ForgeOps Documentation

July 10, 2025

FORGEOPS
Version: 7.3

Table of Contents

Start here

ForgeRock 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 ForgeRock software in a containerized environment requires advanced proficiency in many technologies. See Assess Your Skill Level for details.
- If you don't have experience with complex Kubernetes deployments, then either engage a certified ForgeRock consulting partner or deploy the platform on traditional architecture.
- Don't deploy ForgeRock software in Kubernetes in production until you have successfully deployed and tested the software in a non-production Kubernetes environment.

For information about obtaining support for Ping Identity Platform software, see Support from ForgeRock.

Important

ForgeRock only offers ForgeRock software or services to legal entities that have entered into a binding license agreement with ForgeRock. When you install ForgeRock's Docker images, you agree either that: 1) you are an authorized user of a ForgeRock customer that has entered into a license agreement with ForgeRock governing your use of the ForgeRock software; or 2) your use of the ForgeRock software is subject to the ForgeRock Subscription License Agreement.

Introducing the CDK and CDM

The forgeops repository and DevOps documentation address a range of our customers' typical business needs. The repository contains artifacts for two primary resources to help you with cloud deployment:

- **Cloud Developer's Kit (CDK)**. The CDK is a minimal sample deployment for development purposes. Developers deploy the CDK, and then access AM's and IDM's admin UIs and REST APIs to configure the platform and build customized Docker images for the platform.
- **Cloud Deployment Model (CDM)**. The CDM is a reference implementation for ForgeRock cloud deployments. You can get a sample Ping Identity Platform deployment up and running in the cloud quickly using the CDM. After deploying the CDM, you can use it to explore how you might configure your Kubernetes cluster before you deploy the platform in production.

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

	CDK	CDM
Fully integrated AM, IDM, and DS installations	✔	✔
Randomly generated secrets	✔	✔
Resource requirement	Namespace in a GKE, EKS, AKS, or Minikube cluster	GKE, EKS, or AKS cluster
Can run on Minikube	✔	

	CDK	CDM
Multi-zone high availability		✔
Replicated directory services		✔
Ingress configuration		✔
Certificate management		✔
Prometheus monitoring, Grafana reporting, and alert management		✔

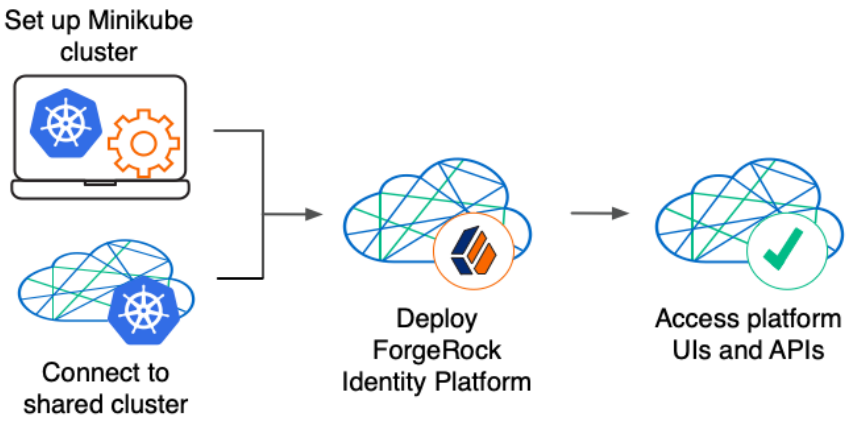
ForgeRock's DevOps documentation helps you deploy the CDK and CDM:

- CDK documentation. Tells you how to install the CDK, modify the AM and IDM configurations, and create customized Docker images for the Ping Identity Platform.
- CDM documentation. Tells you how to quickly create a Kubernetes cluster on Google Cloud, Amazon Web Services (AWS), or Microsoft Azure, install the Ping Identity Platform, and access components in the deployment.
- How-tos. Contains how-tos for customizing monitoring, setting alerts, backing up and restoring directory data, modifying CDM's default security configuration, and running lightweight benchmarks to test DS, AM, and IDM performance.
- ForgeOps 7.3 release notes. Keeps you up-to-date with the latest changes to the `forgeops` repository.

Try out the CDK and the CDM

Before you start planning a production deployment, deploy either the CDK or the CDM—or both. If you're new to Kubernetes, or new to the Ping Identity Platform, deploying these resources is a great way to learn. When you've finished deploying them, you'll have sandboxes suitable for exploring ForgeRock cloud deployment.

Deploy the CDK



The CDK is a minimal sample deployment of the Ping Identity Platform. If you have access to a cluster on Google Cloud, EKS, or AKS, you can deploy the CDK in a namespace on your cluster. You can also deploy the CDK locally in a standalone Minikube environment, and when you're done, you'll have a local Kubernetes cluster with the platform orchestrated on it.

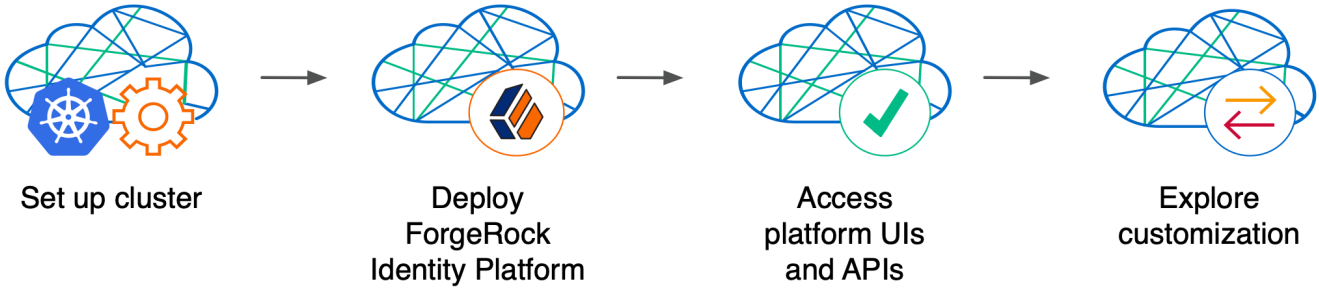
Prerequisite technologies and skills:

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

More information:

- CDK documentation

Deploy the CDM



Deploy the CDM 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 won't have a production-quality deployment. But you will have a robust, reference implementation of the platform.

After you get the CDM up and running, you can use it to test deployment customizations—options that you might want to use in production, but are not part of the CDM. Examples include, but are not limited to:

- Running lightweight benchmark tests
- Making backups of CDM 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

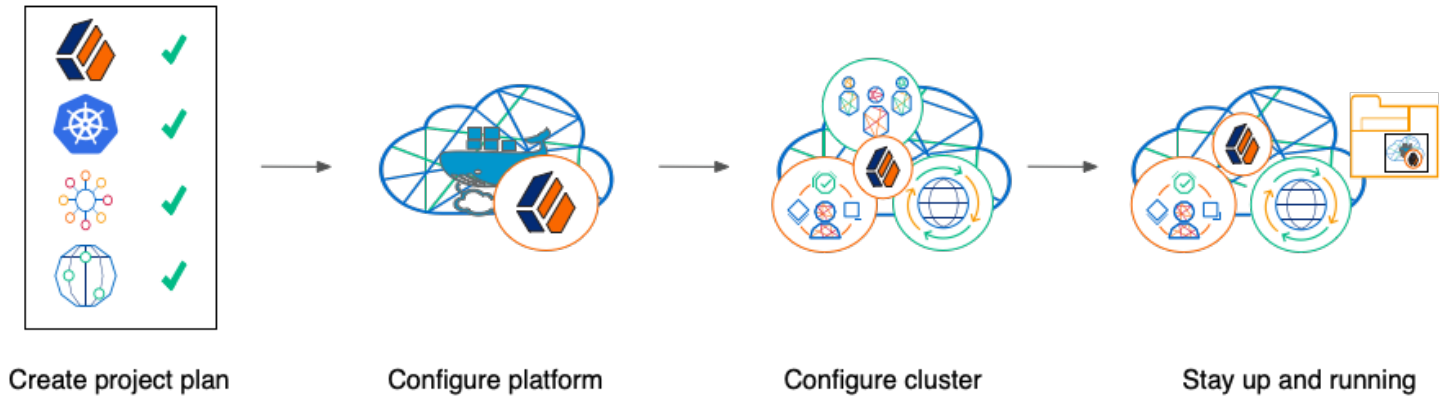
Prerequisite technologies and skills:

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

More information:

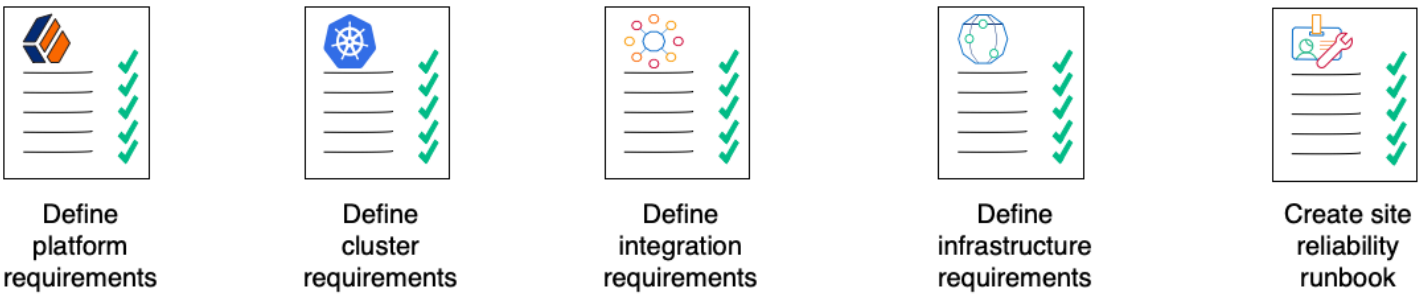
- CDM documentation

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 the CDK and CDM, you're ready to define requirements for your production deployment. *Remember, the CDM is not a production deployment*. Use the CDM to explore deployment customizations, and 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:

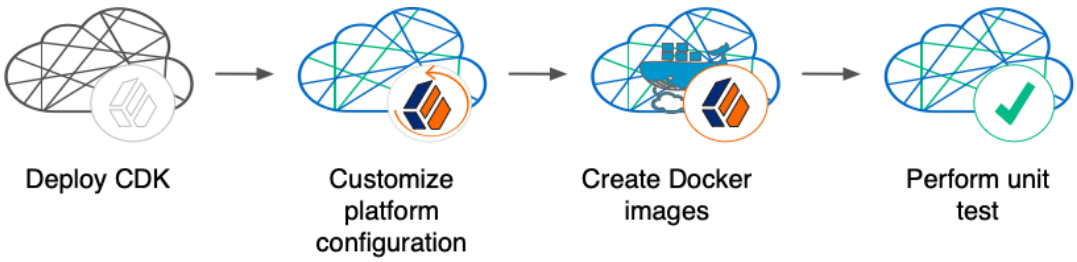
- Project planning and management
- Git

- 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 DevOps 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 the CDK 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.

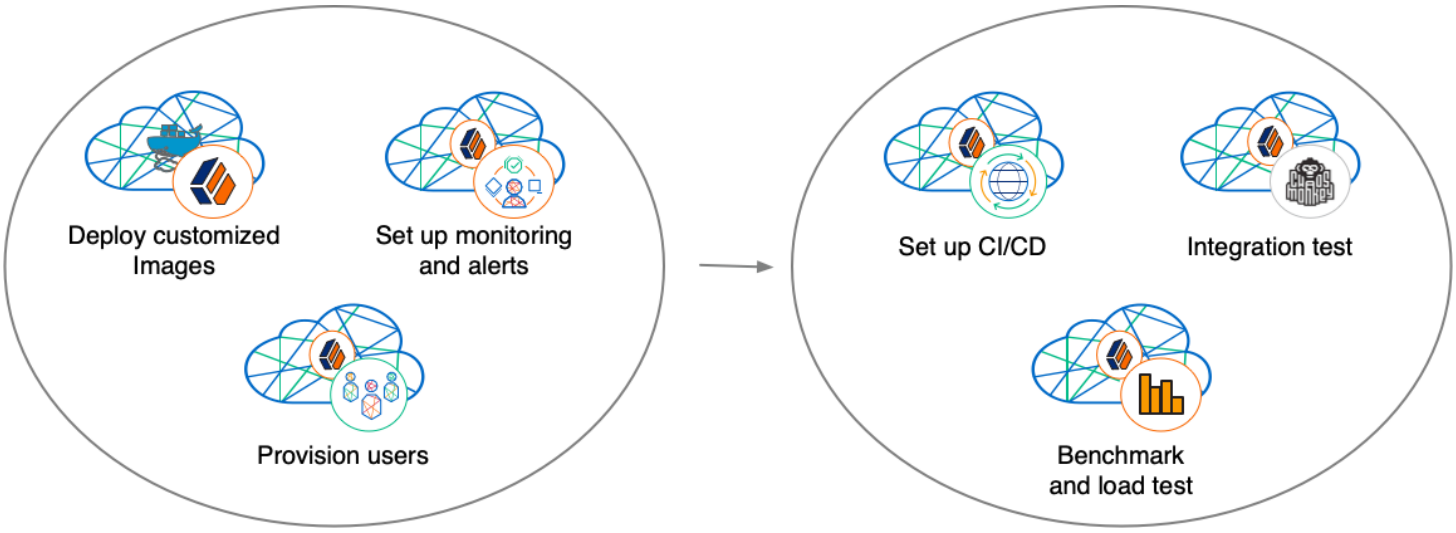
Prerequisite technologies and skills:

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

More information:

- CDK documentation

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 ForgeRock 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.

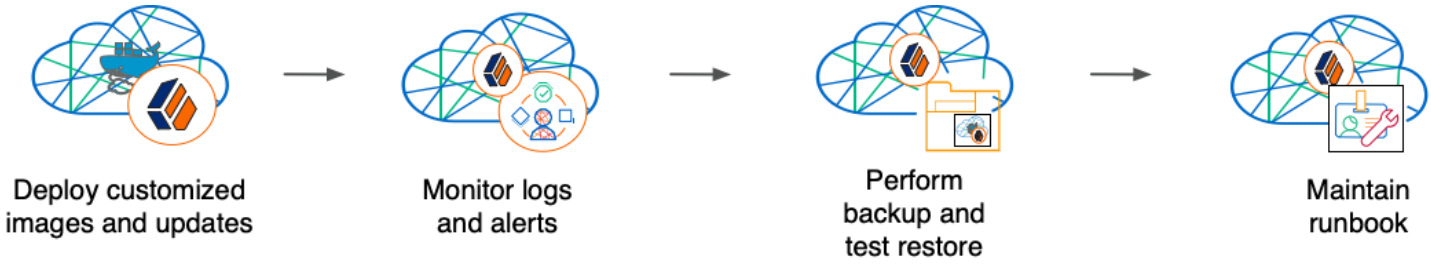
Prerequisite technologies and skills:

- 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:

- How-tos
- CDM documentation

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:

- How-tos

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

- Read the cluster creation shell scripts in the `forgeops` repository to see how the CDM cluster is 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 from ForgeRock

This appendix contains information about support options for the ForgeOps Cloud Developer's Kit, the ForgeOps Cloud Deployment Model, and the Ping Identity Platform.

ForgeOps (ForgeRock DevOps) support

ForgeRock has developed artifacts in the forgeops and forgeops-extras Git repositories for the purpose of deploying the Ping Identity Platform in the cloud. The companion ForgeOps documentation provides examples, including the ForgeOps Cloud Developer's Kit (CDK) and the ForgeOps Cloud Deployment Model (CDM), to help you get started.

These artifacts and documentation are provided on an "as is" basis. ForgeRock does not guarantee the individual success developers may have in implementing the code on their development platforms or in production configurations.

Licensing

ForgeRock only offers ForgeRock software or services to legal entities that have entered into a binding license agreement with ForgeRock. When you install ForgeRock's Docker images, you agree either that: 1) you are an authorized user of a ForgeRock customer that has entered into a license agreement with ForgeRock governing your use of the ForgeRock software; or 2) your use of the ForgeRock software is subject to the ForgeRock Subscription License Agreement.

Support

ForgeRock provides support for the following resources:

- Artifacts in the forgeops Git repository:
  - Files used to build Docker images for the Ping Identity Platform:
    - Dockerfiles
    - Scripts and configuration files incorporated into ForgeRock's Docker images
    - Canonical configuration profiles for the platform
  - Kustomize bases and overlays
- ForgeOps Documentation

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

ForgeRock 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

ForgeRock 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 repository reference.
- Artifacts other than Dockerfiles, Kustomize bases, and Kustomize overlays in the forgeops Git repository. Examples include scripts, example configurations, and so forth.
- Non-ForgeRock infrastructure. Examples include Docker, Kubernetes, Google Cloud Platform, Amazon Web Services, Microsoft Azure, and so forth.
- Non-ForgeRock software. Examples include Java, Apache Tomcat, NGINX, Apache HTTP Server, Certificate Manager, Prometheus, and so forth.
- Deployments that deviate from the published CDK and CDM architecture. Deployments that do not include the following architectural features are not supported:
  - PingAM (AM) and PingIDM (IDM) are integrated and deployed together in a Kubernetes cluster.
  - IDM login is integrated with AM.
  - AM uses PingDS (DS) as its data repository.
  - IDM uses DS as its repository.
- ForgeRock publishes reference Docker images for testing and development, but these images should *not* be used in production. For production deployments, it is recommended that customers build and run containers using a supported operating system and all required software dependencies. Additionally, to help ensure interoperability across container images and the ForgeOps tools, Docker images must be built using the Dockerfile templates as described here.

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. (ForgeRock supports running the identity platform on IBM RedHat OpenShift but does not provide the reference tools for IBM RedHat OpenShift.)

ForgeRock supports running the platform on Kubernetes. ForgeRock does not support Kubernetes itself. You must have a support contract in place with your Kubernetes vendor to resolve infrastructure issues. To avoid any misunderstandings, it must be clear that ForgeRock cannot troubleshoot underlying Kubernetes issues.

Modifications to ForgeRock's deployment assets may be required in order to adapt the platform to your 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 your chosen distribution.

Documentation access

ForgeRock publishes comprehensive documentation online:

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

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

- ForgeRock 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 ForgeRock using the Customer Support Portal to request information, or report a problem with Dockerfiles, Kustomize bases, or Kustomize overlays in the CDK or the CDM.

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 unsupported artifacts

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

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

Contact information

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, refer to https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service-level agreements (SLAs), visit https://www.forgerock.com/support.

About the `forgeops` repository

Use ForgeRock's `forgeops` repository to customize and deploy the Ping Identity Platform on a Kubernetes cluster.

The 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 ForgeRock's Docker images
  - Canonical configuration profiles for the platform
- Kustomize bases and overlays

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

- Deploying ForgeRock's CDK and CDM quickly and easily
- Exploring monitoring, alerts, and security customization
- Modeling a CI/CD solution for cloud deployment

See Repository reference for information about the files in the repository, recommendations about how to work with them, and the support status for the files.

Repository updates

New `forgeops` repository features become available in the `release/7.3-20240131` branch of the repository 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 ForgeRock's public repository on GitHub, or from a fork. See Git clone or Git fork? for more information.

Then, check out the `release/7.3-20240131` branch and create a working branch. For example:

```
$ git checkout release/7.3-20240131
$ git checkout -b my-working-branch
```

ForgeRock recommends that you regularly incorporate updates to the `release/7.3-20240131` into your working branch:

1. Get emails or subscribe to the ForgeOps RSS feed to be notified when there have been updates to ForgeOps 7.3.
2. Pull new commits in the `release/7.3-20240131` branch into your clone's `release/7.3-20240131` 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. 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 Repository reference to reduce merge conflicts, and to better understand how to resolve them when you rebase your working branch with updates that ForgeRock has made to the `release/7.3-20240131` branch.

Repository reference

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

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 ForgeRock.

cluster

Example script that automates Minikube cluster creation.

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

Support Status: Sample file. Not supported by ForgeRock.

config

Deprecated. Supported an older implementation of the CDK.

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.

Dockerfiles

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

Recommendation: Expect to encounter merge conflicts when you rebase changes from ForgeRock 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 ForgeRock.

Support Files Referenced by Dockerfiles

When customizing ForgeRock's default 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 ForgeRock into your branches. However, if you need to modify any files from ForgeRock, 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 ForgeRock that are incorporated into Docker images for the Ping Identity Platform: Support is available from ForgeRock.

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

Configuration Profiles

Add your own configuration profiles to the docker directory using the export command. Do not modify ForgeRock's internal-use only `idm-only` and `ig-only` configuration profiles.

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

Support Status: Configuration profiles. Support is available from ForgeRock.

etc

Files used to support several examples, including the CDM.

Recommendation: Don't modify the files in this directory (or its subdirectories). If you want to use CDM automated cluster creation as a model or starting point for your own automated cluster creation, then create your own subdirectories under etc, and copy the files you want to model into the subdirectories.

Support Status: Sample files. Not supported by ForgeRock.

jenkins-scripts

For ForgeRock internal use only. Do not 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. You'll probably change, at minimum, the kustomize/overlay/all/kustomization.yaml file.

Expect to encounter merge conflicts when you rebase changes into your branches. Be sure to track changes you've made to 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 ForgeRock.

legacy-docs

Documentation for deploying the Ping Identity Platform using DevOps techniques. 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 ForgeRock.

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

Files in the top-level directory

.gcloudignore, .gitchangelog.rc, .gitignore

For ForgeRock internal use only. Do not modify.

LICENSE

Software license for artifacts in the `forgeops` repository. Do not modify.

Makefile

For ForgeRock internal use only. Do not modify.

notifications.json

For ForgeRock internal use only. Do not modify.

README.md

The top-level `forgeops` repository README file. Do not modify.

Git clone or Git fork?

