ForgeOps Documentation

ON THIS PAGE

ForgeRock provides a number of resources to help you get started in the cloud. These resources demonstrate how to deploy the ForgeRock Identity Platform on Kubernetes.

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.

Start Here

ForgeRock provides several resources to help you get started in the cloud. These resources demonstrate how to deploy the ForgeRock 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 ForgeRock Identity Platform software, see Support From ForgeRock.

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 GUI consoles 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 ForgeRock 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

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

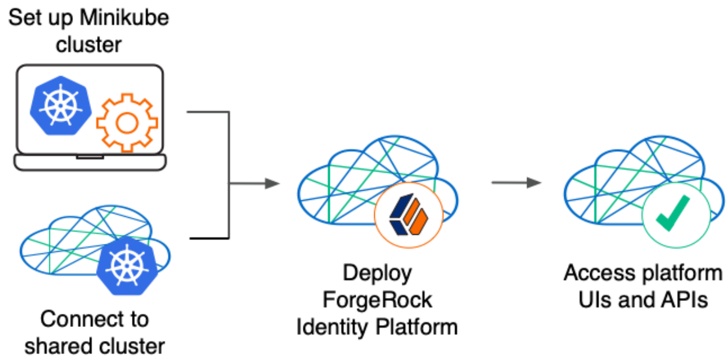
ForgeRock's DevOps documentation helps you deploy the CDK and CDM:

- **Cloud Developer's Kit Documentation**. Tells you how to install the CDK, modify the AM and IDM configurations, and create customized Docker images for the ForgeRock Identity Platform.
- **Cloud Deployment Model Documentation**. Tells you how to quickly create a Kubernetes cluster on Google Cloud, Amazon Web Services (AWS), or Microsoft Azure, install the ForgeRock 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 7.1 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 ForgeRock 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 ForgeRock 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.

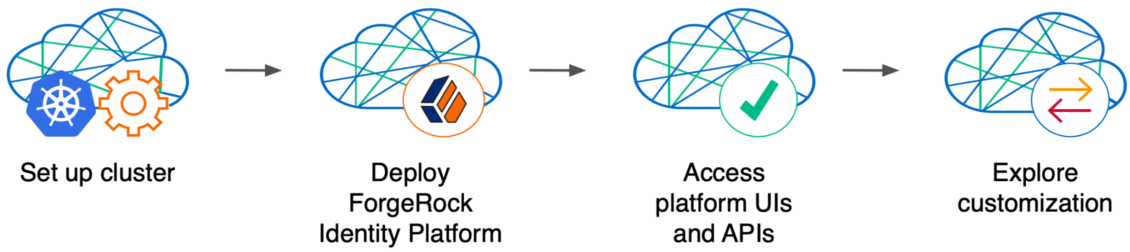
Prerequisite technologies and skills:

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

More information:

- Cloud Developer's Kit 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 NGINX ingress controller
- Resizing the cluster to meet your business requirements
- Configuring Alert Manager to issue alerts when usage thresholds have been reached

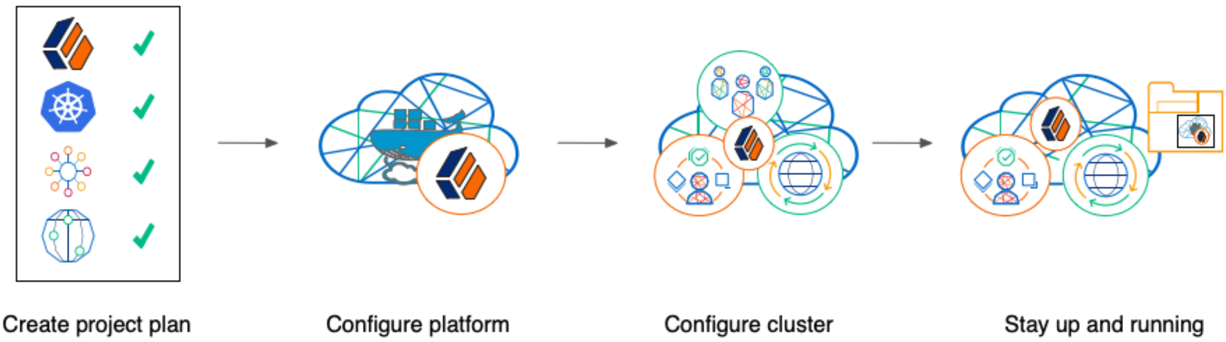
Prerequisite technologies and skills:

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

More information:

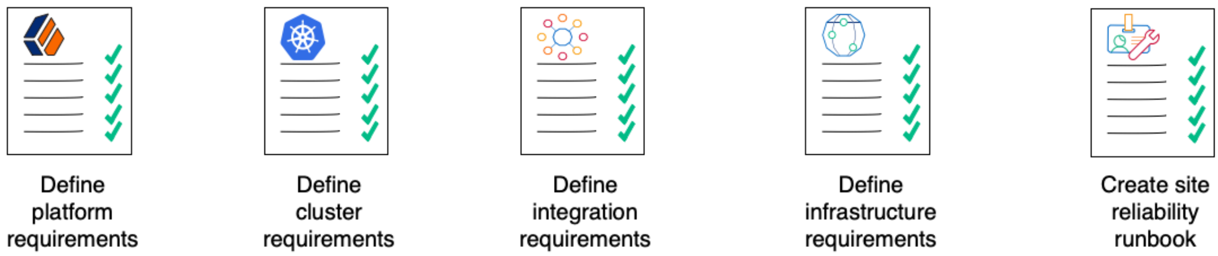
- Cloud Deployment Model Documentation

Build Your Own Service



Perform the following activities to customize, deploy, and maintain a production ForgeRock 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 ForgeRock 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.

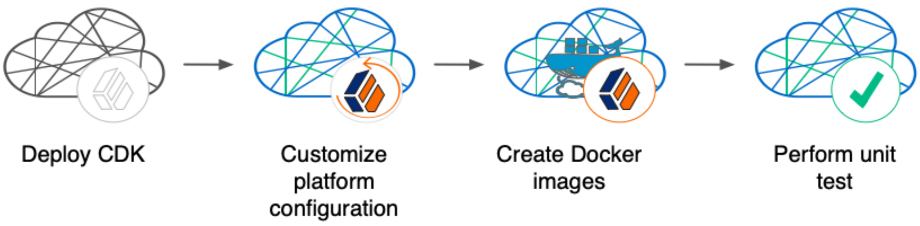
Prerequisite technologies and skills:

- Project planning and management
- Git
- Docker
- Google Cloud, AWS, or Azure
- Kubernetes, running on Google Cloud, AWS, or Azure
- ForgeRock Identity Platform
- Applications and databases that you plan to integrate with ForgeRock 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 ForgeRock 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:

- ForgeRock Identity Platform
- Git

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

More information:

- Cloud Developer's Kit 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.

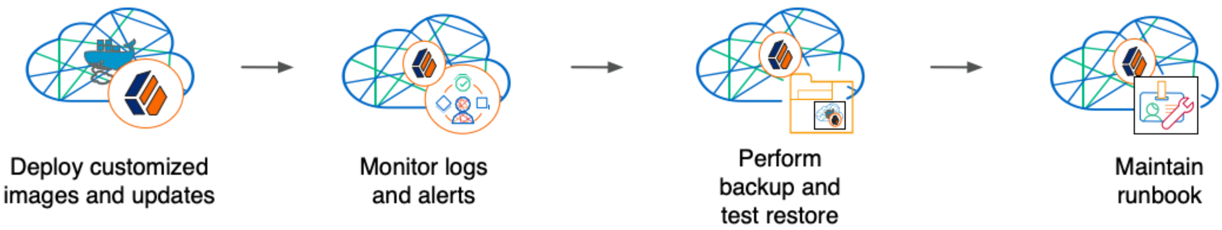
Prerequisite technologies and skills:

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

- How-Tos
- Cloud Deployment Model 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 ForgeRock 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
- ForgeRock Identity Platform
- CI/CD for a production deployment in the cloud
- 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 ForgeRock Identity Platform components to cloud databases.

ForgeRock Identity Platform

I have:

- Attended ForgeRock University training courses.
- Deployed the ForgeRock 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.
- Passed the Cloud Native Certified Kubernetes Administrator exam (highly recommended).

Project Planning and Management for Cloud Deployments

I have planned and managed:

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

Security and Hardening for Cloud Deployments

I can:

- Harden a ForgeRock 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.

About the `forgeops` Repository

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

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

- Files used to build Docker images for the ForgeRock Identity Platform:
  - Dockerfiles
  - Scripts and configuration files incorporated into ForgeRock's Docker images
  - Canonical configuration profiles for the platform
- Kustomize bases and overlays
- Skaffold configuration files

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

- Deploying ForgeRock's 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/7.1-20240223` 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/7.1-20240223` branch and create a working branch. For example:

```
$ git checkout release/7.1-20240223
$ git checkout -b my-working-branch
```

ForgeRock recommends that you regularly incorporate updates to the `release/7.1-20240223` into your working branch:

1. Review the Release Notes from time to time—they provide information about updates.
2. Pull new commits in the `release/7.1-20240223` branch into your clone's `release/7.1-20240223` 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/7.1-20240223` branch.

Repository Reference

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.

`cicd`

Example files for working with Google Cloud Build CI/CD.

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

Support Status: Sample files. Not supported by ForgeRock.

`cluster`

Example scripts and artifacts that automate cluster creation.

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

Support Status: Sample files. Not supported by ForgeRock.

`config`

Configuration profiles, including the canonical `cdk` profile from ForgeRock and user-customized profiles.

Recommendation: Add your own profiles to this directory using the **`config.sh`** command. Do not modify the canonical `cdk` profile.

Support Status: Configuration profiles:

- Support is available from ForgeRock for the canonical `cdk` configuration profile.
- Not supported by ForgeRock:
  - The `am-only`, `ds-only`, `idm-only`, and `ig-only` profiles.
  - Customized configuration profiles you've added to the `config` directory.

`docker`

Dockerfiles and other support files needed to build Docker images for the ForgeRock Identity Platform.

Recommendation: When customizing ForgeRock's default deployments, you'll need to add files under the `docker/7.0` directory. For example, to customize the AM WAR file, you might need to add plugin JAR files, user interface customization files, or image files.

If you only add new files under the `docker/7.0` directory, you should not encounter merge conflicts when you rebase changes from a new release tag 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: Dockerfiles and other files needed to build Docker images for the ForgeRock Identity Platform. 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 ForgeRock 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/7.0` directory. You'll probably change, at minimum, the `kustomize/overlay/7.0/all/kustomization.yaml` file.

Expect to encounter merge conflicts when you rebase changes from a new release tag 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 ForgeRock 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.

`cloudbuild.yaml`

Example files for working with Google Cloud Build.

Recommendation: Don't modify this file. If you want to add your own Cloud Build configuration to the `forgeops` repository, use a different file name.

Support Status: Sample file. Not supported by ForgeRock.

`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.

skaffold.yaml

The declarative configuration for running Skaffold to deploy the ForgeRock Identity Platform.

Recommendation: If you need to customize the `skaffold.yaml` file, you might encounter merge conflicts when you rebase changes from a new release tag into your branches. Be sure to track changes you've made to this file, so that you're prepared to resolve merge conflicts after a rebase.

Support Status:

`small`, `medium`, `large`, `am`, `amster`, `idm`, `ds-cts`, `ds-idrepo`, and `ig` profiles: Support is available from ForgeRock.

All other profiles, including the `am-only`, `am-idm-only`, `ds-only`, `idm-only`, `ig-only`, `no-ui`, and `persistence` profiles are for ForgeRock internal use only. Support is not available from 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` 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 Cloud Developer's Kit Documentation and the Cloud Deployment Model 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.

Cloud Developer's Kit Documentation

The CDK is a minimal sample deployment of the ForgeRock 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

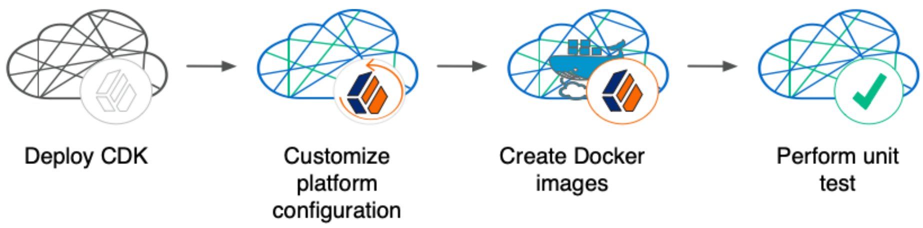
- ❑ Become familiar with the CDK
- ❑ Understand CDK architecture
- ❑ Set up your local environment
- ❑ Deploy the platform
- ❑ Access platform UIs and APIs
- ❑ (Optional) Develop custom Docker images

About the Cloud Developer's Kit

The CDK is a minimal sample deployment of the ForgeRock 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 ForgeRock 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 ForgeRock 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

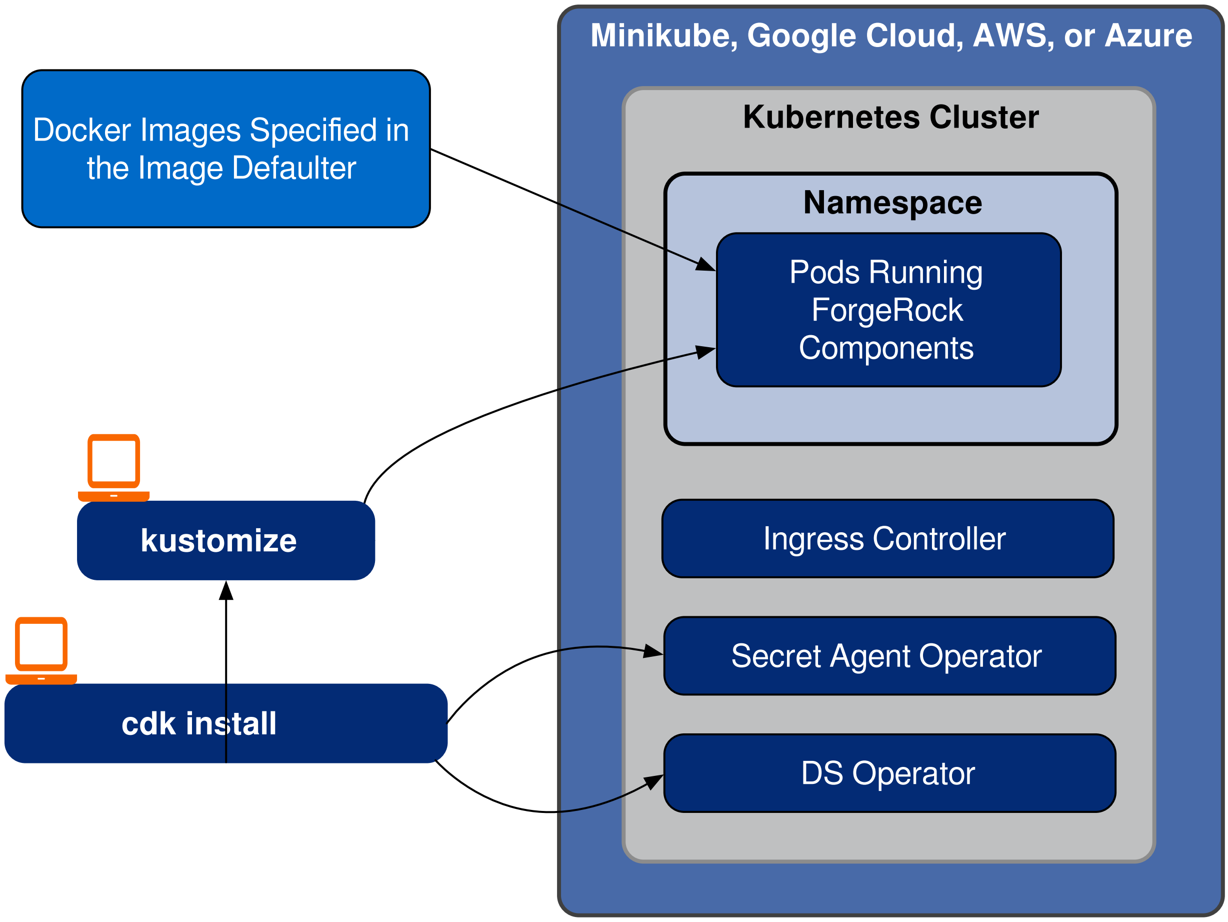
- ✓ Become familiar with the CDK
- ❑ Understand CDK architecture
- ❑ Set up your local environment
- ❑ Deploy the platform
- ❑ Access platform UIs and APIs
- ❑ (Optional) Develop custom Docker images

CDK Architecture

Before you can deploy the CDK, you must have access to a namespace in a Kubernetes cluster. The cluster must have an ingress controller deployed on it.

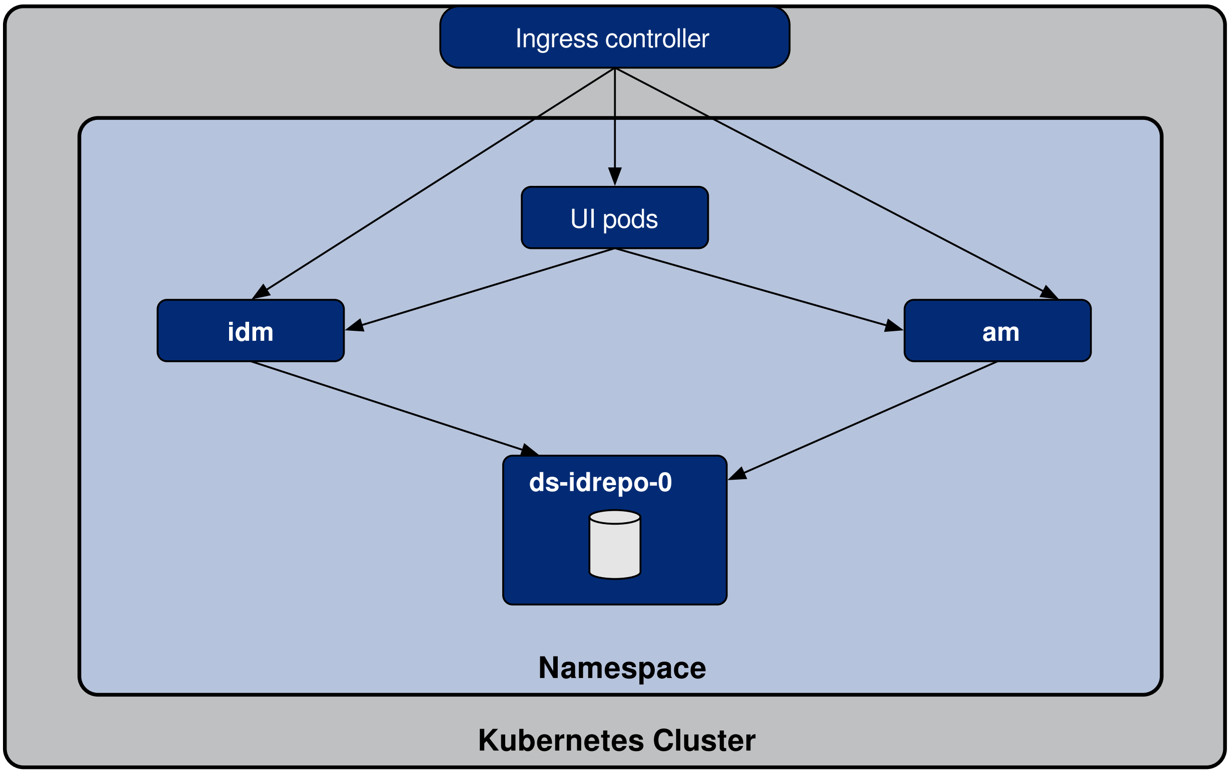
CDK Deployments

- Let you get the ForgeRock Identity Platform up and running on Kubernetes.
- Are suitable for demonstrations and proofs of concept.
- Build Kubernetes manifests based on the Kustomize bases and overlays in your local `forgeops` repository clone.
- Use the image defaulter to specify which Docker images to use to run the platform:
    - By default, the image defaulter specifies the latest evaluation-only Docker images for release 7.1 of the platform, available from ForgeRock's public registry. These images use ForgeRock's canonical configurations for AM and IDM.
    - When you build custom Docker images with customized AM and IDM configurations, the **`cdk build`** command updates the image defaulter to specify your custom images.



