickstart: Create, download, and list blobs with Azure CLI.
2. Log in to Azure Container Registry:

```
$ az acr login --name my-acr-name
```

3. Get the full Azure Container Registry ID:

```
$ ACR_ID=$(az acr show --name my-acr-name --query id | tr -d '"')
```

With the full registry ID, you can connect to a container registry even if you are logged in to a different Azure subscription.

4. Add permissions to connect your AKS cluster to the container registry:

```
$ az aks update --name my-aks-cluster-name --resource-group my-cluster-resource-group --attach-acr
$ACR_ID
```

5. Make sure your current Kubernetes context references the cluster and namespace where the DS pods are running.
6. Create secrets that contain credentials to write to cloud storage. The DS pods use these when performing backups:
    1. Get the name and access key of the Azure storage account for your storage container[1].
    2. Create the `cloud-storage-credentials` secret:

```
$ kubectl create secret generic cloud-storage-credentials \
 --from-literal=AZURE_STORAGE_ACCOUNT_NAME=my-storage-account-name \
 --from-literal=AZURE_ACCOUNT_KEY=my-storage-account-access-key
```

7. Restart the pods that perform backups so that DS can get the credentials needed to write to the backup location:

```
$ kubectl delete pods ds-cts-0
$ kubectl delete pods ds-idrepo-0
```

After the pods have restarted, you can schedule backups.

Schedule backups

1. Make sure you've set up cloud storage for your cloud provider platform.
2. Make sure your current Kubernetes context references the cluster and namespace where the DS pods are running.
3. Make sure you've backed up and saved the shared master key and TLS key for the ForgeOps deployment.
4. Set variable values in the /path/to/forgeops/bin/ds-backup.sh script:

Variable Name	Default	Notes
`HOSTS`	`ds-idrepo-2`	The `ds-idrepo` or `ds-cts` replica or replicas to back up. Specify a comma-separated list to back up more than one replica. For example, to back up the `ds-idrepo-2` and `ds-cts-2` replicas, specify `ds-idrepo-2,ds-cts-2`.
`BACKUP_SCHEDULE_IDREPO`	On the hour and half hour	How often to run backups of the `ds-idrepo` directory. Specify using cron job format.
`BACKUP_DIRECTORY_IDREPO`	n/a	Where the `ds-idrepo` directory is backed up. Specify: ◦ `gs://bucket/path` to back up to Google Cloud Storage ◦ `s3://bucket/path` to back up to Amazon S3 ◦ `az://container/path` to back up to Azure Blob Storage
`BACKUP_SCHEDULE_CTS`	On the hour and half hour	How often to run backups of the `ds-cts` directory. Specify using cron job format.
`BACKUP_DIRECTORY_CTS`	n/a	Where the `ds-cts` directory is backed up. Specify: ◦ `gs://bucket/path` to back up to Google Cloud Storage ◦ `s3://bucket/path` to back up to Amazon S3 ◦ `az://container/path` to back up to Azure Blob Storage

5. Run the ds-backup.sh create command to schedule backups:

```
$ /path/to/forgeops/bin/ds-backup.sh create
```

The first backup is a full backup; all later backups are incremental from the previous backup.

By default, the ds-backup.sh create command configures:

- The backup task name to be `recurringBackupTask`
- The backup tasks to back up all DS backends

If you want to change either of these defaults, configure variable values in the ds-backup.sh script.

> **Note**
>
> To cancel a backup schedule, run the ds-backup.sh cancel command.

Restore

This section covers three options to restore data from dsbackup backups:

- New ForgeOps deployment using DS backup
- Restore all DS directories
- Restore one DS directory

New ForgeOps deployment using DS backup

Creating new instances from previously backed up DS data is useful when a system disaster occurs or when directory services are lost. It is also useful when you want to port test environment data to a production deployment.

To create new DS instances with data from a previous backup:

1. Make sure your current Kubernetes context references the new ForgeOps cluster. Also make sure that the namespace of your Kubernetes context contains the DS pods into which you plan to load data from backup.
2. Create Kubernetes secrets containing your cloud storage credentials:

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --from-file=GOOGLE_CREDENTIALS_JSON=/path/to/my-sa-credential.json
   ```

   In this example, specify the path and file name of the JSON file containing the Google service account key for my-sa-credential.json.

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --from-literal=AWS_ACCESS_KEY_ID=my-access-key \
    --from-literal=AWS_SECRET_ACCESS_KEY=my-secret-access-key
    --from-literal=AWS_REGION=my-region
   ```

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --from-literal=AZURE_STORAGE_ACCOUNT_NAME=my-storage-account-name \
    --from-literal=AZURE_ACCOUNT_KEY=my-storage-account-access-key
   ```

3. Configure the backup bucket location and enable the automatic restore capability:

In a Kustomize-based deployment

1. Change to the /path/to/forgeops/kustomize/base/kustomizeConfig directory.
2. Open the kustomization.yaml file.
3. Set the `DSBACKUP_DIRECTORY` parameter to the location of the backup bucket. For example:

   `DSBACKUP_DIRECTORY="gs://my-backup-bucket"`

   `DSBACKUP_DIRECTORY="s3://my-backup-bucket"`

   `DSBACKUP_DIRECTORY="az://my-backup-bucket"`
4. Set the `AUTORESTORE_FROM_DSBACKUP` parameter to `"true"`.

In a Helm-based deployment

1. Change to the /path/to/forgeops/charts/identity-platform directory.
2. Edit values.yaml file and set up `autoRestore`, `backupLocation`, and `backupHosts` parameters for `ds-idrepo` and `ds-cts`. For example to restore `ds-idrepo-2`:

```
...
ds_restore:
  autoRestore: true
  backupLocation: "gs://my-backup-bucket"
  backupHosts: "ds-idrepo-2"
...
```

```
...
ds_restore:
  autoRestore: true
  backupLocation: "s3://my-backup-bucket"
  backupHosts: "ds-idrepo-2"
...
```

```
...
ds_restore:
  autoRestore: true
  backupLocation: "az://my-backup-bucket"
  backupHosts: "ds-idrepo-2"
...
```

1. Then Deploy the platform.

When the platform is deployed, new DS pods are created, and the data is automatically restored from the most recent backup available in the cloud storage location you configured.

To verify that the data has been restored:

- Use the IDM UI or platform UI.
- Review the logs for the DS pods' `init` container. For example:

```
$ kubectl logs --container init ds-idrepo-0
```

Restore all DS directories

To restore all the DS directories in your ForgeOps deployment from backup:

1. Delete all the PVCs attached to DS pods using the kubectl delete pvc command.
2. Because PVCs might not get deleted immediately when the pods to which they're attached are running, stop the DS pods.

   Using separate terminal windows, stop every DS pod using the kubectl delete pod command. This deletes the pods and their attached PVCs.

   Kubernetes automatically restarts the DS pods after you delete them. The automatic restore feature of ForgeOps deployments recreates the PVCs as the pods restart by retrieving backup data from cloud storage and restoring the DS directories from the latest backup.
3. After the DS pods come up, restart IDM pods to reconnect IDM to the restored PVCs:
   1. List all the pods in the namespace.
   2. Delete all the pods running IDM.

Restore one DS directory

In a ForgeOps deployment with automatic restore enabled, you can recover a failed DS pod if the latest backup is within the replication purge delay:

1. Delete the PVC attached to the failed DS pod using the kubectl delete pvc command.
2. Because the PVC might not get deleted immediately if the attached pod is running, stop the failed DS pod.

   In another terminal window, stop the failed DS pod using the kubectl delete pod command. This deletes the pod and its attached PVC.

   Kubernetes automatically restarts the DS pod after you delete it. The automatic restore feature recreates the PVC as the pod restarts by retrieving backup data from cloud storage and restoring the DS directory from the latest backup.
3. If the DS instance you restored was the `ds-idrepo` instance, restart IDM pods to reconnect IDM to the restored PVC:
   1. List all the pods in the namespace.
   2. Delete all the pods running IDM.

For information about manually restoring DS where the latest available backup is older than the replication purge delay, refer to the Restore section in the DS documentation.

Restore a PingDS deployment after a disaster

The PingDS disaster recovery involves additional steps beyond restoring a complete PingDS environment from backup. The dsrepl disaster-recovery must be run after a normal restore and before the PingDS server starts.

The disaster recovery process resets replication metadata to allow the newly restored version of the PingDS topology. The new topology is identified by a disaster recovery ID. The data pods not being restored have a different disaster recovery ID and don't exchange data with pods already recovered.

The disaster recovery process is automated in Forgeops. When a restore is initiated, the disaster recovery is also initiated using the disaster recovery ID defined in the configuration. If the disaster recovery ID matches the contents of the restored backup, the disaster recovery is stopped; otherwise, the data is disaster recovered.

The disaster recovery ID is configured in the `platform-config` configmap as follows:

- For Helm: update `ds_restore.disasterRecoveryId` in your custom values file
- For Kustomize: update DISASTER_RECOVERY_ID in your custom overlay in base/platform-config.yaml

Best practices for restoring directories

- Use a backup newer than the last replication purge.
- When you restore a single DS replica, the backup must be recent. Learn more at DS README.

1. To get the access key from the Azure portal, go to your storage account. Under Security + networking on the left navigation menu, select Access keys

Upgrade Overview

This section provides the conceptual and procedural details for upgrading your ForgeOps deployment environment.

> **Important**
>
> Because the Ping Identity Platform is highly customizable, testing all possible upgrade scenarios is challenging. It is your responsibility to validate that these upgrade steps work correctly in a test environment with your customized configuration before you upgrade a production environment.

Upgrading ForgeOps deployments involves three main sections:

Upgrading ForgeOps deployment tools from previous releases

- Migrate Kustomize overlays to the new format.
- Migrate from a ForgeOps 7.4 or 7.5 release branch to the 2025.1.x tag.

Upgrading AM, DS, or DS Docker images

- Upgrade the platform product Docker images to a new major or minor version.
- Upgrade AM, DS, or DS Docker images to newer patch release.

Upgrading Helm charts used in ForgeOps deployment

- Upgrade Helm charts

Migrate your Kustomize overlay content

This section covers steps required to migrate your Kustomize overlays from your 7.4 or 7.5 forgeops release branch to overlays in the new ForgeOps deployment environment.

> **Important**
>
> 1. If you are using DS Operator in your deployment, then use step 3 in Upgrade from version 7.3 to migrate away from using the DS Operator and then perform the migration.
> 2. When you migrate Kustomize overlays, **don't copy the files from the old overlay**. Create a new overlay and then copy the content in the old overlay to the new overlay.

The format and layout of the overlays in the new **main** branch have changed from the previous ForgeOps releases. The main changes are:

- Each overlay contains sub-overlays for each product. This enables users to deploy products individually or collectively just as with the previous version of the forgeops command.
- The `image-defaulter` is included in the overlay, so that it's specific for a deployment environment.
- Each product has a separate dedicated ingress file. This enables users to set up product-specific configurations if required. Therefore, set up the FQDN for your new environment using the forgeops env command. **Don't migrate the old ingress overlay.**

Considerations

Using the new forgeops command, you can select the version of products you want to deploy from 7.4 onwards. ForgeOps team recommends you migrate your deployment in the following way:

1. Migrate your overlay to the new overlay layout using the steps below.
2. Upgrade your images to a new version once your overlay is updated. Learn more at Migrate from a ForgeOps 7.4 or 7.5 release branch to the 2025.1.x tag.

Steps to migrate your overlay

To migrate your Kustomize overlays from previous versions, you need either of:

- Your custom overlay and the contents of kustomize/deploy/image-defaulter/kustomization.yaml, or
- Your custom deployment environment directory you have used to create a dedicated `image-defaulter` for your environment using the `--deploy-env` option.

Steps:

1. Ensure your custom overlay or custom deployment environment directory is saved locally so it is accessible when you check out the 2025.1.2 tag.
2. Check out the 2025.1.2 tag.

```
$ cd /path/to/forgeops/
$ git checkout 2025.1.2
```

3. Create a new custom overlay specifying your FQDN and the certificate issuer.

```
$ ./bin/forgeops env --e my-env --fqdn my-fqdn --cluster-issuer my-cluster-issuer
```

> **Important**
>
> 1. Specify your FQDN when creating a new custom overlay as it will populate the required manifests in the new overlay.
> 2. If you want to use a specific issuer for your deployment environment instead of the ClusterIssuer, then replace the `--cluster-issuer` option with `--issuer` option appropriately.

4. Copy the patch information from the previous custom overlay patch files or your deployment directory to the new overlay files.

   For example:

   - **Old overlay**: From `old-overlay/am.yaml` to `new-overlay/am/deployment.yaml`
   - **Environment directory**: From `deploy-custom/apps/am.yaml` to `new-overlay/am/deployment.yaml`

If you need to include additional patches, add them in to the corresponding sub-overlay and update the corresponding `kustomization.yaml` file to include them. The new forgeops command applies the overlays correctly during ForgeOps deployment unlike the forgeops command in previous releases that ignored `kustomization.yaml`.

Other things to watch out for

- Update the `base/base.yaml` file, and ensure that the FQDN is specified correctly.
- A separate ingress file exists for each product. The FQDN is populated in these files when you set up the deployment environment using the `forgeops env` command.
- Update your `image-defaulter/kustomization.yaml` in the new overlay with image URLs and images tags from your old `deploy/image-defaulter/kustomization.yaml` or your custom `my-env/image-defaulter/kustomization.yaml`.

Migrate from a ForgeOps 7.4 or 7.5 release branch to the 2025.1.x tag

If you've already installed Ping Identity Platform using the previous release branch of the `forgeops` repository, such as `release/7.4-20240126` or `release/7.5-20240608`, follow the steps provided on this page to upgrade to the latest platform 2025.1.x branch.

This upgrade methodology has been tested against a deployment based on ForgeOps-provided Docker images with basic configuration settings.

> **Important**
>
> Because the Ping Identity Platform is highly customizable, it is challenging to test all possible upgrade scenarios. It is your responsibility to validate that these upgrade steps work correctly in a test environment with your customized configuration before you upgrade a production environment.

Prerequisites and assumptions

If you've deployed the Ping Identity Platform from a previous release of ForgeOps, such as `release/7.4-20240126` or `release/7.5-20240608`:

- If you are using Kustomize to manage your ForgeOps deployment, Migrate your Kustomize overlay content first.
- You would have created your custom branch with the new ForgeOps release.
- Copy your product configuration profiles from your 7.4 or 7.5 release branch, for example: /path/to/forgeops/docker/am/config-profile/my-profile to the same location in your new custom branch.

To upgrade the platform from release 7.4 or 7.5 to 2025.1.x, you'll need:

- A running 7.4 or 7.5 release of ForgeOps deployment. If you need to port your AM custom configurations, then the running ForgeOps deployment should be a single-instance deployment with your AM and IDM configurations.
- A `forgeops` repository clone with a branch that contains 7.4 or 7.5 artifacts.
- A `forgeops` repository clone with a branch that contains 2025.1.x artifacts.

Example commands in the steps on this page assume:

- `7.4 or 7.5-profile` is the name of the 7.4 or 7.5 configuration profile.
- Your 7.4 or 7.5 ForgeOps deployment is a small cluster.
- Your 7.4 or 7.5 small, medium, or large ForgeOps deployment doesn't include PingGateway.

When you perform the upgrade:

- Choose a different name for the configuration profile if you prefer.
- Specify a different cluster size, if applicable.
- Add commands to upgrade PingGateway, if applicable.

Subscribe to release note updates

Get updates from ForgeOps when there are changes to ForgeOps 2025.1.2.

For more information about getting notifications or subscribing to the ForgeOps 2025.1.2 RSS feed, refer to ForgeOps `2025.1` release notes.

Back up critical data

Before upgrading, back up all critical data, including:

- Directory data stored in the `ds-idrepo` and `ds-cts` backends
- AM and IDM configuration data
- Customized artifacts in your `forgeops` repository clone

After you've started to upgrade, you might not be able to roll back directory data easily because the data is upgraded in place. If you need to roll back directory data, you'll have to redeploy DS and restore directory data from a backup. For a simpler restore scenario, consider backing up directory data on volume snapshots.

Create the new release in your `forgeops` branch

You can manage multiple releases in ForgeOps 2025.1.x using the forgeops image command. Learn more about the forgeops image command.

1. If you don't have the 7.4 or 7.5 release file for your 7.4 or 7.5 deployment, create a 7.4 or 7.5 release file in your `forgeops` branch. For example, to create the release file for 7.4.0 release:

   ```
   $ cd /path/to/forgeops
   $ ./bin/forgeops image --release 7.4.0 platform --release-name 7.4.0
   ```

   This is in case you need to roll back AM or IDM or you have configuration changes you wish to export from your single-instance environment.

2. Create a 2025.1.x release in docker/COMPONENT/releases/2025.1.x in your `forgeops` branch:

```
$ cd /path/to/forgeops
$ ./bin/forgeops image --release 2025.1.x platform --release-name 2025.1.x
```

3. Set the images in your environment to the new release:

```
$ ./bin/forgeops image --release 2025.1.x --env-name my-custom-env platform
```

Export the release 7.4 or 7.5 AM and IDM configurations

If you have AM or IDM configuration changes, in a single-instance deployment, that you haven't yet exported to a configuration profile:

1. Locate a branch of your `forgeops` repository clone that contains release 7.4 or 7.5 artifacts and check out the branch.
2. (Optional) Check out a new branch based on the branch that contains release 7.4 or 7.5 artifacts.
3. Locate a namespace running release 7.4 or 7.5 of the single-instance deployment that contains your AM and IDM configurations.
4. Export the AM and IDM configurations from the 7.4 or 7.5 single-instance deployment:

```
$ cd /path/to/forgeops
$ ./bin/config export am 7.4 or 7.5-profile --sort --release-name 7.4 or 7.5
$ ./bin/config export idm 7.4 or 7.5-profile --sort --release-name 7.4 or 7.5
```

> **Important**
>
> The `--release-name` option is required to ensure you use the release of the `am-config-upgrader` that matches your deployment. This only replaces any default config expressions that are lost during config updates in PingAM. It doesn't carry out any upgrades.

Build new images containing your ForgeOps configuration

1. Run the `am-config-upgrader` utility to upgrade the AM configuration to 2025.1.x:

```
$ cd /path/to/forgeops
$ ./bin/forgeops upgrade-am-config docker/am/config-profiles/my-config-profile --release-name
2025.1.x
```

2. Run the git add . and git commit commands.
3. Build Docker images for the newer patch release that contain your configuration profile:

```
$ cd /path/to/forgeops
$ ./bin/forgeops build am --config-profile my-config-profile \
  --env-name my-custom-env --release-name 2025.1.x --push-to my-repo  \
  --tag custom-am-tag