For the simplest use cases—a single user in an organization installing the CDK or CDM for a proof of concept, or exploration of the platform—cloning ForgeRock's public `forgeops` repository from GitHub provides a quick and adequate way to access the repository.

If, however, your use case is more complex, you might want to fork the `forgeops` repository, and use the fork as your common upstream repository. 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` repository changes from ForgeRock.
- Your organization wants to use pull requests when making repository updates.

If you've forked the `forgeops` repository:

- You'll need to synchronize your fork with ForgeRock's public repository on GitHub when ForgeRock releases a new release tag.
- Your users will need to clone your fork before they start working instead of cloning the public `forgeops` repository on GitHub. Because procedures in the CDK documentation and the CDM documentation tell users to clone the public repository, you'll need to make sure your users follow different procedures to clone the forks instead.
- The steps for initially obtaining and updating your repository clone will differ from the steps provided in the documentation. You'll need to let users know how to work with the fork as the upstream instead of following the steps in the documentation.

About the `forgeops-extras` repository

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

Repository reference

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

Directories

charts

For ForgeRock internal use only. Do not modify or use.

terraform

Example scripts and artifacts that automate CDM 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 ForgeRock.

Git clone or Git fork?

For the simplest use cases—a single user in an organization installing the CDK or CDM for a proof of concept, or exploration of the platform—cloning ForgeRock's public `forgeops-extras` repository from GitHub provides a quick and adequate way to access the repository.

If, however, your use case is more complex, you might want to fork the `forgeops-extras` repository, and use the fork as your common upstream repository. 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-extras` repository changes from ForgeRock.
- Your organization wants to use pull requests when making repository updates.

If you've forked the `forgeops-extras` repository:

- You'll need to synchronize your fork with ForgeRock's public repository on GitHub when ForgeRock releases a new release tag.
- Your users will need to clone your fork before they start working instead of cloning the public `forgeops-extras` repository on GitHub. Because procedures in the documentation tell users to clone the public repository, you'll need to make sure your users follow different procedures to clone the forks instead.
- The steps for initially obtaining and updating your repository clone will differ from the steps provided in the documentation. You'll need to let users know how to work with the fork as the upstream instead of following the steps in the documentation.

CDK documentation

The CDK is a minimal sample deployment of the Ping Identity Platform on Kubernetes that you can use for demonstration and development purposes. It includes fully integrated AM, IDM, and DS installations, and randomly generated secrets.

If you have access to a cluster on Google Cloud, EKS, or AKS, you can deploy the CDK in a namespace on your cluster. You can also deploy the CDK locally in a standalone Minikube environment, and when you're done, you'll have a local Kubernetes cluster with the platform orchestrated on it.

CDK checklist

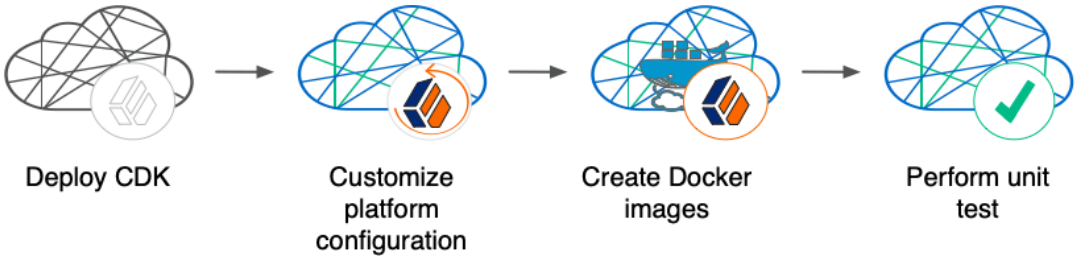
- ☐ Become familiar with the CDK
- ☐ Understand CDK architecture
- ☐ Set up your local environment
- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ (Optional) Develop custom Docker images

About the Cloud Developer's Kit

The CDK is a minimal sample deployment of the Ping Identity Platform on Kubernetes that you can use for demonstration and development purposes. It includes fully integrated AM, IDM, and DS installations, and randomly generated secrets.

CDK deployments orchestrate a working version of the Ping Identity Platform on Kubernetes. They also let you build and run customized Docker images for the platform.

This documentation describes how to deploy the CDK, and then use it to create and test customized Docker images containing your custom AM and IDM configurations.



Before deploying the platform in production, you must customize it using the CDK. To better understand how this activity fits into the overall deployment process, see Configure the Platform.

Containerization

The CDK uses Docker for containerization. Start with evaluation-only Docker images from ForgeRock that include canonical configurations for AM and IDM. Then, customize the configurations, and create your own images that include your customized configurations.

For more information about Docker images for the Ping Identity Platform, see About custom images.

Orchestration

The CDK uses Kubernetes for container orchestration. The CDK has been tested on the following Kubernetes implementations:

- Single-node deployment suitable for demonstrations, proofs of concept, and development:
  - Minikube
- Cloud-based Kubernetes orchestration frameworks suitable for development and production deployment of the platform:
  - Google Kubernetes Engine (GKE)
  - Amazon Elastic Kubernetes Service (Amazon EKS)
  - Azure Kubernetes Service (AKS)

Next step

Become familiar with the CDK

- ☐ Understand CDK architecture
- ☐ Set up your local environment
- ☐ Deploy the platform

- ☐ Access platform UIs and APIs
- ☐ (Optional) Develop custom Docker images

CDK architecture

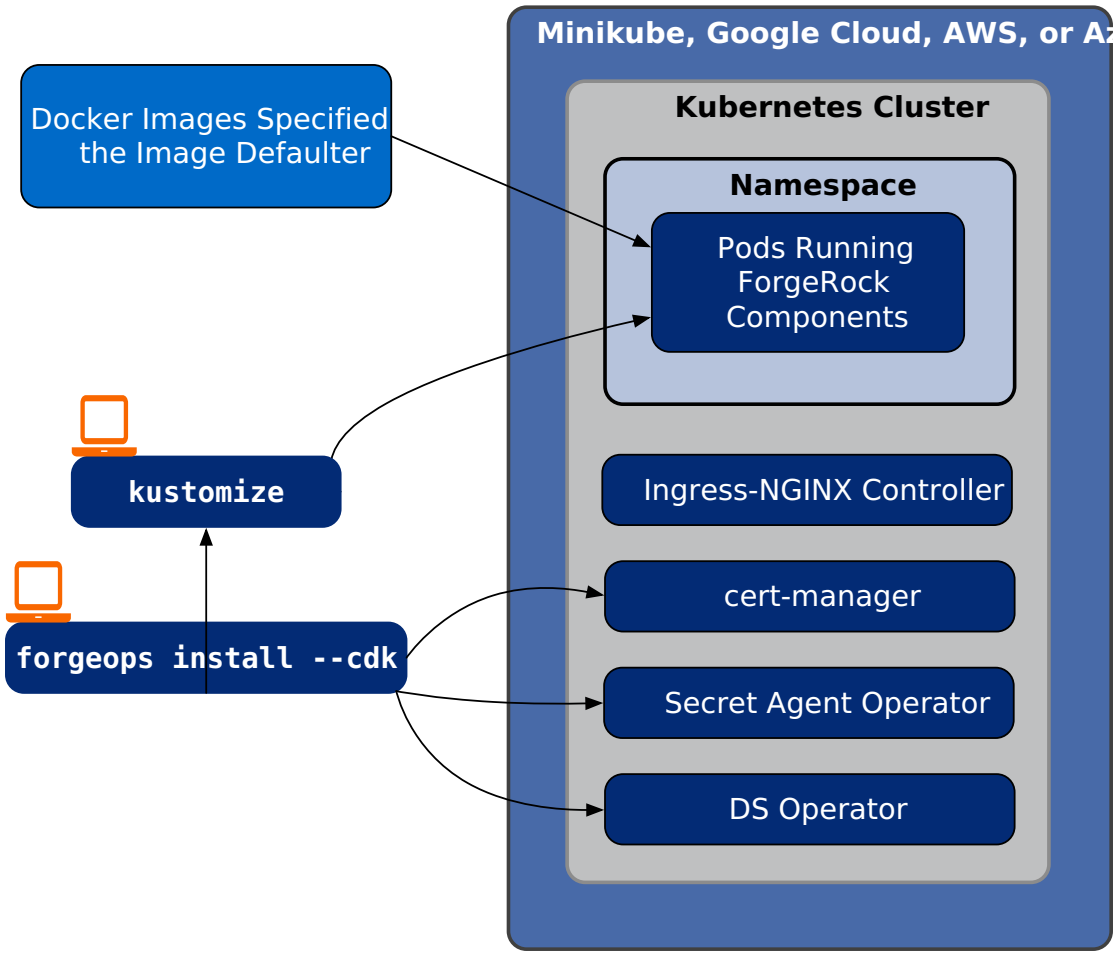
You deploy the CDK to get the Ping Identity Platform up and running on Kubernetes. CDK deployments are useful for demonstrations and proofs of concept. They're also intended for development—building custom Docker images for the platform.

> **Important**
>
> Do not use the CDK as the basis for a production deployment of the Ping Identity Platform.

Before you can deploy the CDK, you must have:

- Access to a Kubernetes cluster with the Ingress-NGINX controller deployed on it.
- Access to a namespace in the cluster.
- Third-party software installed in your local environment, as described in the Setup section that pertains to your cluster type.

This diagram shows the CDK components:



The forgeops install command deploys the CDK in a Kubernetes cluster:

- Installs Docker images for the platform specified in the image defaulter. Initially, the image defaulter specifies the ForgeOps-provided Docker images for ForgeOps 7.3 release, available from the public registry. These images use ForgeRock's canonical configurations for AM and IDM.

- Installs additional software as needed[1]:
  - **Secret Agent operator.** Generates Kubernetes secrets for Ping Identity Platform deployments. More information here.
  - **DS operator.** Deploys and manages DS instances running in a Kubernetes cluster. More information here.
  - **cert-manager software.** Provides certificate management services for the cluster. More information here.

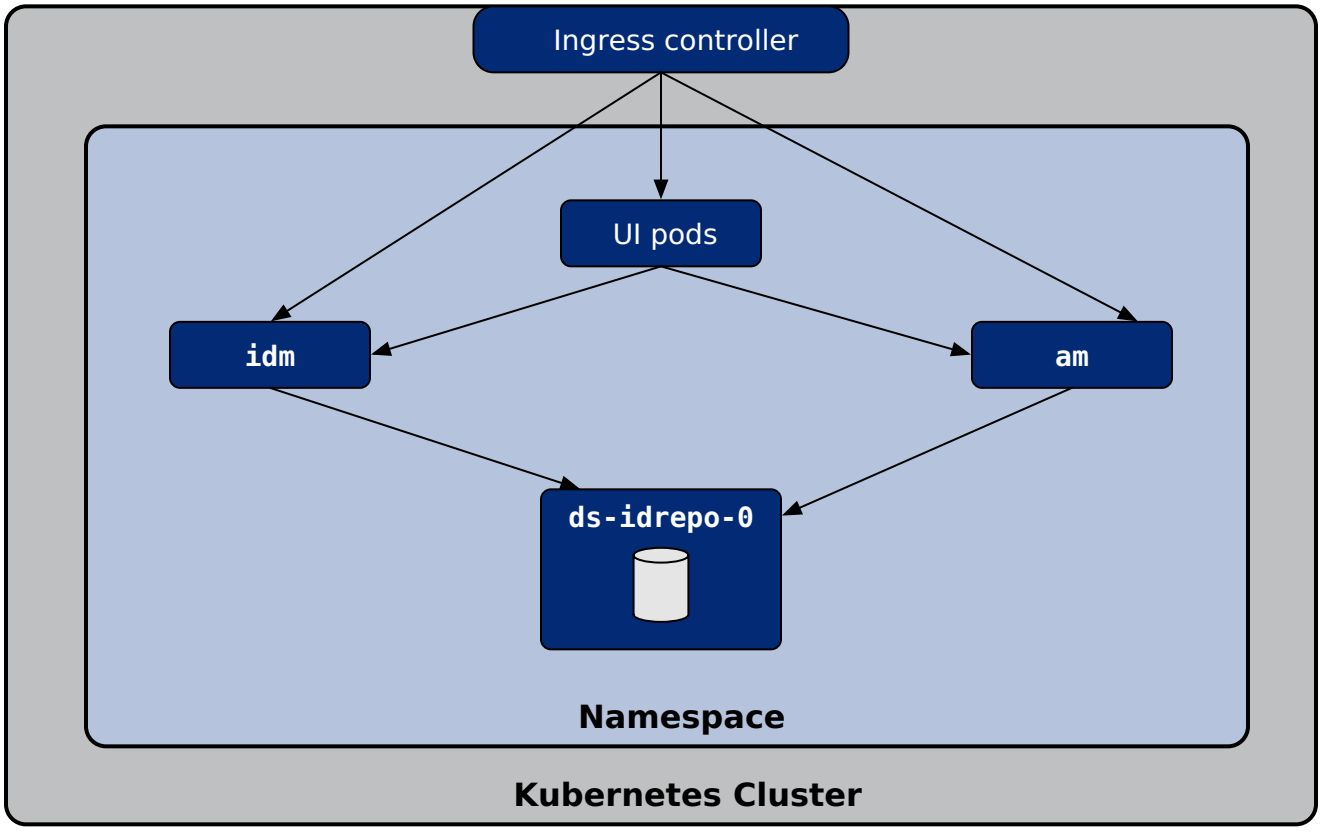
After you've deployed the CDK, you can access AM and IDM UIs and REST APIs to customize the Ping Identity Platform's configuration. You can then create Docker images that contain your customized configuration by using the forgeops build command. This command:

- Builds Kubernetes manifests based on the Kustomize bases and overlays in your local `forgeops` repository clone.
- Updates the image defaulter file to specify the customized images, so that the next time you deploy the CDK, your customized images will be used.

See `am` image and `idm` image for detailed information about building customized AM and IDM Docker images.

CDK pods

After deploying the CDK, you'll see the following pods running in your namespace:



`am`

Runs PingAM.

When AM starts in a CDK deployment, it obtains its configuration from the AM Docker image specified in the image defaulter.

After the `am` pod has started, a job is triggered that populates AM's application store with several agents and OAuth 2.0 client definitions that are used by the CDK.

`ds-idrepo-0`

The `ds-idrepo-0` pod provides directory services for:

- The identity repository shared by AM and IDM
- The IDM repository
- The AM application and policy store
- AM's Core Token Service

`idm`

Runs PingIDM.

When IDM starts in a CDK deployment, it obtains its configuration from the IDM Docker image specified in the image defaulter.

In containerized deployments, IDM must retrieve its configuration from the file system and not from the IDM repository. The default values for the `openidm.fileinstall.enabled` and `openidm.config.repo.enabled` properties in the CDK's system.properties file ensure that IDM retrieves its configuration from the file system. Do not override the default values for these properties.

UI pods

Several pods provide access to ForgeRock common user interfaces:

- `admin-ui`
- `end-user-ui`
- `login-ui`

Next step

Become familiar with the CDK

Understand CDK architecture

- ☐ Set up your local environment
- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ (Optional) Develop custom Docker images

1. If any of these software components are already installed in your cluster, they are not reinstalled.

Setup

Before you can deploy the CDK, you must first set up your local environment to communicate with your working Kubernetes cluster.

Important information for users running Microsoft Windows

Important information for users running macOS on ARM-based (M1) chipsets

On Minikube

Set up a Minikube cluster.

On GKE

Set up your local environment to access a shared cluster on GKE.

On Amazon EKS

Set up your local environment to access a shared cluster on Amazon EKS.

On AKS

Set up your local environment to access a shared cluster on AKS.

After you've completed these environment setup tasks, you're ready to deploy the Ping Identity Platform in your namespace on your Kubernetes cluster.

Important information for users running Microsoft Windows

ForgeRock supports deploying the CDK and CDM using macOS and Linux. If you have a Windows computer, you'll need to create a Linux VM. We tested using 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 deploy CDK or CDM on WSL2 until this issue is resolved.

Important information for users running macOS on ARM-based (M1) chipsets

Running the CDK is currently not supported on macOS systems running an ARM-based chipset, such as the Apple M1 or Apple M1 Max.

See the Release Notes for a workaround.

CDK deployment

After you've set up your environment, deploy the CDK:

1. Set the active namespace in your local Kubernetes context to the namespace that you created when you performed the setup task.
2. Run the forgeops install command:

   ```
   $ cd /path/to/forgeops/bin
   $ ./forgeops install --cdk --fqdn cdk.example.com
   ```

   By default, the forgeops install --cdk command uses the ForgeOps-provided Docker images for ForgeOps 7.3 release, available from the public registry However, if you have built custom images for the Ping Identity Platform, the forgeops install --cdk command uses your custom images.

   If you prefer not to deploy the CDK using a single forgeops install command, see Alternative deployment techniques for more information.

   > **Important**
   >
   > ForgeRock only offers ForgeRock software or services to legal entities that have entered into a binding license agreement with ForgeRock. When you install ForgeRock's Docker images, you agree either that: 1) you are an authorized user of a ForgeRock customer that has entered into a license agreement with ForgeRock governing your use of the ForgeRock software; or 2) your use of the ForgeRock software is subject to the ForgeRock Subscription License Agreement.

3. 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.

   Your namespace should have the pods shown in this diagram.

4. (Optional) Install a TLS certificate instead of using the default self-signed certificate in your CDK deployment. See TLS certificate for details.

Alternative deployment techniques

If you prefer not to deploy the CDK using a single forgeops install command, you can use one of these options:

- Deploy the CDK component by component instead of with a single command. Staging the deployment can be useful if you need to troubleshoot a deployment issue.
- The forgeops install command generates Kustomize manifests that let you recreate your CDK deployment. The manifests are written to the /path/to/forgeops/kustomize/deploy directory of your `forgeops` repository clone. Advanced users who prefer to work directly with Kustomize manifests that describe their CDK deployment can use the generated content in the kustomize/deploy directory as an alternative to using the forgeops command:
  - Generate an initial set of Kustomize manifests by running the forgeops install command. If you prefer to generate the manifests without installing the CDK, you can run the forgeops generate command.
  - Run kubectl apply -k commands to deploy and remove CDK components. Specify a manifest in the kustomize/deploy directory as an argument when you run kubectl apply -k commands.

- Use GitOps to manage CDK configuration changes to the kustomize/deploy directory instead of making changes to files in the kustomize/base and kustomize/overlay directories.

Next step

Become familiar with the CDK

Understand CDK architecture

Set up your local environment

Deploy the platform

- ☐ Access platform UIs and APIs
- ☐ (Optional) Develop custom Docker images

UI and API access

Now that you've deployed the Ping Identity Platform, you'll need to know how to access its administration tools. You'll use these tools to build customized Docker images for the platform.

This page shows you how to access the Ping Identity Platform's administrative UIs and REST APIs.

You access AM and IDM services through the Kubernetes ingress controller using their admin UIs and REST APIs.

You can't access DS through the ingress controller, but you can use Kubernetes methods to access the DS pods.

For more information about how AM and IDM are configured in the CDK, see Configuration in the `forgeops` repository's top-level README file.

AM services

To access the AM admin UI:

1. Set the active namespace in your local Kubernetes context to the namespace in which you have deployed the CDK.
2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./forgeops info | grep amadmin
   179rd8en9rffa82rcf1qap1z0gv1hcej (amadmin user)
   ```

3. Open a new window or tab in a web browser.
4. Go to https://cdk.example.com/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 admin 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: 179rd8en9rffa82rcf1qap1z0gv1hcej" \
 --header "Accept-API-Version: resource=2.0" \
 --data "{}" \
 "https://cdk.example.com/am/json/realms/root/authenticate"
{
    "tokenId":"AQIC5wM2...TU3OQ*",
    "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 have deployed the CDK.
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://cdk.example.com/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 admin 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. The CDK comes 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://cdk.example.com/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://cdk.example.com/am/oauth2/realms/root/authorize?redirect_uri=https://
    cdk.example.com/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: ...
    content-length: 0
    location: https://cdk.example.com/platform/appAuthHelperRedirect.html
     ?code=3cItL9G52DIiBdfXRngv2_dAaYM&iss=http://cdk.example.com: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://cdk.example.com/platform/appAuthHelperRedirect.html" \
 "https://cdk.example.com/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://cdk.example.com/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 the CDK 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 like ldapsearch. You can also use an LDAP editor such as Apache Directory Studio to access the directory.

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

Next step

Become familiar with the CDK

Understand CDK architecture

Set up your local environment

Deploy the platform

Access platform UIs and APIs

☐ (Optional) Develop custom Docker images

CDK shutdown and removal

When you're done working with the CDK, shut it down and remove it from your namespace:

1. Set the active namespace in your local Kubernetes context to the namespace that you created when you performed the setup task.
2. If you've made changes to the AM and IDM configurations in the Git repository on the CDK that you want to save, export the changes to your local `forgeops` repository clone. If you don't export the configurations before you run the forgeops delete command, all the changes that you've made to the configurations will be lost.

   For more information on syncing changes to your local `forgeops` repository clone, see:

   - `am` image
   - `idm` image
3. Run the forgeops delete command, which deletes all CDK artifacts, including PVCs and the AM and IDM configurations in Git:

   ```
   $ cd /path/to/forgeops/bin
   $ ./forgeops delete
   ```

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

Development 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

Concepts:

- ☐ Understand custom images
- ☐ Understand types of configuration
- ☐ Understand property value substitution

Custom Docker images:

- ☐ 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.

Install additional third-party software

You should have already installed third-party software when you set up your local environment before installing the CDK. Depending on how you have installed the CDK, you might need to install additional software before you can build custom Docker images for the platform:

Software	Version	Homebrew package
Docker Desktop	4.26.1	`docker` (cask)

Software	Version	Homebrew package
Docker Desktop	4.26.1	`docker` (cask)

Software	Version	Homebrew package
Docker Engine	24.0.7	n/a

Set up your environment to push to your Docker registry

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

ForgeRock 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, see 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 Docker on your local computer. For more information, see the Docker documentation.
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 Docker on your local computer. For more information, see the Docker documentation.
2. Log in to Amazon ECR. Use the Docker registry location you obtained from your cluster administrator:

```
$ 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 Docker on your local computer. For more information, see the Docker documentation.
2. Install the ACR Docker Credential Helper.