CDK Pods

After deploying the platform, you'll see the following pods running in your namespace. The pods are the same, regardless of whether you performed a demo or developer deployment:



am

Runs ForgeRock Access Management.

When AM starts for the first time in a CDK deployment, it obtains its configuration from the AM Docker image. If you subsequently restart AM, it obtains its configuration from the Git repository running in your namespace.

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 ForgeRock Identity Management.

When IDM starts for the first time in a CDK deployment, it obtains its configuration from the IDM Docker image. If you subsequently restart IDM, it obtains its configuration from the Git repository running in your namespace.

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

Minikube Environment Setup Checklist

- ❑ Get the forgeops repository
- ❑ Install third-party software
- ❑ Create the Minikube cluster
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Optionally install a TLS certificate

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

```
$ cd forgeops
$ git checkout release/7.1-20240223
```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❑ Install third-party software
- ❑ Create the Minikube cluster
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Optionally install a TLS certificate

Third-Party Software

Before performing a demo deployment, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the following tables have been validated for deploying the ForgeRock Identity Platform and building custom Docker images for it. Earlier and later versions will *probably* work. If you want to try using versions that are not in the tables, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
VirtualBox	6.1.32	`virtualbox` (cask)[1]

Software	Version	Homebrew package
Minikube	1.25.2	`minikube`

If you plan to use the CDK to create custom Docker images for the ForgeRock Identity Platform, install the following additional software:

Software	Version	Homebrew package
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

> NOTE
>
> Running the CDK on Minikube on macOS systems with ARM-based chipsets, such as the Apple M1 or M2, is currently available on an experimental basis only. You'll need all the software required to run Minikube on Intel-based macOS systems except for VirtualBox. You'll also need to install Docker and Colima. Refer to this ForgeRock Community article for details.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❏ Create the Minikube cluster
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Optionally install a TLS certificate

Minikube Cluster

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

The **`cluster/minikube/cluster-up`** utility creates a Minikube cluster with a configuration that's suitable for CDK deployments.

To set up Minikube:

```
$ cd /path/to/forgeops/cluster/minikube
$ ./cluster-up
Running: "minikube start --cpus=3 --memory=10g --disk-size=40g
--kubernetes-version=stable --addons=ingress,volumesnapshots --
driver=virtualbox --bootstrapper kubeadm"
?  minikube v1.23.2 on Darwin 11.5.1
```

```
    ▪ MINIKUBE_ACTIVE_DOCKERD=minikube
✨  Using the virtualbox driver based on user configuration
?  Downloading VM boot image …
    > minikube-v1.23.1.iso.sha256: 65 B / 65 B [-------------]
100.00% ? p/s 0s
    > minikube-v1.23.1.iso: 225.22 MiB / 225.22 MiB [ 100.00% 4.00
MiB p/s 1m2s
?  Starting control plane node minikube in cluster minikube
?  Creating virtualbox VM (CPUs=3, Memory=10240MB, Disk=40960MB) .
. .
?  Preparing Kubernetes on Docker 20.10.6 …
    ▪ Generating certificates and keys …
    ▪ Booting up control plane …
    ▪ Configuring RBAC rules …
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image k8s.gcr.io/sig-storage/snapshot-
controller:v4.0.0
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v1.0.0-
beta.3
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-
certgen:v1.0
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-
certgen:v1.0
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    You have selected "virtualbox" driver, but there are better
options !
    For better performance and support consider using a different
driver:
            - hyperkit
    To turn off this warning run:
            $ minikube config set WantVirtualBoxDriverWarning
false
    To learn more about on minikube drivers checkout
https://minikube.sigs.k8s.io/docs/drivers/
    To see benchmarks checkout
https://minikube.sigs.k8s.io/docs/benchmarks/cpuusage/
?  Verifying Kubernetes components. . .
?  Verifying ingress addon. . .
?  Enabled addons: storage-provisioner, default-storageclass,
volumesnapshots, ingress
?  Done! kubectl is now configured to use "minikube" cluster and
"default" namespace by default
```

NOTE

NOTE

The **cluster/minikube/cluster-up** utility uses the VirtualBox driver by default. Version 7.1 of the ForgeRock Identity Platform has been tested on Minikube clusters configured with the VirtualBox driver. If you prefer to configure a different virtual machine driver:

- To configure your Minikube cluster with the Hyperkit driver (macOS systems only), specify the `--driver hyperkit` option when you run the **cluster-up** utility.
- To configure your Minikube cluster with the Docker driver (Linux systems only), specify the `--driver docker` option when you run the **cluster-up** utility.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Create the Minikube cluster
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Optionally install a TLS certificate

Namespace

Create a namespace in your new cluster.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the default namespace. Deploying to a non-default namespace lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your active namespace:

```
$ kubens my-namespace
Context "minikube" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Create the Minikube cluster
- ✓ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Optionally install a TLS certificate

Hostname Resolution

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

1. Run the **`minikube ip`** command to get the Minikube ingress controller's IP address:

   ```
   $ minikube ip
   192.168.99.100
   ```

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

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

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

   ```
   minikube-ip-address dev.example.com
   ```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Create the Minikube cluster
- ✓ Create a Kubernetes namespace
- ✓ Set up hostname resolution
- ❑ Optionally install a TLS certificate

TLS Certificate (Optional)

This page covers several options you can use to encrypt HTTP communications over TLS in CDK deployments.

Self-Signed Certificate

By default, Minikube's ingress controller plugin is configured with a self-signed certificate. This is the simplest encryption option—you don't have to make any changes to the CDK to get encryption.

However, when you access one of the ForgeRock web applications from your browser, you'll get a "Not Secure" message from your browser. You'll need to bypass the message.

Certificate From a Certificate Authority (CA)

If you have a certificate from a CA, you can use the certificate for TLS encryption. Install the certificate and your private key in a Kubernetes secret in your namespace. Minikube's ingress controller plugin gets the certificate from the secret, and then uses it to encrypt communications.

To use a certficate from a CA in a CDK deployment on Minikube:

1. Obtain the certificate:
    - Make sure that the certificate is PEM-encoded.
    - A best practice is to include the entire trust chain in your `.pem` file.
2. Make sure that the deployment FQDN that you specified in your `/etc/hosts` file works with your certificate.
3. Create a secret named `sslcert` in your namespace that contains the certificate. For example:

```
$ kubectl create secret tls sslcert --cert=/path/to/my-cert.crt --key=/path/to/my-key.key
```

Certificate Generated by the mkcert Utility

If you don't have a certificate from a CA, you can use the mkcert utility to generate a locally trusted certificate. In many cases, it's acceptable to use such certificates for development purposes.

To use a certificate generated by the mkcert utility in a CDK deployment on Minikube that uses `dev.example.com` as the deployment FQDN:

1. If you don't have mkcert software installed locally, install it. Firefox users also need to install certutil software. See the mkcert installation instructions for more information.
2. If you haven't ever done so, run the **`mkcert -install`** command to create a local certificate authority (CA) and install it in your system root store. Restart your browser after creating the local CA.

3. Create a wildcard certificate for the `iam.example.com` domain:

```
$ cd
$ mkcert "*.example.com"
```

4. Create a secret named `sslcert` in your namespace that contains the wildcard certificate. For example:

```
$ kubectl create secret tls sslcert --
cert=./_wildcard.example.com.pem --
key=./_wildcard.example.com-key.pem
```

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

GKE Environment Setup Checklist

- ❏ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared GKE cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

```
$ cd forgeops
$ git checkout release/7.1-20240223
```

Depending on your organization’s repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock’s master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared GKE cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Third-Party Software

Before performing a demo deployment, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the following tables have been validated for deploying the ForgeRock Identity Platform and building custom Docker images for it. Earlier and later versions will *probably* work. If you want to try using versions that are not in the tables, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`

Software	Version	Homebrew package
Google Cloud SDK	378.0.0	`google-cloud-sdk` (cask)[1]

If you plan to use the CDK to create custom Docker images for the ForgeRock Identity Platform, install the following additional software:

Software	Version	Homebrew package
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❑ Get details about the shared GKE cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution

Cluster Details

You'll need to get some information about the cluster from your cluster administrator. You'll provide this information as you perform various tasks to access the cluster.

1. Obtain the following cluster details:
   - The name of the Google Cloud project that contains the cluster.
   - The cluster name.
   - The Google Cloud zone in which the cluster resides.
   - The IP address of your cluster's ingress controller.
   - The location of the Docker registry from which your cluster will obtain images for the ForgeRock Identity Platform.
2. Verify that the following operators are installed on the cluster:
   - The Secret Agent operator
   - The DS operator

Next Step

- ✓ Get the forgeops repository

- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Context for the Shared Cluster

Kubernetes uses contexts to access Kubernetes clusters. Before you can access the shared cluster, you must create a context on your local computer if it's not already present.

To create a context for the shared cluster:

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted:
   - If the current context references the shared cluster, there is nothing further to do. Proceed to Namespace.
   - If the context of the shared cluster is present in the **kubectx** command output, set the context as follows:

     ```
     $ kubectx my-context
     Switched to context "my-context".
     ```

     After you have set the context, proceed to Namespace.
   - If the context of the shared cluster is not present in the **kubectx** command output, continue to the next step.
2. Configure the Google Cloud SDK standard component to use your Google account. Run the following command:

   ```
   $ gcloud auth login
   ```

3. A browser window prompts you to log in to Google. Log in using your Google account.

   A second screen requests several permissions. Select Allow.

   A third screen should appear with the heading, "You are now authenticated with the Google Cloud SDK!"
4. Return to the terminal window and run the following command. Use the cluster name, zone, and project name you obtained from your cluster administrator:

```
$ gcloud container clusters \
 get-credentials cluster-name --zone google-zone --project
google-project
Fetching cluster endpoint and auth data.
kubeconfig entry generated for cluster-name.
```

5. Run the **kubectx** command again and verify that the context for our Kubernetes cluster is now the current context.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ✓ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Namespace

Create a namespace in the shared cluster. Namespaces let you isolate your deployments from other developers' deployments.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the `default` namespace. Deploying to a non-default namespace lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository

- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Hostname Resolution

You might need to set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace:

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

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

2. If DNS does not resolve your deployment FQDN, add an entry similar to the following to the `/etc/hosts` file:

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

   For *`ingress-ip-address`*, specify the IP address of your cluster's ingress controller that you obtained from your cluster administrator.

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

Amazon EKS Environment Setup Checklist

- ❏ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared EKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace

- ❏ Set up hostname resolution

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

   ```
   $ cd forgeops
   $ git checkout release/7.1-20240223
   ```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared EKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Third-Party Software

Before performing a demo deployment, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the following tables have been validated for deploying the ForgeRock Identity Platform and building custom Docker images for it. Earlier and later versions will *probably* work. If you want to try using versions that are not in the tables, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
Amazon AWS Command Line Interface	2.8.9	`awscli`
AWS IAM Authenticator for Kubernetes	0.5.5	`aws-iam-authenticator`

If you plan to use the CDK to create custom Docker images for the ForgeRock Identity Platform, install the following additional software:

Software	Version	Homebrew package
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❏ Get details about the shared EKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Cluster Details

You'll need to get some information about the cluster from your cluster administrator. You'll provide this information as you perform various tasks to access the cluster.

1. Obtain the following cluster details:
   - Your AWS access key ID.
   - Your AWS secret access key.
   - The AWS region in which the cluster resides.
   - The cluster name.
   - The IP address of your cluster's ingress controller.
   - The location of the Docker registry from which your cluster will obtain images for the ForgeRock Identity Platform.
2. Verify that the following operators are installed on the cluster:
   - The Secret Agent operator
   - The DS operator

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared EKS cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution

Context for the Shared Cluster

Kubernetes uses contexts to access Kubernetes clusters. Before you can access the shared cluster, you must create a context on your local computer if it's not already present.

To create a context for the shared cluster:

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted:
   - If the current context references the shared cluster, there is nothing further to do. Proceed to Namespace.
   - If the context of the shared cluster is present in the **kubectx** command output, set the context as follows:

     ```
     $ kubectx my-context
     Switched to context "my-context".
     ```

     After you have set the context, proceed to Namespace.

- If the context of the shared cluster is not present in the **kubectx** command output, continue to the next step.

2. Run the **aws configure** command. This command logs you in to AWS and sets the AWS region. Use the access key ID, secret access key, and region you obtained from your cluster administrator. You do not need to specify a value for the default output format:

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

3. Run the following command. Use the cluster name you obtained from your cluster administrator:

```
$ aws eks update-kubeconfig --name my-cluster
Added new context arn:aws:eks:us-east-
1:813759318741:cluster/my-cluster
to /Users/my-user-name/.kube/config
```

4. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

In Amazon EKS environments, the cluster owner must grant access to a user before the user can access cluster resources. For details about how the cluster owner can grant you access to the cluster, refer the cluster owner to Cluster Access for Multiple AWS Users.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared EKS cluster
- ✓ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Namespace

Create a namespace in the shared cluster. Namespaces let you isolate your deployments from other developers' deployments.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the `default` namespace. Deploying to a non-default namespace

lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared EKS cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Hostname Resolution

You might need to set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace:

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

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

2. If DNS does not resolve your deployment FQDN, add an entry similar to the following to the `/etc/hosts` file:

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

   For *`ingress-ip-address`*, specify the IP address of your cluster's ingress controller that you obtained from your cluster administrator.

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

AKS Environment Setup Checklist

- ❑ Get the forgeops repository
- ❑ Install third-party software
- ❑ Get details about the shared AKS cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

   ```
   $ cd forgeops
   $ git checkout release/7.1-20240223
   ```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository

- ❏ Install third-party software
- ❏ Get details about the shared AKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Third-Party Software

Before performing a demo deployment, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the following tables have been validated for deploying the ForgeRock Identity Platform and building custom Docker images for it. Earlier and later versions will *probably* work. If you want to try using versions that are not in the tables, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
Azure Command Line Interface	2.42.0	`azure-cli`

If you plan to use the CDK to create custom Docker images for the ForgeRock Identity Platform, install the following additional software:

Software	Version	Homebrew package
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❏ Get details about the shared AKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Cluster Details

You'll need to get some information about the cluster from your cluster administrator. You'll provide this information as you perform various tasks to access the cluster.

1. Obtain the following cluster details:
   - The ID of the Azure subscription that contains the cluster. Be sure to obtain the hexadecimal subscription ID, not the subscription name.
   - The name of the resource group that contains the cluster.
   - The cluster name.
   - The IP address of your cluster's ingress controller.
   - The location of the Docker registry from which your cluster will obtain images for the ForgeRock Identity Platform.
2. Verify that the following operators are installed on the cluster:
   - The Secret Agent operator
   - The DS operator

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Context for the Shared Cluster

Kubernetes uses contexts to access Kubernetes clusters. Before you can access the shared cluster, you must create a context on your local computer if it's not already present.

To create a context for the shared cluster:

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted:
   - If the current context references the shared cluster, there is nothing further to do. Proceed to Namespace.
   - If the context of the shared cluster is present in the **kubectx** command output, set the context as follows:

     ```
     $ kubectx my-context
     Switched to context "my-context".
     ```

     After you have set the context, proceed to Namespace.
   - If the context of the shared cluster is not present in the **kubectx** command output, continue to the next step.
2. Configure the Azure CLI to use your Microsoft Azure. Run the following command:

   ```
   $ az login
   ```

3. A browser window prompts you to log in to Azure. Log in using your Microsoft account.

   A second screen should appear with the message, "You have logged into Microsoft Azure!"
4. Return to the terminal window and run the following command. Use the resource group, cluster name, and subscription ID you obtained from your cluster administrator:

   ```
   $ az aks get-credentials \
    --resource-group my-fr-resource-group \
    --name my-fr-cluster \
    --subscription your subscription ID \
    --overwrite-existing
   ```

5. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ✓ Create a Kubernetes context
- ❑ Create a Kubernetes namespace

- ❏ Set up hostname resolution

Namespace

Create a namespace in the shared cluster. Namespaces let you isolate your deployments from other developers' deployments.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the default namespace. Deploying to a non-default namespace lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ❏ Set up hostname resolution

Hostname Resolution

You might need to set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace:

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

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

2. If DNS does not resolve your deployment FQDN, add an entry similar to the following to the `/etc/hosts` file:

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

For *`ingress-ip-address`*, specify the IP address of your cluster's ingress controller that you obtained from your cluster administrator.

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 Deployment

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

1. Run the **`cdk install`** command:

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

By default, the **`cdk install`** command uses the latest evaluation-only Docker images for release 7.1 of the platform, available from ForgeRock's public registry.

However, if you have built custom images for the ForgeRock Identity Platform, the **`cdk install`** command uses your custom images.

> NOTE
>
> The **`cdk install`** command in this example deploys the entire ForgeRock Identity Platform. If you prefer, you can deploy the platform component by component. See Staged CDK Installation.

2. 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.

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 ForgeRock 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 ForgeRock Identity Platform's administrative consoles and REST APIs.

You access AM and IDM services through the Kubernetes ingress controller. Access components using their normal interfaces:

- For AM, the console and REST APIs.
- For IDM, the Admin UI 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 console:

1. Make sure that your namespace is the current namespace:

   ```
   $ kubens my-namespace
   ```

2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./print-secrets amadmin
   179rd8en9rffa82rcf1qap1z0gv1hcej
   ```

3. Open a new window or tab in a web browser.

4. Go to https://dev.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 ForgeRock Identity Platform UI appears in the browser.
6. Select Native Consoles > Access Management.

   The AM console 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://dev.example.com/am/json/realms/root/authenticate"
{
    "tokenId":"AQIC5wM2. . .TU3OQ*",
    "successUrl":"/am/console",
    "realm":"/"
}
```

IDM Services

To access the IDM Admin UI:

1. Make sure that your namespace is the current namespace:

```
$ kubens my-namespace
```

2. Obtain the `amadmin` user’s password:

```
$ cd /path/to/forgeops/bin
$ ./print-secrets amadmin
vr58qt11ihoa31zfbjsdxxrqryfw0s31
```

3. Open a new window or tab in a web browser.
4. Go to https://dev.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 ForgeRock 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 **`print-secrets`** 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:
   a. 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://dev.example.com/am/json/realms/root/authenticate"
{
 "tokenId":" AQIC5wM. . .TU3OQ* ",
 "successUrl":"/am/console",
 "realm":"/"}
```

b. 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://dev.example.com/am/oauth2/realms/root/authorize?
redirect_uri=https://dev.example.com/platform/appAuthHelpe
rRedirect.html&client_id=idm-admin-
ui&scope=openid%20fr:idm:*&response_type=code&state=abc123
"
HTTP/2 302
server: nginx/1.17.10
date: Tue, 21 Jul 2020 16:54:20 GMT
content-length: 0
location:
https://dev.example.com/platform/appAuthHelperRedirect.htm
l
 ?
code= 3cItL9G52DIiBdfXRngv2_dAaYM &iss=http://dev.example.co
m: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
strict-transport-security: max-age=15724800;
includeSubDomains
```

c. 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://dev.example.com/platform/appAuthHelp
```

```
erRedirect.html" \

"https://dev.example.com/am/oauth2/realms/root/access_toke
n"
{
 "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://dev.example.com/openidm/config"
{
 "_id":"",
 "configurations":
  [
   {
    "_id":"ui.context/admin",
    "pid":"ui.context.4f0cb656-0b92-44e9-a48b-76baddda03ea",
    "factoryPid":"ui.context"
    },
    . . .
   ]
}
```

Directory Services

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 **print-secrets dsadmin** 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

Overview

This section covers how developers build custom Docker images for the ForgeRock 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 ForgeRock Identity Platform. Complete all of the tasks on this page before proceeding.