$ ./bin/forgeops build idm --config-profile my-config-profile \
  --env-name my-custom-env --release-name 2025.1.x \
  --push-to my-repo --tag custom-idm-tag
```

The newly built Docker images are based on ForgeOps-provided Docker images.

Upgrade the exported configuration profiles to release 2025.1.x

In Kustomize environment

1. Set your Kubernetes context to the cluster on which ForgeOps is deployed.
2. Upgrade the `ds-cts` pods to the new patch release.
    1. Run the `forgeops apply ds-cts` command to update `ds-cts` pods sequentially:

        ```
        $ cd /path/to/forgeops
        $ ./bin/forgeops apply ds-cts --env-name my-custom-env
        ```

    2. Run the `kubectl get pods --watch` command to observe the pod upgrades.
    3. After all the `ds-cts` pods have been upgraded, run the `ds-debug.sh` command to verify that directory replication is working correctly in each ds-cts pod:

        ```
        $ ./bin/ds-debug.sh --pod-name ds-cts-0 rstatus
        ```

3. Similarly, upgrade the `ds-idrepo` pods to the new patch release and verify that directory replication is working correctly in each `ds-idrepo` pod.
4. Upgrade all the Ping Identity Platform pods to the new patch release:

    ```
    $ ./bin/forgeops apply ui --env-name my-custom-env
    ```

    Wait for all the pods to be upgraded. Run the `kubectl get pods --watch` command to observe the progress of upgrade.
5. Start the admin UIs for AM and IDM in the upgraded deployment and verify that:
    - The start page for each admin UI displays the expected component release for the 2025.1.x release.
    - AM and IDM use your custom configuration.

In Helm environment

1. Set your Kubernetes context to the cluster on which ForgeOps is deployed.
2. Upgrade the platform:

```
$ cd /path/to/forgeops
$ helm upgrade --install identity-platform \
  oci://us-docker.pkg.dev/forgeops-public/charts/identity-platform \
  --version 2025.1.x --namespace my-namespace \
  --values helm/my-custom-env/values.yaml
```

3. After all the `ds-cts` pods have been upgraded, run the `ds-debug.sh` command to verify that directory replication is working correctly in each ds-cts pod:

```
$ ./bin/ds-debug.sh --pod-name ds-cts-0 rstatus
```

4. After the `ds-idrepo` pods have been upgraded, run the ds-debug.sh command to verify that directory replication is working correctly:

```
$ ./bin/ds-debug.sh --pod-name ds-idrepo-0 rstatus
```

5. Start the admin UIs for AM and IDM in the upgraded deployment and verify that:
    - The start page for each admin UI displays the expected component release for the 2025.1.x release.
    - AM and IDM use your custom configuration.

Rebuild your new images

If you are using ForgeOps deployment in production, you must rebuild base Docker images and custom Docker images for release 2025.1.x:

- Learn more about building base docker images in Your own base Docker images.
- Learn more about building your Docker images with custom configurations in Creating Docker images for use in production.

Upgrade the platform product Docker images to a new major or minor version

If you've performed ForgeOps deployment using the older AM, IDM, and DS Docker images, you should upgrade your ForgeOps deployment to use the newer version of platform product Docker images.

> **Note**
>
> Using this procedure, you can upgrade all the platform product Docker images sequentially, one at a time.

This upgrade methodology has been tested against a deployment based on ForgeOps-provided Docker images with basic configuration settings.

> **Important**
>
> Because the Ping Identity Platform is highly customizable, testing all possible upgrade scenarios is challenging. It is your responsibility to validate that these upgrade steps work correctly in a test environment with your customized configuration before you upgrade a production environment.

Prerequisites and assumptions

To upgrade platform products in a ForgeOps deployment to a newer release, you'll need:

- A `forgeops` repository clone of ForgeOps 2025.1.0 release tag or later.
- A running ForgeOps deployment environment, which has been configured using the forgeops env command.

Example commands in the steps on this page assume that your ForgeOps deployment:

- Is using the default configuration.
- Doesn't include PingGateway.

Back up critical data

Before upgrading, back up all critical data, including:

- Directory data stored in the `ds-idrepo` and `ds-cts` backends
- AM and IDM configuration data
- Customized artifacts in your `forgeops` repository clone

After you've started upgrading, you might not be able to roll back directory data easily because the data is upgraded in place. To roll back directory data, you must redeploy DS and restore directory data. Consider backing up directory data on volume snapshots for a simpler restore scenario.

Get ready to upgrade

1. Set your Kubernetes context to the cluster running your ForgeOps deployment.
2. View the list of supported product versions:

```
$ cd /path/to/forgeops
$ ./bin/forgeops info --list-releases
```

Upgrade the platform product images to a new major or minor version

> **Important**
>
> Amster and AM images need to be on the same version. So if you're upgrading AM, carry out the same steps to upgrade Amster.

1. Create a new release file for all the platform products. In the forgeops image command, specify:
   - The new product version in the `--release` flag, such as **7.5.1**.
   - The `platform` option to apply the product version to the whole platform.

```
$ cd /path/to/forgeops
$ ./bin/forgeops image --release 7.5.1 --release-name my-custom-release platform
```

2. Upgrade your custom AM configuration profile to the new version:

> **Note**
>
> Use the `--release-name` option to ensure you use the version of the `upgrade-am-config` that matches your deployment.

```
$ cd /path/to/forgeops
$ ./bin/forgeops upgrade-am-config --release-name my-custom-release \
 --config-profile docker/am/config-profiles/my-config-profile
```

3. Build new custom images for all platform products:

```
$ cd /path/to/forgeops
$ ./bin/forgeops build platform --env-name my-custom-env \
  --release-name my-custom-release \
  --config-profile docker/am/config-profiles/my-config-profile
```

4. Deploy your updated images for all platform products:

```
$ cd /path/to/forgeops
$ ./bin/forgeops apply --env-name my-custom-env platform
```

```
$ cd /path/to/forgeops/charts/identity-platform
$ helm upgrade --install identity-platform ./ \
 --values /path/to/forgeops/helm/my-env/values.yaml
```

Upgrade the platform UIs to a new version

This section refers to an upgrade to the platform UIs only.

You don't need to build new Docker images when you upgrade Platform UIs.

1. Update your environment to the new version. Specify the new version in the `--release` flag:

```
$ cd /path/to/forgeops
$ ./bin/forgeops image --release 8.0.0 ui --env-name my-custom-env
```

2. Deploy your updated version (Kustomize only)

```
$ cd /path/to/forgeops
$ ./bin/forgeops apply --env-name my-custom-env ui
```

Upgrade PingAM, PingIDM, or PingDS image to a newer patch release

Patched images are released for each platform product separately. So you may need to update each of PingAM, PingIDM, or PingDS images to a newer image patch release separately.

> **Important**
>
> Because the Ping Identity Platform is highly customizable, testing all possible upgrade scenarios is challenging. It is your responsibility to validate that these upgrade steps work correctly in a test environment with your customized configuration before you upgrade a production environment.

Prerequisites and assumptions

To upgrade PingAM, PingIDM, or PingDS image to a newer patch release, you'll need:

- A local clone of the ForgeOps repository.
- A running ForgeOps deployment deployed using ForgeOps 2025.1.0 or later.
- A configured ForgeOps deployment environment in your forgeops repository clone using the forgeops env command.

Example commands in this section assume that your ForgeOps deployment:

- Is using the default configuration.
- Doesn't include PingGateway.

Back up critical Directory data

If upgrading DS, back up all the directory data stored in the `ds-idrepo` and `ds-cts` backends. After you've started to upgrade, you can't roll back directory data changes easily because the data is upgraded in place. To roll back directory data, you must redeploy DS and restore directory data. Consider backing up directory data on volume snapshots for a simpler restore scenario.

> **Important**
>
> For upgrading a `dev` environment, ensure that you have exported your AM or IDM configuration changes to your custom configuration profile.

Get ready for upgrade

1. Set your Kubernetes context so that you can access the cluster which contains your ForgeOps deployment.
2. Check the current supported product versions available if required:

```
$ cd /path/to/forgeops
$ ./bin/forgeops info --list-releases
```

Upgrade a product to a newer patch release

This section covers the steps to upgrade AM, Amster, IDM, or DS to a new patch release.

> **Important**
>
> Amster and AM need to be on the same version. So if you're upgrading AM, carry out the same steps to upgrade Amster.

1. Create a new release in your ForgeOps repository clone that includes your customized configuration of the product to be updated, using one of the following options:
    1. To update to a new patch release use the forgeops image command and specify:
        - The new patch version in the `--release` flag.
        - Your current release in the `--release-name` flag.

        For example, to upgrade your AM image to 7.5.2 release:

        ```
        $ cd /path/to/forgeops
        $ ./bin/forgeops image --release 7.5.2 --release-name my-custom-release am
        ```

    2. To update to the latest secure image in your current release, use the forgeops image command and specify:
        - The product version you have deployed in the `--release` flag.
        - Specify your current release in the `--release-name` flag.

        > **Note**
        >
        > When you specify the current release in the forgeops image command, it selects the latest available secure image automatically.

        For example, to upgrade your AM image to the latest secure image of 7.5.1 release:

        ```
        $ cd /path/to/forgeops
        $ ./bin/forgeops image --release 7.5.1 --release-name my-custom-release am
        ```

2. If you're upgrading AM, upgrade your custom AM configuration profile to the new version.

> **Important**
>
> The `--release-name` option is required to ensure you use the version of the `am-config-upgrader` that matches your target AM version.

```
$ cd /path/to/forgeops
$ ./bin/forgeops upgrade-am-config --release-name my-custom-release \
 --config-profile docker/am/config-profiles/my-custom-release
```

3. Build your new custom image for the product you are upgrading.

> **Note**
>
> The `--config-profile` option isn't required to build DS image.

```
$ cd /path/to/forgeops
$ ./bin/forgeops build AM --env-name my-custom-env \
 --release-name my-custom-release --config-profile \
 docker/am/config-profiles/my-config-profile
```

4. Deploy your updated version.

```
$ cd /path/to/forgeops
$ helm upgrade --install identity-platform \
oci://us-docker.pkg.dev/forgeops-public/charts/identity-platform \
--version deployed version --namespace my-namespace \
--values helm/my-custom-env/values.yaml
```

```
$ cd /path/to/forgeops
$ ./bin/forgeops apply --env-name my-custom-env product
```

> **Note**
>
> In the forgeops apply command, specify the product, such as `am`, `idm`, or `ds` for product.

Upgrade the platform UIs to a newer patch version

Use the steps in this section to upgrade platform UIs in a ForgeOps deployment. Usually the new platform UI patch versions are available together, so the steps upgrade all the platform UIs together. **You don't need to build new Docker images when you upgrade Platform UIs**.

1. Upgrade your deployment environment to the new patch version.
    1. To upgrade to the new patch release of platform UIs, specify the new patch number in the `--release` flag of the forgeops image command. For example, to upgrade to the **7.5.2** version UIs:

```
$ cd /path/to/forgeops
$ ./bin/forgeops image --release 7.5.2 ui --env-name my-custom-env
```

2. To update to the latest platform UI secure image for the currently deployed release, specify the current platform UI version in the `--release` flag:

```
$ cd /path/to/forgeops
$ ./bin/forgeops image --release 7.5.1 ui --env-name my-custom-env
```

2. Deploy your updated patch image:

```
$ cd /path/to/forgeops
$ helm upgrade --install identity-platform \
oci://us-docker.pkg.dev/forgeops-public/charts/identity-platform \
--version deployed version --namespace my-namespace \
--values helm/my-custom-env/values.yaml
```

```
$ cd /path/to/forgeops
$ ./bin/forgeops apply --env-name [.var}#my-custom-env# ui
```

Update Helm Chart

In ForgeOps deployments using Helm chart version 2025.1.0 version or later, the customized values.yaml files are independent of the Helm chart versions. Therefore, you can update the version of a Helm chart and continue to work with your customized `values.yaml` files in your ForgeOps deployment environment.

Important

- The `values.yaml` files in ForgeOps deployments using 7.4 and 7.5 releases are not independent of the Helm chart versions. You cannot upgrade the Helm chart version in your ForgeOps deployment of 7.4 and 7.5 releases.
- Check the ForgeOps release notes to see what changes are in the new version of the Helm chart.

1. If you used the helm upgrade --install command to perform ForgeOps deployment, you can update the Helm chart version:

```
$ cd /path/to/forgeops
$ helm upgrade --install identity-platform \
  oci://us-docker.pkg.dev/forgeops-public/charts/identity-platform \
  --version new-version --namespace my-namespace \
  --values helm/my-custom-env/values.yaml
```

In the helm upgrade command, specify the new version of the Helm chart, such as 2025.1.1 for new-version.

Troubleshoot a deployment

Troubleshooting

Kubernetes deployments are multi-layered and often complex.

Errors and misconfigurations can crop up in a variety of places. Performing a logical, systematic search for the source of a problem can be daunting.

Here are some techniques you can use to troubleshoot problems with ForgeOps deployments:

Problem	Troubleshooting Technique
Some pods don't start.	Review Kubernetes logs and other diagnostics. Verify if your cluster is resource-constrained. Check for underconfigured clusters by using the `kubectl describe nodes` and `kubectl get events -w` commands. Pods killed with out of memory (OOM) conditions indicate that your cluster is underconfigured. Make sure that you're using tested versions of third-party software. Stage your installation. Install Ping Identity Platform components separately, instead of installing all the components with a single command. Staging your installation lets you make sure each component works correctly before installing the next component.
All the pods have started, but you can't reach the services running in them.	Make sure you don't have any ingress issues.
AM doesn't work as expected.	Set the AM logging level, recreate the issue, and analyze the AM log files. Turn on audit logging in AM.
IDM doesn't work as expected.	Set the IDM logging level, recreate the issue, and analyze the IDM log files. Turn on audit logging in IDM.
Your JVM crashed with an out of memory error or you suspect that you have a memory leak.	Collect and analyze Java thread dumps and heap dumps.

Problem	Troubleshooting Technique
Changes you've made to ForgeOps's Kustomize files don't work as expected.	Fully expand the Kustomize output, and then examine the output for unintended effects.
Your Minikube deployment doesn't work.	Make sure that you don't have a problem with virtual hardware requirements.
You're having name resolution or other DNS issues.	Use diagnostic tools in the debug tools container.
You want to run DS utilities without disturbing a DS pod.	Use the bin/ds-debug.sh script or DS tools in the debug tools container.
You want to keep the `amster` pod running to diagnose AM configuration issues.	Use the forgeops amster command.
You want to troubleshoot AM configuration upgrade issues.	Use the config --no-upgrade option.
The `kubectl` command requires too much typing.	Enable kubectl tab autocompletion.

Kubernetes logs and other diagnostics

Look at pod descriptions and container log files for irregularities that indicate problems.

Pod descriptions contain information about active Kubernetes pods, including their configuration, status, containers (including containers that have finished running), volume mounts, and pod-related events.

Container logs contain startup and run-time messages that might indicate problem areas. Each Kubernetes container has its own log that contains all output written to `stdout` by the application running in the container. The `am` container logs are especially important for troubleshooting AM issues in Kubernetes deployments. AM writes its debug logs to `stdout`. Therefore, the `am` container logs include all the AM debug logs.

debug-logs utility

The debug-logs utility generates the following HTML-formatted output, which you can view in a browser:

- Descriptions of all the Kubernetes pods running the Ping Identity Platform in your namespace
- Logs for all of the containers running in these pods
- Descriptions of the PVCs running in your cluster
- Operator logs

- Information about your local environment, including:
  - The Kubernetes context
  - Third-party software versions
  - CRDs installed in your cluster
  - Kubernetes storage classes
  - The most recent commits in your forgeops repository clone's commit log
  - Details about a variety of Kubernetes objects on your cluster

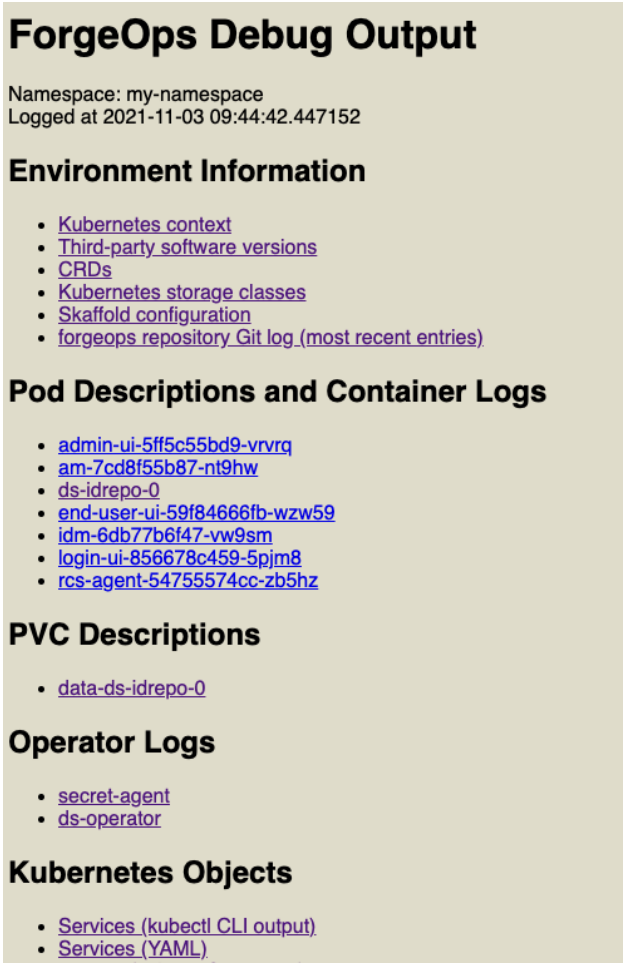
Example troubleshooting steps

Suppose you performed a ForgeOps deployment but noticed that one of the pods had an `ImagePullBackOff` error at startup. Here's an example of how you can use pod descriptions and container logs to troubleshoot the problem:

1. Make sure the active namespace in your local Kubernetes context is the one that contains the component you are debugging.
2. Make sure you've checked out the 2025.1.2 branch of the `forgeops` repository.
3. Change to the /path/to/forgeops/bin directory in your `forgeops` repository clone.
4. Run the debug-logs command:

```
$ ./debug-logs
Writing environment information
Writing pod descriptions and container logs
  admin-ui-5ff5c55bd9-vrvrq
  am-7cd8f55b87-nt9hw
  ds-idrepo-0
  end-user-ui-59f84666fb-wzw59
  idm-6db77b6f47-vw9sm
  login-ui-856678c459-5pjm8
Writing PVC descriptions
  data-ds-idrepo-0
Writing operator logs
  secret-agent
  ds-operator
Writing information about various Kubernetes objects
Open /tmp/forgeops/log.html in your browser.
```

5. In a browser, go to the URL shown in the debug-logs output. In this example, the URL is file:///tmp/forgeops/log.html. The browser displays a screen with a link for each Ping Identity Platform pod in your namespace:



6. Access the information for the pod that didn't start correctly by selecting its link from the Pod Descriptions and Container Logs section of the debug-logs output.

   Selecting the link takes you to the pod's description. Logs for each of the pod's containers follow the pod's description.

After you've obtained the pod descriptions and container logs, here are some actions you might take:

- Examine each pod's event log for failures.
- If a Docker image could not be pulled, verify that the Docker image name and tag are correct. If you are using a private registry, verify that your image pull secret is correct.
- Examine the init containers. Did each init container complete with a zero (success) exit code? If not, examine the logs from that failed init container using the `kubectl logs pod-xxx -c init-container-name` command.
- Look at the pods' logs to check if the main container entered a crashloop.

DS diagnostic tools

Debug script

The bin/ds-debug.sh script lets you obtain diagnostic information for any DS pod running in your cluster. It also lets you perform several cleanup and recovery operations on DS pods.

Run bin/ds-debug.sh -h to refer to the command's syntax.

The following bin/ds-debug.sh subcommands provide diagnostic information:

Subcommand	Diagnostics
status	Server details, connection handlers, backends, and disk space
rstatus	Replication status
idsearch	All the DNs in the `ou=identities` branch
monitor	All the directory entries in the `cn=monitor` branch
list-backups	A list of the backups associated with a DS instance

The bin/ds-debug.sh purge command purges all the backups associated with a DS instance.

Debug tools container

The `ds-util` debug tools container provides a suite of diagnostic tools that you can execute inside of a running Kubernetes cluster.

The container has two types of tools:

- DS tools—A DS instance is installed in the /opt/opendj directory of the `ds-util` container. DS tools, such as the ldapsearch and ldapmodify commands, are available in the /opt/opendj/bin directory.
- Miscellaneous diagnostic tools—A set of diagnostic tools, including `dig`, `netcat`, `nslookup`, `curl`, and `vi`, have been installed in the container. The file, /path/to/forgeops/docker/ds/dsutil/Dockerfile, has the list of operating system packages that have been installed in the debug tools container.

To start the debug tools container:

```
$ kubectl run -it ds-util --image=gcr.io/forgeops-public/ds-util -- bash
```

After you start the tools container, a command prompt appears:

```
root@ds-util:/opt/opendj#
```

You can access all the tools available in the container from this prompt. For example:

```
root@ds-util:/opt/opendj# nslookup am
Server:         10.96.0.10
Address:10.96.0.10#53

Name:   am.my-namespace.svc.cluster.local
Address: 10.100.20.240
```

Troubleshooting the `amster` pod

When ForgeOps deployments start, the `amster` pod starts and imports AM dynamic configuration. Once dynamic configuration is imported, the `amster` pod is stopped and remains in `Completed` status.

```
$ kubectl get pods
NAME                           READY   STATUS      RESTARTS   AGE
admin-ui-b977c857c-2m9pq       1/1     Running     0          10m
am-666687d69c-94thr            1/1     Running     0          12m
amster-4prdg                   0/1     Completed   0          12m
ds-idrepo-0                    1/1     Running     0          13m
end-user-ui-674c4f79c-h4wgb    1/1     Running     0          10m
idm-869679958c-brb2k           1/1     Running     0          12m
login-ui-56dd46c579-gxrtx      1/1     Running     0          10m
```

Start the `amster` pod

After you install AM, use the forgeops amster run command to start the `amster` pod for manually interacting with AM using the forgeops amster run command line interface and perform tasks such as exporting and importing AM configuration and troubleshooting:

```
$ ./bin/forgeops amster run --env-name my-env
starting...
...

$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
admin-ui-b977c857c-2m9pq       1/1     Running   0          22m
am-666687d69c-94thr            1/1     Running   0          24m
amster-852fj                   1/1     Running   0          12s
ds-idrepo-0                    1/1     Running   0          25m
end-user-ui-674c4f79c-h4wgb    1/1     Running   0          22m
idm-869679958c-brb2k           1/1     Running   0          24m
login-ui-56dd46c579-gxrtx      1/1     Running   0          22m
```

The amster jobs have a default time-to-live (TTL) value set to 600 seconds. The amster jobs are removed from the namespace after 10 minutes to allow later runs of amster jobs if the spec is updated in the user's environment and redeployed.

A Kubernetes job cannot be updated after it has started running. If the amster job is running when you apply an update, then an error is thrown. The beginning of the error appears similar to the following:

```
The Job "amster" is invalid: spec.template: Invalid value: ...
...
"batch.kubernetes.io/job-name":"amster", ...
"job-name":"amster"}
```

If an amster job fails due to low TTL, then delete amster jobs using the kubectl delete jobs command and redeploy.

Export and import AM configuration

To export AM configuration, use the forgeops amster export command. Similarly, use the forgeops amster import command to import AM configuration. At the end of the export or import session, the `amster` pod is stopped by default. To keep the `amster` pod running, use the --retain option. You can specify the time (in seconds) to keep the `amster` running.

In the following example, the `amster` pod is kept running for 900 seconds after completing export:

```
$ ./bin/forgeops amster export --env-name my-env --retain 900 /tmp/myexports
Cleaning up amster components
job.batch "amster" deleted
configmap "amster-files" deleted
Packing and uploading configs
configmap/amster-files created
configmap/amster-export-type created
Deploying amster
job.batch/amster created

Waiting for amster job to complete. This can take several minutes.
pod/amster-d6vsv condition met
tar: Removing leading `/' from member names
Updating amster config.
Updating amster config complete.
```

```
$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
admin-ui-b977c857c-2m9pq       1/1     Running   0          27m
am-666687d69c-94thr            1/1     Running   0          29m
amster-d6vsv                   1/1     Running   0          53s
ds-idrepo-0                    1/1     Running   0          30m
end-user-ui-674c4f79c-h4wgb    1/1     Running   0          27m
idm-869679958c-brb2k           1/1     Running   0          29m
login-ui-56dd46c579-gxrtx      1/1     Running   0          27m
```

After 900 seconds notice that the `amster` pod is in `Completed` status:

```
$ kubectl get pods
NAME                          READY   STATUS      RESTARTS   AGE
admin-ui-b977c857c-2m9pq      1/1     Running     0          78m
am-666687d69c-94thr           1/1     Running     0          80m
amster-d6vsv                  0/1     Completed   0          51m
ds-idrepo-0                   1/1     Running     0          81m
end-user-ui-674c4f79c-h4wgb   1/1     Running     0          78m
idm-869679958c-brb2k          1/1     Running     0          80m
login-ui-56dd46c579-gxrtx     1/1     Running     0          78m
```

The `no-upgrade` option of `config export`

When you export AM configuration using the config export am command, the config export command exports the configuration from the AM pod and runs the configuration rules twice:

- First, run rules to reapply any placeholders.
- Next, to run rules to upgrade to the current version of AM.

If you need to troubleshoot issues with the config export am command, then you can separate the steps for applying placeholder rules from the steps for applying upgrade rules.

An example, with `no-upgrade` option:

```
$ config export am my-profile --no-upgrade
[INFO] Running export for am in am-54c87b86cb-rr8mm
[INFO] Updating existing profile: /path/to/forgeops/docker/am/config-profiles/my-profile
[INFO] Clean profile: /path/to/forgeops/docker/am/config-profiles/my-profile
[INFO] Exported AM config
[INFO] Completed export
```

Staged installation

By default, the forgeops apply command installs the entire Ping Identity Platform.

You can also install the platform in stages to help troubleshoot deployment issues.

To install the platform in stages:

1. Verify that the namespace in which the Ping Identity Platform is to be installed is set in your Kubernetes context.
2. Identify the size of the cluster you're deploying the platform on. You'll specify the cluster size as an argument to the forgeops install command:
    - --cdk for a single-instance deployment
    - --small, --medium, or --large, for other ForgeOps deployments
3. Install the `base` and `ds` components first. Other components have dependencies on these two components:
    1. Install the platform `base` component:

```
$ cd /path/to/forgeops/bin
$ ./forgeops apply base --size --fqdn myfqdn.example.com
Checking secret-agent operator and related CRDs: secret-agent CRD not found. Installing
secret-agent.
namespace/secret-agent-system created
...

Waiting for secret agent operator...
customresourcedefinition.apiextensions.k8s.io/secretagentconfigurations.secret-
agent.secrets.forgerock.io condition met
deployment.apps/secret-agent-controller-manager condition met
pod/secret-agent-controller-manager-694f9dbf65-52cbt condition met

Checking ds-operator and related CRDs: ds-operator CRD not found. Installing ds-operator.
namespace/fr-system created
customresourcedefinition.apiextensions.k8s.io/directoryservices.directory.forgerock.io created
...

Waiting for ds-operator...
customresourcedefinition.apiextensions.k8s.io/directoryservices.directory.forgerock.io
condition met
deployment.apps/ds-operator-ds-operator condition met
pod/ds-operator-ds-operator-f974dd8fc-55mxw condition met

Installing component(s): ['base']

configmap/dev-utils created
configmap/platform-config created
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+, unavailable in v1.22+; use
networking.k8s.io/v1 Ingress
ingress.networking.k8s.io/end-user-ui created
ingress.networking.k8s.io/forgerock created
ingress.networking.k8s.io/ig-web created
ingress.networking.k8s.io/login-ui created
ingress.networking.k8s.io/platform-ui created
secretagentconfiguration.secret-agent.secrets.forgerock.io/forgerock-sac created

Waiting for K8s secrets
Waiting for secret: am-env-secrets ...done
Waiting for secret: idm-env-secrets ...done
Waiting for secret: rcs-agent-env-secrets ...done
Waiting for secret: ds-passwords ...done
Waiting for secret: ds-env-secrets ...done

Relevant passwords:
...

Relevant URLs:
https://myfqdn.example.com/platform
https://myfqdn.example.com/admin
https://myfqdn.example.com/am
https://myfqdn.example.com/enduser

Enjoy your deployment!
```

2. After you've installed the `base` component, install the `ds` component:

```
$ ./forgeops apply ds --size
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['ds']

directoryservice.directory.forgerock.io/ds-idrepo created

Enjoy your deployment!
```

4. Install the other Ping Identity Platform components. You can either install all the other components by using the forgeops apply apps command, or install them separately:

    1. Install AM:

```
$ ./forgeops apply am --size
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['am']

service/am created
deployment.apps/am created

Enjoy your deployment!
```

    2. Install Amster:

```
$ ./forgeops apply amster --size
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['amster']

job.batch/amster created

Enjoy your deployment!
```

    3. Install IDM:

```
$ ./forgeops apply idm --size
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['idm']

configmap/idm created
configmap/idm-logging-properties created
service/idm created
deployment.apps/idm created

Enjoy your deployment!
```

5. Install the user interface components. You can either install all the applications by using the forgeops apply ui command, or install them separately:

    1. Install the administration UI:

    ```
    $ ./forgeops apply admin-ui --size
    Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
    Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

    Installing component(s): ['admin-ui']

    service/admin-ui created
    deployment.apps/admin-ui created

    Enjoy your deployment!
    ```

    2. Install the login UI:

    ```
    $ ./forgeops apply login-ui --size
    Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
    Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

    Installing component(s): ['login-ui']

    service/login-ui created
    deployment.apps/login-ui created

    Enjoy your deployment!
    ```

    3. Install the end user UI:

```
$ ./forgeops apply end-user-ui --size
Checking secret-agent operator and related CRDs: secret-agent CRD found in cluster.
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['end-user-ui']

service/end-user-ui created
deployment.apps/end-user-ui created

Enjoy your deployment!
```

6. In a separate terminal tab or window, run the kubectl get pods command to monitor status of the deployment. Wait until all the pods are ready.

Multiple component installation

You can specify multiple components with a single forgeops apply command. For example, to install the `base`, `ds`, `am`, and `amster` components in a ForgeOps deployment:

```
$ ./forgeops apply base ds am amster --size
```

Ingress issues

If the pods in a ForgeOps deployment are starting successfully, but you can't reach the services in those pods, you probably have ingress issues.

To diagnose ingress issues:

1. Use the `kubectl describe ing` and `kubectl get ing ingress-name -o yaml` commands to view the ingress object.
2. Describe the service using the `kubectl get svc; kubectl describe svc xxx` command. Does the service have an `Endpoint:` binding? If the service endpoint binding is not present, the service did not match any running pods.

Third-party software versions

The ForgeOps team recommends installing tested versions of third-party software in environments where you'll run ForgeOps deployments.

Refer to the tables that list the tested versions of third-party software for your deployment:

- On Minikube
- On GKE
- On EKS
- On AKS

You can use the debug-logs utility to get the versions of third-party software installed in your local environment. After you've performed a ForgeOps deployment:

1. Run the /path/to/forgeops/bin/debug-logs utility.
2. Open the log file in your browser.
3. Select Environment Information > Third-party software versions.

Expanded Kustomize output

If you've modified any of the Kustomize bases and overlays that come with the `cdk` canonical configuration, you might want to consider how your changes affect deployment. Use the kustomize build command to assess how Kustomize expands your bases and overlays into YAML files.

For example:

```
$ cd /path/to/forgeops/kustomize/overlay
$ kustomize build all
apiVersion: v1
data:
  IDM_ENVCONFIG_DIRS: /opt/openidm/resolver
  LOGGING_PROPERTIES: /var/run/openidm/logging/logging.properties
  OPENIDM_ANONYMOUS_PASSWORD: anonymous
  OPENIDM_AUDIT_HANDLER_JSON_ENABLED: "false"
  OPENIDM_AUDIT_HANDLER_STDOUT_ENABLED: "true"
  OPENIDM_CLUSTER_REMOVE_OFFLINE_NODE_STATE: "true"
  OPENIDM_CONFIG_REPO_ENABLED: "false"
  OPENIDM_ICF_RETRY_DELAYSECONDS: "10"
  OPENIDM_ICF_RETRY_MAXRETRIES: "12"
  PROJECT_HOME: /opt/openidm
  RCS_AGENT_CONNECTION_CHECK_SECONDS: "5"
  RCS_AGENT_CONNECTION_GROUP_CHECK_SECONDS: "900"
  RCS_AGENT_CONNECTION_TIMEOUT_SECONDS: "10"
  RCS_AGENT_HOST: rcs-agent
  RCS_AGENT_IDM_PRINCIPAL: idmPrincipal
  RCS_AGENT_PATH: idm
  RCS_AGENT_PORT: "80"
  RCS_AGENT_USE_SSL: "false"
  RCS_AGENT_WEBSOCKET_CONNECTIONS: "1"
kind: ConfigMap
metadata:
  labels:
    app: idm
    app.kubernetes.io/component: idm
    app.kubernetes.io/instance: idm
    app.kubernetes.io/name: idm
    app.kubernetes.io/part-of: forgerock
    tier: middle
  name: idm
---
apiVersion: v1
data:
  logging.properties: |
...
```

Minikube hardware resources

Cluster configuration

The minikube start command example in Minikube provides a good default virtual hardware configuration for a Minikube cluster running a single-instance ForgeOps deployment.

Disk space

When the Minikube cluster runs low on disk space, it acts unpredictably. Unexpected application errors can appear.

Verify that adequate disk space is available by logging in to the Minikube cluster and running a command to display free disk space:

```
$ minikube ssh
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.9G     0  3.9G   0% /dev
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           3.9G  383M  3.6G  10% /run
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
tmpfs           3.9G   64K  3.9G   1% /tmp
/dev/sda1        25G  7.7G   16G  33% /mnt/sda1
/Users          465G  219G  247G  48% /Users
$ exit
logout
```

In the preceding example, 16 GB of disk space is available on the Minikube cluster.

`kubectl` shell autocompletion

The kubectl shell autocompletion extension lets you extend the Tab key completion feature of Bash and Zsh shells to the kubectl commands. While not a troubleshooting tool, this extension can make troubleshooting easier, because it lets you enter kubectl commands more easily.

For more information about the Kubernetes autocompletion extension, see Enabling shell autocompletion in the Kubernetes documentation.

Note that to install the autocompletion extension in Bash, you must be running version 4 or later of the Bash shell. To determine your bash shell version, run the bash --version command.

Reference material

Base Docker images

Important

This section is moved into the reference section, because creating Docker images from scratch is only required under special circumstances.
Before you begin building custom images, ensure that you have installed Java version 21 or 17 on your computer.
For example:

```
$ java --version
openjdk 21.0.5 2024-10-15 LTS
OpenJDK Runtime Environment Temurin-21.0.5+11 (build 21.0.5+11-LTS)
OpenJDK 64-Bit Server VM Temurin-21.0.5+11 (build 21.0.5+11-LTS, mixed mode)
```

```
$ java --version
openjdk 17.0.10 2024-01-16
OpenJDK Runtime Environment Temurin-17.0.10+7 (build 17.0.10+7)
OpenJDK 64-Bit Server VM Temurin-17.0.10+7 (build 17.0.10+7, mixed mode)
```

Which Docker images do I deploy?

- I am a developer using a single-instance ForgeOps deployment.
  - UI elements. Deploy the supported images from ForgeOps.
  - Other platform elements. Deploy either:
    - The ForgeOps-provided images.
    - Customized Docker images that are based on ForgeOps-provided images and contain customized configuration profile.
- I am doing a proof-of-concept ForgeOps deployment.
  - UI elements. Deploy the supported images from ForgeOps.
  - Other platform elements. Deploy either:
    - The ForgeOps-provided images.
    - Customized Docker images that are based on ForgeOps-provided images and contain customized configuration profile.

- I am deploying the platform in production.
  - UI elements. Deploy the supported images from ForgeOps.
  - Other platform elements. Deploy Docker images you have built that are based on your own base images, but contain your customized configuration profile.

Your initial base Docker images

> **Note**
>
> The procedures here describe the use of:
>
> 1. Docker container engine to create images for ForgeOps deployment. You can use Podman container engine for the same.
> 2. The latest ForgeOps-provided Docker images. You can select a specific image release suitable to your environment.

Perform the following steps to build base images. After you've built your own base images, push them to your Docker repository:

1. Download the latest versions of the AM, Amster, IDM, and DS `.zip` files from the Ping Identity Download Center. Optionally, you can also download the latest version of the PingGateway `.zip` file.
2. If you haven't already done so, clone the `forgeops` and `forgeops-extras` repositories. For example:

   ```
   $ git clone https://github.com/ForgeRock/forgeops.git
   $ git clone https://github.com/ForgeRock/forgeops-extras.git
   ```

   Both repositories are public; you do not need credentials to clone them.
3. Check out the `forgeops` repository's `2025.1.2` tag:

   ```
   $ cd /path/to/forgeops
   $ git checkout 2025.1.2
   ```
4. Check out the `forgeops-extras` repository's `main` tag:

   ```
   $ cd /path/to/forgeops-extras
   $ git checkout main
   ```
5. Build the Java base image, which is required by several of the other Dockerfiles:

   ```
   $ cd /path/to/forgeops-extras/images/java-21
   $ docker build --tag my-repo/my-java .
   ```

   ```
   $ cd /path/to/forgeops-extras/images/java-17
   $ docker build --tag my-repo/my-java .
   ```

6. Build the base Docker image for Amster. The Amster image is required to build the base image for AM in the next step:

    1. Unzip the Amster `.zip` file.
    2. Change to the amster/samples/docker directory in the expanded `.zip` file output.
    3. Run the setup.sh script:

    ```
    $ ./setup.sh

    + mkdir -p build
    + find ../.. '!' -name .. '!' -name samples '!' -name docker -maxdepth 1 -exec cp -R '{}'
    build/ ';'
    + cp ../../docker/amster-install.sh ../../docker/docker-entrypoint.sh ../../docker/
    export.sh ../../docker/tar.sh build
    ```

    4. Edit the Dockerfile in the samples/docker directory. Change the line:

    ```
    FROM gcr.io/forgerock-io/java-17:latest
    ```

    to:

    ```
    FROM my-repo/my-java
    ```

    5. Build the `amster` Docker image:

```
$ docker build --tag amster:8.0.1 .

 ⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
1.67kB
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for docker.io/my-repo/
java-17:latest
1.1s
 ⇒ [1/8] FROM docker.io/my-repo/my-java
...
 ⇒ exporting to image
 ⇒⇒ exporting layers
 ⇒⇒ writing image
sha256:bc47...f9e52
0.0s
 ⇒⇒ naming to docker.io/library/amster:8.0.1
```

7. Build the empty AM image:

    1. Unzip the AM `.zip` file.
    2. Change to the openam/samples/docker directory in the expanded `.zip` file output.
    3. Run the following command initiate `am-empty` Docker container and get the files missing in the AM zip file:

    > **Note**
    >
    > The following is a single command.

    ```
    $ docker run --rm -v "$PWD:/output" \
      -u $(id -u):$(id -g) gcr.io/forgerock-io/am-empty:8.0.1 \
      bash -c "cp /usr/local/tomcat/lib/am-jul.jar \
      /usr/local/tomcat/lib/openam-shared.jar /usr/local/tomcat/lib/joda-time.jar \
      /usr/local/tomcat/lib/forgerock-util.jar /output/"
    ```

    4. Change to openam/samples/docker and edit the setup.sh script:

        1. Change the line:

        ```
        cp ../../AM-7.*.war images/am-empty/build/openam.war
        ```

        to:

```
cp ../../AM-*.war images/am-empty/build/openam.war
```

2. Then add the following additional lines at the end of the other copy commands:

```
cp ../../am-jul*.jar images/am-empty/build/am-jul.jar
cp ../../forgerock-util*.jar images/am-empty/build/forgerock-util.jar
cp ../../joda-time*.jar images/am-empty/build/joda-time.jar
cp ../../openam-shared*.jar images/am-empty/build/openam-shared.jar
```

5. Run the setup.sh script:

```
$ chmod +x ./setup.sh
./setup.sh
```

6. Change to the images/am-empty directory.

7. Build the `am-empty` Docker image:

```
$ docker build --tag am-empty:8.0.1 .

 ⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
3.60kB
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for docker.io/library/tomcat:9-jdk17-openjdk-slim-
bullseye                                                          1.8s
 ⇒ [internal] load build
context
5.6s
 ⇒⇒ transferring context:
231.59MB
5.6s
 ⇒ [base  1/14] FROM docker.io/library/tomcat:9-jdk17-openjdk-slim-bullseye@...
...
 ⇒ exporting to
image
1.7s
 ⇒⇒ exporting
layers
1.6s
 ⇒⇒ writing image
sha256:9784a73...1d36018c9
0.0s
 ⇒⇒ naming to docker.io/library/am-empty:8.0.1
```

8. Build the base image for AM:

    1. Change to the ../am-base directory.
    2. Edit the Dockerfile in the ../am-base directory and change the line:

    ```
    FROM ${docker.push.repo}/am-empty:${docker.tag}
    ```

    to:

    ```
    FROM am-empty:8.0.1
    ```

    3. Build the `am-base` Docker image:

```
$ docker build --tag am-base:8.0.1 .

 ⇒ [internal] load build definition from
Dockerfile                                                                0.0s
 ⇒ ⇒ transferring dockerfile:
2.72kB                                                                    0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒ ⇒ transferring context:
2B                                                                        0.0s
 ⇒ [internal] load metadata for docker.io/library/amster:
8.0.1                                                      0.0s
 ⇒ [internal] load metadata for docker.io/library/am-empty:
8.0.1                                                     0.0s
 ⇒ [internal] load build
context                                                                   0.4s
 ⇒ ⇒ transferring context:
35.66MB                                                                   0.4s
 ⇒ [generator  1/15] FROM docker.io/library/am-empty:
8.0.1                                                          0.4s
 ⇒ [amster 1/1] FROM docker.io/library/amster:
8.0.1                                                               0.2s
 ⇒ [generator  2/15] RUN apt-get update -y &&     apt-get install -y git jq unzip
...
 ⇒ [am-base  7/11] COPY --chown=forgerock:root docker-entrypoint.sh /home/
forgerock/                                0.0s
 ⇒ [am-base  8/11] COPY --chown=forgerock:root scripts/import-pem-certs.sh /home/
forgerock/                         0.0s
 ⇒ [am-base  9/11] RUN rm "/usr/local/tomcat"/webapps/am/WEB-INF/lib/click-extras-
*.jar                             0.2s
 ⇒ [am-base 10/11] RUN rm "/usr/local/tomcat"/webapps/am/WEB-INF/lib/click-nodeps-
*.jar                             0.3s
 ⇒ [am-base 11/11] RUN rm "/usr/local/tomcat"/webapps/am/WEB-INF/lib/velocity-
*.jar                                0.2s
 ⇒ exporting to
image
0.2s
 ⇒ ⇒ exporting
layers
0.2s
 ⇒ ⇒ writing image
sha256:2c06...87c6c
0.0s
 ⇒ ⇒ naming to docker.io/library/am-base:8.0.1
```

4. Change to the ../am-cdk directory.

5. Edit the Dockerfile in the ../am-cdk directory. Change the line:

```
FROM ${docker.push.registry}/forgerock-io/am-base/${docker.promotion.folder}:${docker.tag}
```

to:

```
FROM am-base:8.0.1
```

6. Build the `am` Docker image:

```
$ docker build --tag my-repo/am:8.0.1 .
[+] Building 5.1s (10/10)
FINISHED                                                                    docker:desktop-linux
 ⇒ [internal] load build definition from
Dockerfile                                                                  0.0s
 ⇒⇒ transferring dockerfile:
1.71kB                                                                                  0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B                                                                                        0.0s
 ⇒ [internal] load metadata for docker.io/library/am-base:
8.0.1                                                          0.0s
 ⇒ [1/5] FROM docker.io/library/am-base:
8.0.1                                                                    0.2s
 ⇒ [internal] load build
context                                                                                     0.2s
 ⇒⇒ transferring context:
403.07kB                                                                                  0.1s
 ⇒ [2/5] RUN apt-get update         && apt-get install -y git         && apt-get clean
&& rm -r /var/lib  3.9s
 ⇒ [3/5] RUN cp -R /usr/local/tomcat/webapps/am/XUI /usr/local/tomcat/webapps/am/
OAuth2_XUI                       0.3s
 ⇒ [4/5] COPY --chown=forgerock:root /config /home/forgerock/cdk/
config                                              0.0s
 ⇒ [5/5] RUN rm -rf /home/forgerock/openam/config/services &&     mkdir /home/forgerock/
openam/config/services    0.5s
 ⇒ exporting to
image
0.1s
 ⇒⇒ exporting
layers
0.1s
 ⇒⇒ writing image
sha256:14b43fb5121cee08341130bf502b7841429b057ff406bbe635b23119a74dec45
0.0s
 ⇒⇒ naming to my-repo/am:
8.0.1                                                                                     0.0s
```

9. Now that the AM image is built, tag the base image for Amster in advance of pushing it to your private repository:

```
$ docker tag amster:8.0.1 my-repo/amster:8.0.1
```

10. Build the `am-config-upgrader` base image:

    1. Change to the `openam` directory in the expanded AM `.zip` file output.
    2. Unzip the `Config-Upgrader-8.0.1.zip` file.
    3. Change to the `amupgrade/samples/docker` directory in the expanded `Config-Upgrader-8.0.1.zip` file output.
    4. Edit the Dockerfile in the amupgrade/samples/docker directory and change line 16 from:

    ```
    FROM gcr.io/forgerock-io/java-17:latest
    ```

    to:

    ```
    FROM my-repo/my-java
    ```

    5. Run the setup.sh script:

    ```
    $ ./setup.sh

    + mkdir -p build/amupgrade
    + find ../.. '!' -name .. '!' -name samples '!' -name docker -maxdepth 1 -exec cp -R '{}'
    build/amupgrade ';'
    + cp ../../docker/docker-entrypoint.sh .
    ```

    6. Create the base `am-config-upgrader` image:

    ```
    $ docker build --tag my-repo/am-config-upgrader:8.0.1 .

    [+] Building 8.5s (9/9) FINISHED                                  docker:desktop-linux
    ⇒ [internal] load build definition from Dockerfile                                0.0s
    ⇒⇒ transferring dockerfile: 1.10kB                                                0.0s
    ⇒ [internal] load .dockerignore                                                   0.0s
    ⇒⇒ transferring context: 2B                                                       0.0s
    ⇒ [internal] load metadata for my-repo/my-java:latest                             0.0s
    ⇒ CACHED [1/4] FROM my-repo/my-java                                               0.0s
    ⇒ [internal] load build context                                                   0.3s
    ⇒⇒ transferring context: 20.58MB                                                  0.3s
    ⇒ [2/4] RUN apt-get update &&     apt-get upgrade -y                              8.3s
    ⇒ [3/4] COPY --chown=forgerock:root docker-entrypoint.sh /home/forgerock/         0.0s
    ⇒ [4/4] COPY build/ /home/forgerock/                                              0.0s
    ⇒ exporting to image                                                              0.1s
    ⇒⇒ exporting layers                                                               0.1s
    ⇒⇒ writing image sha256:3f6845…44011                                              0.0s
    ⇒⇒ naming to my-repo/am-config-upgrader:8.0.1                                     0.0s
    ```

11. Build the base image for DS:

> **Note**
>
> The setup.sh script in DS zip file is currently not working.

1. Unzip the DS `.zip` file.
2. Change to the opendj directory in the expanded `.zip` file output.
3. Run the samples/docker/setup.sh script to create a server:

```
$ ./samples/docker/setup.sh

+ rm -f template/config/tools.properties
+ cp -r samples/docker/Dockerfile samples/docker/README.md ...
+ rm -rf — README README.md bat '*.zip' opendj_logo.png setup.bat upgrade.bat setup.sh
+ ./setup --serverId docker --hostname localhost
...

Validating parameters... Done
Configuring certificates... Done
...
```

4. Edit the Dockerfile in the opendj directory. Change the line:

```
FROM gcr.io/forgerock-io/java-17:latest
```

   to:

```
FROM my-repo/my-java
```

5. Build the `ds` base image:

```
$ docker build --tag my-repo/ds:8.0.0 .

[+] Building 11.0s (9/9) FINISHED

 ⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
1.23kB
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for my-repo/my-java:latest
1.7s
 ⇒ [internal] load build
context
1.2s
 ⇒⇒ transferring context:
60.85MB
1.2s
 ⇒ CACHED [1/4] FROM my-repo/my-java:latest
...
 ⇒ [4/4] WORKDIR /opt/
opendj
0.0s
 ⇒ exporting to
image
0.4s
 ⇒⇒ exporting
layers
0.3s
 ⇒⇒ writing image
sha256:713ac...b107e0f
0.0s
 ⇒⇒ naming to my-repo/ds:8.0.0
```

12. Build the base image for IDM:

    1. Create a new shell script file named build-idm-image.sh and copy the following lines into it:

```
#!/bin/bash

if [ $# -lt 3 ]; then
  echo "$0 <source image> <new base image> <result image>"
  exit 0
fi

sourceImage="$1"
javaImage="$2"
resultImage="$3"

container_id=$(docker create $sourceImage)
docker export $container_id -o image.tar
docker rm $container_id

tar xvf image.tar opt/openidm
rm -f image.tar

cd opt/openidm
# use | separators because image names often have / and :
sed -i.bak 's|^FROM.*$|FROM '$javaImage'|' bin/Custom.Dockerfile
rm bin/Custom.Dockerfile.bak

docker build . --file bin/Custom.Dockerfile --tag "$resultImage"
rm -rf opt
```

2. Change the mode of the file to be executable and run it.

```
$ chmod +x build-idm-image.sh
$ ./build-idm-image.sh us-docker.pkg.dev/forgeops-public/images-base/idm:8.0.0 my-repo/my-java
my-repo/idm:8.0.0
```

> **Note**
>
> The build-idm-image.sh script expands the IDM Docker image, rebuilds the image, and cleans up afterward.

13. (Optional) Build the base image for PingGateway:

    1. Unzip the PingGateway `.zip` file.
    2. Change to the identity-gateway directory in the expanded `.zip` file output.
    3. Edit the Dockerfile in the identity-gateway/docker directory. Change the line:

```
FROM gcr.io/forgerock-io/java-17:latest
```

    to:

```
FROM my-repo/my-java
```

4. Build the `ig` base image:

```
$ docker build . --file docker/Dockerfile --tag my-repo/ig:2025.3.0

[+] Building 2.1s (8/8) FINISHED
⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
1.43kB
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for my-repo/my-java:latest
0.3s
 ⇒ [internal] load build
context
2.2s
 ⇒⇒ transferring context:
113.60MB
2.2s
 ⇒ CACHED [1/3] FROM my-repo/my-java:latest
 ⇒ [2/3] COPY --chown=forgerock:root . /opt/
ig
0.7s
 ⇒ [3/3] RUN mkdir -p "/var/ig"     && chown -R forgerock:root "/var/ig" "/opt/ig"     &&  -R
g+rwx "/var/ig" "/opt/ig"                     0.9s
 ⇒ exporting to
image
0.6s
 ⇒⇒ exporting
layers
0.6s
 ⇒⇒ writing image
sha256:77fc5...6e63
0.0s
 ⇒⇒ naming to my-repo/ig:2025.3.0
```

14. Run the docker images command to verify that you built the base images:

```
$ docker images | grep my-repo

REPOSITORY                  TAG      IMAGE ID       CREATED        SIZE
my-repo/am                  8.0.1    552073a1c000   1 hour ago     795MB
my-repo/am-config-upgrader  8.0.1    d115125b1c3f   1 hour ago     795MB
my-repo/amster              8.0.1    d9e1c735f415   1 hour ago     577MB
my-repo/ds                  8.0.0    ac8e8ab0fda6   1 hour ago     196MB
my-repo/idm                 8.0.0    0cc1b7f70ce6   1 hour ago     387MB
my-repo/ig                  2025.3.0 cc52e9623b3c   1 hour ago     249MB
my-repo/java-17             latest   a504925c2672   1 hour ago     144MB
```

15. Push the new base Docker images to your Docker repository.

    Refer to your registry provider documentation for detailed instructions. For most Docker registries, you run the docker login command to log in to the registry. Then, you run the docker push command to push a Docker image to the registry.

    Be sure to configure your Docker registry so that you can successfully push your Docker images. Each cloud-based Docker registry has its own specific requirements. For example, on Amazon ECR, you must create a repository for each image.

    Push the following images to your repository:

    - `my-repo/am:8.0.1`
    - `my-repo/am-config-upgrader:8.0.1`
    - `my-repo/amster:8.0.1`
    - `my-repo/ds:8.0.0`
    - `my-repo/idm:8.0.0`
    - `my-repo/my-java`

    If you're deploying your own PingGateway base image, also push the `my-repo/ig:2025.3.0` image.

Create Docker images for use in production

After you've built and pushed your own base images to your Docker registry, you're ready to build customized Docker images that can be used in a production deployment of the Ping Identity Platform. These images:

- Contain customized configuration profiles for AM, IDM, and, optionally, PingGateway.
- Must be based on your own base Docker images.

Create your production-ready Docker images, create a Kubernetes cluster to test them, and delete the cluster when you've finished testing the images:

1. Clone the `forgeops` repository.

2. Obtain custom configuration profiles that you want to use in your Docker images from your developer, and copy them into your `forgeops` repository clone:
    - Obtain the AM configuration profile from the /path/to/forgeops/docker/am/config-profiles directory.
    - Obtain the IDM configuration profile from the /path/to/forgeops/docker/idm/config-profiles directory.
    - (Optional) Obtain the PingGateway configuration profile from the /path/to/forgeops/docker/ig/config-profiles directory.
3. Change the `FROM` lines of Dockerfiles in the `forgeops` repositories to refer to your own base Docker images:

In the `forgeops` repository file:	Change the `FROM` line to:
docker/am/Dockerfile	`FROM my-repo/am:8.0.1` [1]
docker/amster/Dockerfile	`FROM my-repo/amster:8.0.1`
docker/ds/ds-new/Dockerfile	`FROM my-repo/ds:8.0.0`
docker/idm/Dockerfile	`FROM my-repo/idm:8.0.0` [2]
(Optional) docker/ig/Dockerfile	`FROM my-repo/ig:2025.3.0`

4. If necessary, log in to your Docker registry.
5. Enable the Python3 virtual environment:

```
$ source .venv/bin/activate
```

6. Set up a ForgeOps deployment environment:

```
$ cd /path/to/forgeops/bin
$ ./forgeops env --env-name my-env --fqdn my-fqdn --cluster-issuer my-cluster-issuer
```

In the command above, replace my-fqdn and my-cluster-issuer with appropriate values from your environment. If you want to use the issuer provided with the platform for demo, then you can use default-issuer.

7. Build Docker images that are based on your own base images.

> **Note**
>
> While the forgeops build command uses the Docker engine by default for ForgeOps deployments, it supports Podman as well. If you are using Podman engine instead of Docker in your environment, then set the `CONTAINER_ENGINE` environment variable to `podman` before running the forgeops build command, for example:
>
> ```
> $ export CONTAINER_ENGINE="podman"
> ```

The AM and IDM images contain your customized configuration profiles:

```
$ cd /path/to/forgeops/bin
$ ./forgeops build --env-name my-env ds --push-to my-repo --tag my-tag
$ ./forgeops build --env-name my-env amster --push-to my-repo --tag my-tag
$ ./forgeops build --env-name my-env am --push-to my-repo --tag my-tag --config-profile my-profile
$ ./forgeops build --env-name my-env idm --push-to my-repo --tag my-tag --config-profile my-profile
```

The forgeops build command:

- Builds Docker images. The AM and IDM images incorporate customized configuration profiles.
- Pushes Docker images to the repository specified in the --push-to argument.
- Updates the image defaulter file, which the forgeops apply command uses to determine which Docker images to run.

8. (Optional) Build and push an PingGateway Docker image that's based on your own base image and contains your customized configuration profile:

```
$ ./forgeops build --env-name my-env ig --config-profile my-profile --push-to my-repo
```

9. Prepare a Kubernetes cluster to test your images:
    1. Create the cluster. This example assumes that you create a cluster suitable for a small-sized ForgeOps deployment.
    2. Make sure your cluster can access and pull Docker images from your repository.
    3. Create a namespace in the new cluster, and then make the new namespace the active namespace in your local Kubernetes context.
10. Perform a ForgeOps deployment in your cluster:

```
$ cd /path/to/forgeops/bin
$ ./forgeops apply --env-name my-env --fqdn my-fqdn --namespace my-namespace
```

11. Access the AM admin UI and the IDM admin UI, and verify that your customized configuration profiles are active.
12. Delete the Kubernetes cluster that you used to test images.

At the end of this process, the artifacts that you'll need to deploy the Ping Identity Platform in production are available:

- Docker images for the Ping Identity Platform, in your Docker repository
- An updated image defaulter file, in your `forgeops` repository clone

You'll need to copy the image defaulter file to your production deployment, so that when you run the forgeops apply command, it will use the correct Docker images.

Typically, you model the image creation process in a CI/CD pipeline. Then, you run the pipeline at milestones in the development of your customized configuration profile.

1. The `FROM` statement originally contained `am-cdk` as part of the repository name. Be sure to use `am`, not `am-cdk`, in the revised statement.
2. The `FROM` statement originally contained `idm-cdk` as part of the repository name. Be sure to use `idm`, not `idm-cdk`, in the revised statement.

forgeops command">

The forgeops command

> **Important**
>
> forgeops — The new generation utility replaces the previous version of `forgeops`. The new forgeops utility simplifies deploying and managing Ping Identity Platform components in a Kubernetes cluster.
> **The previous version of the forgeops utility is not supported in this ForgeOps release. It continues to be supported in ForgeOps 7.5 and 7.4, as long as the corresponding Ping Identity Platform components are supported.**

You can create and manage custom overlays and Helm values files for each deployment. You can then apply the overlays or value files appropriately using Kustomize or Helm accordingly.

The forgeops utility lets you:

- Use Kustomize natively so you can update and use overlays as expected.
- Generate a Kustomize overlay manually when you need the overlay.
- Generate Helm value files from the same environment set up.
- Build and manage Docker images per overlay to allow different images in an environment.
- Create and manage each ForgeOps deployment configuration.
- Apply the environment configuration changes using either Kustomize or Helm.

Features in forgeops

Discrete overlays

The current forgeops command has the following limitations:

- It generates a Kustomize overlay every time it runs.
- It overwrites any post-deployment changes in Kustomize overlays.
- It uses the preconfigured patch files and ignores the customizations during deployment.

The forgeops command doesn't generate overlay files automatically. Instead, overlay files are manually generated as needed.

It is recommended to create an overlay for each environment, such as `test`, `stage`, and `prod`. It is also recommended to create an overlay for each single-instance environment, such as `test-single`, `stage-single`, and `prod-single`. The single-instance overlays help you develop file-based configuration changes, export them, and build new images.

`image-defaulter` in every overlay

Each overlay includes an `image-defaulter` component. When using Kustomize, you can develop and build and test custom images in your single-instance environment. Once you are satisfied with the image, you can copy the image-defaulter's `kustomization.yaml` file into your running overlay.

Sub-overlays

To install and delete individual components, ForgeOps provided overlays are composed of sub-overlays. Each Ping Identity Platform product has its own overlay. There are other overlays to handle shared pieces. You can apply or delete sub-overlay or the entire overlay using `kubectl apply -k` or `kubectl delete -k` commands.

Specify overlay or environment to target

With discrete overlays, you need to specify which overlay you want to target when running the `forgeops` commands. If you forget to specify the overlay, the command exits and lets you know to provide one. Only the apply and info commands allow you to not specify an overlay.

Helm Support

Both Kustomize and Helm are supported by the forgeops command. Use the forgeops env command to generate Helm `values` file and Kustomize overlays for existing environments. The forgeops build command updates the Helm `values` file and the Kustomize `image-defaulter` overlay file for the specified environment.

> **Note**
>
> The forgeops command can generate the `values.yaml` file from an already deployed environment, it cannot generate the `values.yaml` file for a new environment.

The `values.yaml` file contains all the Helm values. While the `values.yaml` file contains all the Helm values for an environment, few more files are created each containing a group of interrelated values that can be copied and used in other environments, if you need to.

Setup

The forgeops command is developed using Python. Run the forgeops configure command to ensure the required packages are set up:

```
$ cd /path/to/forgeops/bin
$ ./forgeops configure
```

You need to run the forgeops configure once before creating and managing your ForgeOps deployment environments.

Workflow

The workflow of `forgeops` is designed to be production first and has three distinct steps:

1. Create an environment

This step is used to manage the overlay and values files on an ongoing basis. Only the requested changes are incorporated, so the customizations are not impacted.

2. Build images for the environment

The `build` step assembles the file-based configuration changes into container images, and updates the `image-defaulter` and `values` files for the targeted environment.

3. Apply the environment

In this step, you deploy the image you configured.

> **Note**
>
> It is recommended that you start with a single-instance deployment to develop your AM and IDM configuration, so you can export them and build your custom container images.

1. Create an environment

You must create an environment first using the forgeops env command. You need to specify an FQDN (`--fqdn`) and an environment name (`--env-name`).

Previously, the t-shirt sized overlays called `small` , `medium` , and `large` were provided, along with the default overlay `cdk` . With `forgeops` , a `single-instance` overlay replaces `cdk` . The `single-instance` overlay is considered the default and is provided in the kustomize-ng/overlay/default directory.

You can use `--small` , `--medium` , and `--large` flags to configure your overlay, and the forgeops env command populates your environment with the size you requested.

For example, the following command creates a medium-sized stage deployment with an FQDN of stage.example.com:

```
$ cd /path/to/forgeops
$ ./bin/forgeops env --fqdn stage.example.com --medium --env-name stage
```

The default deployment size is `single-instance` . The following example command creates a single-instance environment:

```
$ cd /path/to/forgeops
$ ./bin/forgeops env --fqdn stage.example.com --env-name stage-single
```

You will find the generated Kustomize overlay files in the kustomize-ng/overlay/ENV-NAME folder. If you are modifying an existing Helm-based environment, then you will also find the Helm specific value files in the charts/identity-platform folder.

2. Build images for the environment

Use the forgeops build command to create a new container image for the environment you created in the Create an environment step. The forgeops build command applies the config profile from the build docker/am/config-profiles/profile and docker/idm/config-profiles/profile to build AM and IDM container images and push the images to your container registry. It also updates the `image-defaulter` and `values` files for the targeted environment.

To build new AM and IDM images for our stage environment using the stage-cfg profile, run the command:

```
$ ./bin/forgeops build --env-name stage \
 --config-profile stage-cfg \
 --push-to my.registry.com/my-repo/stage am idm
```

3. Apply the environment

Use the overlay you created in the Create an environment step and deploy the environment built in the Build images for the environment step.

Kustomize-based deployment

You have two options to perform ForgeOps deployment in a Kustomize-based environment:

- Using the kubectl apply command, for example:

```
$ kubectl apply -k /path/to/forgops/kustomize-ng/overlay/my-overlay
```

- Using the forgeops apply command, for example:

```
$ ./bin/forgeops apply --env-name stage
```

> **Note**
>
> If you are using Helm-based deployment methods, you cannot use the forgeops command to perform ForgeOps deployment. Instead, use the helm install or helm upgrade command with the Helm values file:
>
> ```
> $ helm upgrade --install ...
> ```

`forgeops` commands

The forgeops command is a Bash wrapper script that calls appropriate scripts in `bin/commands`. These scripts are written in either Bash or Python. All the bash scripts support the new `--dryrun` flag which display the command that would be run and enable you to inspect it before actually running the command. The Python scripts `env` and `info` do not support `--dryrun`.

Helm Support

Both Kustomize and Helm are supported by the forgeops command. Use the forgeops env command to generate Helm `values` files and Kustomize overlays for each environment. The forgeops build command updates the Helm `values` file and the Kustomize `image-defaulter` overlay file for the specified environment.

The `values.yaml` file contains all the Helm values. The other files group the different values so that you can use them individually if you need to.

Custom paths

By default, forgeops uses the `docker`, `kustomize`, and `helm` directories. You can set up your own locations separately and specify the appropriate flags on the command line or set the appropriate environment variable in the path/to/forgeops/forgeops.conf file.

Learn more about the forgeops command options in the forgeops command reference.

forgeops command reference">

forgeops command reference

forgeops — The new generation utility simplifies deploying and managing Ping Identity Platform components in a Kubernetes cluster. You can create and manage custom Kustomize overlays and Helm value files for each deployment. You can then apply the customized overlays or value files using Kustomize or Helm appropriately.

> **Caution**
>
> The `forgeops` command reference documentation is currently in developmental preview stage, and not all command options have been documented yet. To get help in the command-line interface, use the forgeops --help command.

Synopsis

forgeops subcommand options

Description

- Generate custom component overlays and value files.
- Use Kustomize or Helm to install Ping Identity Platform components in a Kubernetes cluster.
- Delete platform components from a Kubernetes cluster.
- Build custom Docker images for the Ping Identity Platform.

Options

The forgeops command takes the following option:

`--help`

Display command usage information.

> **Important**
>
> The following subcommands `clean`, `config`, `install`, and `generate` have been deprecated because their functionality is provided through other existing subcommands.

Subcommands

forgeops apply

forgeops apply components options

Runs the `kubectl apply -k` command to apply Ping Identity Platform Kustomize overlay from the specified overlay directory into a Kubernetes namespace. If the specified overlay directory doesn't exist, a new one is created.

- The `forgeops apply` subcommand subsumes all the functionality of `forgeops install`. Accordingly, `forgeops install` is deprecated.

For components, specify:

- `am`, `amster`, `ds-cts`, `ds-idrepo`, `idm`, or `ig` to deploy each Ping Identity Platform component.
- More than one component or set of components separated by a space to deploy multiple Ping Identity Platform components. For example, forgeops apply ds-idrepo ds-cts am.
- `secrets` to deploy Kubernetes secrets. Secrets generated by cert-manager are not deployed.
- `base` to deploy the `platform-config` configmap Kubernetes ingress resources and Kubernetes secrets. Secrets generated by cert-manager are not deployed.
- `all` to deploy all the Ping Identity Platform components.

The default value for components is `all`.

Options

The forgeops apply subcommand takes the following options:

`--create-namespace`

Create a namespace if it doesn't exist. The default is the current namespace of the user.

`--debug`

Display debug information when executing the command.

`--dryrun`

To perform a dry run without actually applying or installing the components.

`--env-name` *my-env*

Name of environment to apply. The default is `demo`.

`--fqdn` *my-fqdn*

The fully qualified hostname to use in the deployment.

- The namespace specified in the forgeops env command is used by default. For simple demo purposes, the namespace specified in the default overlay file is used.
- Relevant only for the forgeops apply all and forgeops apply base commands. This option is ignored for other forgeops apply commands.

`--namespace` *ns*

The namespace in which to install the ForgeOps platform components. If you need to create the namespace, then specify the `--create-namespace | -c` option.

`--kustomize` *my-kustomize-path*

The directory that contains Kustomize overlays. Specify the full path to the directory or the path relative to the base of your local `forgeops` repository. The default value is `kustomize`.

Examples

Use an environment my-env

forgeops apply --env-name my-env

Do a dry run

forgeops apply --dryrun --env-name my-env

forgeops build

forgeops build --env-name my_env components options

Use the forgeops build command to build custom Docker images for one or more Ping Identity Platform components, and update the Helm `values` file and the Kustomize `image-defaulter` overlay file for the specified environment.

> **Note**
>
> - Building an `amster` image is not supported, so use bin/forgeops amster.
> - The `--config-profile` option is applicable only for AM, idm_abbr, and PingGateway.
> - Use the `--push-to` option or set the PUSH_TO variable in your environment.
> - Use the `--push-to none` option for building local images in Minikube.

For components, specify:

- `am`, `ds`, `idm`, or `ig` to build a custom Docker image for a single Ping Identity Platform component.

- More than one component or set of components separated by a space to build multiple Docker images in a single forgeops build command. For example, forgeops build --env-name [.var]#my-env am idm#.
- `all` to build Docker images for all the Ping Identity Platform components[1] by running a single forgeops build command.

Options

In addition to the global forgeops command options, the forgeops build subcommand takes the following options:

`--build-path path`

The directory path where the build images are to be located. By default, the images are placed in path/to/forgeops/docker.

`--config-profile config-profile-path`

Path that contains the configuration for `am`, `idm`, or `ig`. The forgeops build command incorporates the configuration files located in this path in the custom Docker image it builds.

Configuration profiles reside in subdirectories of one of these paths in a `forgeops` repository clone:

- docker/am/config-profiles
- docker/idm/config-profiles
- docker/ig/config-profiles

Learn more in Configuration profiles.

Customized `ds` images do not use configuration profiles. To customize the `ds` image, add customizations to the docker/ds directory before running the forgeops build ds command.

`--debug`

Display debug information when executing the command.

`--dryrun`

To perform a dry run without actually building the component images.

`--env-name my-env`

The name of the deployment environment that is used for building or deploying the image. Deployment environments let you manage deployment manifests and image defaulters.

You must initialize new deployment environments before using them for the first time. You must specify the `--env-name` option in the `forgeops build` command if you have not set up the `ENV_NAME` shell environment variable.

The forgeops build command updates the image defaulter in the target environment. For example, if you ran forgeops build --env-name prod, the image defaulter in the kustomize/overlay/deploy-prod/image-defaulter directory would be updated.

`--kustomize`

The path to the directory where the Kustomize overlays and the image defaulter files for the environment are located. You can specify the full path or path relative to the local directory of your `forgeops` repository clone.

`--push-to registry`

Docker registry where the Docker image being built is pushed. You must specify the `push-to` option unless you have set the `PUSH_TO` environment variable.

For deployments on Minikube, specify `--push-to none` to push the Docker image to the Docker instance running within Minikube.

If you specify both the `--push-to` option and the `PUSH_TO` environment variable, the value of the `--push-to` option takes precedence.

`--reset`

Revert all the tags and new image names in the image defaulter file to their last committed values.

`--tag my-tag`

Tag to apply to the Docker image being built.

Examples

Normal operation

forgeops build --config-profile prod --env-name prod --tag prod-am-123 am

Do a dry run

forgeops build --config-profile prod --env-name prod --dryrun am

forgeops delete

forgeops delete --env-name my-env <components> <options>

Delete Ping Identity Platform components or sets of components, PVCs, volume snapshots, and Kubernetes secrets from a running Kustomize-based ForgeOps deployment.

By default, the forgeops delete command prompts you to confirm if you want to delete PVCs, volume snapshots, and Kubernetes secrets. You can suppress confirmation prompts as necessary by using the `--yes` option. For example, forgeops delete --env-name test --yes, deletes all Ping Identity Platform components in the `test` environment.

For components, specify:

- `am`, `ds-cts`, `ds-idrepo`, `idm`, or `ig` to delete a single Ping Identity Platform component.

- `secrets` to delete the Kubernetes secrets from the deployment.
    - `base` to delete the `platform-config` configmap, Kubernetes ingress resources, and Kubernetes secrets. Secrets generated by cert-manager are not deleted.
- `all` to delete all the Ping Identity Platform components.
- More than one component or set of components separated by a space to delete multiple Ping Identity Platform components. For example, forgeops delete --env-name my-env am idm.

The default value for components is `all`.

Options

The forgeops delete subcommand takes the following options:

`--debug`

Display debug information when executing the command.

`--dryrun`

To perform a dry run without actually deleting the components.

`--env-name my-env`

The name of the deployment environment that contains the Kustomization overlays. You must specify the `--env-name` option, otherwise the forgeops delete command fails to run.

`--force`

When deleting Ping Identity Platform components, also delete PVCs, volume snapshots, and Kubernetes secrets.

When you specify this option, you still receive the `OK to delete components?` confirmation prompt. Specify the --yes option together with --force to suppress this confirmation prompt.

`--namespace my-namespace`

The namespace from which to delete Ping Identity Platform components.

Defaults to the active namespace in your local Kubernetes context.

`--yes`

Suppress all confirmation prompts.

When you specify this option, PVCs, volume snapshots, and Kubernetes secrets are not deleted. Specify the --force option together with --yes to delete PVCs, volume snapshots, and Kubernetes secrets.

Examples

Normal operation

forgeops delete --env-name prod am

Do a dry run

forgeops delete --env-name prod am --dryrun

forgeops env

forgeops env --env-name my-env --fqdn my-fqdn

Create, configure, and manage a ForgeOps deployment environment. This command lets you define the parameters for your deployment environment, such as FQDN, certificate issuer, and so on by configuring:

- Kustomize overlay files for each component in the /path/to/forgeops/kustomize/overlay/my-env directory.
- A Helm values file in the /path/to/forgeops/helm/my-env directory.

By unifying the parameters in a location, you don't have to specify these parameters when using the other commands, such as `forgeops apply`, `forgeops build`, and so on.

`--amster-retain` *n*

Keep the `amster` pod running for n seconds. The default is 10 seconds.

`--fqdn` *my-fqdn*

A comma separated list of FQDNs. For example:

forgeops env --env-name my-env --fqdn my-fqdn1, my-fqdn2

This is a mandatory parameter. Default: None.

`--helm path/to/helm/directory`

The directory where Helm values files are located. The directory path can be relative to the `forgeops` root directory or an absolute path.

`--ingress my-ingress`

Ingress class name.

Default: None.

`--kustomize my/kustomize`

The directory that contains Kustomize overlays. The directory path can be an absolute or relative to the `forgeops` root directory.

`--namespace my-namespace`

The Kubernetes namespace where the Ping Identity Platform components are deployed.

Default: None.

`--no-namespace`

Remove namespace from Kustomize overlay.

Default: False.

`--env-name my-env`

Name of environment to manage.

Default: None.

`--single-instance`

To use a `single-instance` configuration. In a Minikube environment, you must use the `single-instance` configuration option.

Default: False.

`--source my-kust-source`

Name of the source Kustomize overlay.

Default: None.

`--ssl-secretname my-ssl-secret`

Name of the secret containing private SSL data.

Default: None

`--am-cpu, --am-mem, --am-rep`

Specify the CPU, memory, and the number of AM pod replicas.

`--cts-cpu, --cts-disk, --cts-mem, --cts-rep, --cts-snap-enable`

Specify CPU, disk size, memory, replicas, and volume snapshots for `ds-cts` pods.

`--idm-cpu --idm-mem --idm-rep`

Specify the CPU, memory, and the number of IDM pod replicas.

`--idrepo-cpu, --idrepo-disk, --idrepo-mem, --idrepo-rep, --idrepo-snap-enable`

Specify CPU, disk size, memory, replicas, and enable volume snapshots for `ds-idrepo` pods.

`--pull-policy my-pull-policy`

Set policy for all platform images.

`--no-helm`

Don't create or manage Helm values files.

Default: False.

`--no-kustomize`

Don't create or manage Kustomize overlay.

Default: False.

`--small`, `--medium`, *or* `--large`

The size of ForgeOps deployment used in the environment.

Default: None.

`--issuer my-issuer`

The TLS certificate issuer within the namespace where the ForgeOps components are to be deployed.

Default: None.

`--cluster-issuer my-cluster-issuer`

The TLS certificate issuer that is available across the Kubernetes cluster where ForgeOps components are to be deployed. For demo purposes, you can use the certificate sample certificate issuer provided with ForgeOps, by using the `--cluster-issuer default-issuer`.

Default: None.

`--skip-issuer`

Skip TLS certificate issuer setup. If you use the `--skip-issuer` option when you set up a ForgeOps deployment environment, you must set up your TLS certificate issuer before performing a ForgeOps deployment.

Default: False.

forgeops image

The forgeops image command enables you to maintain ForgeOps deployments with the latest images available. Also, you can work with multiple versions of ForgeOps-provided images, providing more flexibility to upgrade the `forgeops` tool and ForgeOps deployment.

This feature is supported for ForgeOps version 7.4 and later.

Advantages

- You can upgrade forgeops command and ForgeOps deployment separately on your schedule.
- When upgrading, you can create a new release and test it through your different ForgeOps deployment environments.
- Manage a single Git release branch instead of separate branches for each platform version.

• You can use supported container images that are regularly scanned for OS-level security vulnerabilities.

Command details

forgeops image --env-name my-env my-components

Replace my-components with one or more of `platform`, `apps`, `ui`, `am`, `amster`, `idm`, `ds`, `admin-ui`, `end-user-ui`, `login-ui`, `ig`.

Options

`--kustomize-path` ***my-kustomize-loc***

The absolute path or the path relative to the `forgeops` directory where Kustomize overlay files are stored.

Default: kustomize

`--build-path` ***my-docker-loc***

The absolute path or the path relative to the `forgeops` directory where Docker files are stored.

Default: docker

`--helm-path` ***my-helm-loc***

The absolute path or the path relative to the `forgeops` directory where Helm values files are stored.

Default: helm

`--env-name` ***my-env***

Name of ForgeOps deployment environment in which you intend to manage Docker images.

`--source` ***my-src-env***

Name of source environment if you are copying images.

`--tag` ***my-tag***

Set the tag used for images.

`--no-helm`

Don't manage Helm values files.

`--no-kustomize`

Don't manage Kustomize overlay.

`--copy`

Copy images from `--source` to --env-name.

`--release` *platform-release*

Specify platform image release to set, for example `7.5.1`.

`--release-name` *my-release*

Name of the release file in docker/component/releases. Default: my-release in UTC format.

`--releases-src` *my-release-source-url*

URL or path where release files live (default: http://releases.forgeops.com)

`--image-repo` *my-docker-repo*

The URL to the container registry that contains Docker images.

Short form	Default URL
base	us-docker.pkg.dev/forgeops-public/images-base
deploy	us-docker.pkg.dev/forgeops-public/images
dev	gcr.io/forgerock-io

Learn more about the forgeops image command in Managing Ping Identity Platform images.

1. Except for the deprecated `amster` component.

Secrets Reference

ForgeOps authentication relies on AM and IDM signing and encryption methods to protect network communication and to keep data confidential and unalterable. In turn, signing and encryption depend on keys or secrets generated using cryptographic algorithms.

This section describes various secrets and keys used in ForgeOps. Secrets, passwords, and keys used in ForgeOps are configured as environment variables or as files mounted on the Kubernetes pods.

AM configuration passwords

Kubernetes secret name: `am-env-secret`

Passwords stored as environment variables in `am` pod

- Pod: `am`
- Container: `openam`
- Type: Environment variable

Description or role	Location on container
AM_AUTHENTICATION_SHARED_SECRET	
Core authentication secret for the root realm.	◦ `cdk/config/services/realm/root/iplanetamauthservice/1.0/organizationconfig/defaultconfig.json` ◦ Value: `security.sharedSecret`
AM_ENCRYPTION_KEY	
Key used for encrypting information stored in the secure state of authentication trees in AM.	◦ `cdk/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/http___am_80_am.json` ◦ `cdk/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/server-default.json`
AM_OIDC_CLIENT_SUBJECT_IDENTIFIER_HASH_SALT	
Configuration parameter used to specify the Subject Identifier Hash Salt in the OAuth 2.0 and OIDC flows.	`base/config/services/realm/root/oauth2provider/1.0/organizationconfig/defaultconfig.json`
AM_PASSWORDS_AMADMIN_CLEAR	
Password for the amadmin user. Updated to AM_PASSWORDS_AMADMIN_HASHED in `docker-entrypoint.sh`.	`base/config/services/realm/root/sunidentityrepositoryservice/1.0/globalconfig/default/users/amadmin.json`
AM_SELFSERVICE_LEGACY_CONFIRMATION_EMAIL_LINK_SIGNING_KEY	
A 256-bit key (base64-encoded) used for HMAC signing of the legacy self-service confirmation email links.	`base/config/services/realm/root/restsecurity/1.0/organizationconfig/defaultconfig.json`
AM_SESSION_STATELESS_ENCRYPTION_KEY	
Encryption key for encrypting stateless session tokens.	`base/config/services/realm/root/iplanetamsessionservice/1.0/globalconfig/default.json`
AM_SESSION_STATELESS_SIGNING_KEY	
Signing key for validating the security of stateless session tokens.	`base/config/services/realm/root/iplanetamsessionservice/1.0/globalconfig/default.json`

Amster secrets, keys, and passwords

Kubernetes secret name: `amster`

Mounted files on `amster` *pod*

- Description: The key-pair for SSH connectivity to PingAM
- Pod: `amster`
- Container: `amster` or `pause`
- Mount path: `/var/run/secrets/amster`

Description or role	Location on container
`id_rsa`	
Private key for SSH connection to PingAM.	`/var/run/secrets/amster/id_rsa`

Mounted files on `am` *pod*

- Pod: `am`
- Container: `openam`
- Mount path: `/var/run/secrets/amster`

Description or role	Location on container
`id_rsa.pub`	
Public key for SSH connections from Amster.	`/var/run/secrets/amster/authorized_keys`

Kubernetes secret name: `amster-env-secrets`

Environment variables in `amster` *pod*

- Description: The key pairs for SSH connectivity to PingAM
- Pod: `amster`
- Container: `amster`
- Type: Environment variable

Description or role	Location on container
IDM_PROVISIONING_CLIENT_SECRET	
AM nodes in authentication journeys use this confidential client to authenticate through AM and provision identities through IDM.	Used for provisioning Oauth2Client in IDM.
IDM_RS_CLIENT_SECRET	
IDM uses this confidential client to introspect access tokens through the `am/oauth2/introspect` endpoint to get information about users.	Used in the Oauth2Client of the IDM resource server.

Environment variables in `idm` pod

- Pod: `idm`
- Container: `openidm`
- Type: Environment variable

Description or role	Location on container
IDM_RS_CLIENT_SECRET	
IDM uses this confidential client to introspect access tokens through the `am/oauth2/introspect` endpoint to get information about users.	Set in `boot.properties: "rs.client.secret"` to communicate with the Oauth2Client of the IDM resource server.

DS secrets, keys, and passwords

Kubernetes secret name: `ds-env-secrets`

Service account passwords for AM connecting to DS backends. `ldif-importer` is used to update the passwords on the DS backends.

Environment variables in `am` pod

- Pod: `am`

• Container: `openam`

• Type: Environment variables

Description or role	Location on container
AM_STORES_USER_PASSWORD	
Password for AM to access the identities backend on `ds-idrepo`.	1. `cdk/config/services/realm/root/sunidentityrepositoryservice/1.0/organizationconfig/default/opendj.json` 2. `base/config/services/realm/root/amdatastoreservice/1.0/globalconfig/default/datastorecontainer/application-store.json` 3. `base/config/services/realm/root/iplanetamauthldapservice/1.0/organizationconfig/default.json` 4. `base/config/services/realm/root/iplanetamauthldapservice/1.0/organizationconfig/defaultconfig.json` 5. `base/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/server-default.json` 6. `base/config/services/realm/root-sunamhiddenrealmdelegationservicepermissions/iplanetamauthldapservice/1.0/organizationconfig/default.json` ■ **Variables are set in the** `docker-entrypoint.sh`
AM_STORES_APPLICATION_PASSWORD	
Password for AM to access the config backend on `ds-idrepo`.	1. `cdk/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/server-default.json` 2. `base/config/services/realm/root/amdatastoreservice/1.0/globalconfig/default/datastorecontainer/application-store.json` 3. `base/config/services/realm/root/amdatastoreservice/1.0/globalconfig/default/datastorecontainer/policy-store.json` 4. `base/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/server-default.json` 5. `base/config/services/realm/root/iplanetampolicyconfigservice/1.0/organizationconfig/defaultconfig.json` 6. `base/config/services/realm/root-sunamhiddenrealmdelegationservicepermissions/iplanetampolicyconfigservice/1.0/organizationconfig/default.json`
AM_STORES_CTS_PASSWORD	

Description or role	Location on container
Password for AM to access the tokens backend on `ds-cts`.	1. `cdk/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/server-default.json` 2. `base/config/services/realm/root/iplanetamplatformservice/1.0/globalconfig/default/com-sun-identity-servers/server-default.json`

Environment variables in `ldif-importer` pod

- Pod: `ldif-importer`
- Container: `openidm`
- Type: Environment variables

Description or role	Location on container
AM_STORES_USER_PASSWORD	
Password for AM to access the identity backend on the `ds-idrepo`.	
AM_STORES_APPLICATION_PASSWORD	
Password for AM to access the configuration backend on `ds-idrepo`.	
AM_STORES_CTS_PASSWORD	
Password for AM to access the tokens backend on `ds-cts`.	

Kubernetes secret name: `ds-password`

Passwords mounted in `ds-idrepo` or `ds-cts` pods

DS management passwords for administration and monitoring.

- Pod: `ds-idrepo` or `ds-cts`
- Container: `ds`
- Mount path: `/var/run/secrets/admin`

Description or role	Location on container
dirmanager.pw	
Root password for the `uid=admin` user.	Set in `/opt/opendj/data/db/rootUser/rootUser.ldif` as `uid-admin`.

Description or role	Location on container
monitor.pw	
Password for the monitor backend. The monitor backend allows clients to access information provided by the DS server monitor providers.	Set in `/opt/opendj/data/db/monitorUser/monitorUser.ldif` as `uid=monitor`.

Passwords mounted in `idm` pods

- Pod: `idm`
- Container: `idm`
- Type: Environment variables - `OPENIDM_REPO_PASSWORD` and `USERSTORE_PASSWORD`

Description or role	Location on container
`dirmanager.pw`	
Root password for communicating with DS. Configured in `docker/idm/resolver/boot.properties`.	

Kubernetes secret name: `ds-master-keypair`

Master SSL key pair for encrypting DS data

- Pod: `ds-idrepo` or `ds-cts`
- Container: `init` and `ds`
- Mount path: `/var/run/secrets/ds-master-keypair`

Description or role	Location on container
`ca.crt`, `tls.crt`, or `tls.key`	
SSL key pair with ca self-signed cert used to encrypt DS data for backups.	`/var/run/secrets/keys/ds/master-key`. Used by `PEM Key Manager` provider configured in `ds-setup.sh`.

Kubernetes secret name: `ds-ssl-keypair`

The SSL key pair used for encrypting replication traffic. It also used by AM and IDM as a trust store for LDAPS connections to DS.

- Pod: `ds-idrepo` or `ds-cts`
- Container: `init` and `ds`

- Mount path: `/var/run/secrets/keys/ds/ds-ssl-keypair`

Description or role	Location on container
ca.crt/tls.crt/tls.key	
SSL key pair with a self-signed certificate of the certificate authority. Used for encrypting data replicated between servers.	`/var/run/secrets/keys/ds/ds-ssl-keypair`. Used by the `PEM Key Manager` provider configured in `ds-setup.sh`.

- Pod: `idm`
- Container: `truststore-init`
- Mount path: `/var/run/secrets/truststore/ca.crt`

Description or role	Location on container
ca.crt	
SSL key pair with a certificate authority signed certificate. Used for encrypting data replicated between servers.	`IDM_PEM_TRUSTSTORE_DS=/var/run/secrets/truststore/cacerts`, copied to `/opt/openidm/idmtruststore`.

- Pod: `am`
- Container: `truststore-init`
- Mount path: `/var/run/secrets/truststore/ca.crt`

Description or role	Location on container
ca.crt	
SSL key pair with the self-signed certificate of the certificate authority. Used for encrypting data replicated between servers.	`IDM_PEM_TRUSTSTORE_DS=/var/run/secrets/truststore/ca.crt`, copied to `/opt/openidm/idmtruststore`.

IDM admin passwords

Kubernetes secret name: `idm-env-secrets`

IDM admininstration and key store passwords

- Pod: `idm`
- Container: `openidm`
- Type: ENV VARS

Description or role	Location on container
OPENIDM_ADMIN_PASSWORD	
IDM admin password.	Configured in `repo.init.json`
OPENIDM_KEYSTORE_PASSWORD	
IDM key store password.	Configured in `docker/idm/resolver/boot.properties`

ForgeOps benchmarks

The benchmarking instructions in this part of the documentation give you a method to validate performance of your ForgeOps deployment.

The benchmarking techniques we present are a lightweight example, and are not a substitute for load testing a production deployment. Use our benchmarking techniques to help you get started with the task of constructing your own load tests.

When youcreate a project plan, you'll need to think about how you'll put together production-quality load tests that accurately measure your own deployment's performance.

ForgeOps benchmarking checklist

- ☐ Become familiar with ForgeOps benchmarking
- ☐ Install third-party software
- ☐ Generate test users
- ☐ Benchmark the authentication rate
- ☐ Benchmark the OAuth 2.0 authorization code flow

About ForgeOps benchmarking

ForgeOps benchmarks provides instructions for running lightweight benchmarks to give you a means for validating your own ForgeOps deployment.

The ForgeOps team runs the same benchmark tests. Our results are available upon request. To get them, contact your Ping Identity sales representative.

We conduct our tests using the configurations specified for small, medium, and large clusters. We create our clusters using the techniques described in the Setup documentation.

Next, we generate test users:

- 1,000,000 test users for a small cluster.
- 10,000,000 test users for a medium cluster.
- 100,000,000 test users for a large cluster.

Finally, we run tests that measure authentication rates and OAuth 2.0 authorization code flow performance.

If you follow the same method of performing a ForgeOps deployment and running benchmarks, the results you obtain similar results. However, factors beyond the scope of ForgeOps deployment or a failure to use our documented sizing and configuration may affect your benchmark test results. These factors might include (but are not limited to) updates to cloud platform SDKs, changes to third-party software required for Kubernetes, and changes you have made to sizing or configuration to suit your business needs.

ForgeOps deployments are designed to:

- Conform to DevOps best practices
- Facilitate continuous integration and continuous deployment
- Scale and deploy on any Kubernetes environment in the cloud

If you require higher performance than the benchmarks reported here, you can scale your deployment horizontally and vertically. Vertically scaling Ping Identity Platform works particularly well in the cloud. For more information about scaling your deployment, contact your qualified Ping Identity partner or technical consultant.

Next step

Become familiar with ForgeOps benchmarking

- ☐ Install third-party software
- ☐ Generate test users
- ☐ Benchmark the authentication rate
- ☐ Benchmark the OAuth 2.0 authorization code flow

Third-party software

The ForgeOps team uses Gradle 6.8.3 to benchmark ForgeOps deployments. Before you start running benchmarks, install this version of Gradle in your local environment.

Earlier and later versions will *probably* work. If you want to try using another version, it is your responsibility to validate it.

In addition to Gradle, you'll need all the third-party software required to perform a ForgeOps deployment

- GKE

- EKS
- AKS

Next step

Become familiar with ForgeOps benchmarking

Install third-party software

☐ Generate test users

☐ Benchmark the authentication rate

☐ Benchmark the OAuth 2.0 authorization code flow

Test user generation

Running the Authentication rate and OAuth 2.0 authorization code flow benchmarks requires a set of test users. This page provides instructions for generating a set of test users suitable for these two lightweight AM benchmarks. Note that these test users are not necessarily suitable for other benchmarks or load tests, and that they can't be used with IDM.

For small and medium clusters

Follow these steps to generate test users for lightweight AM benchmarks, provision the user stores, and prime the directory servers:

1. Set up your Kubernetes context:
    1. Set the `KUBECONFIG` environment variable so that your Kubernetes context references the cluster where you'll perform the ForgeOps deployment.
    2. Set the active namespace in your Kubernetes context to the Kubernetes namespace where you deployed the platform.
2. Obtain the password for the directory superuser, `uid=admin`:

    ```
    $ cd /path/to/forgeops/bin
    $ ./forgeops info | grep uid=admin
    ```

    Make a note of this password. You'll need it for subsequent steps in this procedure.
3. Change to the directory that contains the source for the `dsutil` Docker container:

    ```
    $ cd /path/to/forgeops/docker/ds/dsutil
    ```

    You'll generate test users from a pod you create from the `dsutil` container.
4. Build and push the `dsutil` Docker container to your container registry, and then run the container.

    The my-registry parameter varies, depending on the location of your registry:

```
$ docker build --tag=my-registry/dsutil .
$ docker push my-registry/dsutil
$ kubectl run -it dsutil --image=my-registry/dsutil --restart=Never -- bash
```

The kubectl run command creates the `dsutil` pod, and leaves you in a shell that lets you run commands in the pod.

5. Generate the test users—1,000,000 users for a small cluster and 10,000,000 for a medium cluster:

   Run these substeps from the `dsutil` pod's shell:

   1. Make an LDIF file that has the number of user entries for your cluster size:

      For example, for a small cluster:

      ```
      $ /opt/opendj/bin/makeldif -o data/entries.ldif \
       -c numusers=1000000 config/MakeLDIF/ds-idrepo.template
      Processed 1000 entries
      Processed 2000 entries
      Processed 3000 entries
      ...
      Processed 1000000 entries
      LDIF processing complete. 1000003 entries written
      ```

      When the ForgeOps team ran the makeldif script, it took approximately:

      - 30 seconds to run on a small cluster.
      - 4 minutes to run on a medium cluster.

   2. Create the user entries in the directory:

      ```
      $ /opt/opendj/bin/ldapmodify \
       -h ds-idrepo-0.ds-idrepo -p 1389 --useStartTls --trustAll \
       -D "uid=admin" -w directory-superuser-password --noPropertiesFile \
       --no-prompt --continueOnError --numConnections 10 data/entries.ldif
      ```

      `ADD operation successful` messages appear as user entries are added to the directory.

      When the ForgeOps team ran the ldapmodify command, it took approximately:

      - 15 minutes to run on a small cluster.
      - 2 hours 35 minutes to run on a medium cluster.

6. Prime the directory servers:

   1. Open a new terminal window or tab.

      Use this new terminal window—not the one running the `dsutil` pod's shell—for the remaining substeps in this step.

2. Prime the directory server running in the `ds-idrepo-0` pod:

    1. Start a shell that lets you run commands in the `ds-idrepo-0` pod:

    ```
    $ kubectl exec ds-idrepo-0 -it -- bash
    ```

    2. Run the following command:

    ```
    $ ldapsearch -D "uid=admin" -w directory-superuser-password \
     -p 1389 -b "ou=identities"  uid=user.*  | grep dn: | wc -l
    10000000
    ```

    3. Exit from the `id-dsrepo-0` pod's shell:

    ```
    $ exit
    ```

3. Prime the directory server running in the `ds-idrepo-1` pod.

For large clusters

Here are some very general steps you can follow if you want to generate test users for benchmarking or load testing a large cluster:

1. Install DS in a VM in the cloud.
2. Run the `makeldif` and `ldapmodify` commands, as described above.
3. Back up your directory.
4. Upload the backup files to cloud storage.
5. Restore an `idrepo` pod from your backup following steps similar to the procedure in Restore.

Next step

Become familiar with ForgeOps benchmarking

Install third-party software

Generate test users

- ☐ Benchmark the authentication rate
- ☐ Benchmark the OAuth 2.0 authorization code flow

Authentication rate

The `AMRestAuthNSim.scala` simulation tests authentication rates using the REST API. It measures the throughput and response times of an AM server performing REST authentications when AM is configured to use CTS-based sessions.

To run the simulation:

1. Make sure the userstore is provisioned, and the PingDS cache is primed.

   Refer to Test user generation.

2. Set environment variables that specify the host on which to run the test, the number of concurrent threads to spawn when running the test, the duration of the test (in seconds), the first part of the user ID, and the user password, and the number of users for the test:

   ```
   $ export TARGET_HOST=
   $ export CONCURRENCY=100
   $ export DURATION=60
   $ export USER_PREFIX=user.
   $ export USER_PASSWORD=T35tr0ck123
   $ export USER_POOL=n-users
   ```

   where *n-users* is `1000000` for a small cluster, `10000000` for a medium cluster, and `100000000` for a large cluster.

3. Configure AM for CTS-based sessions:

   1. Log in to the Ping Identity Platform admin UI as the `amadmin` user. For details, refer to AM Services.
   2. Access the AM admin UI.
   3. Select the top level realm.
   4. Select Properties.
   5. Make sure the Use Client-based Sessions option is disabled.

      If it's not disabled, disable it, and then select Save Changes.

4. Change to the /path/to/forgeops/docker/gatling directory.

5. Run the simulation:

   ```
   $ gradle clean; gradle gatlingRun-am.AMRestAuthNSim
   ```

   When the simulation is complete, the name of a file containing the test results appears near the end of the output.

6. Open the file containing the test results in a browser to review the results.

Next step

Become familiar with ForgeOps benchmarking

Install third-party software

Generate test users

Benchmark the authentication rate

☐ Benchmark the OAuth 2.0 authorization code flow

OAuth 2.0 authorization code flow

The `AMAccessTokenSim.scala` simulation tests OAuth 2.0 authorization code flow performance. It measures the throughput and response time of an AM server performing authentication, authorization, and session token management when AM is configured to use client-based sessions, and OAuth 2.0 is configured to use client-based tokens. In this test, one transaction includes all three operations.

To run the simulation:

1. Make sure the userstore is provisioned, and the PingDS cache is primed.

   Refer to Test user generation.

2. Set environment variables that specify the host on which to run the test, the number of concurrent threads to spawn when running the test, the duration of the test (in seconds), the first part of the user ID, and the user password, and the number of users for the test:

   ```
   $ export TARGET_HOST=my-fqdn
   $ export CONCURRENCY=100
   $ export DURATION=60
   $ export USER_PREFIX=user.
   $ export USER_PASSWORD=T35tr0ck123
   $ export USER_POOL=n-users
   ```

   where *n-users* is `1000000` for a small cluster, `10000000` for a medium cluster, and `100000000` for a large cluster.

3. Configure AM for CTS-based sessions:

   1. Log in to the Ping Identity Platform admin UI as the `amadmin` user. For details, refer to AM Services.
   2. Access the AM admin UI.
   3. Select the top level realm.
   4. Select Properties.
   5. Make sure the Use Client-based Sessions option is disabled.

      If it's not disabled, disable it, and then select Save Changes.

4. Configure AM for CTS-based OAuth2 tokens:

   1. Select Realms > Top Level Realm.
   2. Select Services > OAuth2 Provider.
   3. Make sure the Use Client-based Access & Refresh Tokens option is disabled.

      If it's not disabled, disable it, and then select Save Changes.

5. Change to the /path/to/forgeops/docker/gatling directory.

6. Run the simulation:

```
$ gradle clean; gradle gatlingRun-am.AMAccessTokenSim
```

When the simulation is complete, the name of a file containing the test results appears near the end of the output.

7. Open the file containing the test results in a browser to review the results.

Congratulations!

You've successfully run the lightweight benchmark tests on a ForgeOps deployment.

Become familiar with ForgeOps benchmarking

Install third-party software

Generate test users

Benchmark the authentication rate

Benchmark the OAuth 2.0 authorization code flow

Ingress

By default, ForgeOps deployments use Ingress-NGINX controller.

For deployments on GKE, EKS, and AKS, the tf-apply cluster creation script deploys Ingress-NGINX Controller when it creates new Kubernetes clusters. Alternatively, you can deploy HAProxy Ingress as your ingress controller.

For deployments on Minikube, the minikube start command example installs the ingress add-on in your Minikube cluster.

HAProxy Ingress

This section lists adjustments you'll need to make if you want to perform a ForgeOps deployment that uses HAProxy Ingress as the ingress controller instead of Ingress-NGINX controller.

When you create your GKE, EKS, or AKS cluster:

1. Before you run the tf-apply script, configure Terraform to deploy HAProxy Ingress in your cluster.

   Modify these values under `cluster.tf_cluster_gke_small` in the override.auto.tfvars file:

   1. Set the value of the `helm.ingress-nginx.deploy` variable to `false`.
   2. Set the value of the `helm.ingress-haproxy.deploy` variable to `false`.

2. After you have run the tf-apply script, deploy HAProxy Ingress Controller by running the bin/ingress-controller-deploy.sh script.

   Be sure to specify the `-i haproxy` option when you run the script.

3. To get the ingress controller's external IP address on your GKE, EKS, or AKS cluster, specify --namespace haproxy-ingress (instead of --namespace nginx-ingress) when you run the kubectl get services command. For example:

```
$ kubectl get services --namespace haproxy-ingress
NAME              TYPE           CLUSTER-IP   EXTERNAL-IP  PORT(S)                      AGE
haproxy-ingress   LoadBalancer   10.84.6.68   34.82.11.221 80:32288/TCP,443:32325/TCP   38s
...
```

When you perform your ForgeOps deployment:

1. Specify the --ingress-class haproxy argument. For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops apply --small --ingress-class haproxy --fqdn my-fqdn --namespace my-namespace
```

Glossary

affinity (AM)

AM affinity deployment lets AM spread the LDAP requests load over multiple directory server instances. Once a CTS token is created and assigned to a session, AM sends all further token operations to the same token origin directory server from any AM node. This ensures that the load of CTS token management is spread across directory servers.

Source: CTS Affinity Deployment in the Core Token Service (CTS) documentation

Amazon EKS

Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed service that makes it easy for you to run Kubernetes on Amazon Web Services without needing to set up or maintain your own Kubernetes control plane.

Source: What is Amazon EKS in the Amazon EKS documentation

ARN (AWS)

An Amazon Resource Name (ARN) uniquely identifies an Amazon Web Service (AWS) resource. AWS requires an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies and API calls.

Source: Amazon Resource Names (ARNs) in the AWS documentation

AWS IAM Authenticator for Kubernetes

The AWS IAM Authenticator for Kubernetes is an authentication tool that lets you use Amazon Web Services (AWS) credentials for authenticating to a Kubernetes cluster.

Source: AWS IAM Authenticator for Kubernetes README file on GitHub

Azure Kubernetes Service (AKS)

AKS is a managed container orchestration service based on Kubernetes. AKS is available on the Microsoft Azure public cloud. AKS manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications.

Source: Azure Kubernetes Service in the Microsoft Azure documentation

cloud-controller-manager

The `cloud-controller-manager` daemon runs controllers that interact with the underlying cloud providers. The `cloud-controller-manager` daemon runs provider-specific controller loops only.

Source: cloud-controller-manager in the Kubernetes Concepts documentation

ForgeOps deployment

A ForgeOps deployment is a deployment of the Ping Identity Platform on Kubernetes based on Docker images, Helm charts, Kustomize bases and overlays, utility programs, and other artifacts you can find in the `forgeops` repository on GitHub.

A *single-instance ForgeOps deployment* is a special ForgeOps deployment that you use to configure AM and IDM and build custom Docker images for the Ping Identity Platform. They are called single-instance deployments because unlike small, medium, and large deployments, they have only single pods that run AM and IDM. They are only suitable for developing the AM and IDM configurations and must not be used for testing performance, monitoring, security, and backup requirements in production environments.

Source: Deployment overview

CloudFormation (AWS)

CloudFormation is a service that helps you model and set up your AWS resources. You create a template that describes all the AWS resources that you want. CloudFormation takes care of provisioning and configuring those resources for you.

Source: What is AWS CloudFormation? in the AWS documentation

CloudFormation template (AWS)

An AWS CloudFormation template describes the resources that you want to provision in your AWS stack. AWS CloudFormation templates are text files formatted in JSON or YAML.

Source: Working with AWS CloudFormation Templates in the AWS documentation

cluster

A container cluster is the foundation of Kubernetes Engine. A cluster consists of at least one control plane and multiple worker machines called nodes. The Kubernetes objects that represent your containerized applications all run on top of a cluster.

Source: Standard cluster architecture in the Google Kubernetes Engine (GKE) documentation

ConfigMap

A configuration map, called `ConfigMap` in Kubernetes manifests, binds the configuration files, command-line arguments, environment variables, port numbers, and other configuration artifacts to the assigned containers and system components at runtime. The configuration maps are useful for storing and sharing non-sensitive, unencrypted configuration information.

Source: ConfigMap in the Google Kubernetes Engine (GKE) documentation

container

A container is an allocation of resources such as CPU, network I/O, bandwidth, block I/O, and memory that can be "contained" together and made available to specific processes without interference from the rest of the system. Containers decouple applications from underlying host infrastructure.

Source: Containers in the Kubernetes Concepts documentation

control plane

A control plane runs the control plane processes, including the Kubernetes API server, scheduler, and core resource controllers. GKE manages the lifecycle of the control plane when you create or delete a cluster.

Source: Control plane in the Google Kubernetes Engine (GKE) documentation

DaemonSet

A set of daemons, called `DaemonSet` in Kubernetes manifests, manages a group of replicated pods. Usually, the daemon set follows a one-pod-per-node model. As you add nodes to a node pool, the daemon set automatically distributes the pod workload to the new nodes as needed.

Source: DaemonSet in the Google Cloud documentation

deployment

A Kubernetes deployment represents a set of multiple, identical pods. Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive.

Source: Deployments in the Kubernetes Concepts documentation

deployment controller

A deployment controller provides declarative updates for pods and replica sets. You describe a desired state in a deployment object, and the deployment controller changes the actual state to the desired state at a controlled rate. You can define deployments to create new replica sets, or to remove existing deployments and adopt all their resources with new deployments.

Source: Deployments in the Google Cloud documentation

Docker container

A Docker container is a runtime instance of a Docker image. The container is isolated from other containers and its host machine. You can control how isolated your container's network, storage, or other underlying subsystems are from other containers or from the host machine.

Source: Containers in the Docker Getting Started documentation

Docker daemon

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A Docker daemon can also communicate with other Docker daemons to manage Docker services.

Source: The Docker daemon section in the Docker Overview documentation

Docker Engine

Docker Engine is an open source containerization technology for building and containerizing applications. Docker Engine acts as a client-server application with:

- A server with a long-running daemon process, `dockerd`.
- APIs, which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command-line interface (CLI) client, `docker`. The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manages Docker objects, such as images, containers, networks, and volumes.

Source: Docker Engine overview in the Docker documentation

Dockerfile

A Dockerfile is a text file that contains the instructions for building a Docker image. Docker uses the Dockerfile to automate the process of building a Docker image.

Source: Dockerfile reference in the Docker documentation

Docker Hub

Docker Hub provides a place for you and your team to build and ship Docker images. You can create public repositories that can be accessed by any other Docker Hub user, or you can create private repositories you can control access to.

Source: Docker Hub Quickstart section in the Docker Overview documentation

Docker image

A Docker image is an application you would like to run. A container is a running instance of an image.

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.

An image includes the application code, a runtime engine, libraries, environment variables, and configuration files that are required to run the application.

Source: Docker objects section in the Docker Overview documentation

Docker namespace

Docker namespaces provide a layer of isolation. When you run a container, Docker creates a set of namespaces for that container. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

The `PID` namespace is the mechanism for remapping process IDs inside the container. Other namespaces such as net, mnt, ipc, and uts provide the isolated environments we know as containers. The user namespace is the mechanism for remapping user IDs inside a container.

Source: The underlying technology section in the Docker Overview documentation

Docker registry

A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can also run your own private registry.

Source: Docker registries section in the Docker Overview documentation

Docker repository

A Docker repository is a public, certified repository from vendors and contributors to Docker. It contains Docker images that you can use as the foundation to build your applications and services.

Source: Manage repositories in the Docker documentation

dynamic volume provisioning

The process of creating storage volumes on demand is called dynamic volume provisioning. Dynamic volume provisioning lets you create storage volumes on demand. It automatically provisions storage when it is requested by users.

Source: Dynamic Volume Provisioning in the Kubernetes Concepts documentation

egress

An egress controls access to destinations outside the network from within a Kubernetes network. For an external destination to be accessed from a Kubernetes environment, the destination should be listed as an allowed destination in the allowlist configuration.

Source: Network Policies in the Kubernetes Concepts documentation

firewall rule

A firewall rule lets you allow or deny traffic to and from your virtual machine instances based on a configuration you specify. Each Kubernetes network has a set of firewall rules controlling access to and from instances in its subnets. Each firewall rule is defined to apply to either incoming (ingress) or outgoing (egress) traffic, not both.

Source: VPC firewall rules in the Google Cloud documentation

garbage collection

Garbage collection is the process of deleting unused objects. Kubelets perform garbage collection for containers every minute, and garbage collection for images every five minutes. You can adjust the high and low threshold flags and garbage collection policy to tune image garbage collection.

Source: Garbage Collection in the Kubernetes Concepts documentation

Google Kubernetes Engine (GKE)

The Google Kubernetes Engine (GKE) is an environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple machine instances grouped together to form a container cluster.

Source: GKE overview in the Google Cloud documentation

horizontal pod autoscaler

The horizontal pod autoscaler enables the cluster to automatically increase or decrease the number of pods in a replication controller, deployment, replica set, or stateful set based on observed CPU utilization. Users can specify the CPU utilization target to enable the controller to adjust the number of replicas.

Source: Horizontal Pod Autoscaler in the Kubernetes documentation

ingress

An ingress is a collection of rules that allow inbound connections to reach the cluster services.

Source: Ingress in the Kubernetes Concepts documentation

instance group

An instance group is a collection of virtual machine instances. The instance groups lets you easily monitor and control the group of virtual machines together.

Source: Instance groups in the Google Cloud documentation

instance template

An instance template is a global API resource to create VM instances and managed instance groups. Instance templates define instance properties such as machine type, image, zone, labels, and so on. They are very helpful in replicating the environments.

Source: Instance templates in the Google Cloud documentation

kubectl

The kubectl command-line tool supports several different ways to create and manage Kubernetes objects.

Source: Kubernetes Object Management in the Kubernetes Concepts documentation

kube-controller-manager

The Kubernetes controller manager embeds core controllers shipped with Kubernetes. Each controller is a separate process. To reduce complexity, the controllers are compiled into a single binary and run in a single process.

Source: kube-controller-manager in the Kubernetes Reference documentation

kubelet

A kubelet is an agent that runs on each node in the cluster. It ensures that containers are running in a pod.

Source: kubelet in the Kubernetes Concepts documentation

kube-scheduler

The `kube-scheduler` component is on the master node. It watches for newly created pods that do not have a node assigned to them, and selects a node for them to run on.

Source: kube-scheduler in the Kubernetes Concepts documentation

Kubernetes

Kubernetes is an open source platform designed to automate deploying, scaling, and operating application containers.

Source: Overview in the Kubernetes Concepts documentation

Kubernetes DNS

A Kubernetes DNS pod is a pod used by the kubelets and the individual containers to resolve DNS names in the cluster.