Identify the Docker repository to push to

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

Note that 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:

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

Set the --push-to argument to the GCR repository that you obtained from your cluster administrator.

After it builds the Docker image, the forgeops build command pushes the Docker image to this repository.

Set the --push-to argument to the Amazon ECR repository that you obtained from your cluster administrator.

After it builds the Docker image, the forgeops build command pushes the Docker image to this repository.

Set the --push-to argument to the ACR repository that you obtained 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
- ☐ Understand types of 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.

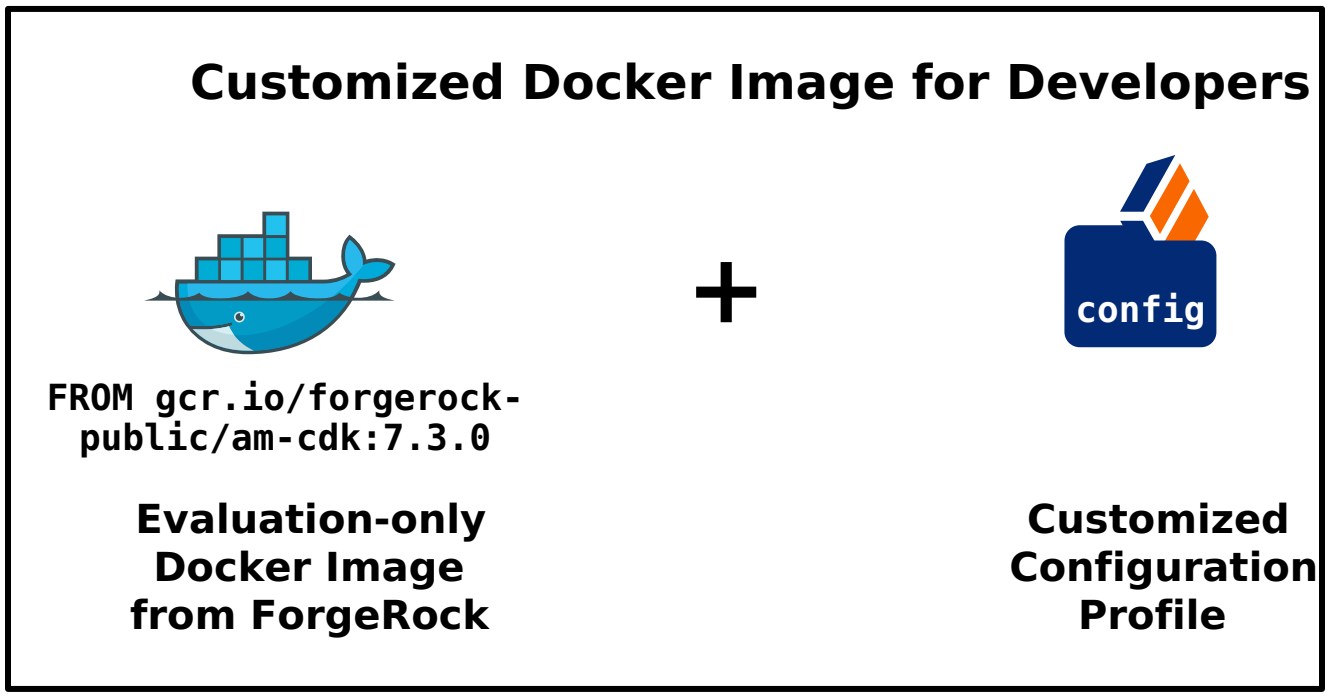
About custom images

In development

To develop customized Docker images, start with ForgeRock's evaluation-only images. Then, build up 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 that are based on ForgeRock's evaluation-only images.

To develop a customized AM Docker image, see `am` image.

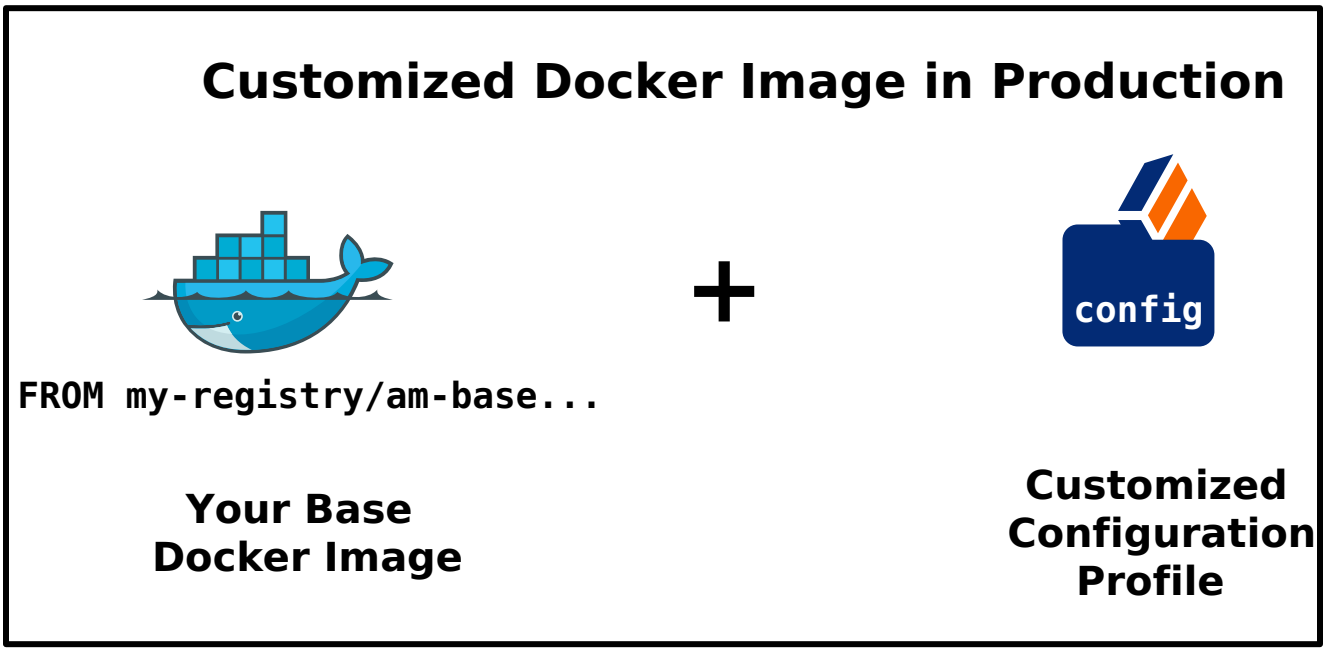
To develop a customized IDM Docker image, see `idm` image.



In production

Before you deploy the platform in production, you'll need to stop using Docker images that are based on ForgeRock's evaluation-only images. Instead, you'll need to build your own base images and integrate your configuration profiles into them.

To create Docker images for production deployment of the platform, see Base Docker images.



Next step

Perform additional setup

Understand custom images

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

Types of configuration

The Ping Identity Platform uses 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 the CDK 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.

See `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 amster utility 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 amster utility. You might manage these files in a Git repository. For example:

```
$ cd /path/to/forgeops/bin
$ mkdir /tmp/amster
$ ./amster export /tmp/amster
Cleaning up amster components
Packing and uploading configs
configmap/amster-files created
configmap/amster-export-type created
configmap/amster-retain 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
configmap "amster-retain" deleted
```

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

Note that the amster utility 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, see Export Utilities and Configuration Expressions.

- Write REST API applications to import and export AM dynamic configuration. For more information, see 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, see Migrate Data in the IDM documentation.
- Write REST API applications to import and export IDM dynamic configuration. For more information, see 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 ForgeRock 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

Understand types of configuration

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

About property value substitution

Many property values in ForgeRock's canonical CDK 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

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, see 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 CDK instance.

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 CDK instances 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 CDK instances to your `forgeops` repository clone. All AM static configuration properties in the CDK, including passwords, have string values. However, after the config utility copies the AM static configuration from the CDK, 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 bin/amster export command exports AM dynamic configuration from running CDK instances to your `forgeops` repository clone. When dynamic configuration is exported, it contains properties with string values. The amster utility transforms the values of 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 does not 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

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

amimage

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:
    - The CDK is deployed.
    - The namespace in which the CDK 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.

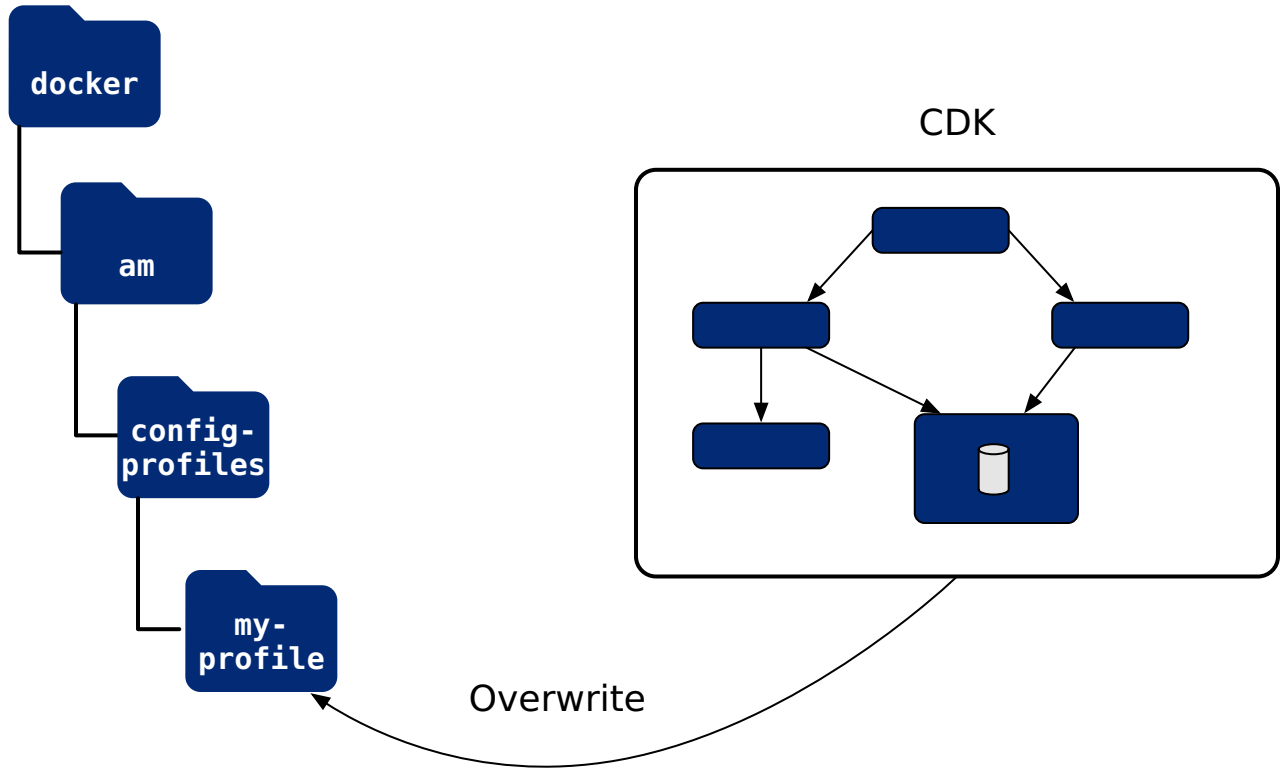
   For information about how to access the AM admin UI or REST APIs, refer to AM Services.

   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 AM configuration in the running Ping Identity Platform 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 gcr.io/forgerock-io/am-config-upgrader:7.3.0-latest-postcommit' locally
7.3.0-latest-postcommit: Pulling from forgerock-io/am-config-upgrader/pit1
...
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 --volume /path/to/forgeops/etc/am-upgrader-rules:/rules gcr.io/forgerock-io/am-config-
upgrader/pit1:7.3.0-latest-postcommit

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 does not exist yet, the config export command creates it.

The config export am my-profile command copies AM static configuration from the running CDK instance 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.

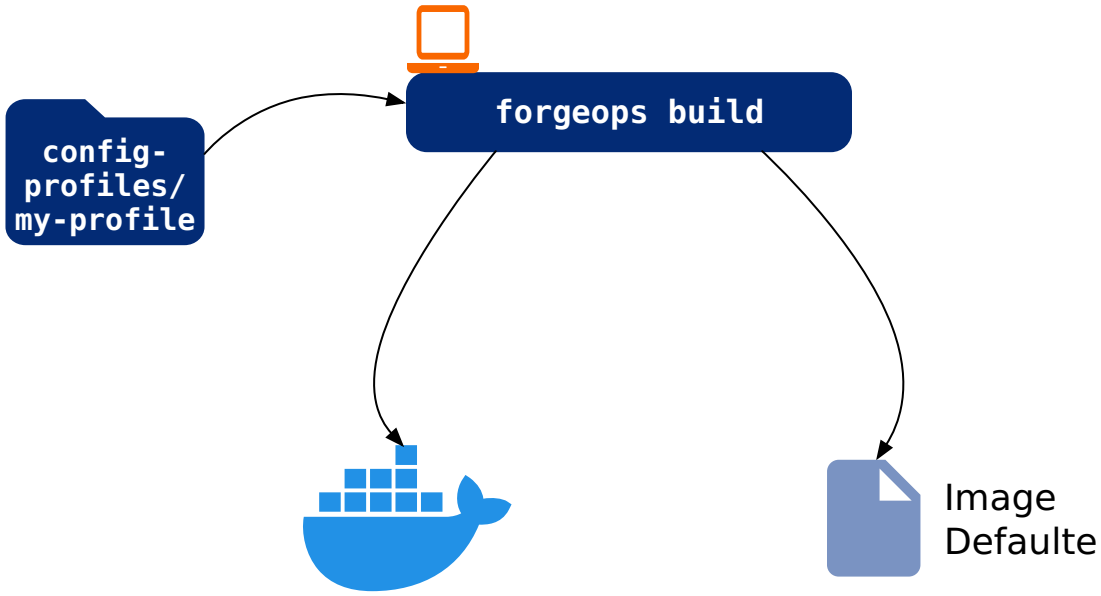
6. Identify the repository to which you'll push the Docker image. You'll use this location to specify the --push-to argument value in the build am image step.

7. Decide on the image tag name, so you can 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:

```
$ ./forgeops build am --config-profile my-profile --push-to my-repo --tag my-am-tag
Flag --short has been deprecated, and will be removed in the future.
Sending build context to Docker daemon  92.67kB
Step 1/6 : FROM gcr.io/forgerock-io/am-cdk:7.3.0
 ---> 7fde2db0c1a7
Step 2/6 : ARG CONFIG_PROFILE=cdk
 ---> Using cache
 ---> ec09a2602d06
...
Step 6/6 : WORKDIR /home/forgerock
 ---> Running in f2093dd17bff
Removing intermediate container f2093dd17bff
 ---> e2e3e16ba613
Successfully built e2e3e16ba613
Successfully tagged am:latest
Updated the image_defaulter with your new image for am: "am".

Generating tags...
 - am → am:da3855f51-dirty
Checking cache...
 - am: Not found. Building
Starting build...
Found [minikube] context, using local docker daemon.
Building [am]...
Sending build context to Docker daemon  1.989MB
Step 1/16 : FROM gcr.io/forgerock-io/am-cdk:7.3.0
 ---> 4e0b979daa5c
...

Updated the image_defaulter with your new image for am: "am:16e5e4048..."
```

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 install command gets AM static configuration from your new custom Docker image.



9. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. Review the state of the kustomize/deploy/image-defaulter/kustomization.yaml file.
    3. (Optional) Run the git commit command to commit changes to the image defaulter file.
10. Remove AM from your CDK installation:

```
$ ./forgeops delete am
"cdk" 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
```

11. Redeploy AM:

```
$ ./forgeops install am --cdk
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.
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['am'] platform: "cdk" in namespace: "my-namespace"

service/am created
deployment.apps/am created

Enjoy your deployment!
```

12. 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

Understand types of configuration

Understand property value substitution

Customize the AM image

☐ Customize the IDM image

idmimage

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:
    - The CDK is deployed.
    - The namespace in which the CDK 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.

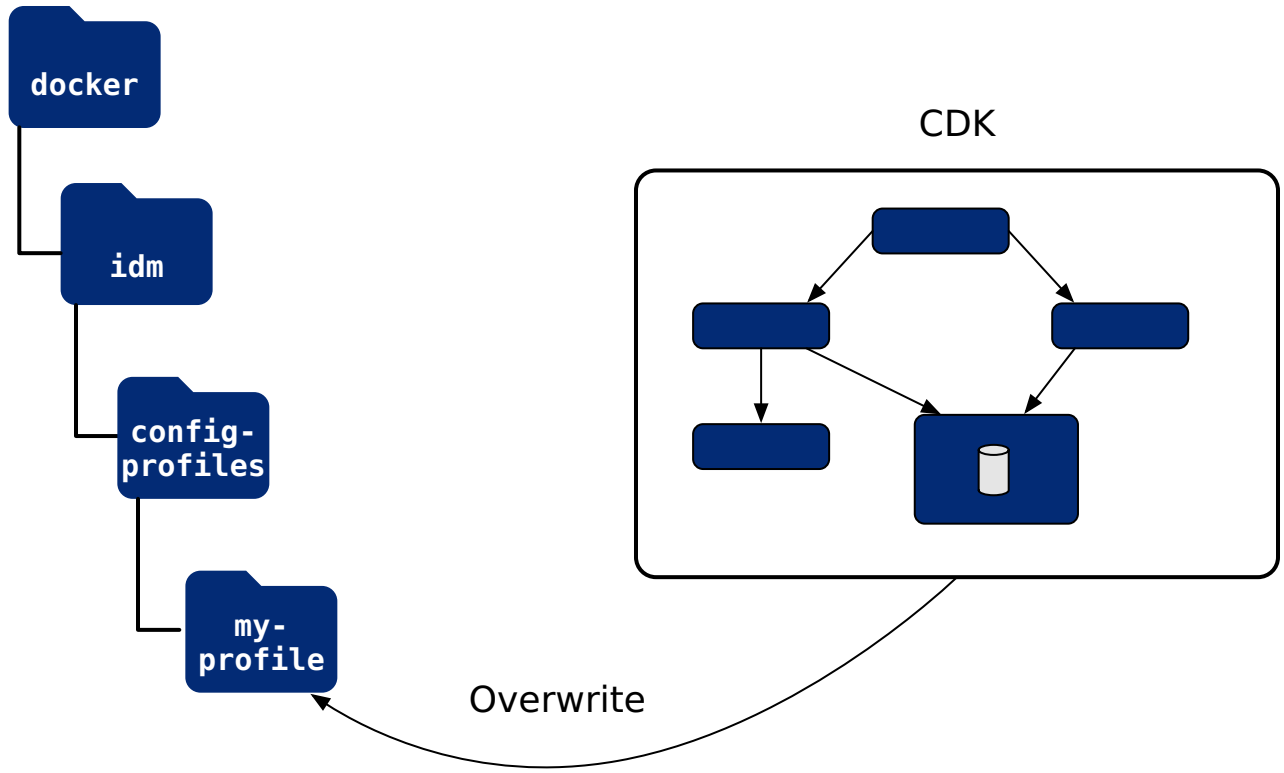
    For information about how to access the IDM admin UI or REST APIs, refer to IDM Services.

    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 Ping Identity Platform to a configuration profile:

```
$ cd /path/to/forgeops/bin
$ ./config export idm my-profile --sort
[INFO] Running export for idm in idm-869679958c-g2dpf
[INFO] Updating existing profile: /path/to/forgeops/docker/idm/config-profiles/my-profile/conf
tar: Removing leading `/' from member names
[INFO] Completed export
[INFO] Sorting configuration.
[INFO] Sorting completed.
```

If the configuration profile does not exist yet, the config export command creates it.