Configure Your Environment to Write to Your Docker Registry

Set up your local environment to write Docker images:

▼ Minikube

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:

1. Run the **`docker-env`** command in your shell:

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

2. Stop Skaffold from pushing Docker images to a remote Docker registry [3]:

```
$ skaffold config set --kube-context minikube local-cluster true
set value local-cluster to true for context minikube
```

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.

▼ GKE shared cluster

In the environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your GKE cluster.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

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. Set up a Docker credential helper:

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

3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location you obtained from your cluster administrator and the Kubernetes context you obtained in Context for the Shared Cluster:

```
$ skaffold config set default-repo my-docker-registry --
kube-context my-kubernetes-context
```

▼ EKS shared cluster

In the environment you’re setting up, Skaffold builds Docker images using the Docker software you’ve installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your EKS cluster.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

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
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.[4]
3. Run the **kubectx** command to obtain the Kubernetes context.

4. Configure Skaffold with the Docker registry location and the Kubernetes context:

```
$ skaffold config set default-repo my-docker-registry --
kube-context my-kubernetes-context
```

▼ AKS shared cluster

In the environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your AKS cluster.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

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.
3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location you obtained from your cluster administrator and the Kubernetes context you obtained in Context for the Shared Cluster:

```
$ skaffold config set default-repo my-docker-registry --
kube-context my-kubernetes-context
```

Create a Configuration Profile

Your configuration profile contains your customizations to ForgeRock's canonical configuration.

To initialize your configuration profile with ForgeRock's canonical configuration:

- First, initialize the staging area with ForgeRock's canonical configuration.
- Then, save the configuration to your configuration profile in the `/path/to/forgeops/config` directory.

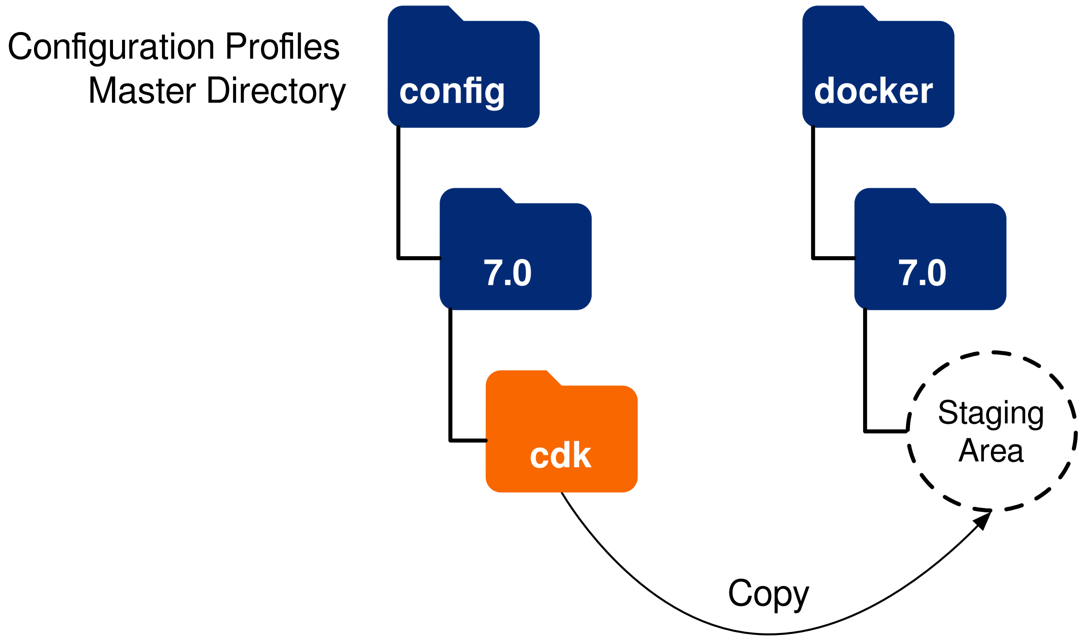
Perform these steps:

1. Change to the `/path/to/forgeops/bin` directory.

2. Initialize the staging area with the canonical CDK configuration profile for the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops/bin
$ ./config.sh init --profile cdk
Removing docker/7.0/am/config/
Removing docker/7.0/amster/config/
Removing docker/7.0/idm/conf/
Removing docker/7.0/idm/ui/
Removing docker/7.0/ig/config/
Copying /path/to/forgeops/config/7.0/cdk/idm.
Copying /path/to/forgeops/config/7.0/cdk/am.
Copying /path/to/forgeops/config/7.0/cdk/ig.
Copying /path/to/forgeops/config/7.0/cdk/amster.
Completed
```

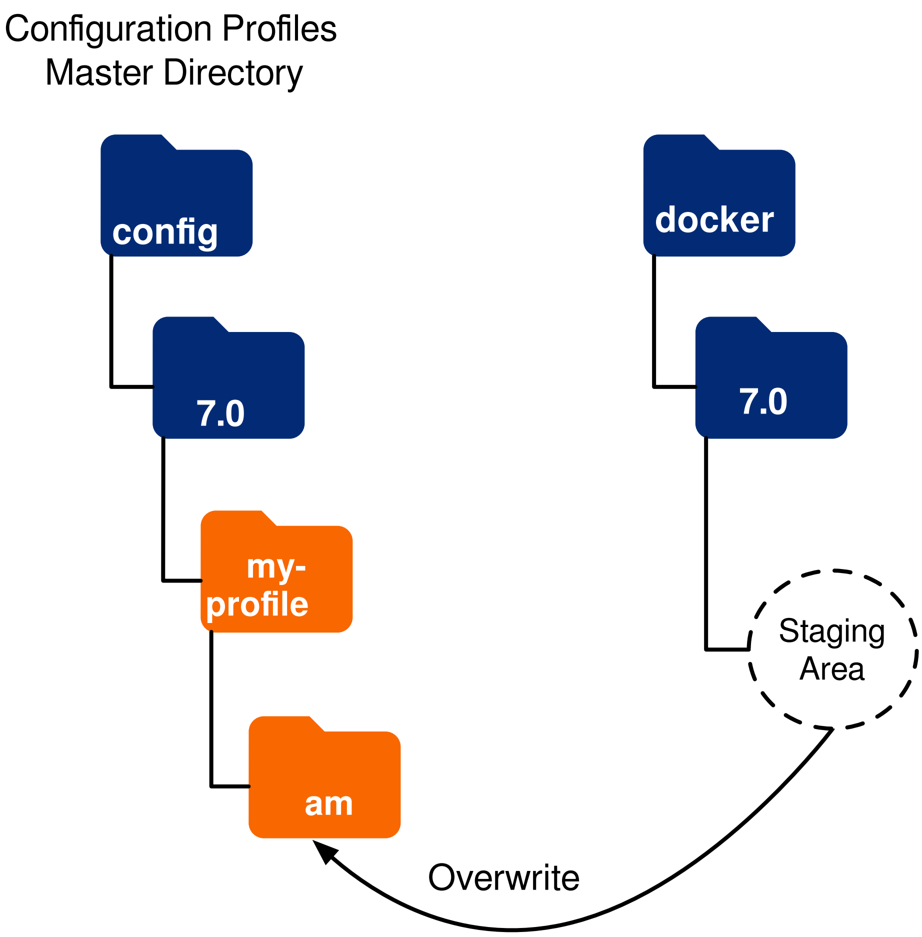
The **`config.sh init --profile cdk`** command clears out the staging area, and then copies the canonical configuration for the CDK from the `config/7.0/cdk` directory to the staging area:



3. Initialize your configuration profile with the canonical AM static configuration:

```
$ ./config.sh save --component am --profile my-profile
Saving AM configuration.
```

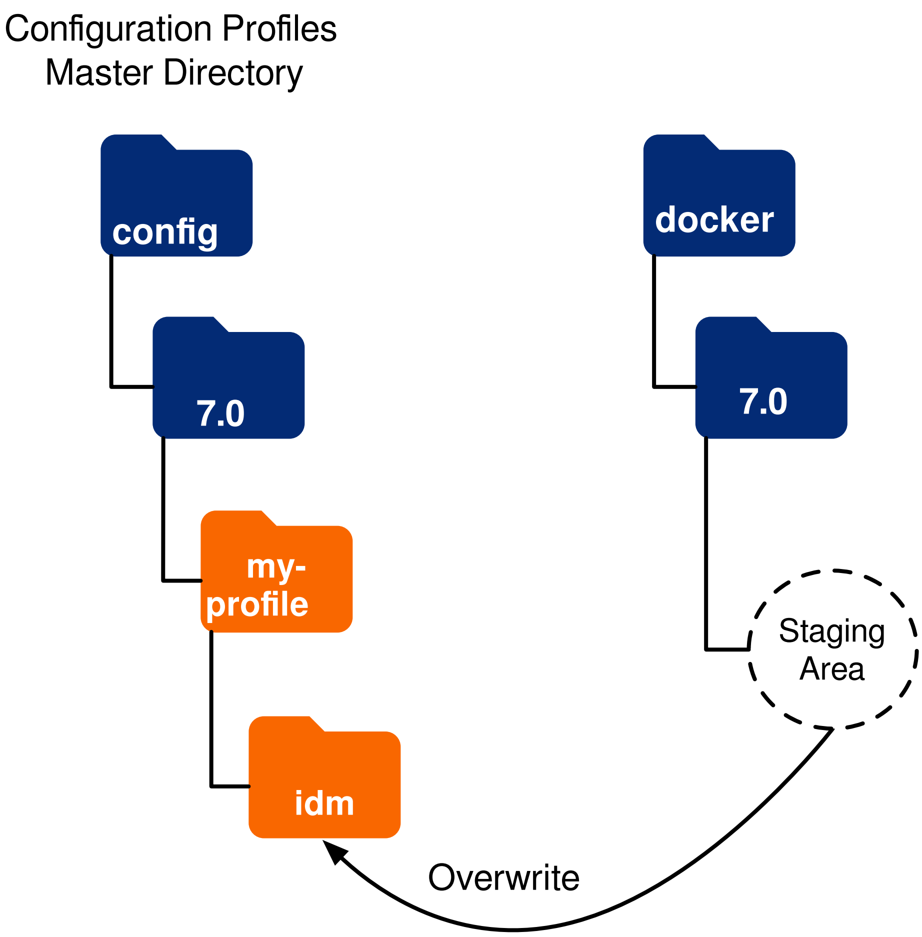
The **`config.sh save --component am --profile`** ***`my-profile`*** command copies AM's static configuration from the staging area to a configuration profile. Because the configuration profile does not exist yet, the **`config.sh save`** command creates it.



4. Initialize your configuration profile with the canonical IDM static configuration:

```
$ ./config.sh save --component idm --profile my-profile
Saving IDM configuration.
```

The **`config.sh save --component idm --profile`** ***`my-profile`*** command copies IDM’s static configuration from the staging area to a configuration profile.



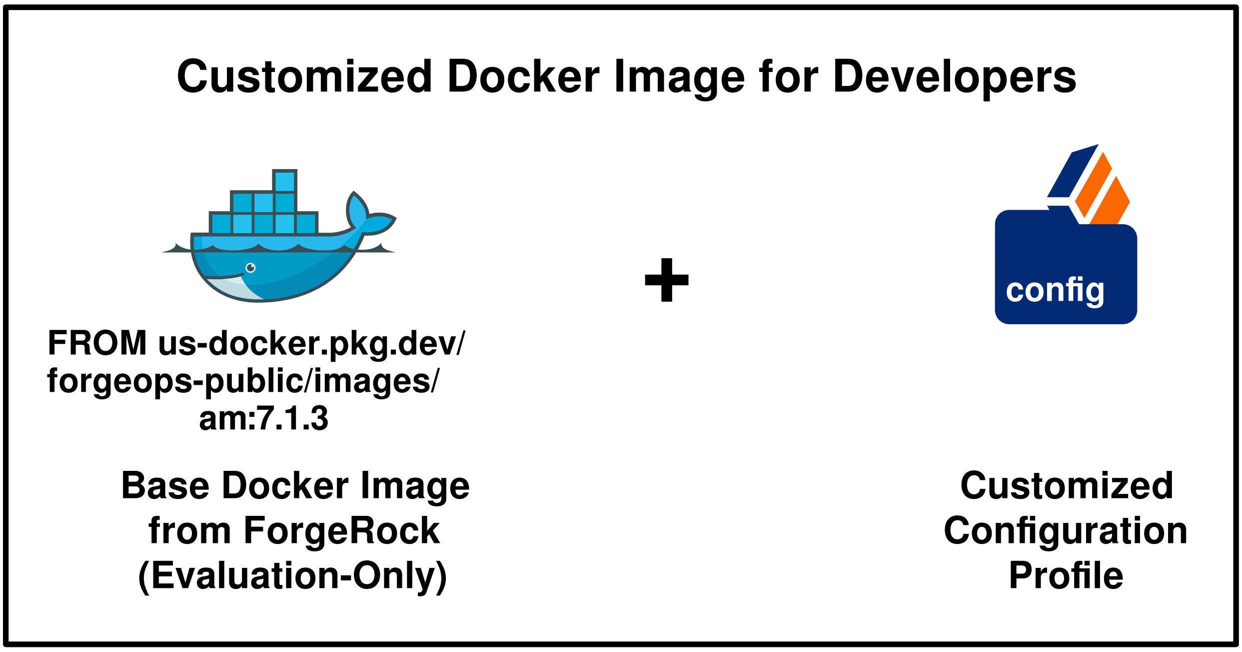
Next Step

- ✓ Perform additional setup
- ❑ Understand custom images
- ❑ Understand types of configuration
- ❑ Understand property value substitution
- ❑ Customize the AM image
- ❑ Customize the IDM image

About Custom Images

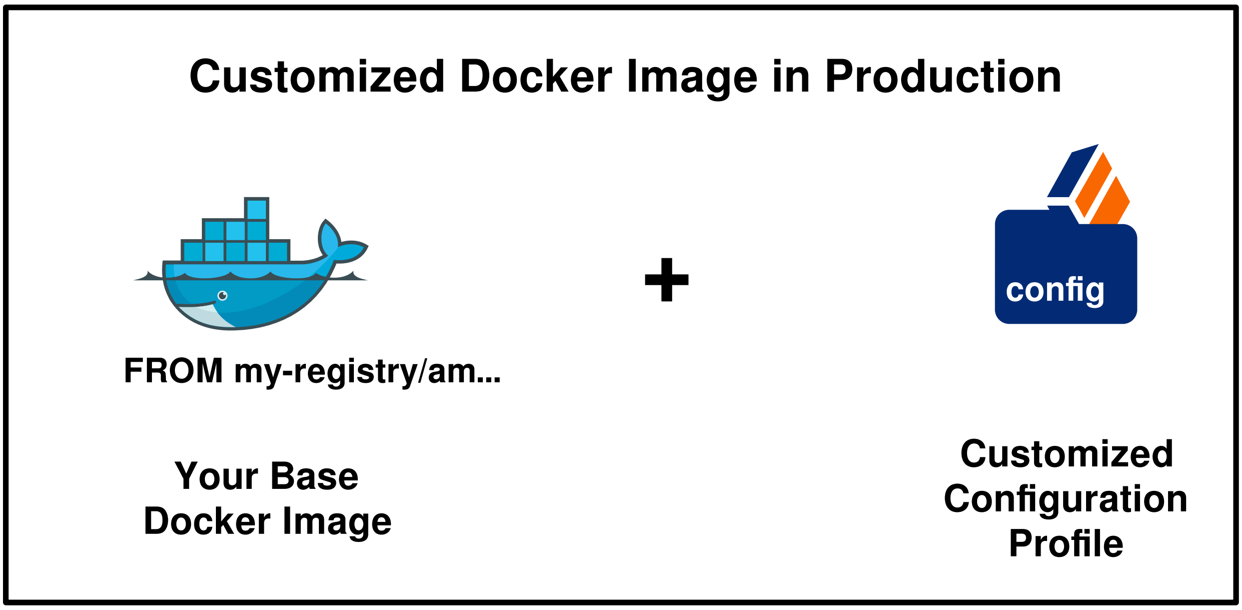
In Development

To develop customized Docker images, start with base images and a canonical configuration profile from ForgeRock. Then, build up a configuration profile, customizing the platform to meet your needs. The configuration profile is integrated into the customized Docker image:



In Production

Before you deploy the platform in production, you'll need to stop using ForgeRock's evaluation-only base images, and start using base images you build yourself. Building your own base images is covered in Base Docker Images. Then, customize your own base images by integrating the configuration profile you've developed into them:



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 ForgeRock Identity Platform uses two types of configuration: static configuration and dynamic configuration.

Static Configuration

Static configuration consists of properties and settings used by the ForgeRock 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 ForgeRock 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 ForgeRock 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 ForgeRock 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 console.
    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
    $ ./amster export ~/Desktop/amster

    Cleaning up amster components
    Deploying amster
    job.batch/amster created

    Waiting for amster job to complete. This can take several
    minutes.
    pod/amster-c8r2l condition met
    tar: Removing leading `/' from member names
    Updating amster config.
    A userpassword key found in
    /Users/me/Desktop/amster/realms/root/OAuth2Clients/Test.js
    on but no replacement rule was found, using default
    /Users/me/Desktop/amster/realms/root/OAuth2Clients/Test.js
    on has password changed to &
    {realms.root.OAuth2Clients.Test.userpassword}
    Updating amster config complete.
    Cleaning up amster components
    job.batch "amster" 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.

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 ForgeRock 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 two locations in the `forgeops` repository:

- **The master directory**. Holds a canonical configuration profile for the CDK and user-customized configuration profiles. User-customized configuration profiles in this directory are considered to be the *source of truth* for ForgeRock Identity Platform deployments.

  The master directory for configuration profiles is located at the path `/path/to/forgeops/config/7.0`. Use Git to manage the configuration profiles in this directory.
- **The staging area**. Holds a single configuration profile. You copy a profile from the master directory to the staging area before building a customized Docker image for the ForgeRock Identity Platform.

  The staging area is located in subdirectories of the path, `/path/to/forgeops/docker/7.0`. Configuration profiles copied to the staging area are transient and are not managed with Git.

The **`config.sh`** script lets you copy configuration profiles between the master directory and the staging area. You run this script before you build customized Docker images for the platform. The script lets you specify which configuration profile to copy to the staging area. The **`cdk build`** command uses the profile that's been copied to the staging area when it builds a Docker image.

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

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

How Property Value Substitution Works

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

1. Search the `/path/to/forgeops/config/7.0/cdk` directory for the string `&{`.
2. Locate this line in your search results:

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

   Because the property `am.encryption.pwd` is being set to a configuration expression, its value will be determined when AM starts up.

3. Search the `forgeops` repository for the string `AM_ENCRYPTION_KEY`. You'll see that the secret agent operator sets the environment variable, `AM_ENCRYPTION_KEY`. The property, `am.encryption.pwd`, will be set to the value of the environment variable, `AM_ENCRYPTION_KEY` when AM starts up.

Configuration expressions take their values from environment variables as follows:

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

For more information about configuration expressions, see Property Value Substitution and environment variables, Java system properies, and configuration files 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.sh export idm`** command exports IDM static configuration from running CDK instances to your `forgeops` repository clone. The script 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 console is *not* aware of configuration expressions.

Properties can not be specified as configuration expressions in the AM console; 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.sh 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.sh`** script copies the AM static configuration from the CDK, it calls the AM configuration upgrader. The upgrader transforms the AM configuration, following rules in the `config/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 `config/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 `config/am-upgrader-rules/placeholders.groovy` file is using the **`jq`** command to modify the output from the **`config.sh`** script.

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

`am` Image

The `am` Docker image contains the AM configuration.

Customization Overview

- Customize AM's configuration data by using the console and the 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

Perform the following steps iteratively when developing a custom `am` Docker image:

1. If this is your first time building a custom Docker image, verify that you performed these setup activities, which are required for developers:
    - Configuration profile creation
    - Docker registry configuration
    - Installation of all required third-party software in your local environment (Minikube|GKE|EKS|AKS)
2. Verify that:
    - The CDK is deployed.
    - The namespace in which the CDK is deployed is set in your Kubernetes context.
3. Perform version control activities on your `forgeops` repository clone:
    a. Run the **`git status`** command.
    b. Review the state of the `config` directory.
    c. (Optional) Run the **`git commit`** command to commit changes to files that have been modified.

4. Modify the AM configuration using the AM console or the REST APIs.

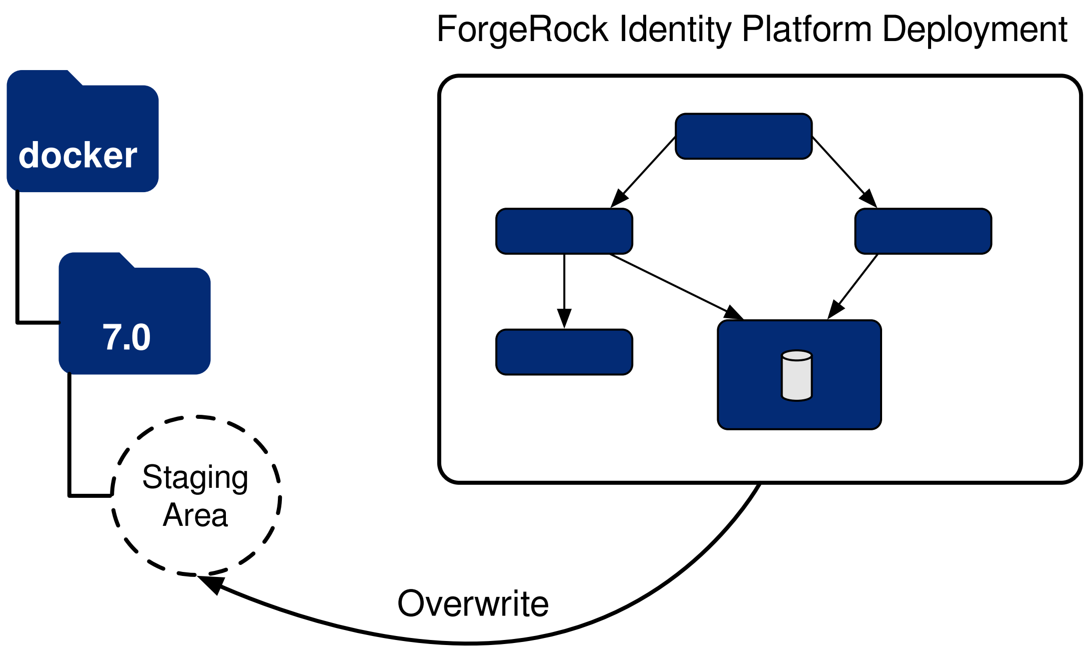
   For information about how to access the AM console or REST APIs, see AM Services.

   See About Property Value Substitution for important information about configuring values that vary at run-time, such as passwords and host names.

5. Export the changes you made to the AM configuration in the running ForgeRock Identity Platform to the staging area:

```
$ cd /path/to/forgeops/bin
$ ./config.sh export --component am

AM configuration files have been exported to
docker/7.0/am/config.

Reading existing configuration from files in /am-
config/config/services…
Modifying configuration based on rules in
[/rules/placeholders.groovy]…
reading configuration from file-based config files
SLF4J: Failed to load class
"org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder
 for further details.
Writing configuration to new location at /am-
config/config/services…
Upgrade Completed, modified configuration saved to /am-
config/config/services
```

   The **config.sh export --component am** command copies AM static configuration from the running CDK instance to the staging area.



6. Review the differences between the files you exported to the staging area and files that you previously saved to your configuration profile.

   Use the **config.sh diff** command to review the changes. For example:

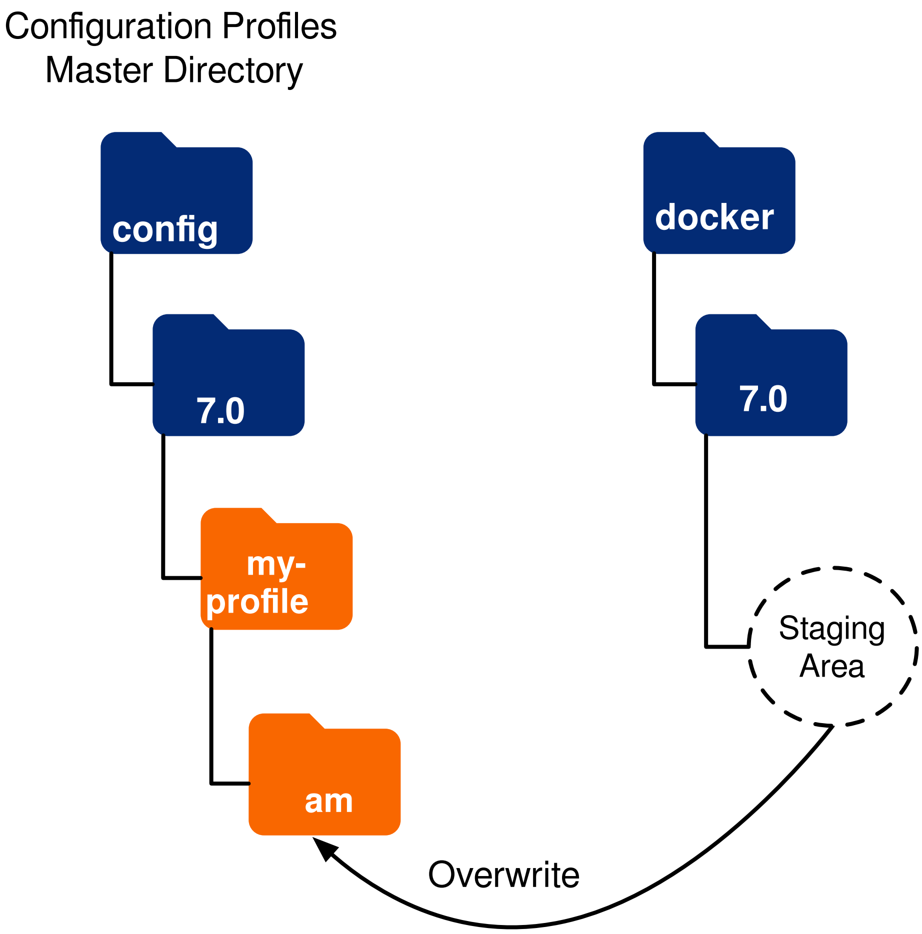
```
$ ./config.sh diff --component am --profile my-profile
Only in docker/7.0/am/config/services: global
diff -u --recursive -x '.' -x Dockerfile -x '.sh'
config/7.0/my-
profile/am/config/services/realm/root/configurationversionserv
ice/1.0/globalconfig/default.json
docker/7.0/am/config/services/realm/root/configurationversions
ervice/1.0/globalconfig/default.json
--- config/7.0/my-
profile/am/config/services/realm/root/configurationversionserv
ice/1.0/globalconfig/default.json       2022-01-06
11:35:23.000000000 -0800
+
docker/7.0/am/config/services/realm/root/configurationversions
ervice/1.0/globalconfig/default.json    2022-01-06
11:38:05.000000000 -0800
@@ -23,6 +23,6 @@
     },
     "_id" : "default",
     "configurationVersion" : "3.0.0.1",
-    "configurationCommit" :
"1c17cc27b8237484b5c7b49ccabfd712da0c0f3e"
+    "configurationCommit" :
"4e72fe392c000b0a15027eb41267d01bfd2d2220"
   }
```

```
  }
. . .
```

7. Save the AM configuration to your configuration profile:

```
$ ./config.sh save --component am --profile my-profile

Saving AM configuration.
```

The **`config.sh save --component am`** command copies AM static configuration from the staging area to your configuration profile.



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

   a. Run the **`git status`** command.

   b. Review the state of the `config` directory.

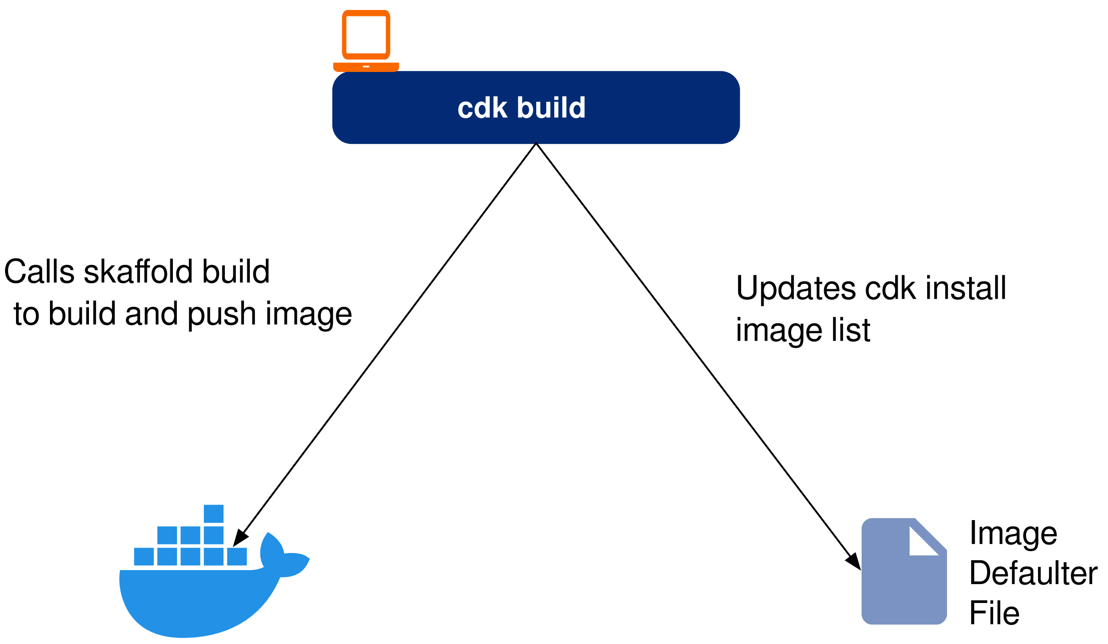
   c. (Optional) Run the **`git commit`** command to commit changes to files that have been modified.

9. Build a new `am` image that includes your changes to AM static configuration:

```
$ ./cdk build am
Generating tags…
 - am → am:584ce9b20
```

```
Checking cache…
 - am: Not found. Building
Starting build…
Found [minikube] context, using local docker daemon.
Building [am]…
Sending build context to Docker daemon  463.9kB
Step 1/14 : FROM us-docker.pkg.dev/forgeops-
public/images/am:7.1.4
7.1.4: Pulling from us-docker.pkg.dev/forgeops-
public/images/am
345e3491a907: Pulling fs layer
. . .
Step 14/14 : WORKDIR /home/forgerock
 ---→ Running in c0d17bb09b92
 ---→ e44e3b0256cb
Successfully built e44e3b0256cb
Successfully tagged am:584ce9b20
. . .
Updated the image_defaulter with your new image for am:
"am:e44e3b0256cbe477b158adc3fa188f9c5ef5f117bd4cf844580421c848
bad61a"
```

The **`cdk build`** command calls Skaffold to build a new `am` Docker image, and to push the image to your Docker registry[5]. It also updates the image defaulter file so that the next time you install AM, the **`cdk install`** command gets AM static configuration from your new custom Docker image.



10. Redeploy AM:

    a. Remove AM from your CDK installation:

```
$ ./cdk delete am
Uninstalling component(s): ['am']
OK to delete these components? [Y/N] Y
service "am" deleted
deployment.apps "am" deleted
```

b. Redeploy AM:

```
$ ./cdk install am
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!
```

c. 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.

11. To validate that AM has the expected configuration:
    - 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 console 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

`idm` Image

The `idm` Docker image contains the IDM configuration.

Customization Overview

- Customize IDM's configuration data by using the Admin UI and the 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

Perform the following steps iteratively when developing a custom `idm` Docker image:

1. If this is your first time building a custom Docker image, verify that you performed these setup activities, which are required for developers:
   - Configuration profile creation
   - Docker registry configuration
   - Installation of all required third-party software in your local environment (Minikube|GKE|EKS|AKS)
2. Verify that:
   - The CDK is deployed.
   - The namespace in which the CDK is deployed is set in your Kubernetes context.
3. Perform version control activities on your `forgeops` repository clone:
   a. Run the **`git status`** command.
   b. Review the state of the `config` directory.
   c. (Optional) Run the **`git commit`** command to commit changes to files that have been modified.
4. Modify the IDM configuration using the IDM Admin UI or the REST APIs.

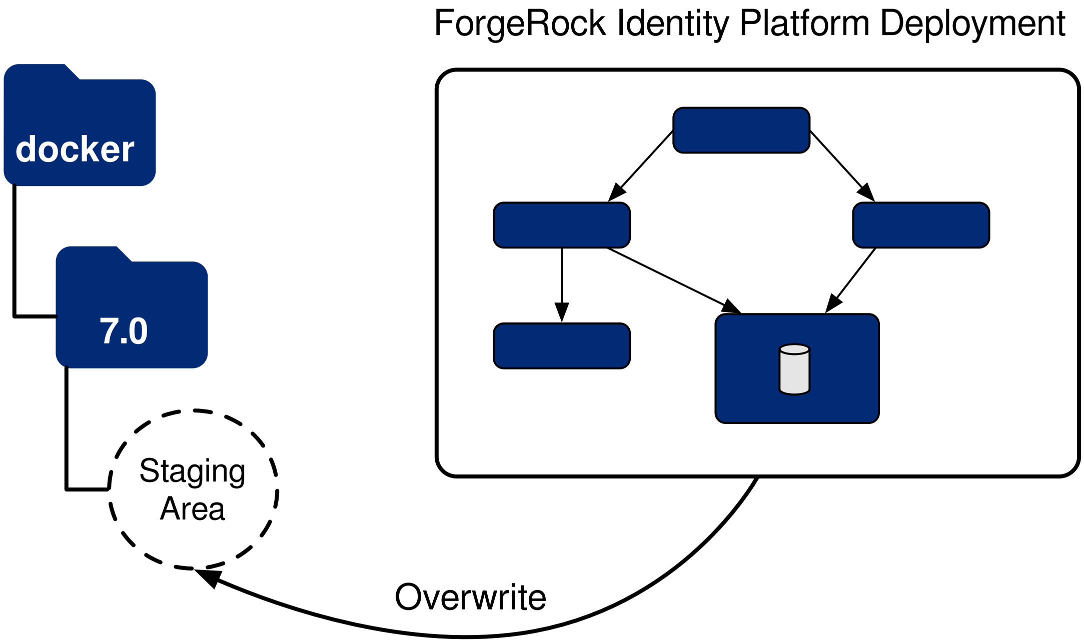
   For information about how to access the IDM Admin UI or REST APIs, see IDM Services.

   See About Property Value Substitution for important information about configuring values that vary at run-time, such as passwords and host names.
5. Export the changes you made to the IDM configuration in the running ForgeRock Identity Platform to the staging area:

```
$ cd /path/to/forgeops/bin
$ ./config.sh export --component idm
```

```
Exporting IDM configuration…

tar: Removing leading `/' from member names

IDM configuration files have been exported to
docker/7.0/idm/config.
```

The **`config.sh export --component idm`** command copies IDM static configuration from the running CDK instance to the staging area.



6. Review the differences between the files you exported to the staging area and files that you previously saved to your configuration profile.

   Use the **`config.sh diff`** command to review the changes. For example:

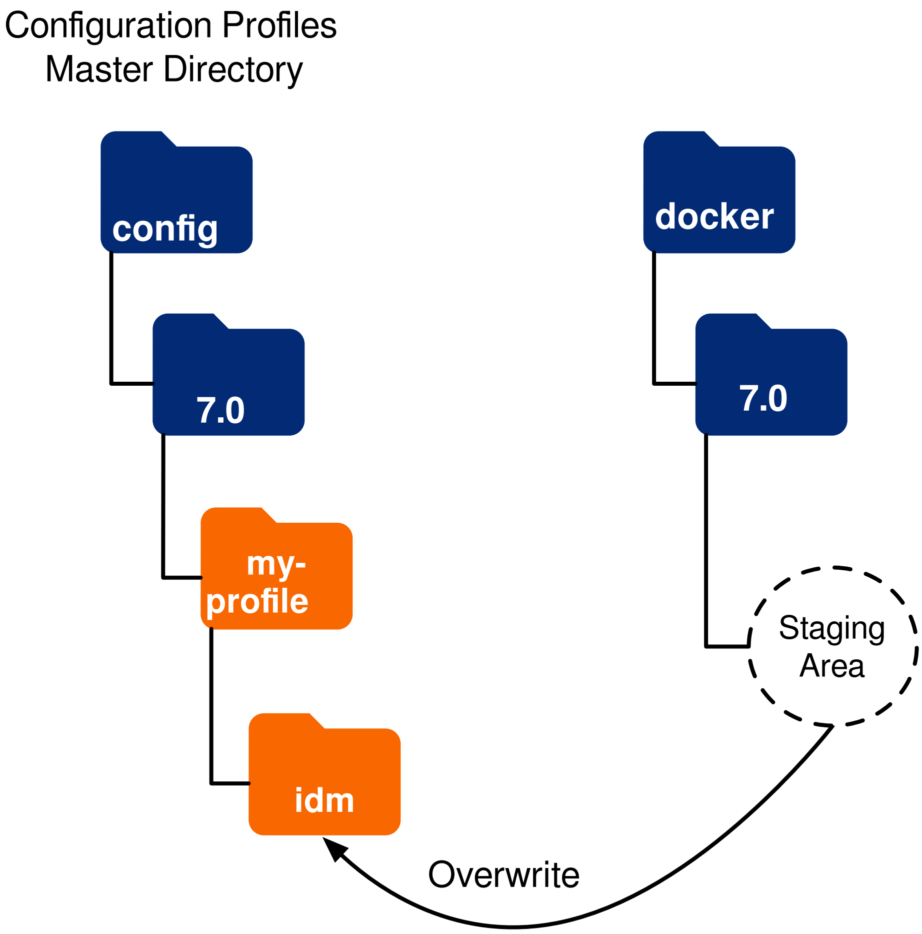
```
$ ./config.sh diff --component idm --profile my-profile
diff  -u --recursive config/7.0/my-profile/idm docker/7.0/idm
diff -u --recursive -x '.' -x Dockerfile -x '.sh'
config/7.0/my-profile/idm/conf/audit.json
docker/7.0/idm/conf/audit.json
--- config/7.0/my-profile/idm/conf/audit.json   2022-01-06
11:35:36.000000000 -0800
+ docker/7.0/idm/conf/audit.json        2022-01-06
11:54:19.000000000 -0800
@@ -135,6 +135,9 @@
     },
     "exceptionFormatter" : {
         "type" : "text/javascript",
+        "globals" : {
+            "fred" : "aaa"
```