Source: DNS for Services and Pods in the Kubernetes Concepts documentation

Kubernetes namespace

Kubernetes supports multiple virtual clusters backed by the same physical cluster. A Kubernetes namespace is a virtual cluster that provides a way to divide cluster resources between multiple users. Kubernetes starts with three initial namespaces:

- `default` : The default namespace for user created objects which don't have a namespace.
- `kube-system` : The namespace for objects created by the Kubernetes system.
- `kube-public` : The automatically created namespace that is readable by all users.

Source: Namespaces in the Kubernetes Concepts documentation

Let's Encrypt

Let's Encrypt is a free, automated, and open certificate authority.

Source: Let's Encrypt website

Microsoft Azure

Microsoft Azure is the Microsoft cloud platform, including infrastructure as a service (IaaS) and platform as a service (PaaS) offerings.

Source: What is Azure? in the Microsoft Azure documentation

network policy

A Kubernetes network policy specifies how groups of pods are allowed to communicate with each other and with other network endpoints.

Source: Network Policies in the Kubernetes Concepts documentation

node (Kubernetes)

A Kubernetes node is a virtual or physical machine in the cluster. Each node is managed by the master components and includes the services needed to run the pods.

Source: Nodes in the Kubernetes documentation

node controller (Kubernetes)

A Kubernetes node controller is a Kubernetes master component that manages various aspects of the nodes, such as lifecycle operations, operational status, and maintaining an internal list of nodes.

Source: Node Controller in the Kubernetes Concepts documentation

node pool (Kubernetes)

A Kubernetes node pool is a collection of nodes with the same configuration. At the time of creating a cluster, all the nodes created in the `default` node pool. You can create your custom node pools for configuring specific nodes that have different resource requirements such as memory, CPU, and disk types.

Source: About node pools in the Google Kubernetes Engine (GKE) documentation

persistent volume

A persistent volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins that have a lifecycle independent of any individual pod that uses the PV.

Source: Persistent Volumes in the Kubernetes Concepts documentation

persistent volume claim

A persistent volume claim (PVC) is a request for storage by a user. A PVC specifies size and access modes such as:

- Mounted once for read and write access
- Mounted many times for read-only access

Source: Persistent Volumes in the Kubernetes Concepts documentation

pod anti-affinity (Kubernetes)

Kubernetes pod anti-affinity constrains which nodes can run your pod, based on labels on the pods that are already running on the node, rather than based on labels on nodes. Pod anti-affinity lets you control the spread of workload across nodes and also isolate failures to nodes.

Source: Assigning Pods to Nodes in the Kubernetes Concepts documentation

pod (Kubernetes)

A Kubernetes pod is the smallest, most basic deployable object in Kubernetes. A pod represents a single instance of a running process in a cluster. Containers within a pod share an IP address and port space.

Source: Pods in the Kubernetes Concepts documentation

region (Azure)

An Azure region, also known as a location, is an area within a geography, containing one or more data centers.

Source: region in the Microsoft Azure glossary

replication controller (Kubernetes)

A replication controller ensures that a specified number of Kubernetes pod replicas are running at any one time. The replication controller ensures that a pod or a homogeneous set of pods is always up and available.

Source: ReplicationController in the Kubernetes Concepts documentation

resource group (Azure)

A resource group is a container that holds related resources for an application. The resource group can include all the resources for an application, or only those resources that are logically grouped together.

Source: resource group in the Microsoft Azure glossary

secret (Kubernetes)

A Kubernetes secret is a secure object that stores sensitive data, such as passwords, OAuth 2.0 tokens, and SSH keys in your clusters.

Source: Secrets in the Kubernetes Concepts documentation

security group (AWS)

A security group acts as a virtual firewall that controls the traffic for one or more compute instances.

Source: Amazon EC2 security groups for Linux instances in the AWS documentation

service (Kubernetes)

A Kubernetes service is an abstraction that defines a logical set of pods and a policy by which to access them. This is sometimes called a microservice.

Source: Service in the Kubernetes Concepts documentation

service principal (Azure)

An Azure service principal is an identity created for use with applications, hosted services, and automated tools to access Azure resources. Service principals let applications access resources with the restrictions imposed by the assigned roles instead of accessing resources as a fully privileged user.

Source: Create an Azure service principal with Azure PowerShell in the Microsoft Azure PowerShell documentation

shard

Sharding is a way of partitioning directory data so that the load can be shared by multiple directory servers. Each data partition, also known as a shard, exposes the same set of naming contexts, but only a subset of the data. For example, a distribution might have two shards. The first shard contains all users whose names begin with A-M, and the second contains all users whose names begin with N-Z. Both have the same naming context.

Source: Class Partition in the DS Javadoc

single-instance ForgeOps deployment

Refer to ForgeOps deployment.

stack (AWS)

A stack is a collection of AWS resources that you can manage as a single unit. You can create, update, or delete a collection of resources by using stacks. The AWS template defines all the resources in a stack.

Source: Working with stacks in the AWS documentation

stack set (AWS)

A stack set is a container for stacks. You can provision stacks across AWS accounts and regions by using a single AWS template. A single template defines the resources included in each stack of a stack set.

Source: StackSets concepts in the AWS documentation

subscription (Azure)

An Azure subscription is used for pricing, billing, and payments for Azure cloud services. Organizations can have multiple Azure subscriptions, and subscriptions can span multiple regions.

Source: subscription in the Microsoft Azure glossary

volume (Kubernetes)

A Kubernetes volume is a storage volume that has the same lifetime as the pod that encloses it. Consequently, a volume outlives any containers that run within the pod, and data is preserved across container restarts. When a pod ceases to exist, the Kubernetes volume also ceases to exist.

Source: Volumes in the Kubernetes Concepts documentation

volume snapshot (Kubernetes)

In Kubernetes, you can copy the content of a persistent volume at a point in time, without having to create a new volume. You can efficiently back up your data using volume snapshots.

Source: Volume Snapshots in the Kubernetes Concepts documentation

VPC (AWS)

A virtual private cloud (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud.

Source: What Is Amazon VPC? in the AWS documentation

worker node (AWS)

An Amazon Elastic Container Service for Kubernetes (Amazon EKS) worker node is a standard compute instance provisioned in Amazon EKS.

Source: Self-managed nodes in the AWS documentation

workload (Kubernetes)

A Kubernetes workload is the collection of applications and batch jobs packaged into a container. Before you deploy a workload on a cluster, you must first package the workload into a container.

Source: Workloads in the Kubernetes Concepts documentation

Beyond the docs

Useful links that cover topics beyond the scope of this documentation.

Development topics

- Get a full Amster export out of a ForgeOps deployment

Deployment topics

- Deploy and customize Prometheus, Grafana, and Alertmanager in a ForgeOps deployment
- Deploy the platform in a multi-cluster environment using Google Cloud Multi Cluster Ingress and Cloud DNS for GKE
- Import a certificate into the truststore in a ForgeOps deployment
- Enable the IDM workflow in a ForgeOps deployment
- ForgeOps deployment to Minikube on M1 or M2 based Mac running Colima

DS script guide

- An overview of DS scripts to customize, build and deploy DS Docker images

Troubleshooting

- Enable and modify the AM logging level (applies to ForgeOps 2025.1.1)
- Enable and modify the IDM logging level (applies to ForgeOps 2025.1.1)
- Enable and modify the audit logging level (applies to ForgeOps 2025.1.1)

ForgeOps2025.1release notes

Get an email when there's an update to ForgeOps `2025.1` documentation. Go to the Notifications page in your Backstage profile and select ForgeOps `2025.1` Changes in the Documentation Digests section.

Or subscribe to the ForgeOps 2025.1 RSS feed.

> **Note**
>
> Learn about how to configure GitHub notifications here so you can get notified on ForgeOps releases.

Important information for this ForgeOps release:

Validated Kubernetes, Ingress-NGINX Controller, HAProxy Ingress, cert-manager, and operator versions for deploying Ping Identity Platform 2025.1.2	Link
Limitations when deploying Ping Identity Platform `2025.1.2` on Kubernetes	Link
More information about the rapidly evolving nature of the `forgeops` repository, including technology previews, legacy features, and feature deprecation and removal	Link
Legal notices	Link
Archive of release notes in 2024 and before are available from ForgeOps release 7.5 documentation	Link
Archive of release notes in 2023 and before are available from ForgeOps release 7.4 documentation	Link

2025

June 4, 2025

Documentation updates

Added reference for secrets

Included a section to describe the different Kubernetes secrets used in ForgeOps. Learn more in Secrets Reference.

May 21, 2025

Documentation updates

Workaround for AM base image creation

The script used for generating AM base image from AM zip file had a flaw. A workaround has been documented. Learn more in Base Docker images.

May 6, 2025

Documentation updates

Updated Java version

Steps to build customized base images are updated to use Java version 21. Learn more at Base Docker images.

May 1, 2025

New ForgeOps 2025.1.2 released

New features and updated functionality

New PingGateway version available

PingGateway 2025.3.0 Docker image has been released. The forgeops command has been updated to deploy PingGateway in a ForgeOps deployment.

Updated PingGateway deployment to use the new `admin` endpoint

PingGateway has two endpoints now:

- `/ig` the main entry point to PingGateway
- `/admin` the API of the PingGateway administration, containing the `/ping` handler used for live check, for example.

Updated the Kubernetes version to 1.32

The Terraform cluster creation manifests have been updated to use Kubernetes version 1.32 on all platforms.

Custom environment variables in Helm chart

Implemented a mechanism to define extra environment variables for AM, IDM and custom variables to the platform configuration map.

Update the `values.yaml` file for your environments with the desired configuration. The `env` arrays should contain maps of Kubernetes environment configurations. The following sections in the `charts/identity-platform/values.yaml` file contain examples:

- platform.configMap.data: Map of custom key,value pairs for `platform-config`
- platform.env: Shared custom environment variables
- am.env: AM custom environment variables
- idm.env: IDM custom environment variables

The install-prereqs script is updated

The following new features have been added to the install-prereqs script:

- A usage statement.

- The `--upgrade` flag for easy upgrading of prereqs.
- The ability to provide a config file to pin versions.

Prometheus and Grafana added to the Helm chart

Added the ability to enable Prometheus and Grafana in the Helm chart.

Improved release detection

Using forgeops image and forgeops info commands, you can now look for and select a newer version, skipping the version you specified in the command if it isn't available.

Bugfixes

Fix `--amster-retain` option

Added the `--amster-retain` option to the forgeops env command. You can configure a ForgeOps deployment environment to keep the `amster` pod running for troubleshooting purposes.

Fix VolumeSnapshots in Kustomize deployments

The forgeops env command now adds a patch to update the namespace when enabling volume snapshots.

Removed Features

Removed the forgeops generate command

The deprecated forgeops generate command has been removed.

Removed the separate scripts to deploy `certmanager` and `secret-agent`

The `certmanager-deploy.sh` and `secret-agent` scripts have been removed in favor of the `charts/scripts/install-prereqs` script, which includes steps to deploy `certmanager` and `secret-agent`.

April 24, 2025

ForgeOps updates

Platform product Docker images released

The following platform Docker images have been released:

PingAM	**8.0.1**
PingAM configuration upgrader	**8.0.1**
Amster	**8.0.1**
PingDS	**8.0.0**
PingIDM	**8.0.0**

PingGateway	**2025.3.0**
admin-ui	**8.0.0**
end-user-ui	**8.0.0**

April 22, 2025

ForgeOps updates

The `debug-logs` utility updated

The `debug-log` utility is updated to use the new ForgeOps deployment environments and parameters. Learn more at Kubernetes logs and other diagnostics.

April 17, 2025

ForgeOps updates

Updated Docker image for PingGateway

The Docker image for PingGateway is updated to 2025.3.0.

April 15, 2025

Documentation update

Moved Ingress and Benchmark sections

Ingress and Benchmark sections are moved from the Prepare branch to the Reference navigation branch.

April 9, 2025

Documentation updates

Third-party software

Updated the list of third-party software required for ForgeOps deployment. Included third-party software requirement in the Getting Started section. Learn more at your cluster environment at Setup overview.

ForgeOps updates

Prometheus update

Monitoring tools Grafana and Prometheus have been updated to use the latest versions, along with newer monitoring endpoints. Learn more at About ForgeOps deployment monitoring.

April 4, 2025

Documentation updates

Removed the `disaster` subcommand from the ds-debug command

The DS team has removed the `disaster` subcommand from the ds-debug command. Accordingly, that subcommand description is removed from the Troubleshooting section.

Fixed the name of the ingress controller used

The name of the ingress controller used by default in ForgeOps deployment is corrected to Ingress-NGINX controller.

Corrected steps to install PingGateway

Procedures to install PingGateway are corrected. Learn more at Deploy PingGateway and Custom PingGateway image.

March 19, 2025

Documentation updates

Revise steps to enable volume snapshots

The steps to enable volume snapshots have been simplified with the use of the forgeops env command. Learn more in Backup and restore using volume snapshots.

Command reference for forgeops image

Added the command reference for the forgeops image command. Learn more at the forgeops image command reference page.

March 05, 2025

Documentation updates

Revamp the Upgrade section

The Upgrade document section is updated to cover the new format of the forgeops command and the ForgeOps deployment environment. Learn more in the Upgrade Overview section.

Update the Troubleshooting `amster` section

The amster command has been subsumed in the forgeops amster command. Learn more in the Troubleshooting `amster` pod section.

February 19, 2025

New ForgeOps 2025.1 released

New features and updated functionality

Ability to set `FORGEOPS_ROOT`

You can set `FORGEOPS_ROOT` parameter to specify the local folder that contains the Docker, Helm, and Kustomize configurations. This allows you to keep your changes in a separate Git repository. You can create a `~/.forgeops.conf` file with your overrides. Your development team can place a `forgeops.conf` file in their `FORGEOPS_ROOT` location which contains team-wide settings.

You can clone the `forgeops` repository and check out only the version tag you need. This makes it easier to keep track of the ForgeOps version you're using and upgrade to a newer version consistently.

> **Important**
>
> Don't create or modify the `forgeops.conf` file in the `/path/to/forgeops_repo/` directory.

forgeops info command can provide release information

You can now get a list of supported platform releases and their latest flags using the forgeops info --list-releases command.

You can get details for any release on `releases.forgeops.com` using the forgeops info --release xyz command.

forgeops env command supports PingGateway

You can now define and update PingGateway node configuration parameters, such as CPU, memory, replicas, and pull policy in a ForgeOps deployment environment. This lets you install PingGateway quickly in a ForgeOps deployment.

Version of `pyyaml` *is updated*

The version of `pyyaml` is updated. Run the [.command]forgeops configure# command to update your libraries.

Bugfixes

`forgeops info --env-name` *command has been fixed*

The timestamp issue in the forgeops info --env-name has been fixed.

DS certificates are now deployed in Helm pre-install

Helm pre-install hooks are now used to deploy DS certificates. These certificates are no longer deleted when the Helm chart is uninstalled.

AM service target ports are updated

Updated the AM service in the Helm chart to use HTTPS target port.

Prometheus ports are updated

Prometheus default ports and labels have been updated to match the new Helm chart.

Documentation updates

Upgrade procedures revised

The procedures to upgrade ForgeOps artifacts and component images are revised. Learn more in Upgrade Overview.

February 10, 2025

New features and updated functionality

Added sample storage class definition files

We've added sample storage class definition files required for ForgeOps deployment. This helps users who are setting up Kubernetes clusters without using the ForgeOps-provided Terraform manifests.

Documentation updates

Updated the procedure to set up Minikube cluster

Because we've removed the `forgeops-minikube` script, we've revised the steps to create a Minikube cluster to use the generic minikube command. Learn more about creating a Minikube cluster here.

Updated the procedure to perform ForgeOps deployment on Minikube

We've added the step to create the `fast` storage class required for ForgeOps deployment on Minikube.

January 27, 2025

Documentation updates

Revised instruction for deployment on Minikube

Revised the procedure to perform ForgeOps deployment on Minikube using generic Kubernetes tools rather than proprietary `forgeops-minikube` utility.

Learn the revised steps to perform ForgeOps deployment on Minikube:

- Using Helm.
- Using Kustomize.

January 13, 2025

New features and updated functionality

The ForgeOps releases are based on the `main` branch

The `master` branch of `forgeops` repository is no longer used. The ForgeOps artifacts are released from the `main` branch. The latest Docker images are tagged as `dev` images. You can view the available Docker images using the forgeops image command.

New forgeops command

- The forgeops-ng command has been renamed forgeops. The new forgeops command subsumes all the functionality provided by the previous version of forgeops command. The previous version of the forgeops-ng command has been removed.
- The process of deploying and managing ForgeOps deployments has been improved with the use of provisioning environments with the forgeops env command for both Kustomize and Helm user environments. Learn more about the forgeops env command in the forgeops env command].
- Provided an option to select the Docker image as appropriate for a user deployment with the forgeops image command.
- You can view configured environments and product versions using the forgeops info command.

Learn more in forgeops command reference

ForgeOps-provided Docker images are now supported

Ping Identity now supports ForgeOps-provided Docker images. We've revised the documentation and removed the "unsupported" admonition.

New supported product versions

Platform UI	7.5.1
PingAM	7.4.1, 7.5.1
PingDS	7.4.3, 7.5.1
PingGateway	2024.6.0, 2024.9.0, 2024.11.0
PingIDM	7.5.0

Removed legacy DS docker directories

Removed the legacy docker/ds/idrepo and docker/ds/cts directories. The content that was in docker/ds/ds-new is now in docker/ds.

Removed the requirement to build `ldif-importer`

The `ldif-importer` component uses the DS Docker image, you don't need to build a separate Docker image. The required `ldif-importer` scripts are mounted to the `ldif-importer` pod using a configmap.

Documentation updates

New forgeops command reference

The new forgeops command reference contains more information on the new forgeops command.

Description of the release process

Learn more about the new ForgeOps release process here

New section on customizing DS image

Learn more about customizing DS image in the new section on Customizing DS image.

2024

December 05, 2024

Documentation updates

Added description of the release process

Learn more about the new ForgeOps release process

Moved the forgeops command description and reference to the Reference section

The new forgeops command is supported, so we've moved the corresponding documentation pages to the Reference section. Learn more in the forgeops command reference.

> **Note**
>
> The previous version of the forgeops utility is not supported in this ForgeOps release. It continues to be supported in ForgeOps 7.5 and 7.4, as long as the corresponding Ping Identity Platform components are supported.

Moved Base Docker Image page to the Reference section

Considering the ForgeOps-provided docker images are supported, you need to build base Docker images only in special cases. Accordingly, we've moved the Base Docker Images section to the Reference section.