The config export idm my-profile command copies IDM static configuration from the running CDK instance 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"
   }
 }
```

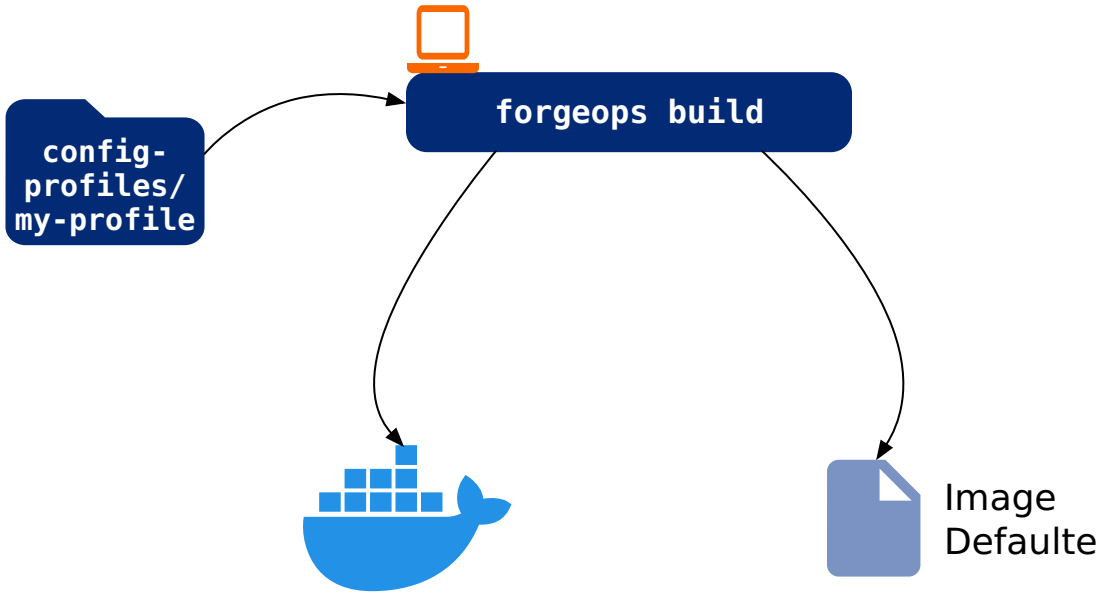
Note that if this is the first time that you have exported IDM configuration changes to this configuration profile, the git diff command will not show any changes.

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 in the build idm image step.
8. Build a new `idm` image that includes your changes to IDM static configuration:

```
$ ./forgeops build idm --config-profile my-profile --push-to my-repo --tag my-idm-tag

Flag --short has been deprecated, and will be removed in the future.
Sending build context to Docker daemon    618kB
Step 1/8 : FROM gcr.io/forgerock-io/idm-cdk:7.3.0
7.3.0: Pulling from forgerock-io/idm-cdk
...
Step 8/8 : COPY --chown=forgerock:root . /opt/openidm
 ---> 42f15d2e0544
Successfully built 42f15d2e0544
Successfully tagged idm:latest
Updated the image_defaulter with your new image for idm: "idm".
```

The forgeops build command calls Docker to build a new `idm` Docker image and 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 install command gets IDM static configuration from your new custom Docker image.



9. Perform version control activities on your `forgeops` repository clone:
    1. Run the git status command.
    2. Review the state of the kustomize/deploy/image-defaulter/kustomization.yaml file.
    3. (Optional) Run the git commit command to commit changes to the image defaulter file.
10. Remove IDM from your CDK installation:

```
$ ./forgeops delete idm
"cdk" platform detected in namespace: "my-namespace".
Uninstalling component(s): ['idm'] from namespace: "my-namespace".
OK to delete components? [Y/N] Y
configmap "idm" deleted
configmap "idm-logging-properties" deleted
service "idm" deleted
deployment.apps "idm" deleted
```

11. Redeploy IDM:

```
$ ./forgeops install idm --cdk
Flag --short has been deprecated, and will be removed in the future.
Flag --short has been deprecated, and will be removed in the future.
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.

Checking secret-agent operator is running...
customresourcedefinition.apiextensions.k8s.io/secretagentconfigurations.secret-
agent.secrets.forgerock.io condition met
deployment.apps/secret-agent-controller-manager condition met
NAME                                              READY   STATUS    RESTARTS   AGE
secret-agent-controller-manager-5577479db5-6jfjw   2/2     Running   0          21h
secret-agent operator is running
Checking ds-operator and related CRDs: ds-operator CRD found in cluster.

Installing component(s): ['idm'] platform: "cdk" in namespace: "my-namespace" from deployment
manifests in "None".

configmap/idm created
configmap/idm-logging-properties created
service/idm created
deployment.apps/idm created

Enjoy your deployment!
```

12. To 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.

CDM documentation

Deploy the CDM on GKE, Amazon EKS, or AKS 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 won't have a production-quality deployment. But you will have a robust, reference implementation of the Ping Identity Platform.

CDM checklist

- ☐ Become familiar with the CDM
- ☐ Understand CDM architecture
- ☐ Set up your local environment and create a cluster
- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

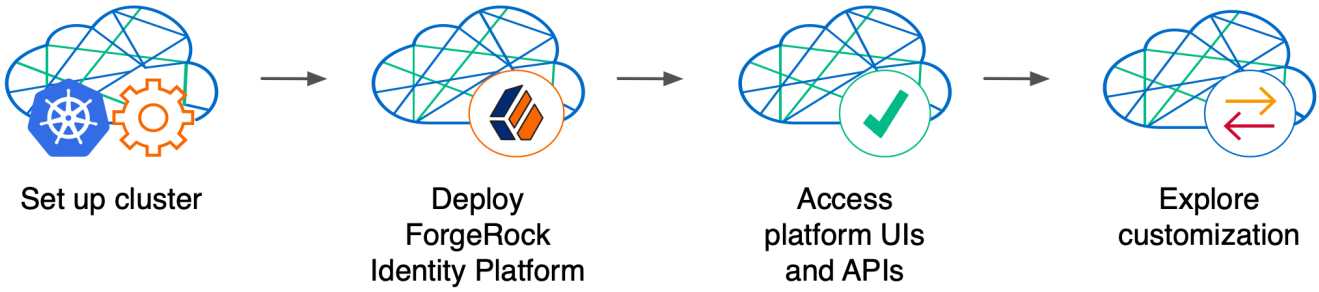
About the Cloud Deployment Model

The ForgeOps Team has developed Docker images, Kustomize bases and overlays, utility programs, and other artifacts expressly to deploy the Cloud Deployment Model (CDM). The `forgeops` repository on GitHub contains the CDM artifacts you can use to deploy the Ping Identity Platform in a cloud environment.

The CDM is a reference implementation for ForgeRock cloud deployments. You can get a sample Ping Identity Platform deployment up and running in the cloud quickly using the CDM. After deploying the CDM, you can use it to explore how you might configure your Kubernetes cluster before you deploy the platform in production.

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

This documentation describes how to use the CDM to stand up a Kubernetes cluster in the cloud that runs the Ping Identity Platform, and then access the platform's GUIs and REST APIs. When you're done, you can use the CDM to explore deployment customizations.



Standing up a Kubernetes cluster and deploying the platform using the CDM is an activity you might want to perform as a learning and exploration exercise before you put together a project plan for deploying the platform in production. To better understand how this activity fits in to the overall deployment process, see Deploy the CDM.

Using the CDM artifacts and this documentation, you can quickly get the Ping Identity Platform running in a Kubernetes cloud environment. You deploy the CDM to begin to familiarize yourself with some of the steps you'll need to perform when deploying the platform in the cloud for production use. These steps include creating a cluster suitable for deploying the Ping Identity Platform, installing the platform, and accessing its UIs and APIs.

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

Simplifies baseline deployment. We then developed artifacts—Dockerfiles, Kustomize bases and overlays, 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 CDM artifacts and follow the instructions in this documentation without deviation, you can successfully deploy the Ping Identity Platform in the cloud. The CDM takes the guesswork out of setting up a cloud environment. It bypasses 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 deployed the CDM, you'll be ready to start working with experts on deploying in production. We strongly recommend that you engage a ForgeRock technical consultant or partner to assist you with deploying the platform in production.

Next step

- ☐ Become familiar with the CDM
- ☐ Understand CDM architecture
- ☐ Set up your local environment and create a cluster
- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

CDM architecture

Once you deploy the CDM, the Ping Identity Platform is fully operational within a Kubernetes cluster. `forgeops` artifacts provide well-tuned JVM settings, memory, CPU limits, and other CDM configurations.

Here are some of the characteristics of the CDM:

Multi-zone Kubernetes cluster

Ping Identity Platform is deployed in a Kubernetes cluster.

For high availability, CDM clusters are distributed across three zones.

Go here for a diagram that shows the organization of pods in zones and node pools in a CDM cluster.

Cluster sizes

When deploying the CDM, you specify one of three cluster 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

Third-party deployment and monitoring tools

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

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. The AM instances are configured to access the DS data stores.
- Multiple IDM instances are deployed for higher availability. The IDM instances are configured to access the DS data stores.

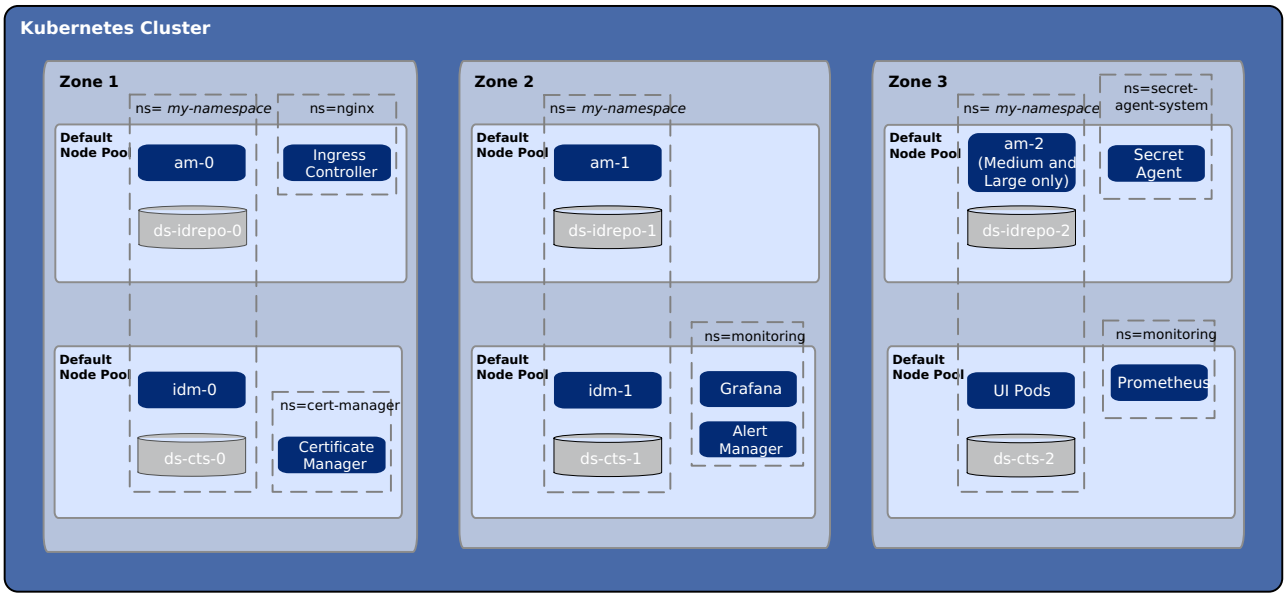
Highly available, distributed deployment

Deployment across the 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 CDM 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 CDM deployments. Ingress controller pods run in the `nginx` namespace. Implementation varies by cloud provider.

Optionally, you can deploy HAProxy Ingress as the CDM's ingress controller instead of Ingress-NGINX Controller.

Secret generation and management

ForgeRock's 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.

Directory services management

ForgeRock's open source Directory Services operator provides management of PingDS instances deployed with the CDM, including:

- Creation of stateful sets, services, and persistent volume claims
- Addition of new directory pods to the replication topology
- Modification of service account passwords in the directory, using a Kubernetes secret

- Taking volume snapshots of the directory disk, and restoring directories from snapshots
- Backing up and restoring directory data using LDIF

Secured communication

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

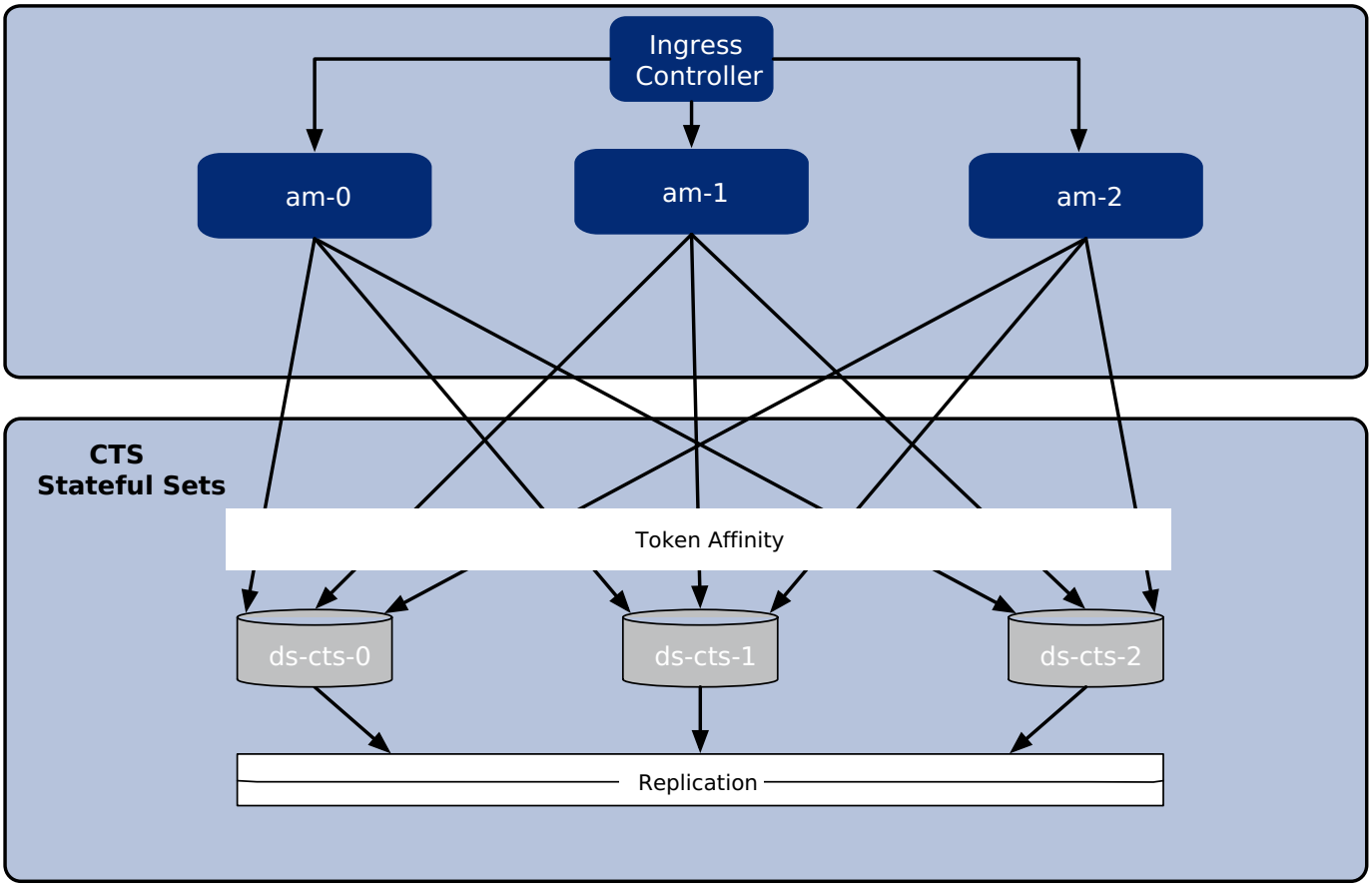
Inbound communication to DS instances occurs over secure LDAP (LDAPS).

For more information, see Secure HTTP.

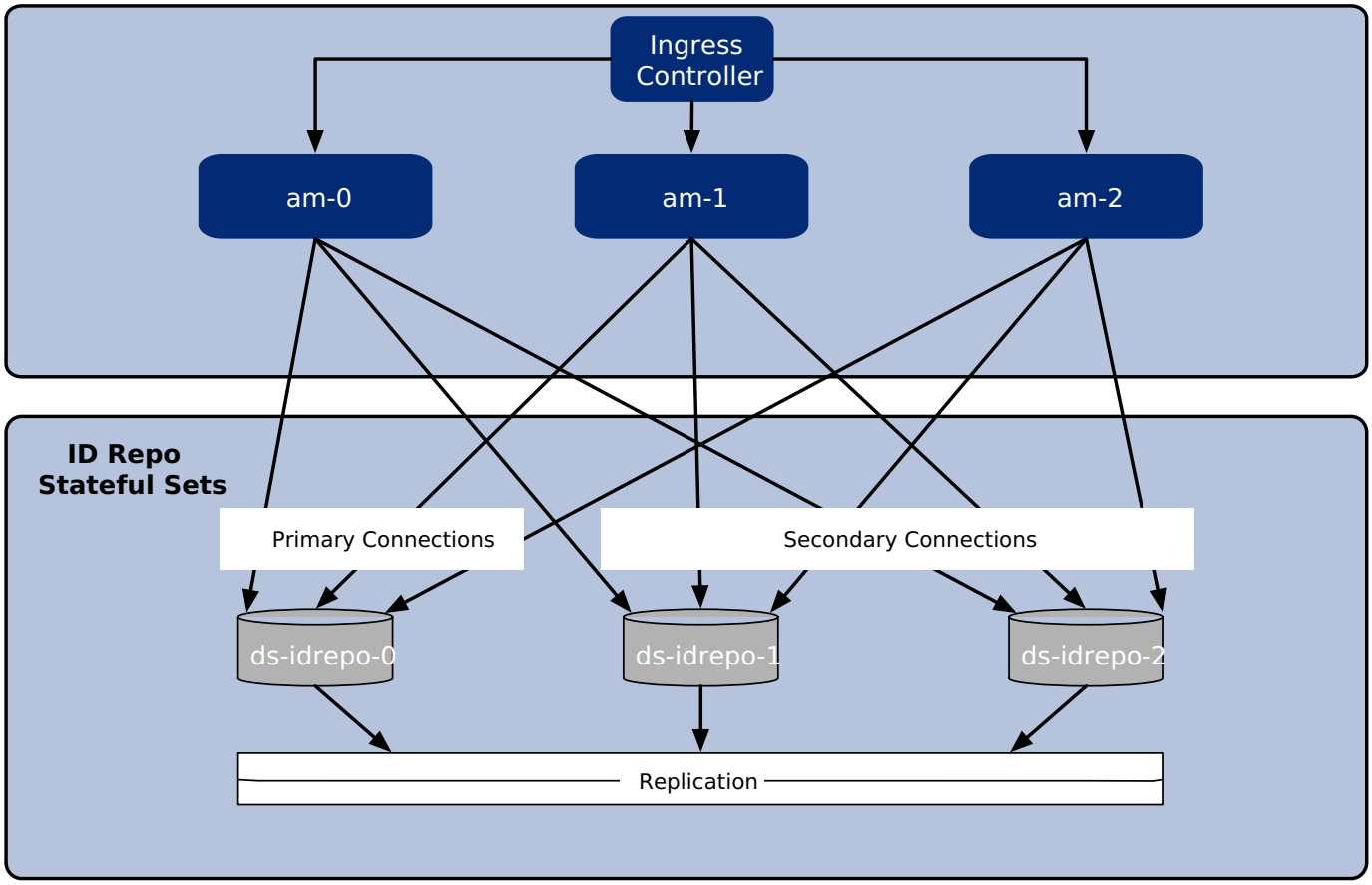
Stateful sets

The CDM uses Kubernetes stateful sets to manage the DS pods. Stateful sets protect against data loss if Kubernetes client containers fail.

The CTS data stores are configured for affinity load balancing for optimal performance.



The AM policies, application data, and identities reside in the `idrepo` directory service. The deployment uses a single `idrepo` master that can fail over to one of two secondary directory services.



Authentication

IDM is configured to use AM for authentication.

DS replication

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

Backup and restore

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

- Use the volume snapshot capability in GKE, EKS, or AKS. The cluster that the CDM is deployed in must be configured with a volume snapshot class before you can take volume snapshots, and that persistent volume claims must use a CSI driver that supports volume snapshots.
- 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, see Backup and restore overview.

Initial data loading

When it starts up, the CDM runs the `amster` job, which loads application data, such as OAuth 2.0 client definitions, to the `idrepo` DS instance.

Next step

Become familiar with the CDM

Understand CDM architecture

- ☐ Set up your local environment and create a cluster
- ☐ Deploy the platform
- ☐ Access platform UIs and APIs
- ☐ Plan for production deployment

Setup

Before deploying the CDM, you must set up your local computer, configure your cloud platform environment, and create a Kubernetes cluster.

On Google Cloud

Set up your Google Cloud environment.

On AWS

Set up your AWS environment.

On Azure

Set up your Azure environment.

CDM deployment

Now that you've set up your deployment environment following the instructions in the Setup section for your cloud platform, you're ready to deploy the CDM:

1. 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 at /path/to/forgeops/kustomize/deploy/image-defaulter/kustomization.yaml.

     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 ForgeRock's evaluation-only Docker images for the platform, do not modify the image defaulter file.
2. Set up your Kubernetes context:
   1. Set the `KUBECONFIG` environment variable so that your Kubernetes context references the cluster in which you'll deploy the CDM.
   2. Create a Kubernetes namespace in the cluster for the CDM.
   3. Set the active namespace in your Kubernetes context to the CDM's namespace.
3. Change to the bin directory in your `forgeops` repository clone:

   ```
   $ cd /path/to/forgeops/bin
   ```
4. Run the forgeops install command. For example, to install a small-sized CDM deployment:

   ```
   $ ./forgeops install --small --fqdn cdm.example.com --namespace my-namespace
   ```

   The forgeops install command examines the image defaulter file to determine which Docker images to use.

   If you prefer not to deploy the CDM using a single forgeops install command, refer to Alternative deployment techniques for more information.

   > **Important**
   >
   > ForgeRock only offers ForgeRock software or services to legal entities that have entered into a binding license agreement with ForgeRock. When you install ForgeRock's Docker images, you agree either that: 1) you are an authorized user of a ForgeRock customer that has entered into a license agreement with ForgeRock governing your use of the ForgeRock software; or 2) your use of the ForgeRock software is subject to the ForgeRock Subscription License Agreement.
5. Check the status of the pods in the namespace in which you deployed the CDM 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].
- Three AM and two IDM pods are present.

3. If necessary, continue to query your deployment's status until all the pods are ready.

6. Back up and save the Kubernetes secrets that contain the master and TLS keys created by the DS operator:

   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.

7. (Optional) Deploy Prometheus, Grafana, and Alertmanager monitoring and alerts[1]:

   1. Deploy Prometheus, Grafana, and Alertmanager pods in the CDM:

```
$ /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
```

8. (Optional) Install a TLS certificate instead of using the default self-signed certificate in your CDM deployment. See TLS certificate for details.

Alternative deployment techniques

If you prefer not to deploy the CDM using a single forgeops install command, you can use one of these options:

1. Deploy the CDM in stages component by component instead of with a single command.

   Staging the deployment can be useful if you need to troubleshoot a deployment issue. Make sure you specify a CDM size (such as --small) instead of --cdk when you run the forgeops install command to install components.

2. Back up and save the master and TLS key pairs created by the DS operator. Refer to this step for details.

3. Generate Kustomize manifests, and then deploy the CDM with the kubectl apply -k command.

   The forgeops install command generates Kustomize manifests that let you recreate your CDM deployment. The manifests are written to the /path/to/forgeops/kustomize/deploy directory of your `forgeops` repository clone. Advanced users who prefer to work directly with Kustomize manifests that describe their CDM deployment can use the generated content in the kustomize/deploy directory as an alternative to using the forgeops command:

   1. Generate an initial set of Kustomize manifests by running the forgeops install command. If you prefer to generate the manifests without installing the CDM, you can run the forgeops generate command.

   2. Run kubectl apply -k commands to deploy and remove CDM components. Specify a manifest in the kustomize/deploy directory as an argument when you run kubectl apply -k commands.

   3. Use GitOps to manage CDK configuration changes to the kustomize/deploy directory instead of making changes to files in the kustomize/base and kustomize/overlay directories.

Next step

Become familiar with the CDM

Understand CDM architecture

Set up your local environment and create a cluster

Deploy the platform

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

---

1. Installing Prometheus, Grafana, and Alertmanager technology in the CDM provides an example of how you might set up monitoring and alerting in a Ping Identity Platform deployment in the cloud. Remember, the CDM is a reference implementation 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.

UI and API access

This page shows you how to access and monitor the Ping Identity Platform components that make up the CDM.

AM and IDM are configured for access through the CDM cluster's Kubernetes 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.

For more information about how AM and IDM have been configured in the CDM, see Configuration in the `forgeops` repository's top-level README file for more information about the configurations.

AM services

To access the AM admin UI:

1. Set the active namespace in your local Kubernetes context to the namespace in which you have deployed the CDM.
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://cdm.example.com/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://cdm.example.com/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 have deployed the CDM.
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://cdm.example.com/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. The CDM comes 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://cdm.example.com/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://cdm.example.com/am/oauth2/realms/root/authorize?redirect_uri=https://
    cdm.example.com/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://cdm.example.com/platform/appAuthHelperRedirect.html
     ?code=3cItL9G52DIiBdfXRngv2_dAaYM&iss=http://cdm.example.com: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://cdm.example.com/platform/appAuthHelperRedirect.html" \
 "https://cdm.example.com/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://cdm.example.com/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 the CDM 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 CDM directory pods, the directory superuser DN is `uid=admin`. Obtain this user's password by running the **forgeops info** command.

CDM monitoring

This section describes how to access Grafana dashboards and Prometheus UI.

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, enter Ctrl+c in the terminal window where you initiated port forwarding.

For information about Grafana, see 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, enter Ctrl+c in the terminal window where you initiated port forwarding.

For information about Prometheus, see the Prometheus documentation.

For a description of the CDM monitoring architecture and information about how to customize CDM monitoring, see CDM monitoring.

Next step

Become familiar with the CDM

Understand CDM architecture

Set up your local environment and create a cluster

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

CDM removal

To remove your CDM cluster when you're done working with it:

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

Next steps

If you've followed the instructions for deploying the CDM *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 ForgeRock component.
- DS replication and failover work as expected.
- Monitoring tools are installed and running. You can access a monitoring console for DS, AM, and IDM.

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 in the cloud. Congratulations!

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

- Running lightweight benchmark tests
- Making backups of CDM 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

Now that you're familiar with the CDM—ForgeRock's reference implementation—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 ForgeRock 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 the CDK, and build custom Docker images for the Ping Identity Platform. The CDK documentation 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 How-tos 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 might include use cases, service-level agreements, thresholds, and load test profiles. The How-tos, and READMEs in the `forgeops` repository, provide information about site reliability.

How-tos

After you get the CDM up and running, you can use it to test deployment customizations—options that are not part of the CDM, but which you may want to use when you deploy in production.

Start Here

Important information for deploying the platform on Kubernetes.

Base Docker Images

Create Docker images for deploying the platform in production.

Identity Gateway

Add PingGateway to your deployment.

Monitoring

Customize Prometheus monitoring and alerts.

Benchmarks

Run ForgeRock's lightweight benchmarks.

Security

Customize the security features built into the CDM.

Ingress

Use an alternate ingress controller when deploying the CDM.

Backup and Restore

Back up and restore CDM data, such as identities and tokens.

Upgrade

Back up and restore CDM data, such as identities and tokens.

The Ping Identity Platform serves as the basis for our simple and comprehensive identity and access management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

Base Docker images

ForgeRock provides 13 Docker images for deploying the Ping Identity Platform:

- Eight unsupported, evaluation-only base images:
  - `amster`
  - `am-cdk`
  - `am-config-upgrader`
  - `ds`
  - `idm-cdk`
  - `ig`
  - `java-11`
  - `java-17`
- Five supported base images that implement the platform's user interface elements and ForgeOps operators:
  - `ds-operator`
  - `platform-admin-ui`
  - `platform-enduser-ui`
  - `platform-login-ui`
  - `secret-agent`

The Docker images are publicly available in ForgeRock's Docker repository, `gcr.io/forgerock-io`.

Which Docker images do I deploy?

- I am a developer using the CDK.
  - UI elements. Deploy the supported images from ForgeRock.
  - Other platform elements. Deploy either:
    - The evaluation-only images from ForgeRock.
    - Docker images that are based on the evaluation-only images, but contain a customized configuration profile.
- I am doing a proof-of-concept CDM deployment.
  - UI elements. Deploy the supported images from ForgeRock.
  - Other platform elements. Deploy either:
    - The evaluation-only images from ForgeRock.
    - Docker images that are based on the evaluation-only images, but contain a customized configuration profile.

- I am deploying the platform in production.
    - UI elements. Deploy the supported images from ForgeRock.
    - Other platform elements. Deploy Docker images that are based on your own base images, but contain a customized configuration profile. ForgeRock does not support production deployments with Docker images based on the evaluation-only images.

Your own base Docker images

Perform the following steps to build base images for the eight unsupported, evaluation-only Docker 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 `release/7.3-20240131` branch:

    ```
    $ cd /path/to/forgeops
    $ git checkout release/7.3-20240131
    ```

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

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

5. Build Java base images, which are required by several of the other Dockerfiles:
    1. Build the Java 11 Docker image:

```
$ cd /path/to/forgeops-extras/images/java-11
$ docker build --tag my-repo/java-11 .

 ⇒ [internal]
load .dockerignore
0.0s
 ⇒ ⇒ transferring context:
2B
0.0s
 ⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒ ⇒ transferring dockerfile:
2.42kB
0.0s
 ⇒ [internal] load metadata for docker.io/library/debian:buster-
slim                                                                    2.9s
 ⇒ [internal] load metadata for docker.io/azul/zulu-openjdk-debian:11-
latest                                                                 2.7s
 ⇒ [stage-0 1/4] FROM docker.io/azul/zulu-openjdk-debian:11-
latest@sha256:aa1df513d9f6d0025e4e1a890732192a1aee473437a01a885136fc0f5db2622e
22.4s
...
 ⇒ exporting to
image
0.3s
 ⇒ ⇒ exporting
layers
0.3s
 ⇒ ⇒ writing image
sha256:76742b285ddf975ab6b36e1ad91d63cd6d5920d0f096d222f93ffa6026b7f7f5
0.0s
 ⇒ ⇒ naming to docker.io/my-repo/java-11
```

2. Build the Java 17 Docker image:

```
$ cd /path/to/forgeops-extras/images/java-17
$ docker build --tag my-repo/java-17 .

⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
2.38kB
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for docker.io/library/debian:bullseye-
slim                                                                    1.1s
 ⇒ [internal] load metadata for docker.io/azul/zulu-openjdk-debian:
17                                                                     1.3s
 ⇒ [jdk 1/3] FROM docker.io/azul/zulu-openjdk-debian:
17@sha256:420a137d0576e3fd0d6f6332f5aa1aef85314ed83b3797d7f965e0b9169cbc57
17.7s
...
⇒ exporting to
image
0.3s
 ⇒⇒ exporting
layers
0.3s
 ⇒⇒ writing image
sha256:cc52e9623b3cd411682ca221a6722e83610b6b7620f126d3f7c4686e79ff1797
0.0s
 ⇒⇒ naming to docker.io/my-repo/
java-17
0.0s
```

6. Build the base image for Amster. This image must be available to build the base image for AM in the next step:

    1. Make a directory named `amster`.
    2. Unzip the Amster `.zip` file into the new `amster` directory.
    3. Change to the samples/docker directory in the expanded `.zip` file output.
    4. 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
```

5. Edit the Dockerfile in the samples/docker directory. Change the line:

```
FROM gcr.io/forgerock-io/java-11:latest
```

to:

```
FROM my-repo/java-11
```

6. Build the `amster` Docker image:

```
$ docker build --tag amster:7.3.0 .

 ⇒ [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-11:latest
1.1s
 ⇒ [1/8] FROM docker.io/my-repo/java-11
...
 ⇒ exporting to image
 ⇒ ⇒ exporting layers
 ⇒ ⇒ writing image
sha256:bc47...f9e52
0.0s
 ⇒ ⇒ naming to docker.io/library/amster:7.3.0
```

7. Build the AM empty image:

1. Unzip the AM `.zip` file.
2. Change to the openam/samples/docker directory in the expanded `.zip` file output.
3. Run the setup.sh script:

```
$ chmod +x ./setup.sh
./setup.sh
```

4. Change to the images/am-empty directory.
5. Edit the Dockerfile in the samples/docker directory. Change the line:

```
FROM tomcat:9-jdk11 AS base
```

to:

```
FROM my-repo/java-11
```

6. Build the `am-empty` Docker image:

```
$ docker build --tag am-empty:7.3.0 .

 ⇒ [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-jdk11-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-jdk11-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:7.3.0
```

8. Build the AM image:

    1. Change to the ../am-base directory.

    2. Edit the Dockerfile in the ../am-base directory and change the line:

```
FROM amster:${docker_tag} as amster

FROM am-empty:${docker_tag} AS generator
```

to:

```
FROM amster:7.3.0

FROM am-empty:7.3.0
```

3. Build the `am-base` Docker image:

```
$ docker build --build-arg docker_tag=7.3.0 --tag am-base:7.3.0 .

 ⇒ [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:
7.3.0                                                       0.0s
 ⇒ [internal] load metadata for docker.io/library/am-empty:
7.3.0                                                     0.0s
 ⇒ [internal] load build
context                                                                       0.4s
 ⇒ ⇒ transferring context:
35.66MB                                                                     0.4s
 ⇒ [generator  1/15] FROM docker.io/library/am-empty:
7.3.0                                                          0.4s
 ⇒ [amster 1/1] FROM docker.io/library/amster:
7.3.0                                                               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:7.3.0
```

4. Change to the ../am-cdk directory.

5. Edit the Dockerfile in the ../am-cdk directory. Change the line:

```
FROM am-base:${docker_tag}
```

to:

```
FROM am-base:7.3.0
```

6. Build the `am` Docker image:

```
$ docker build --build-arg docker_tag=7.3.0 --tag my-repo/am:7.3.0 .

[+] Building 8.7s (11/11) FINISHED
 ⇒ [internal] load build definition from
Dockerfile                                                              0.0s
 ⇒⇒ transferring dockerfile:
2.02kB                                                                  0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B                                                                      0.0s
 ⇒ [internal] load metadata for docker.io/library/am-base:
7.3.0                                                   0.0s
 ⇒ [1/6] FROM docker.io/library/am-base:
7.3.0                                                    0.3s
 ⇒ [internal] load build
context                                                                 0.1s
 ⇒⇒ transferring context:
397.45kB                                                                0.1s
 ⇒ [2/6] RUN apt-get update         && apt-get install -y git         && apt-get clean
&& rm -r /var/lib  6.7s
 ⇒ [3/6] RUN cp -R /usr/local/tomcat/webapps/am/XUI /usr/local/tomcat/webapps/am/
OAuth2_XUI                      0.8s
 ⇒ [4/6] COPY --chown=forgerock:root /config /home/forgerock/cdk/
config                                          0.0s
 ⇒ [5/6] COPY --chown=forgerock:root logback.xml /usr/local/tomcat/webapps/am/WEB-INF/
classes                    0.0s
 ⇒ [6/6] RUN rm -rf /home/forgerock/openam/config/services &&     mkdir /home/forgerock/
openam/config/services    0.3s
 ⇒ exporting to
image
0.5s
 ⇒⇒ exporting
layers
0.5s
 ⇒⇒ writing image
sha256:9b1119bd37c8810cebb446d1b1fb67841bbeba6416a0aed73e7a5bf2b84deff3
0.0s
 ⇒⇒ naming to my-repo/am:
7.3.0                                                                   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:7.3.0 my-repo/amster:7.3.0
```

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-7.3.0.zip` file.
    3. Change to the `amupgrade/samples/docker` directory in the expanded `Config-Upgrader-7.3.0.zip` file output.
    4. Edit the Dockerfile in the amupgrade/samples/docker directory. Change the line:

    ```
    FROM gcr.io/forgerock-io/java-11:latest
    ```

    to:

    ```
    FROM my-repo/java-11
    ```

    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:7.3.0 .

 ⇒ [internal] load build definition from
Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
1.14kB
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for docker.io/my-repo/
java-11:latest
0.2s
 ⇒ [internal] load build
context
0.4s
 ⇒⇒ transferring context:
15.44MB
0.4s
 ⇒ CACHED [1/4] FROM docker.io/my-repo/
java-11
0.0s
 ⇒ [2/4] RUN apt-get update &&     apt-get upgrade -
y                                                                                4.3s
 ⇒ [3/4] COPY --chown=forgerock:root docker-entrypoint.sh /home/
forgerock/                                                                  0.0s
 ⇒ [4/4] COPY build/ /home/
forgerock/
0.1s
 ⇒ exporting to
image
0.1s
 ⇒⇒ exporting
layers
0.1s
 ⇒⇒ writing image
sha256:c06eb12006468f50eb79621a3e945ce52dec0775c46879d2ad4d07296fd5b818
0.0s
 ⇒⇒ naming to my-repo/am-config-upgrader:7.3.0
```

11. Build the base image for DS:

    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/java-17
```

5. Build the `ds` base image:

```
$ docker build --tag my-repo/ds-empty:7.3.0 .

[+] 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/
java-17:latest
1.7s
 ⇒ [internal] load build
context
1.2s
 ⇒⇒ transferring context:
60.85MB
1.2s
 ⇒ CACHED [1/4] FROM my-repo/java-17: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:7.3.0
```

12. Build the base image for IDM:

    1. Unzip the IDM `.zip` file.
    2. Change to the openidm directory in the expanded `.zip` file output.
    3. Edit the Custom.Dockerfile in the openidm/bin directory. Change the line:

    ```
    FROM gcr.io/forgerock-io/java-11:latest
    ```

    to:

```
FROM my-repo/java-11
```

4. Build the `idm` base image:

```
$ docker build . --file bin/Custom.Dockerfile --tag my-repo/idm:7.3.0

[+] Building 8.1s (9/9) FINISHED
 ⇒ [internal] load build definition from
Custom.Dockerfile
0.0s
 ⇒⇒ transferring dockerfile:
648B
0.0s
 ⇒ [internal]
load .dockerignore
0.0s
 ⇒⇒ transferring context:
2B
0.0s
 ⇒ [internal] load metadata for my-repo/
java-11:latest
0.3s
 ⇒ CACHED [1/4] FROM my-repo/java-11:latest
 ⇒ [internal] load build
context
9.7s
 ⇒⇒ transferring context:
322.62MB
9.7s
 ⇒ [2/4] RUN apt-get update &&     apt-get install -y ttf-
dejavu                                                                          10.3s
 ⇒ [3/4] COPY --chown=forgerock:root . /opt/
openidm
2.3s
 ⇒ [4/4] WORKDIR /opt/
openidm
0.0s
 ⇒ exporting to
image
3.3s
 ⇒⇒ exporting
layers
3.3s
 ⇒⇒ writing image
sha256:9550...5788
0.0s
 ⇒⇒ naming to my-repo/idm:7.3.0
```

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-11:latest
```

   to:

```
FROM my-repo/java-11
```

4. Build the `ig` base image:

```
$ docker build . --file docker/Dockerfile --tag my-repo/ig:2023.9.0

[+] 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/
java-11:latest
0.3s
 ⇒ [internal] load build
context
2.2s
 ⇒⇒ transferring context:
113.60MB
2.2s
 ⇒ CACHED [1/3] FROM my-repo/java-11: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:2023.9.0
```

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                   7.3.0      552073a1c000   1 hour ago     795MB
my-repo/am-config-upgrader   7.3.0      d115125b1c3f   1 hour ago     795MB
my-repo/amster               7.3.0      d9e1c735f415   1 hour ago     577MB
my-repo/ds-empty             7.3.0      ac8e8ab0fda6   1 hour ago     196MB
my-repo/idm                  7.3.0      0cc1b7f70ce6   1 hour ago     387MB
my-repo/ig                   2023.9.0   cc52e9623b3c   1 hour ago     249MB
my-repo/java-11              latest     76742b285ddf   1 hour ago     146MB
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:

    - `my-repo/am:7.3.0`
    - `my-repo/am-config-upgrader:7.3.0`
    - `my-repo/amster:7.3.0`
    - `my-repo/ds-empty:7.3.0`
    - `my-repo/idm:7.3.0`
    - `my-repo/java-11`
    - `my-repo/java-17`

    If you're deploying your own PingGateway base image, also push the `my-repo/ig:2023.9.0` 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.
- Must *not* be based on ForgeRock's evaluation-only 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:7.3.0` [1]
docker/amster/Dockerfile	`FROM my-repo/amster:7.3.0`
docker/ds/ds-new/Dockerfile	`FROM my-repo/ds-empty:7.3.0`
docker/idm/Dockerfile	`FROM my-repo/idm:7.3.0` [2]
(Optional) docker/ig/Dockerfile	`FROM my-repo/ig:2023.9.0`

4. If necessary, log in to your Docker registry.
5. Build Docker images that are based on your own base images. The AM and IDM images contain your customized configuration profiles:

```
$ cd /path/to/forgeops/bin
$ ./forgeops build ds --push-to my-repo
$ ./forgeops build amster --push-to my-repo
$ ./forgeops build am --push-to my-repo --config-profile my-profile
$ ./forgeops build idm --push-to my-repo --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 install command uses to determine which Docker images to run.
6. (Optional) Build and push an PingGateway Docker image that's based on your own base image and contains your customized configuration profile:

```
$ ./forgeops build ig --config-profile my-profile --push-to my-repo
```

7. 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 CDM 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.
8. Install the CDM in your cluster:

```
$ ./forgeops install --small --fqdn cdm.example.com
```

9. Access the AM admin UI and the IDM admin UI, and verify that your customized configuration profiles are active.
10. 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 install 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.

Identity Gateway

IG Deployment

Add PingGateway to a CDK or CDM deployment.

Custom IG Image

Build a custom PingGateway image, and deploy it in the CDK.

CDM monitoring

The CDM uses 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 the CDM:

Overview

Monitoring installation and architecture.

Monitoring Pods

Prometheus and Grafana pods that monitor the CDM and provide reporting services.

Grafana Dashboards

Grafana dashboards for the platform that are available in the CDM.

Prometheus Alerts

Prometheus alerts for the platform that are available in the CDM.

CDM security

This topic describes several options for securing a CDM deployment of the Ping Identity Platform:

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.

CDM benchmarks

The benchmarking instructions in this part of the documentation give you a method to validate performance of your CDM 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.

Remember, the CDM is a reference implementation and not for production use. When you create 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.

CDM Benchmarking checklist

- ☐ Become familiar with CDM benchmarking
- ☐ Install third-party software
- ☐ Generate test users
- ☐ Benchmark the authentication rate
- ☐ Benchmark the OAuth 2.0 authorization code flow

Ingress

The CDK and CDM use Ingress-NGINX Controller.

For the CDK:

- The cdk-minikube start command installs the ingress add-on in Minikube clusters.
- Cluster administrators deploy Ingress-NGINX Controller in shared GKE, EKS, and AKS clusters.

For the CDM:

- The tf-apply cluster creation script deploys NGINX Ingress Controller by default when it creates new Kubernetes clusters.
- Alternatively, you can deploy HAProxy Ingress as your ingress controller.

HAProxy Ingress

This section lists adjustments you'll need to make if you want to deploy the CDM using HAProxy Ingress as your ingress controller instead of NGINX Ingress 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 deploy the CDM:

1. Specify the --ingress-class haproxy argument. For example:

```
$ cd /path/to/forgeops/bin
$ ./forgeops install --small --ingress-class haproxy --fqdn cdm.example.com --namespace my-namespace
```

Backup and restore

Backup and restore overview

CDM 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

These two directory services, implemented as Kubernetes stateful sets, are managed by the DS operator.

Before deploying the Ping Identity Platform in production, create and test a backup plan that will let you recover these two directory services should you experience data loss.

Choose a backup solution

There are numerous options you can use when implementing data backup. The CDM provides two solutions:

- Kubernetes volume snapshots.
- Backup DS utilities.

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 PX-Backup. These third-party products are cloud-platform agnostic, and can be used across cloud platforms.