```
+        },
         "file" :
"bin/defaults/script/audit/stacktraceFormatter.js"
     }
-}
+}
. . .
Only in docker/7.0/idm: resolver
Only in docker/7.0/idm: ui
. . .
```

7. Save the IDM configuration to your configuration profile:

```
$ ./config.sh save --component idm --profile my-profile
Saving IDM configuration.
```

The **`config.sh save --component idm`** command copies IDM static configuration from the staging area to your configuration profile.



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

   a. Run the **`git status`** command.

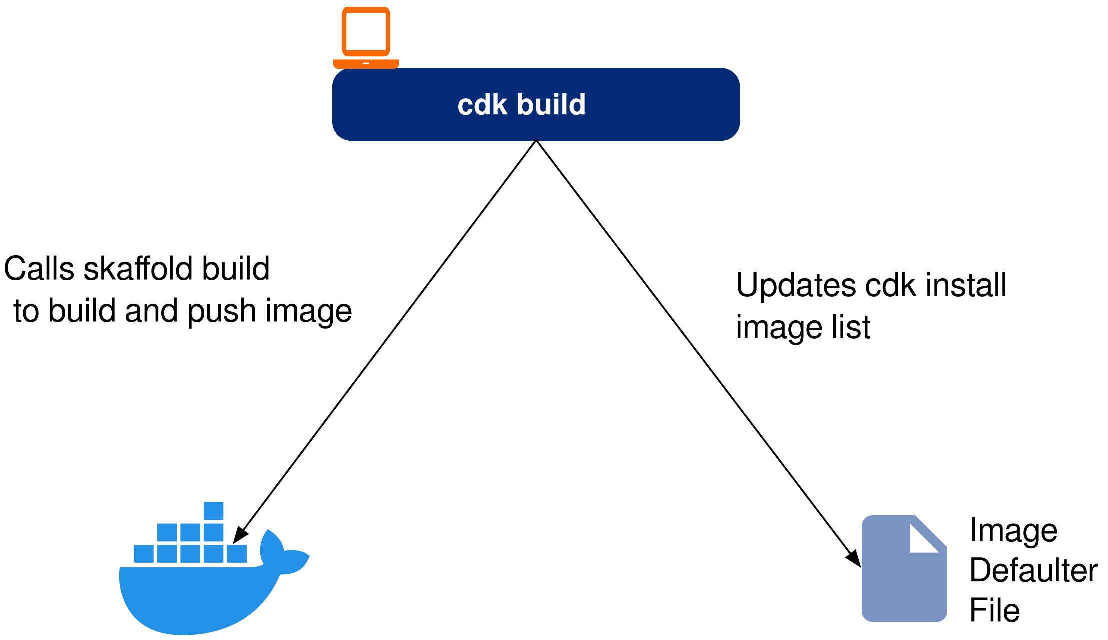
   b. Review the state of the `config` directory.

c. (Optional) Run the **`git commit`** command to commit changes to files that have been modified.

9. Build a new `idm` image that includes your changes to IDM static configuration:

```
$ ./cdk build idm
Building [idm]…
Sending build context to Docker daemon    276kB
FROM us-docker.pkg.dev/forgeops-public/images/idm:7.1.5
7.1.5: Pulling from us-docker.pkg.dev/forgeops-
public/images/idm
79d3b412d726: Already exists
. . .
Step 7/7 : COPY --chown=forgerock:root . /opt/openidm
 --→ 4c47ecbce819
Successfully built 4c47ecbce819
Successfully tagged idm:24f2f9a16

Updated the image_defaulter with your new image for idm:
"idm:4c47ecbce819a8cc9b1b4af9821bf3653b33d06469ae6d25f82caae17
805c195"
```

The **`cdk build`** command calls Skaffold to build a new `idm` Docker image and push the image to your Docker registry[6]. It also updates the image defaulter file so that the next time you install IDM, the **`cdk install`** command gets IDM static configuration from your new custom Docker image.



10. Redeploy IDM:

a. Remove IDM from your CDK installation:

```
$ cd /path/to/forgeops/bin
$ ./cdk delete idm
OK to delete these components? [Y/N] Y
configmap "idm" deleted
configmap "idm-logging-properties" deleted
service "idm" deleted
deployment.apps "idm" deleted
```

b. Redeploy IDM:

```
$ ./cdk install idm
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!
```

c. 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.

11. To validate that IDM has the expected configuration:
    - 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.

Additional Topics of Interest

Cloud Deployment Model

Deploy the CDM, ForgeRock's reference

How-Tos

Customize CDM deployments.

implementation for cloud deployment.

Shutdown

Shut down and remove your CDK deployment.

CDK Shutdown and Removal

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

1. If you've made changes to the AM and IDM configurations in the Git repository on the CDK that you want to save, sync the changes to your local `forgeops` repository clone. If you don't sync the configurations before you run the **`cdk 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

2. Run the **`cdk delete`** command which deletes all CDK artifacts, including PVCs and the AM and IDM configurations in Git:

   ```
   $ cd /path/to/forgeops/bin
   $ ./cdk delete --namespace my-namespace
   ```

   Respond `Y` to the `OK to delete?` prompt.

Cloud Deployment Model 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 ForgeRock 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

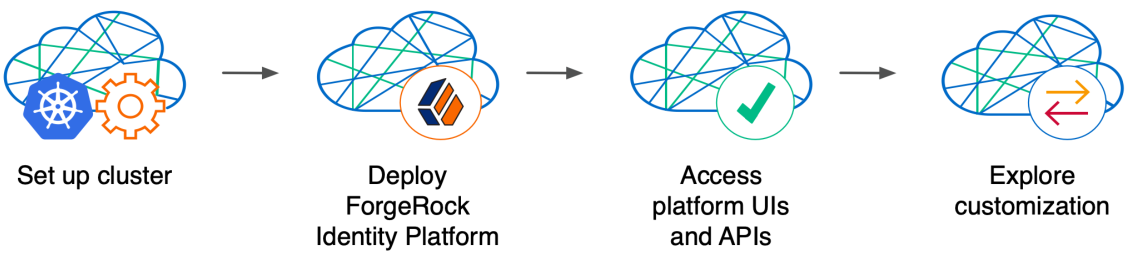
About the Cloud Deployment Model

The ForgeRock Cloud Deployment Team has developed Docker images, Kustomize bases and overlays, Skaffold workflows, shell scripts, and other artifacts expressly to build the Cloud Deployment Model (CDM). The `forgeops` repository on GitHub contains the CDM artifacts you can use to deploy the ForgeRock Identity Platform in a cloud environment.

The CDM is a reference implementation for ForgeRock cloud deployments. You can get a sample ForgeRock 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 ForgeRock 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 ForgeRock 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 ForgeRock Identity Platform, installing the platform, and accessing its UIs and APIs.

Standardizes the process. The ForgeRock Cloud Deployment Team's mission is to standardize a process for deploying ForgeRock 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, Skaffold workflows, and shell scripts—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 ForgeRock Identity Platform in the cloud. The CDM takes the guesswork out out of setting up a cloud environment. It bypasses the deploy-test-integrate-test-repeat cycle many customers struggle through when spinning up the ForgeRock 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 ForgeRock 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

ForgeRock 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

Before 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

- NGINX Ingress 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 NGINX Ingress Controller, Prometheus, and Grafana.

Ready-to-use ForgeRock 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.

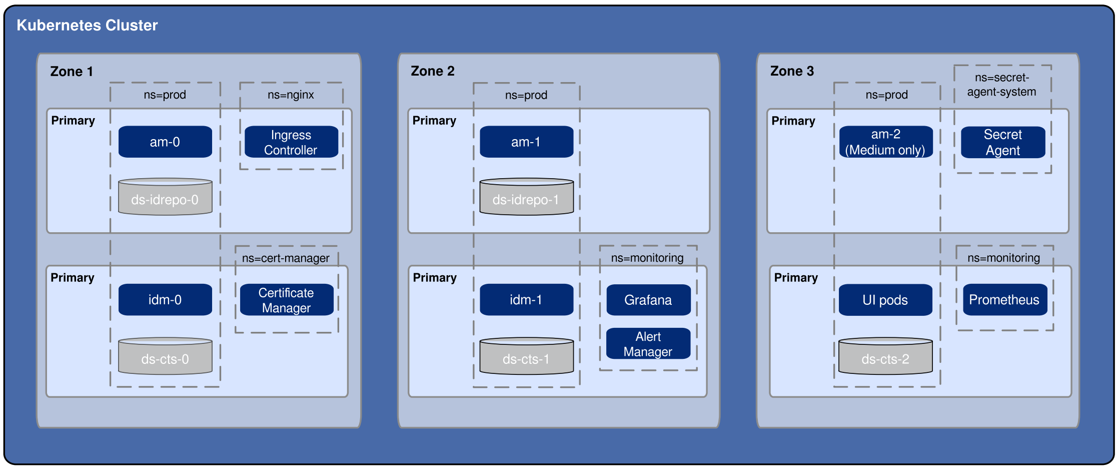
Highly available, distributed deployment

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

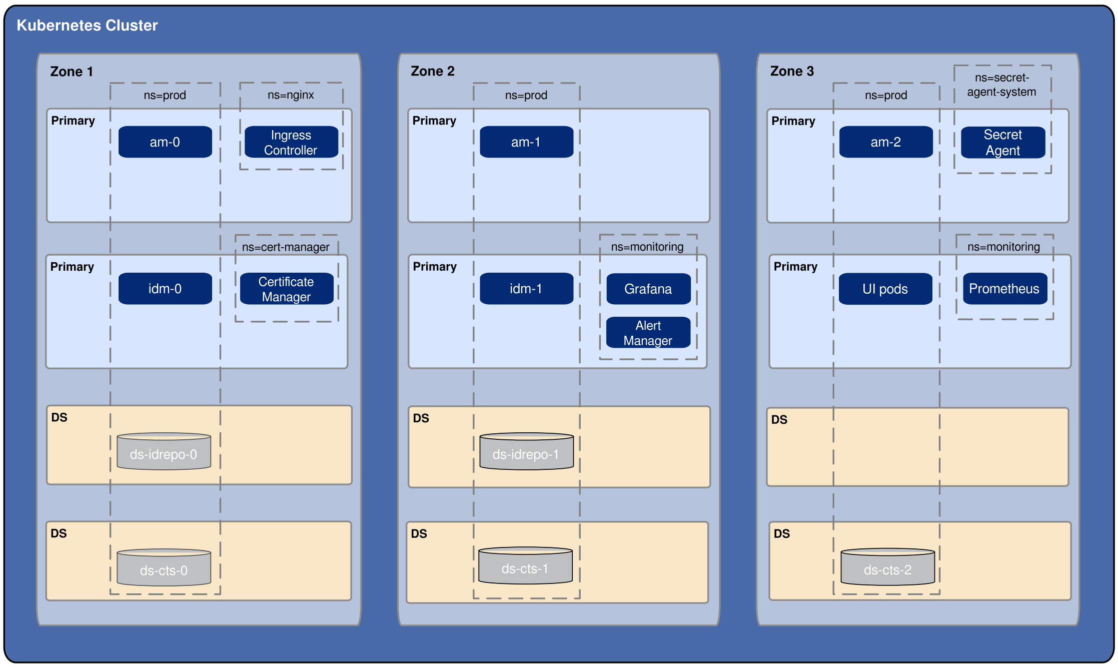
Pods that run DS are configured to use soft anti-affinity. Because of this, Kubernetes schedules DS pods to run on nodes that don't have any other DS pods whenever possible.

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

In small and medium CDM clusters, pods are organized across three zones in a single primary node pool [7] with six nodes. Pod placement among the nodes might vary, but the DS pods should run on nodes without any other DS pods:



In large CDM clusters, pods are distributed across two node pools — primary [7] and DS. Each node pool has six nodes. Again, pod placement among the nodes might vary, but the DS pods should run on nodes without any other DS pods:



Load balancing

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

Secret generation and management

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

Secured communication

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

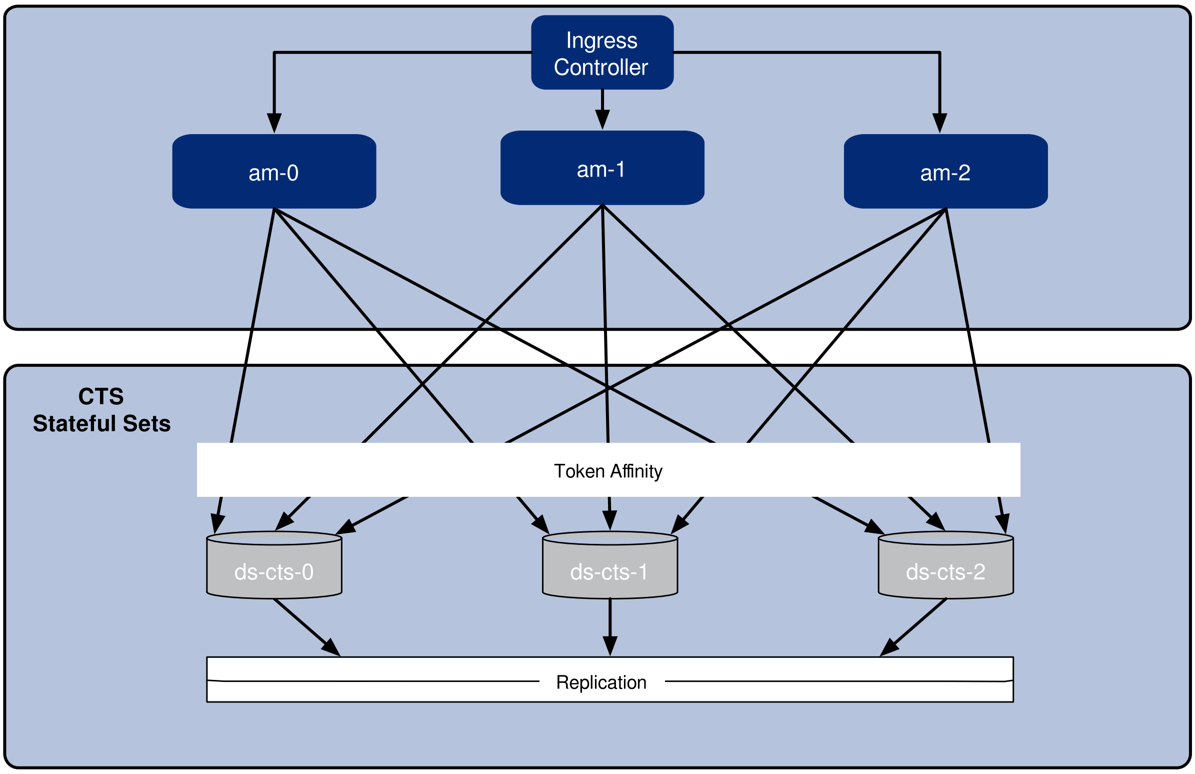
Inbound communication to DS instances occurs over secure LDAP (LDAPS).

For more information, see Secure HTTP and Secure LDAP.

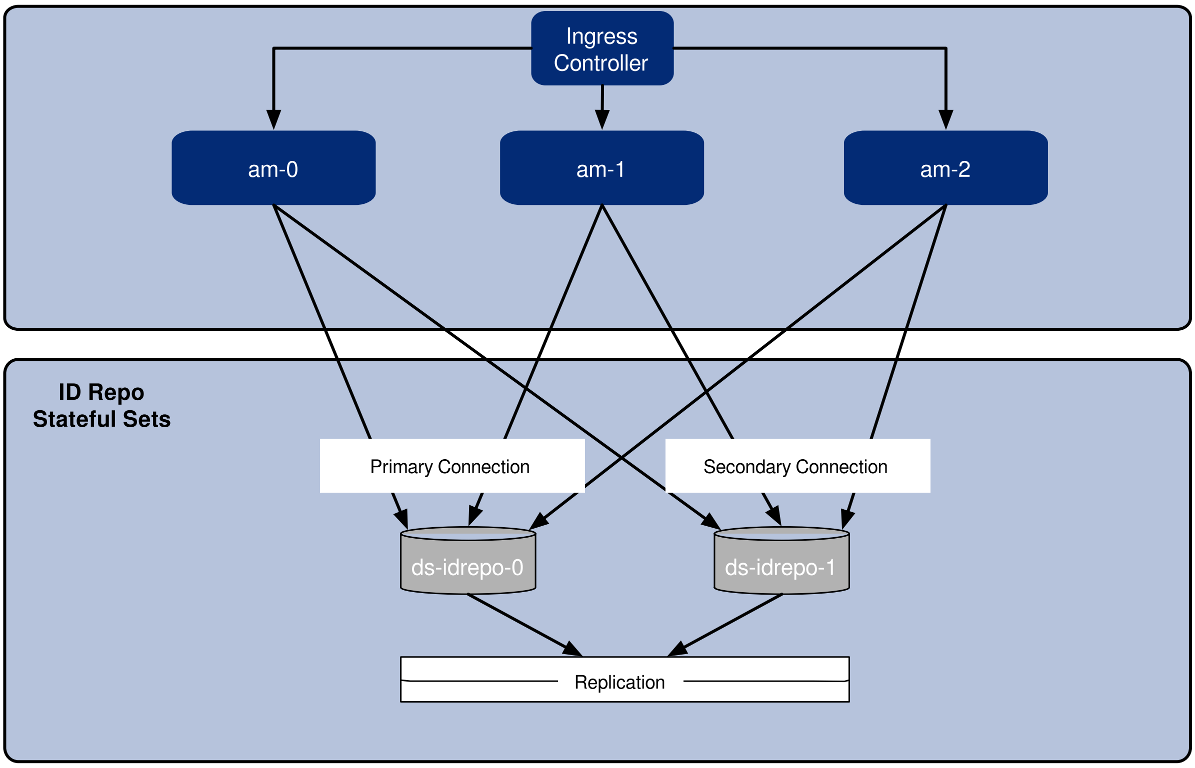
Stateful Sets

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 a secondary directory service:



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

The CDM is ready to back up directory data, but backups are not scheduled by default. To schedule backups, see Backup and Restore.

You can enable the automatic restore capability in CDM to create new DS instances with data from the backup of another CDM deployment with the same DS topology.

Initial data loading jobs

When it starts up, the CDM runs two jobs to load data into the environment:

- The `amster` job, which loads application data, such as OAuth 2.0 client definitions, to the `idrepo` DS instance.
- The `ldif-importer` job, which sets passwords for the DS `idrepo` and `cts` instances.

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

Environment Setup: GKE

Before deploying the CDM, you must set up your local computer, configure a Google Cloud project, and create a GKE cluster.

▼ Windows users

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: Ubuntu 19.10 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.

Environment Setup Checklist

- ❏ Install third-party software
- ❏ Set up a Google Cloud project
- ❏ Get the forgeops repository
- ❏ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager

- ❏ Prepare to push Docker images

After you've completed all of these environment setup tasks, you'll be ready to deploy the ForgeRock Identity Platform on your new Kubernetes cluster.

Third-Party Software

Before installing the CDM, you must obtain non-ForgeRock software and install it on your local computer.

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

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

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
Helm	3.8.1	`helm`
Google Cloud SDK	378.0.0	`google-cloud-sdk` (cask)[1]
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Install third-party software
- ❏ Set up a Google Cloud project
- ❏ Get the forgeops repository
- ❏ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator

- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Google Cloud Project Setup

This page outlines the steps that the Cloud Deployment Team took when setting up a Google Cloud project before deploying the CDM.

Perform these steps before you deploy the CDM:

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

      ```
      $ gcloud auth login
      ```

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

      A second screen requests several permissions. Select Allow.

      A third screen should appear with the heading, "You are now authenticated with the Google Cloud SDK!"

   c. Set the Google Cloud SDK configuration to reference your new project. Specify the project ID, not the project name, in the `gcloud config set project` command:

      ```
      $ gcloud config set project my-project-id
      ```

3. Assign the following roles to users who will be creating Kubernetes clusters and deploying the CDM:
   - Editor
   - Kubernetes Engine Admin
   - Kubernetes Engine Cluster Admin

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

4. Determine the region where you'll deploy the CDM. Then, set that region as the default region in your Google Cloud SDK configuration. For example:

```
$ gcloud config set compute/region us-west1
```

5. Determine the cluster size: small, medium, or large.
6. Ensure that the cluster creation script will support your region:
   a. Go to Google's Regions and Zones page.
   b. Determine if the `a`, `b`, and `c` zones are available in your region.

      If these zones are available, no additional action needs to be taken.

      If they're not available:

      i. Change to the `/path/to/forgeops/cluster/gke` directory.
      ii. Open the script that sets environment variables for your selected cluster size. For example, open the `small.sh` script if you're going to deploy a small-sized cluster.
      iii. Locate the statement that sets the `NODE_LOCATIONS` environment variable.
      iv. Uncomment this statement.
      v. Change the statement to configure CDM to use three zones available in your region.
      vi. Save your changes to the script.
7. Ensure that your region has an adequate CPU quota for the CDM:
   a. Change to the `/path/to/forgeops/cluster/gke` directory.
   b. Open the script that sets environment variables for your selected cluster size. For example, open the `small.sh` script if you're going to deploy a small-sized cluster.
   c. Locate the statements that set the `MACHINE` and `DS_MACHINE` environment variables.
   d. Your quotas need to let you allocate six machines each of `MACHINE` and `DS_MACHINE` types in your region. If your quotas are too low, request and wait for a quota increase from Google Cloud before attempting to create your CDM cluster.

Next Step

- ✓ Install third-party software
- ✓ Set up a Google Cloud project
- ❑ Get the forgeops repository

- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

   ```
   $ cd forgeops
   $ git checkout release/7.1-20240223
   ```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Install third-party software
- ✓ Set up a Google Cloud project
- ✓ Get the forgeops repository
- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Kubernetes Cluster Creation

ForgeRock provides shell scripts based on the Google Cloud SDK to use for GKE cluster creation. Use them when you deploy the CDM. After you've finished deploying the CDM, you can use the CDM as a sandbox to explore a different infrastructure-as-code solution, if you like.

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

Here are the steps the Cloud Deployment Team followed to create a Kubernetes cluster on GKE:

1. Create the cluster:
   a. Change to the directory that contains the cluster creation script:

      ```
      $ cd /path/to/forgeops/cluster/gke
      ```

   b. Source the script that contains the configuration for your cluster size. For example:

      ```
      $ source ./small.sh
      ```

   c. Run the cluster creation script[8]:

      ```
      $ ./cluster-up.sh
      ```

      If you're prompted to install Google Cloud SDK beta components, enter **Y** to install them.

      The script creates:

      - The cluster
      - The `ds-pool` node pool (for large clusters only)
      - The `fast` storage class
      - The `prod` namespace
      - The `cluster-admin-binding` cluster role binding

   d. To verify that the script created the cluster, log in to the Google Cloud console. Select the Kubernetes Engine option. You should see the new cluster in the list of Kubernetes clusters.
   e. Run the **`kubectx`** command.

      The output should contain your newly created cluster and any existing clusters.

The current context should be set to the context for your new cluster.

2. Set context to the `prod` namespace:

```
$ kubens prod
```

3. Check the status of the pods in your cluster until all the pods are ready:

   a. List all the pods in the cluster:

```
$ kubectl get pods --all-namespaces
NAMESPACE                              NAME
READY   STATUS    RESTARTS   AGE
cnrm-system                            cnrm-deletiondefender-0
1/1     Running   0          6m52s
cnrm-system                            cnrm-resource-stats-
recorder-7cc75c4d59-tjwvx    2/2     Running   0
6m52s
cnrm-system                            cnrm-webhook-manager-
656ffb69d5-6d7cp            1/1     Running   0
6m52s
configconnector-operator-system   configconnector-
operator-0                       1/1     Running   0
7m46s
kube-system                            event-exporter-gke-
59b99fdd9c-qtvzw             2/2     Running   0
7m49s
kube-system                            fluentbit-gke-dm42n
2/2     Running   0          6m37s
. . .
```

   b. Review the output. Deployment is complete when:

      - The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
      - All entries in the `STATUS` column indicate `Running` or `Completed`.

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

Next Step

- ✔ Install third-party software
- ✔ Set up a Google Cloud project
- ✔ Get the forgeops repository
- ✔ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator

- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Secret Agent Operator

Install ForgeRock's Secret Agent operator before you deploy the CDM.

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 manage secrets in production.

See Secret Agent Operator for further details on the Secret Agent operator.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore secret management options.

To install the Secret Agent operator in your cluster:

```
$ kubectl apply -f https://github.com/ForgeRock/secret-
agent/releases/latest/download/secret-agent.yaml
namespace/secret-agent-system created
customresourcedefinition.apiextensions.k8s.io/secretagentconfigura
tions.secret-agent.secrets.forgerock.io created
mutatingwebhookconfiguration.admissionregistration.k8s.io/secret-
agent-mutating-webhook-configuration created
serviceaccount/secret-agent-manager-service-account created
role.rbac.authorization.k8s.io/secret-agent-leader-election-role
created
clusterrole.rbac.authorization.k8s.io/secret-agent-manager-role
created
rolebinding.rbac.authorization.k8s.io/secret-agent-leader-
election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/secret-agent-manager-
rolebinding created
service/secret-agent-webhook-service created
deployment.apps/secret-agent-controller-manager created
validatingwebhookconfiguration.admissionregistration.k8s.io/secret
-agent-validating-webhook-configuration created
```

Next Step

- ✓ Install third-party software
- ✓ Set up a Google Cloud project

- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

NGINX Ingress Controller

Use the NGINX ingress controller when you deploy the CDM.

Remember, the CDM is a reference implementation and not for production use. When you create a project plan, you'll need to determine which ingress controller to use in production.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different ingress controller.

To deploy an NGINX ingress controller in a GKE cluster:

1. Verify that you initialized your cluster by performing the steps in Kubernetes Cluster Creation.

   If you did not set up your cluster using this technique, the cluster might be missing some required configuration.

2. Deploy the NGINX ingress controller in your cluster:

   ```
   $ /path/to/forgeops/bin/ingress-controller-deploy.sh --gke
   Deploying Ingress Controller to GKE…

   namespace/nginx created
   Detected cluster of type: small
   Setting ingress pod count to 1
   "ingress-nginx" has been added to your repositories
   Release "ingress-nginx" does not exist. Installing it now.
   NAME: ingress-nginx
   LAST DEPLOYED: Mon May 10 14:15:40 2021
   NAMESPACE: nginx
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   . . .
   ```

3. Check the status of the pods in the `nginx` namespace until all the pods are ready:

```
$ kubectl get pods --namespace nginx
NAME                                        READY   STATUS
RESTARTS   AGE
ingress-nginx-controller-d794bb476-xxx6j    1/1     Running
0          4m38s
```

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

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

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

5. Add an entry to the `/etc/hosts` file to resolve the deployment FQDN used by the platform UIs and APIs. For example:

```
ingress-ip-address prod.iam.example.com
```

For *`ingress-ip-address`*, specify the external IP address of the ingress controller service in the previous command.

Next Step

- ✓ Install third-party software
- ✓ Set up a Google Cloud project
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Certificate Manager

Use cert-manager when you deploy the CDM.

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 manage certificates in production.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different certificate manager.

To deploy the Certificate Manager:

```
$ /path/to/forgeops/bin/certmanager-deploy.sh
customresourcedefinition.apiextensions.k8s.io/certificaterequests.
cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert
-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-
manager.io created
namespace/cert-manager created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector
created
. . .
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
deployment.apps/cert-manager-webhook created
mutatingwebhookconfiguration.admissionregistration.k8s.io/cert-
manager-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/cert-
manager-webhook created
deployment.apps/cert-manager-webhook condition met
clusterissuer.cert-manager.io/default-issuer created
secret/certmanager-ca-secret created
```

After you've deployed the Certificate Manager, check the status of the pods in the `cert-manager` namespace until all the pods are ready:

```
$ kubectl get pods --namespace cert-manager
NAME                                         READY  STATUS
RESTARTS AGE
cert-manager-6d5fd89bdf-khj5w                1/1    Running
0        3m57s
cert-manager-cainjector-7d47d59998-h5b48     1/1    Running
0        3m57s
cert-manager-webhook-6559cc8549-8vdtp        1/1    Running
0        3m56s
```

Next Step

- ✓ Install third-party software
- ✓ Set up a Google Cloud project
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ✓ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Prometheus, Grafana, and Alertmanager

IMPORTANT

There's an outstanding issue (CLOUD-4064) logged against the **`bin/prometheus-deploy.sh`** script described on this page. Do not attempt to run this script until this issue has been resolved.

Use Prometheus, Grafana, and Alertmanager when you deploy the CDM.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different monitoring and alerting framework.

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.

To deploy Prometheus, Grafana, and Alertmanager:

1. Deploy Prometheus, Grafana, and Alertmanager in your cluster:

```
$ /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          2m49s
prometheus-operator-grafana-7fb6687584-g7mll                  2/2
Running   0          2m55s
prometheus-operator-kube-p-operator-648f6dc47f-qsdkl          1/1
Running   0          2m55s
prometheus-operator-kube-state-metrics-957fc5f95-jwvdz        1/1
Running   0          2m55s
prometheus-operator-prometheus-node-exporter-dd4vz            1/1
Running   0          2m55s
prometheus-operator-prometheus-node-exporter-dtblt            1/1
Running   0          2m56s
prometheus-operator-prometheus-node-exporter-jvlj5            1/1
Running   0          2m55s
prometheus-prometheus-operator-kube-p-prometheus-0            2/2
Running   1          2m49s
```

Next Step

- ✓ Install third-party software
- ✓ Set up a Google Cloud project
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ✓ Deploy certificate manager
- ✓ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

`docker push` Setup

In the deployment environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your GKE cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

Perform the following steps to let Skaffold to push Docker images to a registry accessible to your cluster:

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
   ```

3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location for your project and the Kubernetes context. Use your project ID (*not* your project name) and the Kubernetes context that you obtained in the previous step:

   ```
   $ skaffold config set default-repo gcr.io/my-project-ID -k my-kubernetes-context
   ```

Next Step

You've completed all the setup tasks for GKE. Now you're ready to deploy the platform in your new cluster:

✓ 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

Environment Setup: EKS

Before deploying the CDM, you must set up your local computer, configure your AWS account, and create an EKS cluster.

▼ Windows users

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: Ubuntu 19.10 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.

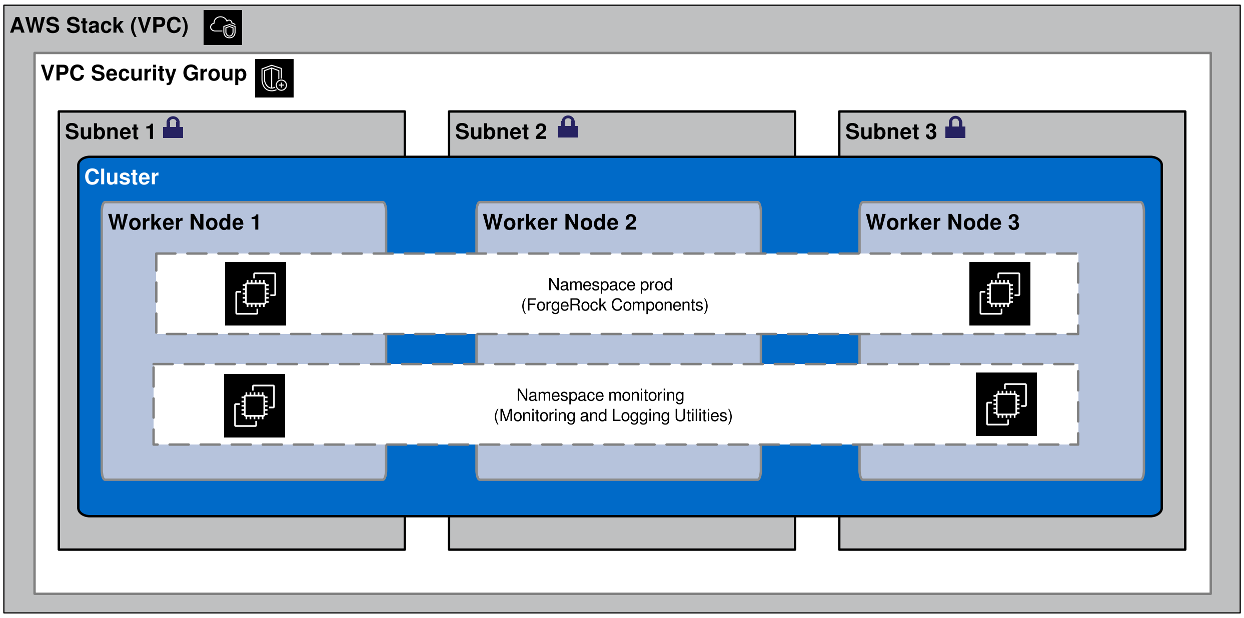
Environment Setup Checklist

- ❑ Understand CDM architecture on EKS
- ❑ Install third-party software
- ❑ Set up your AWS environment
- ❑ Get the forgeops repository
- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

After you've completed all of these environment setup tasks, you'll be ready to deploy the ForgeRock Identity Platform on your new Kubernetes cluster.

Architecture Overview

The following diagram provides an overview of a CDM deployment in the Amazon EKS environment:



- An AWS stack template is used to create a virtual private cloud (VPC).
- Three subnets are configured across three availability zones.
- A Kubernetes cluster is created over the three subnets.
- Three worker nodes are created within the cluster. The worker nodes contain the computing infrastructure to run the CDM components.
- A local file system is mounted to the DS pod for storing directory data backup.

Next Step

- ✓ Understand CDM architecture on EKS
- ❏ Install third-party software
- ❏ Set up your AWS environment
- ❏ Get the forgeops repository
- ❏ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Third-Party Software

Before installing the CDM, you must obtain non-ForgeRock software and install it on your local computer.

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

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

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
Helm	3.8.1	`helm`
Amazon AWS Command Line Interface	2.8.9	`awscli`
AWS IAM Authenticator for Kubernetes	0.5.5	`aws-iam-authenticator`
eksctl	0.117.0	`eksctl`
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ❑ Set up your AWS environment
- ❑ Get the forgeops repository
- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

AWS Environment Setup

This page outlines the steps that the Cloud Deployment Team took when setting up AWS before deploying the CDM.

Perform these steps before you deploy the CDM:

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

      Remember, the CDM is a reference implementation and is not for production use. The policies you create in this procedure are suitable for the CDM. When you create a project plan, you'll need to determine how to configure AWS permissions.
   e. Assign one or more AWS users who will set up CDM to the `cdm-users` group.
2. If you haven't already done so, set up your **aws** command-line interface environment using the **`aws configure`** command.
3. Verify that your AWS user is a member of the `cdm-users` group:

```
$ aws iam list-groups-for-user --user-name my-user-name --
output json
{
    "Groups": [
        {
            "Path": "/",
            "GroupName": "cdm-users",
            "GroupId": "ABCDEFGHIJKLMNOPQRST",
```

```
            "Arn": "arn:aws:iam::048497731163:group/cdm-
users",
            "CreateDate": "2020-03-11T21:03:17+00:00"
        }
    ]
}
```

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

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

5. Determine the region where you'll deploy the CDM. Then, configure that region as your default AWS region. For example:

```
$ aws configure set default.region us-east-1
```

   Note the following:

   - The region must support Amazon EKS.
   - The region must have at least three availability zones. (Use the **`aws ec2 describe-availability-zones --region region-name`** command to determine the availability zones for an AWS region.)
   - Objects required for your EKS cluster should reside in the same region to get the best performance. To make sure that AWS objects are created in the correct region, be sure to set your default region as shown above.

6. Determine your cluster size: small, medium, or large.
7. Ensure that the cluster creation script will support your region:
   a. Change to the `/path/to/forgeops/cluster/eks` directory.
   b. Open the configuration file for your selected cluster size. For example, open the `small.yaml` file if you're going to deploy a small-sized cluster.
   c. Specify your region as the `metadata` / `region` value.

d. Specify the three availability zones in your region as the `availabilityZones` values.

8. Ensure that your region has an adequate CPU quota for the CDM:

   a. Change to the `/path/to/forgeops/cluster/eks` directory.

   b. Open the YAML file that contains the configuration for your selected cluster size. For example, open the `small.yaml` file if you're going to deploy a small-sized cluster.

   c. Locate the two `instanceType` statements in the `nodeGroups` section.

   d. Your quotas need to let you allocate six machines of each type in your region. If your quotas are too low, request and wait for a quota increase from AWS before attempting to create your CDM cluster.

9. Create Amazon ECR repositories for the ForgeRock Identity Platform Docker images:

```
$ for i in am am-config-upgrader amster ds-cts ds-idrepo idm
ldif-importer ig ds-proxy;
do
  aws ecr create-repository --repository-name "forgeops/${i}";
done

{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:. .
.:repository/forgeops/am",
        "registryId": ". . .",
        "repositoryName": "forgeops/am",
        "repositoryUri": ". . . .dkr.ecr.us-east-
1.amazonaws.com/forgeops/am",
        "createdAt": "2020-08-03T14:19:54-08:00"
    }
}
. . .
```

Next Step

- ✔ Understand CDM architecture on EKS
- ✔ Install third-party software
- ✔ Set up your AWS environment
- ❑ Get the forgeops repository
- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller

- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

   ```
   $ cd forgeops
   $ git checkout release/7.1-20240223
   ```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ✓ Set up your AWS environment
- ✓ Get the forgeops repository
- ❏ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Kubernetes Cluster Creation

ForgeRock provides shell scripts based on AWS CloudFormation to use for EKS cluster creation. Use them when you deploy the CDM. After you've finished deploying the CDM, you can use the CDM as a sandbox to explore a different infrastructure-as-code solution, if you like.