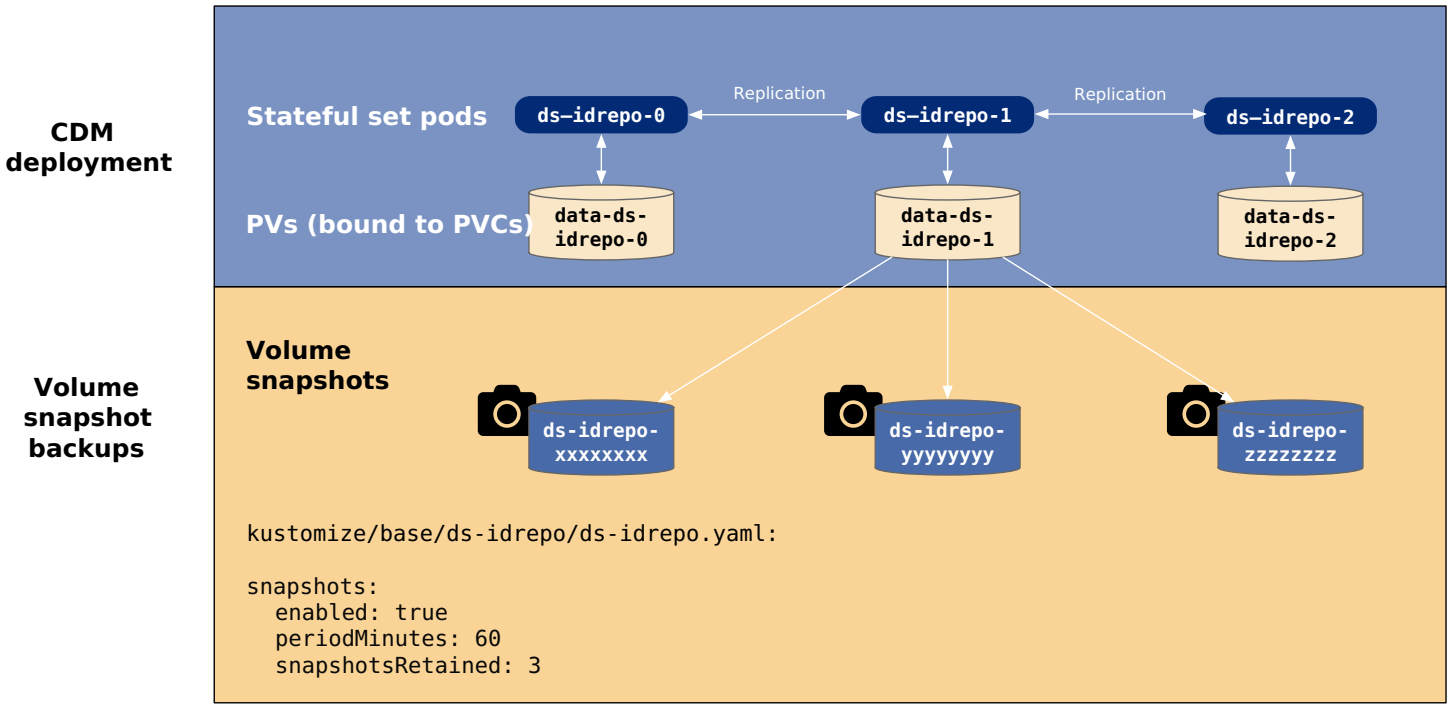
Your organization might have specific needs for its backup solution. Some factors to consider include:

- 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, in which 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 the content 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 can also facilitate testing by letting you initialize a directory with sample data.

When you create a Kubernetes cluster for deploying the CDM, you create a Kubernetes volume snapshot class named `ds-snapshot-class`. The DS operator uses this class for creating snapshots. Volume snapshot backups are based on configuration in the /path/to/forgeops/kustomize/base/ds-idrepo/ds-idrepo.yaml file:



The next sections include example steps to back up and restore the `ds-idrepo` directory. To back up and restore the `ds-cts` directory, follow similar steps.

Back up the `ds-idrepo` directory

To start taking volume snapshot backups of the `ds-idrepo` directory:

1. Set the active namespace in your local Kubernetes context to the namespace in which the CDM is deployed.
2. Run the kubectl get pvc command to get the size of the volume that holds the `ds-idrepo` directory's data. The `CAPACITY` column contains the volume size:

```
$ kubectl get pvc
NAME                STATUS   VOLUME                                     CAPACITY   ...
...
data-ds-idrepo-0    Bound    pvc-04293c38-05a8-44b0-b137-0db259854971   100Gi      ...
data-ds-idrepo-1    Bound    pvc-04ab2617-a9a2-4f71-9094-6d3a4b7c0082   100Gi      ...
data-ds-idrepo-2    Bound    pvc-19a9915e-46f4-4ba5-b3fa-7d1ff83f38aa   100Gi      ...
...
```

3. Update the /path/to/forgeops/kustomize/base/ds-idrepo/ds-idrepo.yaml file, which contains the snapshot backup and restore configuration for the `ds-idrepo` directory instance:

    1. Set the value of `replicas` to `3`.
    2. Set the value of `storage` in the `volumeClaimSpec/resources/requests:` section to the size of the volume that holds the `ds-idrepo` directory's data.
    3. Uncomment the `dataSource` section by removing the `#` character from the four lines staring with `#dataSource:`.

        The `dataSource` section tells the CDM which snapshot to use when restoring one of the `data-ds-idrepo` PVCs. The PVCs are restored from a snapshot if:

        - The PVC does not exist.
        - The snapshot backup configured in the `dataSource` section does exist.

    4. Configure the `snapshots` section so that snapshot backups start on the `ds-idrepo-1` pod:

        1. Set `enabled` to `true`.
        2. Set `periodMinutes` to the interval, in minutes, between snapshots.
        3. Set `snapshotsRetained` to the number of snapshots to keep.
        4. Set `directoryInstance` to `1`, and uncomment the line if it is commented. This setting configures the DS operator to snapshot the `ds-idrepo-1` instance—a secondary instance.

    5. Save and close the file.

4. Apply the changes to the DS configuration:

```
$ cd /path/to/forgeops/kustomize/base
$ kubectl apply -f ds-idrepo/ds-idrepo.yaml
directoryservice.directory.forgerock.io/ds-idrepo configured
```

5. After allowing enough time for one or more snapshots to be created, run the kubectl get volumesnapshots command.

    You should see one or more snapshots that are ready to use listed in the command output:

```
NAME                    READYTOUSE   SOURCEPVC          ...    AGE
ds-idrepo-1653077404    true         data-ds-idrepo-1   ...    44s
```

Restore the `ds-idrepo` directory

To test restoring DS instances from a snapshot:

1. In a browser window, log in to the Ping Identity Platform admin UI, and then create an example identity using the **Identities > Manage** option.

    You'll use this identity to verify that the restore test worked correctly.

2. Log out of the Ping Identity Platform admin UI.

3. Run the kubectl get volumesnapshots command until you can verify that a new snapshot was created after you created the example identity:

```
NAME                    READYTOUSE   SOURCEPVC          ...    AGE
ds-idrepo-1653077404    true         data-ds-idrepo-1   ...    16m3s
ds-idrepo-1653077584    true         data-ds-idrepo-1   ...    6m3s
ds-idrepo-1653077765    true         data-ds-idrepo-1   ...    3s
```

   Note the name of the latest snapshot. Because the data source `name` has the value `"$(latest)"` in the ds-idrepo.yaml file, the latest snapshot is used when you restore the `ds-idrepo` directory service.

4. Disable taking snapshots:

   1. Set `enabled : false` in the `snapshots` section of the ds-idrepo.yaml file.

   2. Apply the changes:

```
$ cd /path/to/forgeops/kustomize/base
$ kubectl apply -f ds-idrepo/ds-idrepo.yaml
directoryservice.directory.forgerock.io/ds-idrepo configured
```

5. Delete the `ds-idrepo` directory service custom resource. Be sure to reply `N` when you're prompted to delete volume snapshots and secrets:

```
$ cd /path/to/forgeops/bin
$ ./forgeops delete ds-idrepo
"small" platform detected in namespace: "my-namespace".
Uninstalling component(s): ['ds-idrepo'] from namespace: "my-namespace".
OK to delete components? [Y/N] Y
OK to delete PVCs? [Y/N] Y
OK to delete volume snapshots? [Y/N] N
OK to delete secrets? [Y/N] N
directoryservice.directory.forgerock.io "ds-idrepo" deleted
```

6. Redeploy `ds-idrepo`:

```
$ cd /path/to/forgeops
$ ./bin/forgeops install ds-idrepo
```

7. Use the kubectl get pods command to monitor the status of the `ds-idrepo` pods. Wait until these pods are in the `Running` state before proceeding to the next step.

8. The preceding events also force the IDM pods to restart. Wait until these pods have restarted before proceeding to the next step.

9. Log back in to the Ping Identity Platform admin UI, and then select the **Identities > Manage** option.

   You should see your example identity.

10. Run the kubectl describe pvc data-ds-idrepo-0 command and review the output under the label, `DataSource`:

```
DataSource:
  APIGroup:  snapshot.storage.k8s.io
  Kind:      VolumeSnapshot
  Name:      ds-idrepo-1653077765
```

The `Kind` field should have a value of `VolumeSnapshot`, indicating that the source of the PVC was a volume snapshot.

The value in the `Name` field should match the name of the latest volume snapshot that was taken before you deleted the `ds-idrepo` directory instance.

11. Run the kubectl describe pvc data-ds-idrepo-1 and kubectl describe pvc data-ds-idrepo-2 commands. The output should be similar to what you observed in the previous step.
12. Optionally, re-enable taking volume snapshots:
    1. Set `enabled : true` in the 'snapshots` section of the ds-idrepo.yaml file.
    2. Apply the changes:

```
$ cd /path/to/forgeops/kustomize/base
$ kubectl apply -f ds-idrepo/ds-idrepo.yaml
directoryservice.directory.forgerock.io/ds-idrepo configured
```

Backup and restore using DS utilities

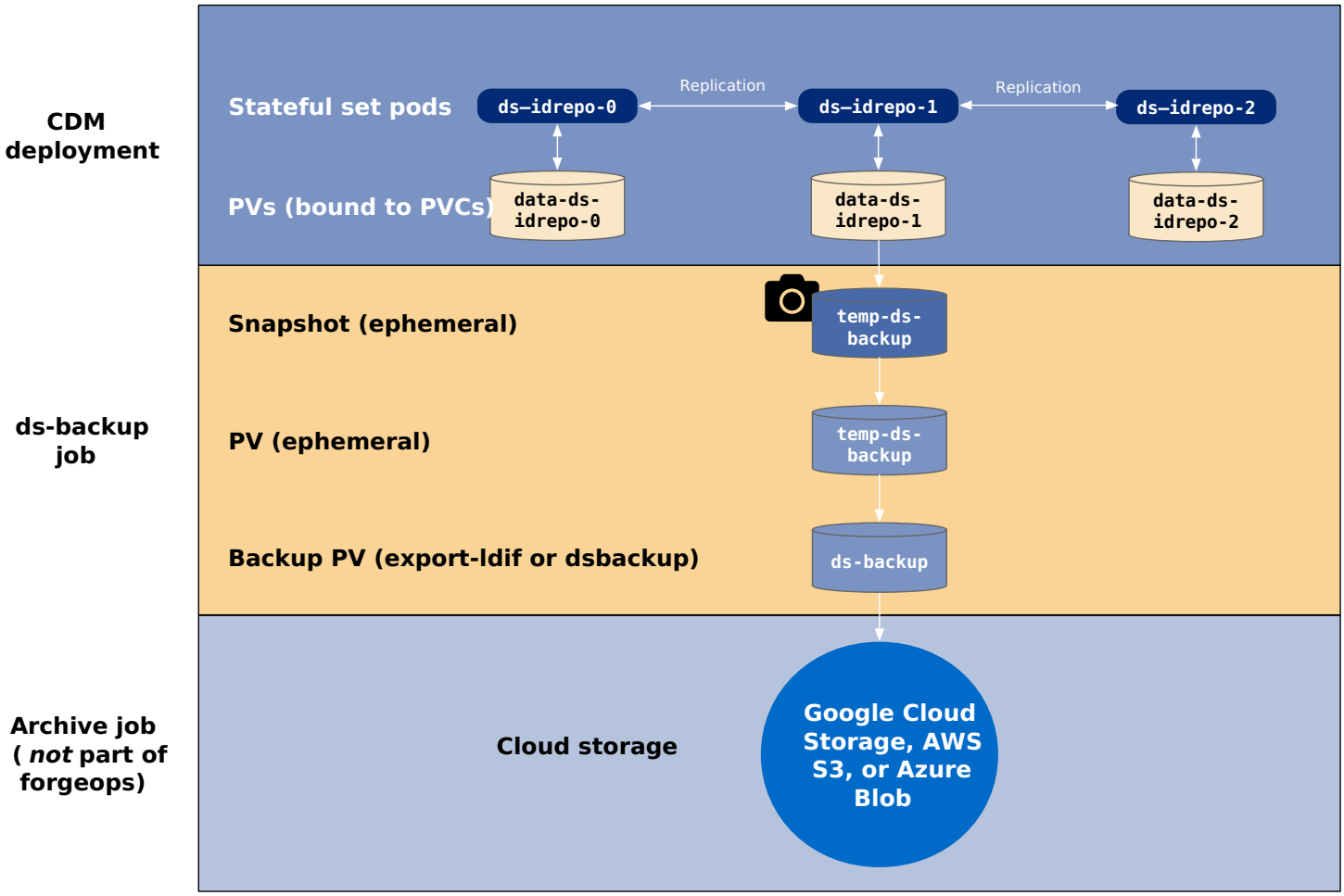
The DS operator supports two DS utilities that back up directory data:

- dsbackup
- export-ldif

To back up directory data using one of these DS utilities, update the DS operator's configuration to create a backup job. The backup job:

- Takes a snapshot of the directory data volume.
- Copies the data from the snapshot to an ephemeral persistent volume.
- Runs the dsbackup or export-ldif utility on the data on the ephemeral volume. The DS utility writes the utility's output to another persistent volume.

You then archive the backup to cloud storage, where you can maintain it for as long as you need to.



The next sections include example steps to back up and restore the `ds-idrepo` directory. To back up and restore the `ds-cts` directory, follow similar steps.

Back up the `ds-idrepo` directory

To back up the `ds-idrepo` directory using the dsbackup or export-ldif utility:

1. In a browser window, log in to the Ping Identity Platform admin UI, and then create an example identity using the **Identities > Manage** option.

   You'll use this identity later to verify that backup and restore were successful.

2. Log out of the Ping Identity Platform admin UI.

3. Set the active namespace in your local Kubernetes context to the namespace in which the CDM is deployed.

4. Run the kubectl get pvc command to get the size of the volume that holds the `ds-idrepo` directory's data. The `CAPACITY` column contains the volume size:

```
$ kubectl get pvc
NAME               STATUS   VOLUME                                     CAPACITY   ...
...
data-ds-idrepo-0   Bound    pvc-04293c38-05a8-44b0-b137-0db259854971   100Gi      ...
data-ds-idrepo-1   Bound    pvc-04ab2617-a9a2-4f71-9094-6d3a4b7c0082   100Gi      ...
data-ds-idrepo-2   Bound    pvc-19a9915e-46f4-4ba5-b3fa-7d1ff83f38aa   100Gi      ...
...
```

5. Update the /path/to/forgeops/etc/backup/ds-backup-restore-ds-operator/ds-backup.yaml file, which contains a default configuration for backing a CDM directory:
    1. Set `volumeClaimSpec/resources/requests/storage` to the size of the volume that holds the `ds-idrepo` directory's data.
    2. Set `env/value` to `tar,ldif` (for an LDIF backup) or to `tar,dsbackup` (for a standard DS backup).
    3. Set the value of `claimToBackup` to `data-ds-idrepo-1`.
6. Apply your changes to the DS operator's backup configuration. Changing this configuration initiates the backup process:

```
$ cd /path/to/forgeops/etc
$ kubectl apply -f backup/ds-backup-restore-ds-operator/ds-backup.yaml
directorybackup.directory.forgerock.io/ds-backup created
```

7. Verify that the backup process started:
    1. Run the kubectl get jobs command. You should see a job named `ds-backup`.
    2. Run the kubectl get pods command. You should see a pod whose name starts with the string, `ds-backup-`.
    3. Run the kubectl get volumesnapshot command. You should see that the `temp-ds-backup` snapshot has been created. This snapshot is ephemeral, used only during the backup process.
    4. Run the kubectl get pvc command. You should see that two new PVCs have been created:
        - The `temp-ds-backup` PVC, which is an ephemeral PVC, is used as input to the DS backup utility that you're running.
        - The `ds-backup` PVC, which is created from the `temp-ds-backup` PVC, contains the output from the backup. This PVC should be archived to cloud storage after the backup process has completed.
8. Verify that the backup process completed:
    1. Review the value in the `COMPLETIONS` column of the kubectl get jobs ds-backup command output:

```
$ kubectl get jobs ds-backup
NAME        COMPLETIONS   DURATION   AGE
ds-backup   1/1           3m13s      3m40s
```

    2. Review the `ds-backup` pod's log, which contains the output from the DS utility—either `dsbackup` or `export-ldif`.

       The text `done` in the last line of the log indicates that the backup completed.

With the backup job completed, you're ready to archive the `ds-backup` PVC to cloud storage.

Archive

After you've backed up your data using one of the DS utilities, move the data to cloud storage.

Archive your backup data by using a tool from your cloud provider or a third-party product. The CDM does not include a data archival tool.

Restore the `ds-idrepo` directory

To test restoring DS instances using DS utility backups:

1. Set the active namespace in your local Kubernetes context to the namespace in which the CDM is deployed.
2. Delete your CDM deployment. Be sure to reply `N` when you're prompted to delete PVCs, volume snapshots, and secrets:

```
$ cd /path/to/forgeops/bin
$ ./forgeops delete
"small" platform detected in namespace: "my-namespace".
Uninstalling component(s): ['all'] from namespace: "my-namespace".
OK to delete components? [Y/N] Y
OK to delete PVCs? [Y/N] N
OK to delete volume snapshots? [Y/N] N
OK to delete secrets? [Y/N] N
service "admin-ui" deleted
...
```

3. Delete the `ds-idrepo` PVCs:

```
$ kubectl delete pvc data-ds-idrepo-0 data-ds-idrepo-1 data-ds-idrepo-2
persistentvolumeclaim "data-ds-idrepo-0" deleted
persistentvolumeclaim "data-ds-idrepo-1" deleted
persistentvolumeclaim "data-ds-idrepo-2" deleted
```

4. Transfer your archived backup data from cloud storage to a persistent volume using a tool from your cloud provider, or a third-party product.

   Note that the CDM does not include a tool to transfer data from cloud storage to a persistent volume.

5. Update the /path/to/forgeops/etc/backup/ds-backup-restore-ds-operator/ds-restore.yaml file, which contains a default configuration for restoring a CDM directory:

   1. Set `volumeClaimSpec/resources/requests/storage` to the size of the volume that holds the `ds-idrepo` directory's data.
   2. Set `env/value` to `tar,ldif` (for an LDIF restore) or to `tar,dsbackup` (for a standard DS restore). Use the same value that you used when you backed up the DS instance.
   3. Set the value of `sourcePvcName` to the name of the PVC that contains the backup that you transferred from cloud storage.

6. Apply your changes to the DS operator's restore configuration:

```
$ cd /path/to/forgeops/etc
$ kubectl apply -f backup/ds-backup-restore-ds-operator/ds-restore.yaml
directorybackup.directory.forgerock.io/ds-restore created
```

7. Verify that the restore process started:

    1. Run the kubectl get jobs command. You should see a job named `ds-restore`.
    2. Run the kubectl get pods command. You should see a pod whose name starts with the string, `ds-restore-`.
    3. Run the kubectl get volumesnapshots command. You should see a snapshot named `temp-ds-restore`:
        - The DS operator creates this snapshot from your backup PVC.
        - The `temp-ds-restore` snapshot will be used to initialize the `ds-idrepo` directory service's PVCs.

8. Verify that the restore process completed:

    1. Review the value in the `COMPLETIONS` column of the kubectl get jobs ds-restore command output:

```
$ kubectl get jobs ds-restore
NAME         COMPLETIONS   DURATION   AGE
ds-restore   1/1           3m13s      3m40s
```

    2. Review the `ds-restore` pod's log, which contains the output from the DS utility—either `dsbackup` or `export-ldif`.
    3. Review output from the kubectl get volumesnapshots command. For the `temp-ds-restore` volume snapshot, verify that the value in the `READYTOUSE` column is `true`:

```
$ kubectl get volumesnapshots
NAME              READYTOUSE   SOURCEPVC           RESTORESIZE   ...
temp-ds-restore   true         temp-ds-restore     100Gi         ...
temp-ds-backup    true         data-ds-idrepo-1    100Gi         ...
```

    Do not proceed to the next step until the `temp-ds-restore` volume snapshot is ready to use.

9. Update the `ds-idrepo` overlay file for your deployment. Specify that the source of the `ds-idrepo` PVC should be the `temp-ds-restore` snapshot:

    1. Determine the path to the `ds-idrepo` overlay file for your deployment. For example, the overlay file for a small CDM deployment is at the path, /path/to/forgeops/kustomize/overlay/small/ds-idrepo.yaml.
    2. Add the following `dataSource` key to the `podTemplate/volumeClaimSpec` section of the overlay file:

```
dataSource:
  name: temp-ds-restore
  kind: VolumeSnapshot
  apiGroup: snapshot.storage.k8s.io
```

```
# Sample DirectoryService deployment
apiVersion: directory.forgerock.io/v1alpha1
kind: DirectoryService
metadata:
  name: ds-idrepo
spec:
  # The number of DS servers in the topology
  replicas: 3
  # The resources assigned to each DS pod
  podTemplate:
    resources:
      requests:
        memory: 4Gi
        cpu: 1500m
      limits:
        memory: 6Gi
    volumeClaimSpec:
      storageClassName: fast
      resources:
        requests:
          storage: 100Gi
      # The following triggers restoring from a snapshot:
      dataSource:
        name: temp-ds-restore
        kind: VolumeSnapshot
        apiGroup: snapshot.storage.k8s.io
```

The values in the `dataSource` key tell the CDM which snapshot to use when restoring the `ds-idrepo` PVC. The PVC is restored from a snapshot if:

- The PVC does not exist.
- The snapshot backup configured in the `dataSource` section exists.

10. Reinstall the CDM using the forgeops install command you had used when you deployed the CDM, for example:

```
$ ./bin/forgeops install --small --fqdn cdm.example.com
```

11. Run the kubectl describe pvc data-ds-idrepo-0 command and review the output under the label, `DataSource`:

```
DataSource:
  APIGroup:  snapshot.storage.k8s.io
  Kind:      VolumeSnapshot
  Name:      temp-ds-restore
```

The `Kind` field should have a value of `VolumeSnapshot`, indicating that the source of the PVC was the ephemeral volume snapshot created by the restore job.

The value in the `Name` field should be `temp-ds-restore`, which is the name of the ephemeral snapshot.

12. Run the kubectl describe pvc data-ds-idrepo-1 and kubectl describe pvc data-ds-idrepo-2 commands. The output should be similar to what you observed in the previous step.
13. Log back in to the Ping Identity Platform admin UI, and then select the **Identities > Manage** option.

    You should see the example identity you created before you took a backup.

Upgrade the platform from version 7.2 to 7.3

If you've already installed Ping Identity Platform version 7.2 using artifacts from the `forgeops` repository, follow the steps provided on this page to upgrade to version 7.3.

Use these steps to upgrade the platform *in place, with no downtime*.

This upgrade methodology has been tested against a deployment based on ForgeRock's evaluation-only Docker images with basic configuration settings.

> **Important**
>
> Because the Ping Identity Platform is highly customizable, it is difficult for ForgeRock 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

To upgrade the platform from version 7.2 to 7.3, you'll need:

- A running version 7.2 CDK deployment with your current AM and IDM configurations
- A running version 7.2 CDM deployment
- A `forgeops` repository clone with a branch that contains 7.2 artifacts
- A `forgeops` repository clone with a branch that contains 7.3 artifacts

Example commands in the steps on this page assume:

- `7.2-profile` is the name of the 7.2 configuration profile.
- Your 7.2 CDM deployment is a small cluster.
- Your 7.2 CDM deployment does not 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.

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 may 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. Consider backing up directory data on volume snapshots for a simpler restore scenario.

Export the version 7.2 AM and IDM configurations

1. Locate a branch of your `forgeops` repository clone that contains version 7.2 artifacts and check out the branch.
2. (Optional) Check out a new branch based on the branch that contains version 7.2 artifacts.
3. Locate a namespace running version 7.2 of the CDK that contains your current AM and IDM configurations.
4. If you've never exported the AM and IDM configurations on this system, initialize directories where the configuration profiles will be exported:

   ```
   $ cd /path/to/forgeops/docker/am/config-profiles
   $ cp -r cdk 7.2-profile
   $ cd /path/to/forgeops/docker/idm/config-profiles
   $ mkdir -p 7.2-profile/conf
   ```

   This step is not necessary for the latest 7.2 branches of the `forgeops` repository, but performing it causes no ill effect.
5. Export the AM and IDM configurations from the running 7.2 CDK deployment:

   ```
   $ cd /path/to/forgeops
   $ ./bin/config export am 7.2-profile --sort
   $ ./bin/config export idm 7.2-profile --sort
   ```
6. Run the git add . and git commit commands.

Upgrade the exported configuration profiles to version 7.3

1. Locate the branch of your `forgeops` repository clone that contains version 7.3 artifacts and check out the branch.

   The latest branch with 7.3 artifacts is the `release/7.3-20240131` branch.
2. (Optional) Check out a new branch based on the branch that contains version 7.3 artifacts.
3. Copy the configuration profiles you exported from your 7.2 CDK into the 7.3 branch:
   - Copy the AM 7.2 configuration profile into the /path/to/forgeops/docker/am/config-profiles directory.
   - Copy the IDM 7.2 configuration profile into the /path/to/forgeops/docker/idm/config-profiles directory.
4. Upgrade the AM configuration in the 7.3 branch.

   Run the am-config-upgrader utility:

   ```
   $ cd /path/to/forgeops
   $ ./bin/am-config-upgrader docker/am/config-profiles/7.2-profile
   ```

5. Upgrade the IDM configuration in the 7.3 branch.

   Follow the steps in Migrate your configuration in the IDM documentation.

6. Run the git add . and git commit commands.

Upgrade the 7.2 pods to 7.3 and build custom 7.3 Docker images

1. Set your Kubernetes context so that you can access the cluster on which you deployed the version 7.2 CDM.
2. Check out the branch of your `forgeops` repository clone that contains version 7.3 artifacts.

   This is important! If you've checked out a branch that contains version 7.2 artifacts, the forgeops install command reinstalls version 7.2 instead of upgrading your pods to version 7.3.

3. Upgrade the `ds-cts` pods from 7.2 to 7.3:

   ```
   $ cd /path/to/forgeops
   $ ./bin/forgeops install ds-cts --small
   ```

   This command updates one `ds-cts` pod at a time. Run the `kubectl get pods --watch` command to observe the pod upgrades.

   After all the `ds-cts` pods have been upgraded, run the ds-debug.sh command to verify that directory replication is working correctly. Run commands similar to the following for each `ds-cts` pod:

   ```
   $ ./bin/ds-debug.sh rstatus podname
   ```

4. Upgrade the `ds-idrepo` pods from 7.2 to 7.3:

   ```
   $ cd /path/to/forgeops
   $ ./bin/forgeops install ds-idrepo --small
   ```

   This command updates one `ds-idrepo` pod at a time. Run the `kubectl get pods --watch` command to observe the pod upgrades.

   After all the `ds-idrepo` pods have been upgraded, run the ds-debug.sh command to verify that directory replication is working correctly. Run commands similar to the following for each `ds-idrepo` pod:

   ```
   $ ./bin/ds-debug.sh rstatus podname
   ```

5. Check out the branch of your `forgeops` repository clone that contains version 7.3 artifacts.

   This branch should contain the `7.2-profile` configuration profile you upgraded to work with version 7.3.

6. Build Docker images for version 7.3 that contain the `7.2-profile` configuration profile:

```
$ cd /path/to/forgeops
$ ./bin/forgeops build am --config-profile 7.2-profile --push-to my-repo
$ ./bin/forgeops build idm --config-profile 7.2-profile --push-to my-repo
```

The newly-built Docker images are based on ForgeRock's evaluation-only Docker images.

7. Upgrade the Ping Identity Platform pods from 7.2 to 7.3:

```
$ ./bin/forgeops install ui --small
$ ./bin/forgeops install am --small
$ ./bin/forgeops install idm --small
```

Wait for the pod upgrades to complete. Run the `kubectl get pods --watch` command to observe the pod upgrades.

In version 7.2, the default number of AM pods in a small configuration is three. In version 7.3, that's been changed to two. If you use the default small size cluster configuration, you'll have one fewer AM pod after completing the AM upgrade.

8. Start the AM and IDM admin UIs in your upgraded CDM deployment. Verify that:
    - The start page for each admin UI indicates that the component version is 7.3, not 7.2.
    - AM and IDM use your custom configuration.
9. If you are using a Kubernetes-based Ping Identity Platform deployment in production, you must rebuild base Docker images for version 7.3, and then build custom Docker images based on those images:
    1. Build your own Docker base images. Refer to Your own base Docker images for more information.
    2. Rebuild your custom Docker images, basing them on the images you built in the previous step. Refer to Create Docker images for use in production for more information.

Upgrade the platform to a newer 7.3 patch release

If you've installed version 7.3 of the Ping Identity Platform using artifacts from the `forgeops` repository, follow the steps provided on this page to upgrade to a new patch release of Ping Identity Platform 7.3.

Use these steps to upgrade the platform *in place, with no downtime*.

This upgrade methodology has been tested against a deployment based on ForgeRock's evaluation-only Docker images with basic configuration settings.

> **Important**
>
> Because the Ping Identity Platform is highly customizable, it is difficult for ForgeRock 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

To upgrade the platform to a newer patch release, you'll need:

- A running version 7.3 CDM deployment
- A `forgeops` repository clone with a branch that contains the artifacts for the newer patch release

Example commands in the steps on this page assume:

- Your 7.3 CDM deployment is a small cluster.
- Your 7.3 CDM deployment does not include PingGateway.

When you perform the upgrade:

- Specify a different cluster size, if applicable.
- Add commands to upgrade PingGateway, if applicable.

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 may 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.

Upgrade the CDM to the new patch release

1. If you have AM or IDM configuration changes that you haven't already exported to a configuration profile:
    1. Locate a branch of your `forgeops` repository clone that contains version 7.3 artifacts and check out the branch.
    2. Locate the namespace running version 7.3 of the CDK that contains the AM and IDM configuration changes.
    3. Export the AM and IDM configurations from the running 7.3 CDK deployment:

    ```
    $ cd /path/to/forgeops
    $ ./bin/config export am my-config-profile --sort
    $ ./bin/config export idm my-config-profile --sort
    ```

    4. Run the am-config-upgrader utility to upgrade the AM configuration:

    ```
    $ cd /path/to/forgeops
    $ ./bin/am-config-upgrader docker/am/config-profiles/my-config-profile
    ```

2. Run the git add . and git commit commands.
3. Set your Kubernetes context so that you can access the cluster on which the CDM is deployed.
4. Upgrade the `ds-cts` pods to the new patch release:

    ```
    $ cd /path/to/forgeops
    $ ./bin/forgeops install ds-cts --small
    ```

    This command updates one `ds-cts` pod at a time. Run the `kubectl get pods --watch` command to observe the pod upgrades.

    After all the `ds-cts` pods have been upgraded, run the ds-debug.sh command to verify that directory replication is working correctly. Run commands similar to the following for each `ds-cts` pod:

    ```
    $ ./bin/ds-debug.sh rstatus podname
    ```

5. Upgrade the `ds-idrepo` pods to the new patch release:

    ```
    $ cd /path/to/forgeops
    $ ./bin/forgeops install ds-idrepo --small
    ```

    This command updates one `ds-idrepo` pod at a time. Run the `kubectl get pods --watch` command to observe the pod upgrades.

    After all the `ds-idrepo` pods have been upgraded, run the ds-debug.sh command to verify that directory replication is working correctly. Run commands similar to the following for each `ds-idrepo` pod:

```
$ ./bin/ds-debug.sh rstatus podname
```

6. 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 --push-to my-repo
$ ./bin/forgeops build idm --config-profile my-config-profile --push-to my-repo
```

   The newly-built Docker images are based on ForgeRock's evaluation-only Docker images.

7. Upgrade the Ping Identity Platform pods to the new patch release:

```
$ ./bin/forgeops install ui --small
$ ./bin/forgeops install am --small
$ ./bin/forgeops install idm --small
```

   Wait for the pod upgrades to complete. Run the `kubectl get pods --watch` command to observe the pod upgrades.

8. Start the AM and IDM admin UIs in your upgraded CDM deployment. Verify that:
   - The start page for each admin UI displays the expected component version for the newer patch release.
   - AM and IDM use your custom configuration.

9. If you are using a Kubernetes-based Ping Identity Platform deployment in production, you must rebuild Docker images based on the newer patch release, and then build custom Docker images based on those images:
   1. Build your own Docker base images. Refer to Your own base Docker images for more information.
   2. Rebuild your custom Docker images, and base them on your new base Docker images. Refer to Create Docker images for use in production for more information.

forgeops command">

forgeopscommand

forgeops — Manage Ping Identity Platform components in a Kubernetes cluster

Synopsis

forgeops subcommand options

Description

- 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 | -h`

Display command usage information.

Subcommands

forgeops build

forgeops build components options

Build a custom Docker image for one or more Ping Identity Platform components, and update the image defaulter file.

For components, specify:

- `am`, `ds`, `idm`, or `ig`, to build a custom Docker image for a single Ping Identity Platform component.[1]
- More than one component, to build multiple Docker images by running a single forgeops build command. Separate multiple components with a space. For example, forgeops build am idm.
- `all`, to build Docker images for all the Ping Identity Platform components[2] by running a single forgeops build command.

Options

In addition to the global forgeops command options, the forgeops build subcommand takes the following options:

`--config-profile | -p configuration profile path`

Path that contains the configuration for `am`, `idm`, or `ig`. The forgeops build command incorporates the configuration files located at 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

For more information, refer to Configuration profiles.

The default value for the --config-profile option is cdk:

- The docker/ig/config-profiles/cdk directory contains a starter configuration that you can use when you begin customizing the `ig` Docker image.
- The docker/am/config-profiles/cdk and docker/idm/config-profiles/cdk directories are intentionally empty. The base images for the customized `am` and `idm` Docker images already contain starter configurations, so a starter configuration in a configuration profile is not needed.

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.

`--deploy-env environment`

The deployment environment.

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.

You must initialize new deployment environments before using them for the first time. Refer to Initialize deployment environments.

`--push-to | -r registry`

Docker registry to which the Docker image being built is pushed. Required 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` takes precedence.

`--reset`

Revert all the tags and new image names in the image defaulter file to their original values.

`--tag | -t tag`

Tag to apply to the Docker image being built.

forgeops clean

forgeops clean

Remove Kustomize manifests for a Ping Identity Platform deployment from a `forgeops` repository clone.

The forgeops clean command removes Kustomize manifests from:

- The kustomize/deploy directory, if you do not specify the `--deploy-env` option when you run the command.
- The kustomize/deploy-environment directory, if you specify the `--deploy-env` option when you run the command.

Options

In addition to the global forgeops command options, the forgeops clean subcommand takes the following option:

`--deploy-env environment`

Deployment environment to remove.

Specify this option if you specified a deployment environment when you ran the forgeops install or forgeops generate command. Note that by default, these two commands generate Kustomize manifests in the kustomize/deploy directory, but when you run them with the `--deploy-env` option, they generate the manifests in the kustomize/deploy-environment directory.

forgeops delete

forgeops delete components options

Delete Ping Identity Platform components or sets of components, PVCs, volume snapshots, and Kubernetes secrets from a running deployment.

By default, the forgeops delete command prompts you to verify whether you want to delete Ping Identity Platform components, PVCs, volume snapshots, and Kubernetes secrets. You can modify the default behavior to suppress confirmation prompts as necessary.

For components, specify:

- `admin-ui`, `am`, `amster`, `ds-cts`, `ds-idrepo`, `end-user-ui`, `idm`, `ig`, or `login-ui`, to delete a single Ping Identity Platform component.
- `secrets`, to delete the Kubernetes secrets from the deployment.
- A named set of components:
  - `apps`, to delete the `am`, `amster`, `idm`, and `ig` components.
  - `base`, to delete the `dev-utils` and `platform-config` configmaps, Kubernetes ingress resources, and Kubernetes secrets. Secrets generated by the ds-operator and cert-manager are not deleted.
  - `ds`, to delete all the DS components.

    - `ui`, to delete the `admin-ui`, `end-user-ui`, and `login-ui` components.
- `all`, to delete all the Ping Identity Platform components.
- More than one component or set of components, to delete multiple Ping Identity Platform components by running a single forgeops delete command. Separate multiple components with a space. For example, forgeops delete ui am.

The default value for components is `all`.

Options

In addition to the global forgeops command options, the forgeops delete subcommand takes the following options:

`--debug`

Display debug information when executing the command.

`--deploy-env environment`

The deployment environment that was specified when you installed components.

If you specified a deployment environment when you installed platform components, you must specify the same deployment environment when deleting the components.

`--force | -f`

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 | -n namespace`

The namespace from which to delete Ping Identity Platform components.

Defaults to the active namespace in your local Kubernetes context.

`--yes | -y`

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.

forgeops generate

forgeops generate components options

Generate Kustomize manifests for a Ping Identity Platform deployment.

By default, the forgeops generate command places manifests in the kustomize/deploy directory. You can alter this location by specifying a deployment environment.

The forgeops generate and forgeops install commands are similar, except that the forgeops generate command does not deploy Ping Identity Platform components after generating Kustomize manifests. If you generate manifests for Ping Identity Platform components by running the forgeops generate command, you can then deploy them by running kubectl apply -k commands. For more information, refer to the CDK and CDM deployment documentation.

For components, specify:

- `admin-ui`, `am`, `amster`, `ds-cts`, `ds-idrepo`, `end-user-ui`, `idm`, `ig`, or `login-ui`, to generate a manifest for a single Ping Identity Platform component.
- `secrets`, to generate a manifest for Kubernetes secrets.
- A named set of components:
  - `apps`, to generate a manifest for the `am`, `amster`, `idm`, and `ig` components.
  - `base`, to generate a manifest for the `dev-utils` and `platform-config` configmaps, Kubernetes ingress resources, and another manifest for Kubernetes secrets.
  - `ds`, to generate a manifest for all the DS components.
  - `ui`, to generate a manifest for the `admin-ui`, `end-user-ui`, and `login-ui` components.
- `all`, to generate manifests for all the Ping Identity Platform components.
- More than one component or set of components, to generate manifests for multiple Ping Identity Platform components by running a single forgeops generate command. Separate multiple components with a space. For example, forgeops generate ui am.

The default value for components is `all`.

Options

In addition to the global forgeops command options, the forgeops generate subcommand takes the following options:

`--cdk | --custom overlay path | --large | --medium | --mini | --small`

Deployment size. References a Kustomize overlay that contains YAML patch files that alter the behavior of the related base Kustomize files. Kustomize overlays provided by ForgeRock reside in the kustomize/overlay directory. Base Kustomize files reside in the kustomize/base directory.

If none of these options are specified, the deployment size option defaults to `--cdk`.

Refer to CDK architecture and CDM architecture for information about deployment sizing and contents options provided with the CDK and the CDM.

About the `--custom` option:

Specify the `--custom` option if you want to provide your own overlay that specifies Kubernetes deployment environment characteristics rather than using one of the deployment sizes provided by ForgeRock. For overlay path, specify the full path where the patch files are located.

The names of the patch files residing in overlay path must align with the names expected by the forgeops generate command:

- am.yaml for the `am`, `apps`, and `all` components

- idm.yaml for the `idm`, `apps`, and `all` components
- ig.yaml for the `ig` and `all` components
- ingress.yaml and/or secret_agent_config.yaml for the `base` and `all` components

`--debug`

Display debug information when executing the command.

`--deploy-env environment`

The deployment environment.

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

By default, the forgeops generate command generates Kustomize manifests in the kustomize/deploy directory.

When you specify a deployment environment, the forgeops generate command generates the manifests in the kustomize/deploy-environment directory. For example, if you ran forgeops generate --deploy-env production, Kustomize manifests would be placed in the kustomize/deploy-production directory.

You must initialize new deployment environments before using them for the first time. Refer to Initialize deployment environments.

`--fqdn | -n fqdn`

The fully-qualified hostname to use in the deployment.

Defaults to `namespace.iam.example.com`, where namespace is the active namespace in your local Kubernetes context.

Relevant only for the forgeops generate all and forgeops generate base commands; ignored for other forgeops generate commands.

`--ingress-class | -i`

The type of ingress controller used in the deployment.

Possible values are `nginx` and `haproxy`. The default value is `nginx`.

Relevant only for the forgeops generate all and forgeops generate base commands; ignored for other forgeops generate commands.

forgeops info

forgeops info options

Write administrative passwords and URLs for accessing Ping Identity Platform admin UIs to standard output.

Options

In addition to the global forgeops command options, the forgeops info subcommand takes the following options:

`--debug`

Display debug information when executing the command.

`--json`

Display output in JSON format.

`--namespace | -n namespace`

The namespace that contains Ping Identity Platform components.

Defaults to the active namespace in your local Kubernetes context.

forgeops install

forgeops install components options

Generate Kustomize manifests for a Ping Identity Platform deployment, and then deploy the components in a Kubernetes cluster.

By default, the forgeops install command places manifests in the kustomize/deploy directory. You can alter this location by specifying a deployment environment.

The forgeops generate and forgeops install commands are similar, except that the forgeops generate command does not deploy Ping Identity Platform components.

For components, specify:

- `admin-ui`, `am`, `amster`, `ds-cts`, `ds-idrepo`, `end-user-ui`, `idm`, `ig`, or `login-ui`, to deploy a single Ping Identity Platform component.
- `secrets`, to deploy Kubernetes secrets. Secrets generated by the ds-operator and cert-manager are not deployed.
- A named set of components:
  - `apps`, to deploy the `am`, `amster`, `idm`, and `ig` components.
  - `base`, to deploy the `dev-utils` and `platform-config` configmaps, Kubernetes ingress resources, and Kubernetes secrets. Secrets generated by the ds-operator and cert-manager are not deployed.
  - `ds`, to deploy all the DS components.
  - `ui`, to deploy the `admin-ui`, `end-user-ui`, and `login-ui` components.
- `all`, to deploy all the Ping Identity Platform components.
- More than one component or set of components, to deploy multiple Ping Identity Platform components by running a single forgeops install command. Separate multiple components with a space. For example, forgeops install ui am.

The default value for components is `all`.

Options

In addition to the global forgeops command options, the forgeops install subcommand takes the following options:

`--amster-retain | -a seconds`

Amount of time, in seconds, to leave the Amster pod up and running after the Amster job to restore dynamic configuration finishes.

Specify either a number of seconds to retain the Amster pod, or `infinity` if you want the pod to run indefinitely. The default value is `10`.

`--cdk | --custom overlay path | --large | --medium | --mini | --small`

Deployment size. References a Kustomize overlay that contains YAML patch files that alter the behavior of the related base Kustomize files. Kustomize overlays provided by ForgeRock reside in the kustomize/overlay directory. Base Kustomize files reside in the kustomize/base directory.

If none of these options are specified, the deployment size option defaults to `--cdk`.

Refer to CDK architecture and CDM architecture for information about deployment sizing and contents options provided with the CDK and the CDM.

About the `--custom` option:

Specify the `--custom` option if you want to provide your own overlay that specifies Kubernetes deployment environment characteristics rather than using one of the deployment sizes provided by ForgeRock. For overlay path, specify the full path where the patch files are located.

The names of the patch files residing in overlay path must align with the names expected by the forgeops install command:

- am.yaml for the `am`, `apps`, and `all` components
- idm.yaml for the `idm`, `apps`, and `all` components
- ig.yaml for the `ig` and `all` components
- ingress.yaml and/or secret_agent_config.yaml for the `base` and `all` components

`--debug`

Display debug information when executing the command.

`--deploy-env environment`

The deployment environment.

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

By default, the forgeops install command generates Kustomize manifests in the kustomize/deploy directory and runs Docker images defined in the image defaulter in the kustomize/deploy/image-defaulter directory.

When you specify a deployment environment, the forgeops install command generates the manifests in the kustomize/deploy-environment directory. For example, if you ran forgeops generate --deploy-env production, Kustomize manifests would be placed in the kustomize/deploy-production directory.

It then runs Docker images specified in the environment's image defaulter, located in the kustomize/deploy-production/image-defaulter directory.

You must initialize new deployment environments before using them for the first time. Refer to Initialize deployment environments.

`--fqdn | -n fqdn`

The fully-qualified hostname to use in the deployment.

Defaults to `namespace.iam.example.com`, where namespace is the active namespace in your local Kubernetes context.

Relevant only for the forgeops install all and forgeops install base commands; ignored for other forgeops install commands.

`--ingress-class | -i`

The type of ingress controller used in the deployment.

Possible values are `nginx` and `haproxy`. The default value is `nginx`.

Relevant only for the forgeops install all and forgeops install base commands; ignored for other forgeops install commands.

`--timeout | -t seconds`

The maximum number of seconds to pause before terminating the forgeops install command if an intermediate process does not complete.

The default value for the --timeout option is `600`.

forgeops wait

forgeops wait component options

Wait for Ping Identity Platform components to fully start up.

Use the forgeops wait command to pause further execution until a Ping Identity Platform component is fully deployed. For example:

- When deploying components using a technique other than the forgeops install command, such as deploying Kustomize manifests by using the kubectl apply -k command.
- When deploying components in one shell while performing another operation that depends on deployment completion in another shell.

Because the forgeops install command waits for completion of component deployment before proceeding, it is generally not necessary to use the forgeops wait command when you deploy the platform by using the forgeops install command.

For component, specify:

- `am`, `amster`, `ds-cts`, `ds-idrepo`, `idm`, `ig`, to wait for a single Ping Identity Platform component to be deployed.
- A named set of components:
  - `apps`, to wait for the `am`, `amster`, `idm`, and `ig` components to be deployed.
  - `ds`, to wait for all the DS components to be deployed.

You must specify a single component or set of components as an argument to the forgeops wait command. You cannot specify multiple components, and there is no default component.

Options

In addition to the global forgeops command options, the forgeops wait subcommand takes the following options:

`--debug`

Display debug information when executing the command.

`--namespace | -n namespace`

The namespace that contains Ping Identity Platform components.

Defaults to the active namespace in your local Kubernetes context.

`--timeout | -t seconds`

The maximum number of seconds to pause before terminating the forgeops wait command.

The default value for the --timeout option is `600`.

---

1. Building a Docker image for the `amster` component is deprecated.
2. Except for the deprecated `amster` component.

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 CDK and CDM deployments:

Problem	Troubleshooting Technique
Pods in the CDK or CDM don't start up as expected.	Review pod descriptions and container logs. See 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 deployment. Install Ping Identity Platform components separately, instead of installing all the components with a single command. Staging your deployment 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.
Changes you've made to ForgeRock'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.

Problem	Troubleshooting Technique
You want to keep the `amster` pod running to diagnose AM configuration issues.	Use the amster command.
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

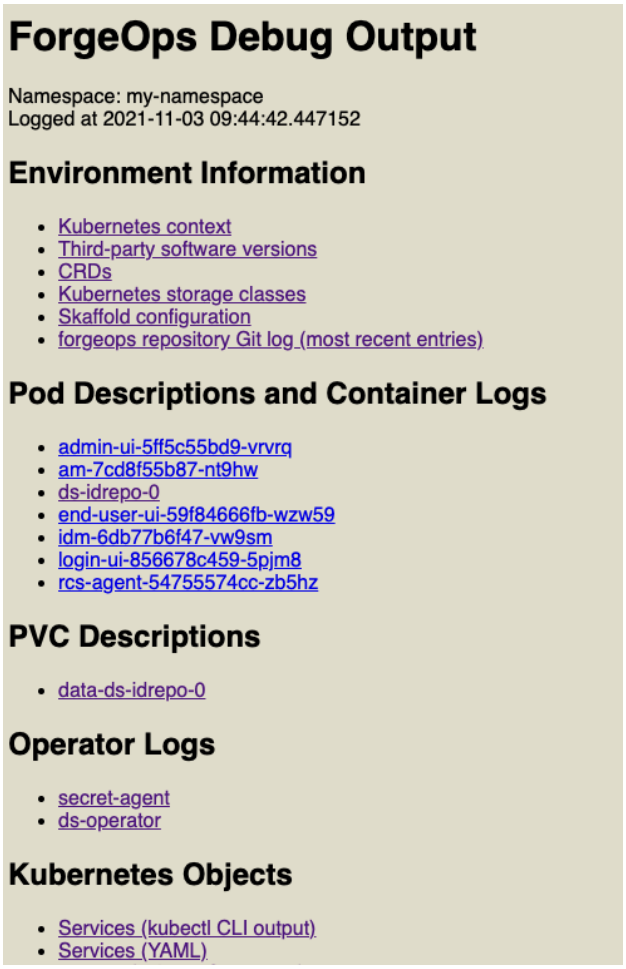
Example troubleshooting steps

Suppose you installed the CDK, but noticed that one of the CDK pods had an `ImagePullBackOff` error at startup. Here's an example of how you might use pod descriptions and container logs to troubleshoot the problem:

1. Make sure that 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 release/7.3-20240131 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 see 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 see 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 following bin/ds-debug.sh subcommands are operational:

Subcommand	Action
purge	Purges all the backups associated with a DS instance
disaster	Performs a disaster recovery operation by executing the dsrepl start-disaster-recovery -X command, and then the the dsrepl end-disaster-recovery -X command

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
```

Theamsterpod

When you deploy the CDM or the CDK, 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 amster run command to start the `amster` pod for manually interacting with AM using the amster run command line interface and perform tasks such as exporting and importing AM configuration and troubleshooting:

```
$ ./bin/amster run
starting...
Cleaning up amster components
job.batch "amster" deleted
configmap "amster-files" deleted
configmap "amster-retain" deleted
configmap/amster-files created
Deploying amster
job.batch/amster created

Waiting for amster pod to be running. This can take several minutes.
pod/amster-852fj condition met

$ 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
```

Export and import AM configuration

To export AM configuration, use the amster export command. Similarly, use the 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. To keep it running indefinitely, specify --retain infinity.

In the following example, the `amster` pod is kept running for 300 seconds after completing export:

```
$ ./bin/amster export --retain 300 /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
configmap/amster-retain 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 300 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
```

Staged CDK and CDM installation

By default, the forgeops install 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 CDK deployment
    - --small, --medium, or --large, for a CDM deployment
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 install 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 install 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 install apps command, or install them separately:

    1. Install AM:

```
$ ./forgeops install 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 install 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 install 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 install ui command, or install them separately:

    1. Install the administration UI:

    ```
    $ ./forgeops install 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 install 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 install 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 install command. For example, to install the `base`, `ds`, `am`, and `amster` components in the CDK or CDM:

```
$ ./forgeops install base ds am amster --size
```

Ingress issues

If the CDK or CDM pods 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

ForgeRock recommends installing tested versions of third-party software in environments where you'll run the CDK and the CDM.

See the tables that list the tested versions of third-party software for your deployment:

- CDK:
  - On Minikube
  - On a shared cluster:
    - On GKE
    - On EKS
    - On AKS
- CDM:
  - 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 installed the CDK or the CDM:

- Run the /path/to/forgeops/bin/debug-logs utility.
- Open the log file in your browser.
- 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 see how your changes affect deployment. Use the kustomize build command to see 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 cdk-minikube command example in Minikube cluster provides a good default virtual hardware configuration for a Minikube cluster running the CDK.

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.

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

- About the DS Dockerfile for the PingDS operator
- Deploy and customize Prometheus, Grafana, and Alertmanager in a CDM cluster
- 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

Troubleshooting

- Enable and modify the AM logging level in ForgeOps 7.3
- Enable and modify the IDM logging level in ForgeOps 7.3
- Enable and modify the audit logging level in ForgeOps 7.3

Previews

- Deploying ForgeOps to Minikube on an M1 mac with Colima

ForgeOps 7.3 release notes

Get an email when there's an update to ForgeOps 7.3. Go to the Notifications page in your Backstage profile and select ForgeOps 7.3 Changes in the Documentation Digests section.

Or subscribe to the ForgeOps 7.3 RSS feed.

Important information for this ForgeOps release:

Validated Kubernetes versions for deploying Ping Identity Platform 7.3	Link
Validated Ingress-NGINX versions for deploying Ping Identity Platform 7.3	Link
Limitations when deploying Ping Identity Platform 7.3 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
Archive of release notes prior to April 4, 2023	Link

2025

March 28, 2025

Documentation update

Updated the name of ingress controller

Updated the name of ingress controller to Ingress-NGINX controller.

2024

April 30, 2024

Documentation update

AM, IDM, and audit logging links in the Troubleshooting section

Added links to AM, IDM, and audit logging community articles in the Troubleshooting section.

February 2, 2024

Changes

New evaluation-only Docker images are now available from ForgeRock

New evaluation-only Docker images are now available for the following versions of Ping Identity Platform components:

- PingDS: 7.3.0

- PingGateway: 2023.9.0

This documentation has been updated to refer to these new versions of Docker images.

For more information about changes to the Ping Identity Platform, refer to the Release Notes for platform components at https://backstage.forgerock.com/docs.

To upgrade to the new versions, you'll need to rebuild your custom Docker images. Refer to Base Docker Images for instructions.

2023

November 15, 2023

Documentation updates

New task to initialize deployments

A new task to initialize deployment environments has been added to the instructions for developing custom Docker images using the CDK.

Before you can use a new deployment environment, you must initialize a directory that supports the environment.

Clarification about support for environments that deviate from the published CDK and CDM architecture

The Support from ForgeRock page has been updated to state that environments that deviate from the published CDK and CDM architecture are not supported. For details, refer to Support limitations.

October 13, 2023

Changes

CDM deployments now use Kubernetes version 1.27

When you create a cluster for deploying version 7.3 of the platform, use Kubernetes version 1.27.

August 10, 2023

Documentation updates

New how-to: Upgrade the platform to a newer patch release

A new how-to provides steps for upgrading to newer patch releases of version 7.3.

August 3, 2023

Changes

Running the CDK on Minikube on macOS systems with ARM-based chipsets is now available on an experimental basis

Running the CDK on Minikube on macOS systems with ARM-based chipsets, such as the Apple M1 or M2, is now available on an experimental basis.

Refer to this ForgeRock Community article for details.

July 11, 2023

Highlights

Updates to the `forgeops` repository for Ping Identity Platform version 7.3

Updates for Ping Identity Platform version 7.3 are available in the `release/7.3-20240131` branch of the `forgeops` repository.

The `release/7.3-20240131` branch replaces the `release/7.3-20230609` branch. Upgrade to the new branch as soon as possible.

New evaluation-only Docker images are now available from ForgeRock

New evaluation-only Docker image versions are now available for the following Ping Identity Platform components:

- PingDS: 7.3.0
- PingGateway: 2023.9.0

This documentation has been updated to refer to these new versions of Docker images.

For more information about changes to the Ping Identity Platform, refer to the Release Notes for platform components at https://backstage.forgerock.com/docs.

To upgrade to the new versions, you'll need to rebuild your custom Docker images. Refer to Base Docker Images for instructions.

June 23, 2023

Documentation updates

Updates to the Base Docker images page

New steps describe how to build Docker images for Java, and how to base your own base Docker images on those Java images.

June 16, 2023

Highlights

Updates to the `forgeops` repository for Ping Identity Platform version 7.3

Updates for Ping Identity Platform version 7.3 are available in the `release/7.3-20240131` branch of the `forgeops` repository.

The `release/7.3-20240131` branch replaces the `release/7.3-20230404` branch. Upgrade to the new branch as soon as possible.

New evaluation-only Docker images are now available from ForgeRock

New evaluation-only Docker image versions are now available for the following Ping Identity Platform components:

- PingDS: 7.3.0
- PingGateway: 2023.9.0

This documentation has been updated to refer to these new versions of Docker images.

For more information about changes to the Ping Identity Platform, refer to the Release Notes for platform components at https://backstage.forgerock.com/docs.

To upgrade to the new versions, you'll need to rebuild your custom Docker images. Refer to Base Docker Images for instructions.

June 15, 2023

Changes

Base Docker image for DS updated to 7.3.1

The evaluation-only base image for DS version 7.3 was updated from 7.3.0 to 7.3.0.

June 5, 2023

Documentation updates

New how-to: Upgrade the platform from version 7.2 to 7.3

A new how-to provides steps for upgrading a version 7.2 CDM to version 7.3.

April 4, 2023

Highlights

Terraform for CDM cluster creation and deletion

Use Terraform to create clusters in which you can install the CDM. Terraform artifacts are now available in the top-level terraform directory of the new `forgeops-extras` repository. Install Terraform software before you install the CDM to take advantage of this new capability.

The cluster-up.sh and cluster-down.sh scripts are no longer available. Use Terraform for cluster creation and deletion instead.

You'll find changes on the following pages in the documentation:

- Terraform directory in the `forgeops-extras` repository
- Google Cloud project setup
- GKE cluster creation
- Setup for AWS
- EKS cluster creation
- Azure subscription setup
- AKS cluster creation
- CDM deployment
- CDM removal
- Ingress how-to

Deployment environments

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

Specify a deployment environment by using the forgeops command's new `--deploy-env` option.

By default, the image defaulter and generated Kustomize manifests reside in the kustomize/deploy directory.

Each deployment environment has its own image defaulter, located in the kustomize/deploy-environment/image-defaulter directory.

When you specify a deployment environment, Kustomize manifests are generated in the kustomize/deploy-environment directory. For example, if you ran forgeops generate --deploy-env production, Kustomize manifests would be placed in the kustomize/deploy-production directory.

HAProxy Ingress as the CDM ingress controller

You can now install HAProxy Ingress as the ingress controller for CDM deployments.

Ingress-NGINX Controller remains the default ingress for controller for CDK and CDM deployments.

New forgeops command reference

A reference for the forgeops command is now available here.

Changes

CDM deployments now use Kubernetes version 1.25

When you create a cluster for deploying version 7.3 of the platform, use Kubernetes version 1.25.

CDK deployments on Minikube now use the stable Kubernetes version

The cdk-minikube script has been modified to use the stable Kubernetes version instead of version 1.23.3. Refer to the Minikube start command reference for details about which Kubernetes version is currently considered to be the stable version.

CDM deployments should now use Ingress-NGINX Controller version 1.4.0 or higher

When you deploy the NGINX Ingress Controller in your CDM cluster, use version 1.4.0[1] or higher.

Additional documented DS limitations in CDK and CDM deployments

Three additional limitations on DS in CDK and CDM deployments are now documented here:

- Database encryption is not supported
- DS starts successfully even when it cannot decrypt a backend
- Root file system write access is required to run the DS Docker image

Please note that these are not new limitations. They had inadvertently been omitted from the DS limitations section in the documentation.

Large CDM deployments now run in a single node pool

Large CDM deployments are now configured to run in a single node pool.

The CDM architecture previously used two node pools: one for the DS pods, and another for all the other pods in the CDM deployment.

Automatic configuration profile creation

The config export am and config export idm commands now have added functionality to create new configuration profiles.

You are no longer required to create and populate subdirectories under the /path/to/forgeops/docker/am/config-profiles and /path/to/forgeops/docker/idm/config-profiles directories the first time you export configuration from the CDK to a new configuration profile in your `forgeops` repository clone.

Availability and usage of `KUBECONFIG` environment variables

The tf-apply script creates a kubeconfig file when it creates a cluster. The documentation has been modified to support changing the Kubernetes context using `KUBECONFIG` environment variables instead of assuming you use the default kubeconfig file, $HOME/.kube/config.

CDM deployment no longer defaults to the `prod` namespace

You can now use any namespace you want for CDM deployment.

Previously, cluster creation scripts created the `prod` namespace, and some scripts defaulted to using this namespace for CDM deployment.

Skaffold is no longer used to build Docker images

The forgeops build command now uses Docker rather than Skaffold to build and push Docker images.

Because of this, you no longer need to install Skaffold software when deploying the CDM or the CDK.

New --push-to option replaces the forgeops build command's --default-repo option

The forgeops build command's --push-to option replaces the --default-repo option.

When running the forgeops build command on Minikube, you must now specify --push-to none with the forgeops build command to push a Docker image to the Docker registry embedded in the Minikube cluster. Previously, it was not necessary to specify the --default-repo option when running the forgeops build command on Minikube.

The forgeops delete command issues new confirmation prompts and has a new option

The forgeops delete command now issues multiple confirmation prompts, letting you choose to delete all PVCs, volume snapshots, and/or secrets from a CDK or CDM deployment.

Previously, you could only choose to delete all three deployment artifacts, or none of them.

The forgeops delete command's new --force --yes option lets you suppress all confirmation prompts.

eksctl is no longer used to create EKS clusters

The tf-apply command uses Terraform rather than eksctl to create EKS clusters.

Because of this, you no longer need to install eksctl software when deploying the CDM.

AM evaluation-only Docker image repository name change

The name of the AM evaluation-only Docker image repository has been changed to `gcr.io/forgerock-io/am-cdk`. This image repository was formerly named `gcr.io/forgerock-io/am-base`.

The AM canonical configuration is now built into the `am-cdk` Docker image

The AM canonical configuration for the CDK has been incorporated into the `am-cdk` Docker image.

Because of this, you no longer need to copy files from the docker/am/config-profiles/cdk directory when you initialize a new configuration profile. Simply create a new subdirectory under the docker/am/config-profiles directory.

Deprecated

ForgeOps artifacts for deploying Ping Identity Platform 7.2

The ForgeOps artifacts for deploying Ping Identity Platform 7.2 are deprecated. You should migrate to version 7.3 as soon as you're able to.

Removed

The cluster-up and cluster-down scripts

The cluster-up and cluster-down scripts are no longer available. Use Terraform for cluster creation and deletion instead.

The forgeops build command's --default-repo option

The forgeops build command's --default-repo option is no longer available. It's been replaced by the new --push-to option.

The cicd directory

The cicd directory has been removed from the `forgeops` repository.

Documentation updates

New deployment step: back up the secrets that contain the DS master and TLS keys

A new step to back up the Kubernetes secrets that contain the DS master and TLS keys has been added to the instructions for deploying the CDM.

It is extremely important to back up these secrets and retain them in a secure location. Loss of these secrets could result in the inability to restore data from backups.

Secret generation documentation corrected

The Secret Agent operator page previously stated that the Secret Agent operator generates all secrets required for a Ping Identity Platform deployment.

This page has been corrected to state that the Secret Agent operator generates all secrets required for a Ping Identity Platform deployment except for the DS master and TLS keys. In version 7.3, the DS operator calls the certificate manager to generate these two keys.

Secret management recommendations changed

The recommendation that you always configure cloud secret management has been relaxed. ForgeRock now recommends that you configure cloud secret management only when you have multiple deployments that need to use the same secrets.

Base Docker images page updated

The Base Docker images page has been significantly updated. A new section, Create Docker images for use in production, explains how to build customized Docker images for the Ping Identity Platform that:

- Contain customized configuration profiles for AM, IDM, and, optionally, PingGateway.
- Must be based on your own base Docker images.
- Must *not* be based on ForgeRock's evaluation-only Docker images.

1. Ingress-NGINX Controller Helm chart version 4.3.0 installs NGINX Ingress Controller version 1.4.0.

Glossary

affinity (AM)

AM affinity deployment lets AM spread the LDAP reqests load over multiple directory server instances. Once a CTS token is created and assigned to a session, AM sends all subsequent 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

Cloud Developer's Kit (CDK)

The developer artifacts in the `forgeops` Git repository, together with the Ping Identity Platform documentation, form the Cloud Developer's Kit (CDK). Use the CDK to set up the platform in your developer environment.

Source: About the Cloud Developer's Kit

Cloud Deployment Model (CDM)

The Cloud Deployment Model (CDM) is a common use Ping Identity Platform architecture, designed to be easy to deploy and easy to replicate. The ForgeOps Team has developed Kustomize bases and overlays, Docker images, and other artifacts expressly to build the CDM.

Source: About the Cloud Deployment Model

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. AWS 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. The lifecycle of the control plane is managed by GKE 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 manage 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 whitelist 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 lets a Kubernetes cluster to automatically scale 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 athe 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 instances of virtual machines. 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 the machine type, image, zone, labels, and other instance properties. 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 is a process that 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 web site

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 a 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 of 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 which 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 begins with A-M, and the second contains all users whose names begins with N-Z. Both have the same naming context.

Source: Class Partition in the DS Javadoc

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. All the resources in a stack are defined by the AWS template.

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. All the resources included in each stack of a stack set are defined by the same template.

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 backup 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

Legal notices

Click here for legal information about product documentation published by ForgeRock.

About ForgeRock Identity Platform software

The ForgeRock® Identity Platform serves as the basis for our simple and comprehensive identity and access management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

The platform includes the following components:

- ForgeRock® Access Management (AM)
- ForgeRock® Identity Management (IDM)
- ForgeRock® Directory Services (DS)
- ForgeRock® Identity Gateway (IG)

FontAwesome copyright