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

Here are the steps the Cloud Deployment Team followed to create a Kubernetes cluster on EKS:

1. Create your cluster:

   a. Change to the directory that contains the cluster creation script:

   ```
   $ cd /path/to/forgeops/cluster/eks
   ```

   b. Run the cluster creation script. Specify the YAML file that contains the configuration for your cluster size. For example[9]:

   ```
   $ ./cluster-up.sh small.yaml
   ```

   To verify that the cluster has been created, log in to the AWS console. Select the EKS service link. You should see the new cluster in the list of Amazon EKS clusters.

   c. Run the **kubectx** command:

   ```
   $ kubectx
   . . .
   user.name@small.us-east-1.eksctl.io
   ```

   The output should contain your newly created cluster and any existing clusters.

   The current context should be set to the context for your new cluster.

2. Set context to the `prod` namespace:

   ```
   $ kubens prod
   ```

3. Check the status of the pods in your cluster until all the pods are ready:

   a. List all the pods in the cluster:

   ```
   $ kubectl get pods --all-namespaces
   NAMESPACE     NAME                                  READY
   ```

```
STATUS     RESTARTS   AGE
kube-system   aws-node-4fzmz                        1/1
Running   0          9m28s
kube-system   aws-node-btkf9                        1/1
Running   0          9m23s
kube-system   aws-node-gbw6b                        1/1
Running   0          9m3s
kube-system   aws-node-gtp2x                        1/1
Running   0          9m1s
kube-system   aws-node-klv4t                        1/1
Running   0          9m28s
kube-system   aws-node-znjt6                        1/1
Running   0          9m4s
kube-system   coredns-75b44cb5b4-nm824              1/1
Running   0          25m
kube-system   coredns-75b44cb5b4-xlwpn              1/1
Running   0          25m
kube-system   kube-proxy-4gxgw                      1/1
Running   0          9m3s
kube-system   kube-proxy-bb4sr                      1/1
Running   0          9m28s
kube-system   kube-proxy-knw77                      1/1
Running   0          9m28s
kube-system   kube-proxy-kwq22                      1/1
Running   0          9m1s
kube-system   kube-proxy-ptpnf                      1/1
Running   0          9m23s
kube-system   kube-proxy-zt4t2                      1/1
Running   0          9m4s
kube-system   metrics-server-5f956b6d5f-nhpmd   1/1
Running   0          12m8s
```

b. Review the output. Deployment is complete when:

- The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
- All entries in the `STATUS` column indicate `Running` or `Completed`.

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

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ✓ Set up your AWS environment

- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Secret Agent Operator

Install ForgeRock's Secret Agent operator before you deploy the CDM.

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 manage secrets in production.

See Secret Agent Operator for further details on the Secret Agent operator.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore secret management options.

To install the Secret Agent operator in your cluster:

```
$ kubectl apply -f https://github.com/ForgeRock/secret-
agent/releases/latest/download/secret-agent.yaml
namespace/secret-agent-system created
customresourcedefinition.apiextensions.k8s.io/secretagentconfigura
tions.secret-agent.secrets.forgerock.io created
mutatingwebhookconfiguration.admissionregistration.k8s.io/secret-
agent-mutating-webhook-configuration created
serviceaccount/secret-agent-manager-service-account created
role.rbac.authorization.k8s.io/secret-agent-leader-election-role
created
clusterrole.rbac.authorization.k8s.io/secret-agent-manager-role
created
rolebinding.rbac.authorization.k8s.io/secret-agent-leader-
election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/secret-agent-manager-
rolebinding created
service/secret-agent-webhook-service created
deployment.apps/secret-agent-controller-manager created
validatingwebhookconfiguration.admissionregistration.k8s.io/secret
-agent-validating-webhook-configuration created
```

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ✓ Set up your AWS environment
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

NGINX Ingress Controller

Use the NGINX ingress controller when you deploy the CDM.

Remember, the CDM is a reference implementation and not for production use. When you create a project plan, you'll need to determine which ingress controller to use in production.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different ingress controller.

To deploy an NGINX ingress controller in an EKS cluster:

1. Verify that you initialized your cluster by performing the steps in Kubernetes Cluster Creation.

   If you did not set up your cluster using this technique, the cluster might be missing some required configuration.

2. Deploy the NGINX ingress controller in your cluster:

   ```
   $ /path/to/forgeops/bin/ingress-controller-deploy.sh --eks
   Deploying Ingress Controller to EKS…

   namespace/nginx created
   Detected cluster of type: cdm-small
   Setting ingress pod count to 2
   "ingress-nginx" has been added to your repositories
   Release "ingress-nginx" does not exist. Installing it now.
   NAME: ingress-nginx
   LAST DEPLOYED: Wed Dec 22 14:54:43 2021
   NAMESPACE: nginx
   STATUS: deployed
   ```

```
REVISION: 1
TEST SUITE: None
NOTES:
. . .
```

3. Check the status of the pods in the `nginx` namespace until all the pods are ready:

```
$ kubectl get pods --namespace nginx
NAME                                       READY   STATUS
RESTARTS   AGE
ingress-nginx-controller-bb566bf7b-c9kmk   1/1     Running   0
2m45s
ingress-nginx-controller-bb566bf7b-l6dz6   1/1     Running   0
2m45s
```

4. Obtain the DNS name (shown under the EXTERNAL-IP column) of the load balancer :

```
$ kubectl get services --namespace nginx
NAME                                 TYPE           CLUSTER-IP
EXTERNAL-IP                                    PORT(S)
AGE
ingress-nginx-controller             LoadBalancer
10.100.43.88   ac5f2939. . .ca4.elb.us-east-1.amazonaws.com
80:30005/TCP,443:30770/TCP   62s
ingress-nginx-controller-admission   ClusterIP
10.100.2.215   <none>
443/TCP                      62s
```

5. Wait for a couple of minutes for the load balancer to be assigned the external IP address. Then, get the external IP addresses of the load balancer. For example:

```
$ host ac5f2939. . .42d085.elb.us-east-1.amazonaws.com
ac5f2939. . .42d085.elb.us-east-1.amazonaws.com has address
3.210.123.210
```

The **`host`** command returns several IP addresses. You can use any of the IP addresses when you modify your local hosts file in the next step for evaluation purposes. You must create an appropriate entry in your DNS servers for production deployments.

6. Add an entry to the `/etc/hosts` file to resolve the deployment FQDN used by the platform UIs and APIs. For example:

```
ingress-ip-address prod.iam.example.com
```

For *ingress-ip-address*, specify the external IP of the ingress controller service in the previous command.

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ✓ Set up your AWS environment
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Certificate Manager

Use cert-manager when you deploy the CDM.

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 manage certificates in production.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different certificate manager.

To deploy the Certificate Manager:

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

customresourcedefinition.apiextensions.k8s.io/certificaterequests.
cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert
-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-
manager.io created
```

```
namespace/cert-manager created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector
created
. . .
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
deployment.apps/cert-manager-webhook created
mutatingwebhookconfiguration.admissionregistration.k8s.io/cert-
manager-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/cert-
manager-webhook created
deployment.extensions/cert-manager-webhook condition met
clusterissuer.cert-manager.io/default-issuer created
secret/certmanager-ca-secret created
```

After you've deployed the Certificate Manager, check the status of the pods in the `cert-manager` namespace until all the pods are ready:

```
$ kubectl get pods --namespace cert-manager

NAME                                         READY  STATUS
RESTARTS  AGE
cert-manager-6d5fd89bdf-khj5w                1/1    Running
0         3m57s
cert-manager-cainjector-7d47d59998-h5b48     1/1    Running
0         3m57s
cert-manager-webhook-6559cc8549-8vdtp        1/1    Running
0         3m56s
```

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ✓ Set up your AWS environment
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller

- ✓ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Prometheus, Grafana, and Alertmanager

> IMPORTANT
>
> There's an outstanding issue (CLOUD-4064) logged against the **`bin/prometheus-deploy.sh`** script described on this page. Do not attempt to run this script until this issue has been resolved.

Use Prometheus, Grafana, and Alertmanager when you deploy the CDM.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different monitoring and alerting framework.

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.

To deploy Prometheus, Grafana, and Alertmanager:

1. Deploy Prometheus, Grafana, and Alertmanager in your cluster:

```
$ /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:
. . .
pod/prometheus-operator-prometheus-node-exporter-klgw2
condition met
pod/prometheus-operator-prometheus-node-exporter-nv2jn
condition met
pod/prometheus-prometheus-operator-kube-p-prometheus-0
condition met
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          6m20s
prometheus-operator-grafana-5c7677f88b-jzd22                  2/2
Running   0          6m24s
prometheus-operator-kube-p-operator-58cbbf67b5-4lchb          1/1
Running   0          2m1s
prometheus-operator-kube-state-metrics-957fc5f95-fdvgz        1/1
Running   0          6m24s
prometheus-operator-prometheus-node-exporter-6prrn            1/1
Running   0          6m24s
prometheus-operator-prometheus-node-exporter-6wr4b            1/1
Running   0          6m24s
prometheus-operator-prometheus-node-exporter-9qmtw            1/1
Running   0          95s
prometheus-operator-prometheus-node-exporter-g4p7f            1/1
Running   0          110s
prometheus-operator-prometheus-node-exporter-hjf28            1/1
Running   0          6m24s
prometheus-operator-prometheus-node-exporter-nv2jn            1/1
```

```
Running   0          6m24s
prometheus-prometheus-operator-kube-p-prometheus-0      2/2
Running   1          2m1s
```

Next Step

- ✓ Understand CDM architecture on EKS
- ✓ Install third-party software
- ✓ Set up your AWS environment
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ✓ Deploy certificate manager
- ✓ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

`docker push` *Setup*

In the deployment environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your Amazon EKS cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to the shared cluster.
- Skaffold needs to know the location of the Docker registry.

Perform the following steps to let Skaffold to push Docker images to a registry accessible to your cluster:

1. If it's not already running, start Docker on your local computer. For more information, see the Docker documentation.
2. Obtain your 12 digit AWS account ID. You'll need it when you run subsequent steps in this procedure.
3. Log in to Amazon ECR:

```
$ aws ecr get-login-password | docker login --username AWS  \
 --password-stdin my-account-id.dkr.ecr.my-region.amazonaws.com
Login Succeeded
```

ECR login sessions expire after 12 hours. Because of this, you'll need to log in again whenever your login session expires [10].

4. Run the **kubectx** command to obtain the Kubernetes context.
5. Configure Skaffold with your Docker registry location and the Kubernetes context:

```
$ skaffold config \
 set default-repo my-account-id.dkr.ecr.my-region.amazonaws.com/forgeops \
 -k my-kubernetes-context
set value default-repo to my-account-id.dkr.ecr.my-region.amazonaws.com/forgeops
for context my-kubernetes-context
```

Next Step

You've completed all the setup tasks for EKS. Now you're ready to deploy the platform in your new cluster:

- ✓ 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

Environment Setup: AKS

Before deploying the CDM, you must set up your local computer, configure an Azure subscription, and create a AKS cluster.

▼ Windows users

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: Ubuntu 19.10 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.

Environment Setup Checklist

- ❑ Install third-party software
- ❑ Set up an Azure subscription
- ❑ Get the forgeops repository
- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

After you've completed all of these environment setup tasks, you'll be ready to deploy the ForgeRock Identity Platform on your new Kubernetes cluster.

Third-Party Software

Before installing the CDM, you must obtain non-ForgeRock software and install it on your local computer.

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

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

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`

Software	Version	Homebrew package
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
Helm	3.8.1	`helm`
Azure Command Line Interface	2.42.0	`azure-cli`
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Install third-party software
- ❑ Set up an Azure subscription
- ❑ Get the forgeops repository
- ❑ Create a Kubernetes cluster
- ❑ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Azure Subscription Setup

This page outlines the steps that the Cloud Deployment Team took when setting up an Azure subscription before deploying the CDM.

Perform these steps before you deploy the CDM:

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

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

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

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

3. View your current subscription ID:

   ```
   $ az account show
   ```

4. If necessary, set the current subscription ID to the one you will use to deploy the CDM:

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

5. Determine the region where you'll deploy the CDM. Then, set that region as the default location for the Azure CLI. For example:

   ```
   $ az configure --defaults location=westus2
   ```

   When changing the region for deploying the CDM:

   - The region must support AKS.
   - The subscription, resource groups, and resources you create for your AKS cluster must reside in the same region.

6. Determine the cluster size: small, medium, or large.
7. Ensure that your region has an adequate CPU quota for the CDM:
   a. Change to the `/path/to/forgeops/cluster/aks` directory.
   b. Open the script that sets environment variables for your selected cluster size. For example, open the `small.sh` script if you're going to deploy a small-sized cluster.
   c. Locate the statements that set the `VM_SIZE` and `DS_VM_SIZE` environment variables.
   d. Your quotas need to let you allocate six machines of each type in your region. If your quotas are too low, request and wait for a quota increase from Microsoft before attempting to create your CDM cluster.
8. The CDM uses Azure Container Registry (ACR) for storing Docker images.

   If you do not have a container registry in your subscription, create one.

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ❏ Get the forgeops repository
- ❏ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

`forgeops` Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

   ```
   $ cd forgeops
   $ git checkout release/7.1-20240223
   ```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ✓ Get the forgeops repository
- ❏ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager

- ☐ Deploy Prometheus, Grafana, and Alertmanager
- ☐ Prepare to push Docker images

Kubernetes Cluster Creation

ForgeRock provides shell scripts based on the Azure CLI to use for AKS cluster creation. Use them when you deploy the CDM. After you've finished deploying the CDM, you can use the CDM as a sandbox to explore a different infrastructure-as-code solution, if you like.

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

Here are the steps the Cloud Deployment Team followed to create a Kubernetes cluster on AKS:

1. Set the value of the `ACR_NAME` environment variable to the name of your ACS container registry. For example, *my-container-registry*, **not** *my-container-registry.azurecr.io*:

   ```
   $ export ACR_NAME=my-container-registry
   ```

2. Create the cluster:

   a. Change to the directory that contains the cluster creation script:

      ```
      $ cd /path/to/forgeops/cluster/aks
      ```

   b. Source the script that contains the configuration for your cluster size. For example:

      ```
      $ source ./small.sh
      ```

   c. Run the cluster creation script[11]:

      ```
      $ ./cluster-up.sh
      ```

      The script creates:

      - The cluster
      - The DS node pool (for large clusters only)
      - The `fast` storage class
      - The `prod` namespace
      - A public static IP address

d. To verify that the script created the cluster, log in to the Azure portal. Select the Kubernetes Engine option. You should see the new cluster in the list of Kubernetes clusters.

e. Run the **kubectx** command.

   The output should contain your newly created cluster and any existing clusters.

   The current context should be set to the context for your new cluster.

3. Check the status of the pods in your cluster until all the pods are ready:

   a. List all the pods in the cluster:

```
$ kubectl get pods --all-namespaces
NAMESPACE     NAME
READY   STATUS    RESTARTS   AGE
kube-system   azure-cni-networkmonitor-7jhxs
1/1     Running   0          22m
kube-system   azure-cni-networkmonitor-7js4b
1/1     Running   0          20m
. . .
kube-system   azure-ip-masq-agent-2vdnb
1/1     Running   0          22m
. . .
```

   b. Review the output. Deployment is complete when:

      - The `READY` column indicates all running containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
      - All entries in the `STATUS` column indicate `Running` or `Completed`.

   c. If necessary, continue to query your cluster’s status until all the pods are ready.

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ❏ Install the Secret Agent operator
- ❏ Deploy the NGINX ingress controller
- ❏ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Secret Agent Operator

Install ForgeRock's Secret Agent operator before you deploy the CDM.

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 manage secrets in production.

See Secret Agent Operator for further details on the Secret Agent operator.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore secret management options.

To install the Secret Agent operator in your cluster:

```
$ kubectl apply -f https://github.com/ForgeRock/secret-
agent/releases/latest/download/secret-agent.yaml
namespace/secret-agent-system created
customresourcedefinition.apiextensions.k8s.io/secretagentconfigura
tions.secret-agent.secrets.forgerock.io created
mutatingwebhookconfiguration.admissionregistration.k8s.io/secret-
agent-mutating-webhook-configuration created
serviceaccount/secret-agent-manager-service-account created
role.rbac.authorization.k8s.io/secret-agent-leader-election-role
created
clusterrole.rbac.authorization.k8s.io/secret-agent-manager-role
created
rolebinding.rbac.authorization.k8s.io/secret-agent-leader-
election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/secret-agent-manager-
rolebinding created
service/secret-agent-webhook-service created
deployment.apps/secret-agent-controller-manager created
validatingwebhookconfiguration.admissionregistration.k8s.io/secret
-agent-validating-webhook-configuration created
```

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ❑ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager

- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

NGINX Ingress Controller

Use the NGINX ingress controller when you deploy the CDM.

Remember, the CDM is a reference implementation and not for production use. When you create a project plan, you'll need to determine which ingress controller to use in production.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different ingress controller.

To deploy an NGINX ingress controller in an AKS cluster:

1. Verify that you initialized your cluster by performing the steps in Kubernetes Cluster Creation.

   If you did not set up your cluster using this technique, the cluster might be missing some required configuration.

2. Deploy the NGINX ingress controller in your cluster:

   ```
   $ /path/to/forgeops/bin/ingress-controller-deploy.sh --aks
   namespace/nginx created
   Release "nginx-ingress" does not exist. Installing it now.
   NAME: nginx-ingress
   . . .
   ```

3. Check the status of the pods in the `nginx` namespace until all the pods are ready:

   ```
   $ kubectl get pods --namespace nginx
   NAME                                              READY STATUS
   RESTARTS AGE
   nginx-ingress-controller-69b755f68b-9l5n8         1/1
   Running   0       4m38s
   ```

4. Get the ingress controller's public IP address:

   ```
   $ kubectl get services --namespace nginx
   NAME                                 TYPE          CLUSTER-IP
   EXTERNAL-IP   PORT(S)                     AGE
   ingress-nginx-controller             LoadBalancer
   10.0.149.206   20.51.97.25   80:30378/TCP,443:31333/TCP   25s
   ```

```
ingress-nginx-controller-admission   ClusterIP
10.0.87.188    <none>        443/TCP                        25s
```

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

5. Add an entry to the `/etc/hosts` file to resolve the deployment FQDN used by the platform UIs and APIs. For example:

```
ingress-ip-address prod.iam.example.com
```

For *`ingress-ip-address`*, specify the public IP address of the ingress controller service in the previous command.

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ❑ Deploy certificate manager
- ❑ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

Certificate Manager

Use cert-manager when you deploy the CDM.

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 manage certificates in production.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different certificate manager.

To deploy the Certificate Manager:

```
$ /path/to/forgeops/bin/certmanager-deploy.sh
customresourcedefinition.apiextensions.k8s.io/certificaterequests.
cert-manager.io created
```

```
customresourcedefinition.apiextensions.k8s.io/certificates.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert
-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-
manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-
manager.io created
namespace/cert-manager created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector
created
. . .
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
deployment.apps/cert-manager-webhook created
mutatingwebhookconfiguration.admissionregistration.k8s.io/cert-
manager-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/cert-
manager-webhook created
deployment.extensions/cert-manager-webhook condition met
clusterissuer.cert-manager.io/default-issuer created
secret/certmanager-ca-secret created
```

After you've deployed the Certificate Manager, check the status of the pods in the `cert-manager` namespace until all the pods are ready:

```
$ kubectl get pods --namespace cert-manager
NAME                                            READY STATUS
RESTARTS AGE
cert-manager-6d5fd89bdf-khj5w                   1/1   Running
0        3m57s
cert-manager-cainjector-7d47d59998-h5b48        1/1   Running
0        3m57s
cert-manager-webhook-6559cc8549-8vdtp           1/1   Running
0        3m56s
```

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ✓ Deploy certificate manager
- ❏ Deploy Prometheus, Grafana, and Alertmanager
- ❏ Prepare to push Docker images

Prometheus, Grafana, and Alert Manager

> IMPORTANT
>
> There's an outstanding issue (CLOUD-4064) logged against the **`bin/prometheus-deploy.sh`** script described on this page. Do not attempt to run this script until this issue has been resolved.

Use Prometheus, Grafana, and Alertmanager when you deploy the CDM.

After you've finished deploying the CDM, you can use the CDM as a sandbox to explore deployment with a different monitoring and alerting framework.

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.

To deploy Prometheus, Grafana, and Alertmanager:

1. Deploy Prometheus, Grafana, and Alertmanager in your cluster:

```
$ /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
. . .
Update Complete. ⎈Happy Helming!⎈
Release "prometheus-operator" does not exist. Installing it
now.
NAME: prometheus-operator
```

```
LAST DEPLOYED: . . .
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
. . .
pod/prometheus-operator-prometheus-node-exporter-lj6qb
condition met
pod/prometheus-operator-prometheus-node-exporter-tblbz
condition met
pod/prometheus-prometheus-operator-kube-p-prometheus-0
condition met
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          5m33s
prometheus-operator-grafana-76985fcf94-gm557              2/2
Running   0          5m39s
prometheus-operator-kube-p-operator-7884fb6cf7-7vtkt      1/1
Running   0          5m39s
prometheus-operator-kube-state-metrics-847485d6bb-jgkk9   1/1
Running   0          5m39s
prometheus-operator-prometheus-node-exporter-44fpz        1/1
Running   0          5m39s
prometheus-operator-prometheus-node-exporter-cdl4g        1/1
Running   0          5m39s
prometheus-operator-prometheus-node-exporter-fvvqq        1/1
Running   0          5m39s
prometheus-operator-prometheus-node-exporter-gv5lq        1/1
Running   0          5m39s
prometheus-operator-prometheus-node-exporter-lj6qb        1/1
Running   0          5m39s
prometheus-operator-prometheus-node-exporter-tblbz        1/1
Running   0          5m39s
```

```
prometheus-prometheus-operator-kube-p-prometheus-0       2/2
Running   1          5m33s
```

Next Step

- ✓ Install third-party software
- ✓ Set up an Azure subscription
- ✓ Get the forgeops repository
- ✓ Create a Kubernetes cluster
- ✓ Install the Secret Agent operator
- ✓ Deploy the NGINX ingress controller
- ✓ Deploy certificate manager
- ✓ Deploy Prometheus, Grafana, and Alertmanager
- ❑ Prepare to push Docker images

`docker push` *Setup*

In the deployment environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your AKS cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker repository available to your cluster.
- Skaffold needs to know the location of the Docker repository.

Perform the following steps to let Skaffold to push Docker images to a registry accessible to your cluster:

1. If it's not already running, start Docker on your local computer. For more information, see the Docker documentation.
2. If you don't already have the name of the container registry that will hold ForgeRock Docker images, obtain it from your Azure administrator.
3. Log in to your container registry:

```
$ az acr login --name registry-name
```

Azure repository logins expire after 4 hours. Because of this, you'll need to log in to ACR whenever your login session expires [12].

4. Run the **kubectx** command to obtain the Kubernetes context.
5. Configure Skaffold with your Docker repository location and Kubernetes context:

```
$ skaffold config \
 set default-repo registry-name.azurecr.io/cdm -k my-
kubernetes-context
```

For example:

```
$ skaffold config set default-repo my-container-
registry.azurecr.io/cdm -k small
```

Next Step

You've completed all the setup tasks for AKS. Now you're ready to deploy the platform in your new cluster:

- ✓ 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 Deployment

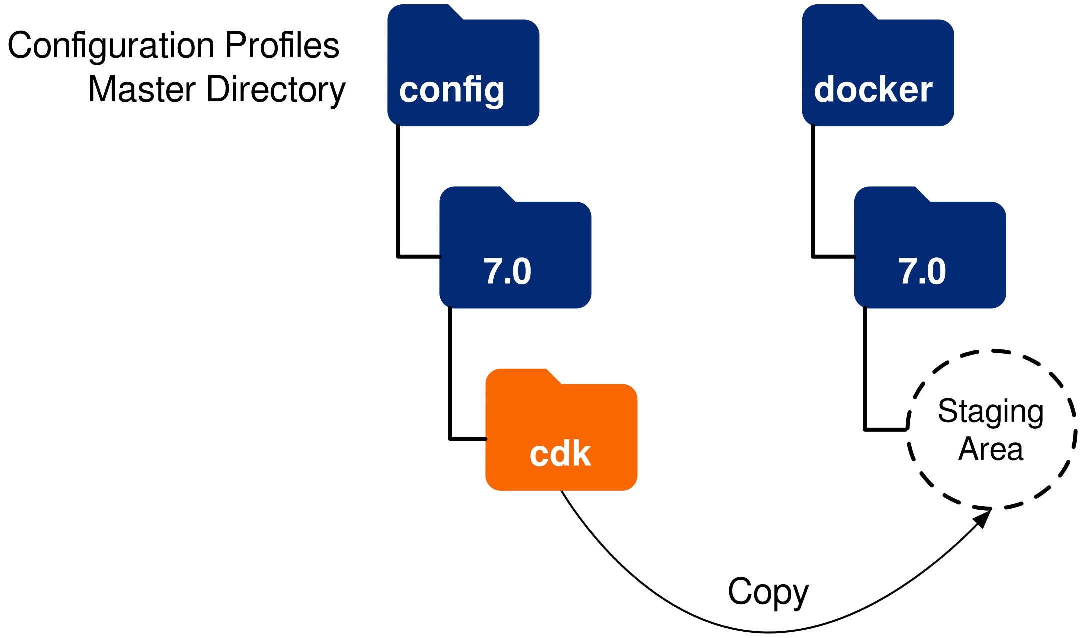
Now that you've set up your deployment environment following the instructions in the Environment Setup section for your cloud platform, you're ready to deploy the CDM.

To deploy the CDM in your Kubernetes cluster using artifacts from the `forgeops` repository:

1. Initialize the staging area for configuration profiles with the canonical CDK configuration profile [13] for the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops/bin
$ ./config.sh init --profile cdk
```

The **config.sh init** command copies the canonical CDK configuration profile from the master directory for configuration profiles to the staging area:



For more information about the management of ForgeRock Identity Platform configuration profiles in the `forgeops` repository, see Configuration Profiles.

2. Configure secrets for the ForgeRock Identity Platform:

   a. Make sure that context is set to the `prod` namespace:

   ```
   $ kubens prod
   ```

   b. Deploy the secrets:

   ```
   $ cd /path/to/forgeops/kustomize/base/secrets
   $ kubectl apply --filename secret_agent_config.yaml
   ```

   c. Verify that all the ForgeRock Identity Platform secrets have been created:

   ```
   $ kubectl get sac
   NAME            STATUS      NUMSECRETS   NUMK8SSECRETS
   forgerock-sac   Completed   14           14
   ```

   When the `forgerock-sac` entry reaches `Completed` status, all the secrets have been created.

3. Change to the `/path/to/forgeops` directory and execute the **`skaffold run`** command. For example:

   ```
   $ cd /path/to/forgeops
   $ skaffold run --profile small
   ```

4. Check the status of the pods in the `prod` namespace until all the pods are ready:

   a. Run the **`kubectl get pods`** command:

```
$ kubectl get pods
NAME                           READY   STATUS
RESTARTS   AGE
admin-ui-69bc8b89bb-dtmj8      1/1     Running     0
3m30s
am-cfc95954d-wqz6d             1/1     Running     0
3m29s
am-cfc95954d-dfl8h             1/1     Running     0
3m21s
amster-j87dl                   0/1     Completed   0
3m27s
ds-cts-0                       1/1     Running     0
3m28s
ds-cts-1                       1/1     Running     0
2m55s
ds-cts-2                       1/1     Running     0
2m21s
ds-idrepo-0                    1/1     Running     0
3m28s
ds-idrepo-1                    1/1     Running     0
2m32s
end-user-ui-6985574b49-dz8t9   1/1     Running     0
3m29s
idm-57b6b86b98-hl8mj           1/1     Running     0
3m29s
idm-57b6b86b98-klj8r           1/1     Running     0
3m29s
ldif-importer-m6n6x            0/1     Completed   0
3m27s
login-ui-64b994b944-9qv7n      1/1     Running     0
3m29s
```

b. 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].
- Two AM and IDM pods are present.
- The initial loading jobs (`amster` and `ldif-importer`) have reached `Completed` status.

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

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

UI and API Access

This page shows you how to access and monitor the ForgeRock 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 normal interfaces:

- For AM, the console and REST APIs.
- For IDM, the Admin UI 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 console:

1. Make sure that the `prod` namespace is the current namespace:

   ```
   $ kubens prod
   ```

2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./print-secrets amadmin
   vr58qt11ihoa31zfbjsdxxrqryfw0s31
   ```

3. Open a new window or tab in a web browser.
4. Go to https://prod.iam.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 ForgeRock Identity Platform UI appears in the browser.

6. Select Native Consoles > Access Management.

   The AM console 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://prod.iam.example.com/am/json/realms/root/authenticate
"


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

IDM Services

To access the IDM Admin UI:

1. Make sure that the `prod` namespace is the current namespace:

```
$ kubens prod
```

2. Obtain the `amadmin` user's password:

```
$ cd /path/to/forgeops/bin
$ ./print-secrets amadmin
vr58qt11ihoa31zfbjsdxxrqryfw0s31
```

3. Open a new window or tab in a web browser.
4. Go to https://prod.iam.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 ForgeRock 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 **`print-secrets`** 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:
   a. 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://prod.iam.example.com/am/json/realms/root/authenti
cate'
{
 "tokenId":" AQIC5wM. . .TU3OQ* ",
```

```
 "successUrl":"/am/console",
 "realm":"/"}
```

b. 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://prod.iam.example.com/am/oauth2/realms/root/author
ize?
redirect_uri=https://prod.iam.example.com/platform/appAuth
HelperRedirect.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://prod.iam.example.com/platform/appAuthHelperRedirec
t.html
 ?
code=3cItL9G52DIiBdfXRngv2_dAaYM&iss=http://prod.iam.examp
le.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
```

c. 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://prod.iam.example.com/platform/appAut
hHelperRedirect.html" \

"https://prod.iam.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://prod.iam.example.com/openidm/config
{
 "_id":"",
 "configurations":
  [
   {
    "_id":"ui.context/admin",
    "pid":"ui.context.4f0cb656-0b92-44e9-a48b-76baddda03ea",
    "factoryPid":"ui.context"
    },
    . . .
```

```
    ]
}
```

Directory Services

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 **print-secrets dsadmin** 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: GKE

To remove the CDM from GKE when you're done working with it:

1. Run the **skaffold delete** command to shut down your deployment and remove it from your namespace. For example:

```
$ cd /path/to/forgeops
$ skaffold delete --profile small
Cleaning up…
. . .
```

2. Remove your cluster:

   a. Change to the directory that contains the cluster removal script:

   ```
   $ cd /path/to/forgeops/cluster/gke
   ```

   b. Source the script that contains the configuration for your cluster size. For example:

```
$ source ./small.sh
```

c. Run the cluster removal script:

```
$ ./cluster-down.sh
The "small" cluster will be deleted. This action cannot be
undone.
Press any key to continue, or CTRL+C to quit
```

i. Press Enter to proceed with cluster and data removal.

```
Getting the cluster credentials for small in Zone my-
region-a
Fetching cluster endpoint and auth data.
kubeconfig entry generated for small.
Do you want to delete all PVCs allocated by this
cluster (recommended for dev clusters)? [Y/N]
```

ii. Enter Y to confirm that you want to remove all the data created by this cluster, or enter N if you don't want to delete the data.

```
Draining all nodes
node/gke-small-default-pool-05ac2046-c1xd cordoned
node/gke-small-default-pool-5dd53b95-mr80 cordoned
node/gke-small-default-pool-e835e4bb-0gwp cordoned
. . .
The following clusters will be deleted.
 - [small] in [my-region]
Deleting cluster small…done.
Deleted
[https://container.googleapis.com/v1/projects/my-
project/zones/my-region/clusters/small].
Check your GCP console for any orphaned project
resources such as disks!
```

3. Run the **kubectx** command.

   The Kubernetes context for the CDM cluster should not appear in the **kubectx** command output.

NOTE

Use the Google Cloud Console to review your use of resources such as Compute Engine disks, static IP addresses, load balancers, and storage buckets. Avoid unnecessary charges by deleting resources that you're no longer using.

CDM Removal: EKS

To remove the CDM from EKS when you're done working with it:

1. Run the **`skaffold delete`** command to shut down your deployment and remove it from your namespace. For example:

   ```
   $ cd /path/to/forgeops
   $ skaffold delete --profile small
   Cleaning up…
   ```

2. Remove your cluster:

   a. Change to the directory that contains the cluster removal script:

   ```
   $ cd /path/to/forgeops/cluster/eks
   ```

   b. Run the cluster removal script. Specify the YAML file that contains the configuration for your cluster size. For example:

   ```
   $ ./cluster-down.sh small.yaml
   The "small" cluster will be deleted. This action cannot be
   undone.
   Press any key to continue, or CTRL+C to quit
   ```

   i. Press Enter to proceed with cluster and data removal.

   ```
   Do you want to delete all PVCs allocated by this
   cluster (recommended for dev clusters)? [Y/N]
   ```

   ii. Enter Y to confirm that you want to remove all the data created by this cluster, or enter N if you don't want to delete the data.

   ```
   Draining all nodes
   node/ip-. . . .ec2.internal cordoned
   node/ip-. . . .ec2.internal cordoned
   . . .
   Deleting cluster "small"
   . . . [i]  eksctl version 0.54.0
   . . . [i]  using region us-east-1
   . . . [i]  deleting EKS cluster "small"
   . . . [i]  deleted 0 Fargate profile(s)
   . . . [✔]  kubeconfig has been updated
   . . . [i]  cleaning up AWS load balancers created by
   Kubernetes objects of Kind Service or Ingress
   ```

```
. . . [i]  2 sequential tasks: { delete nodegroup
"primary", delete cluster control plane "small" }
. . . [i]  will delete stack "eksctl-small-nodegroup-
primary"
. . . [i]  waiting for stack "eksctl-small-nodegroup-
primary" to get deleted

. . .

. . . [i]  will delete stack "eksctl-small-cluster"
. . . [i]  waiting for CloudFormation stack "eksctl-
small-cluster"

. . .

. . . [✔]  all cluster resources were deleted
```

3. Run the **kubectx** command.

   The Kubernetes context for the CDM cluster should not appear in the **kubectx** command output.

CDM Removal: AKS

To remove the CDM from AKS when you're done working with it:

1. Run the **skaffold delete** command to shut down your deployment and remove it from your namespace. For example:

   ```
   $ cd /path/to/forgeops
   $ skaffold delete --profile small
   Cleaning up…
   ```

2. Remove your cluster:

   a. Change to the directory that contains the cluster removal script:

   ```
   $ cd /path/to/forgeops/cluster/aks
   ```

   b. Source the script that contains the configuration for your cluster size. For example:

   ```
   $ source ./small.sh
   ```

   c. Run the cluster removal script:

```
$ ./cluster-down.sh
The "small" cluster will be deleted. This action cannot be
undone.
Press any key to continue, or CTRL+C to quit
```

i. Press Enter to proceed with cluster and data removal.

```
Getting the cluster credentials for small in Zone my-
region-a
Fetching cluster endpoint and auth data.
kubeconfig entry generated for small.
Do you want to delete all PVCs allocated by this
cluster (recommended for dev clusters)? [Y/N]
```

ii. Enter Y to confirm that you want to remove all the data created by this cluster, or enter N if you don't want to delete the data.

```
Draining all nodes
node/aks-primsmall-18811554-vmss000000 cordoned
node/aks-primsmall-18811554-vmss000001 cordoned
node/aks-primsmall-18811554-vmss000002 cordoned
. . .

Deleting AKS cluster small…
Deleting resource group small-res-group…
```

3. Run the **`kubectx`** command.

   The Kubernetes context for the CDM cluster should not appear in the **`kubectx`** command output.

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 is provisioned with sample users. 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 ForgeRock 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 NGINX ingress 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 ForgeRock 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. ForgeRock Identity Platform experts configure AM and IDM using the CDK, and build custom Docker images for the ForgeRock Identity Platform. The Cloud Developer's Kit Documentation provides information about platform configuration tasks.

Cluster configuration. Cloud technology experts configure the Kubernetes cluster that will host the ForgeRock 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 ForgeRock 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 might 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 ForgeRock Identity Gateway to your deployment.

Monitoring

Customize Prometheus monitoring and alerts.

Benchmarks

Run ForgeRock's lightweight benchmarks.

Security

Customize the security features built in to the CDM.

Backup and Restore

Back up and restore CDM data, such as

identities and tokens.

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.

Base Docker Images

ForgeRock provides eleven Docker images for deploying the ForgeRock Identity Platform:

- Seven unsupported, evaluation-only base images:
  - `amster`
  - `am-base`
  - `am-config-upgrader`
  - `ds`
  - `ldif-importer`
  - `idm`
  - `ig`
- Three supported base images that implement the platform's user interface elements:
  - `platform-admin-ui`
  - `platform-enduser-ui`
  - `platform-login-ui`

All of the Docker images are publicly available in ForgeRock's Docker registry.

Which Docker Images Do I Deploy?

- I am a developer using the CDK.
  - UI elements. Deploy the supported images from ForgeRock.
  - Other platform elements. Either deploy the evaluation-only images from ForgeRock or your own base images.
- I am doing a proof-of-concept CDM deployment.
  - UI elements. Deploy the supported images from ForgeRock.
  - Other platform elements. Either deploy the evaluation-only images from ForgeRock or your own base images.

- I am deploying the platform in production.
    - UI elements. Deploy the supported images from ForgeRock.
    - Other platform elements. Deploy your own base images. The evaluation-only images are not supported for production deployments of the ForgeRock Identity Platform.

Your Own Base Docker Images

Perform the following steps to build base images for the seven unsupported, evaluation-only Docker images. After you've built your own base images, push them to your Docker registry:

1. Download the latest versions of the AM, Amster, IDM, and DS `.zip` files from the ForgeRock Download Center. Optionally, you can also download the latest version of the IG `.zip` file.
2. Build the base image for Amster. This image must be available in order to build the base image for AM in the next step:
    a. Make a directory named `amster`.
    b. Unzip the Amster `.zip` file into the new `amster` directory.
    c. Change to the `samples/docker` directory in the expanded `.zip` file output.
    d. 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
    ```

    e. Build the `amster` Docker image:

    ```
    $ docker build --tag amster:7.1.4 .

    [+] Building 45.1s (13/13) FINISHED
     ⇒ [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 gcr.io/forgerock-io/java-
11:latest
1.1s
 ⇒ [1/8] FROM gcr.io/forgerock-io/java-
11:latest@sha256:8828befbed5cb57cd1d0fa3aa01aceb849fe5a71a
a124871207d6c417bf1d8a9
. . .
⇒ exporting to image
1.2s
 ⇒ ⇒ exporting layers
1.2s
 ⇒ ⇒ writing image
sha256:bc474cb6c189e253278f831f178b8d51f63a958a6526c0189fd
f122ddf8f9e52
0.0s
 ⇒ ⇒ naming to docker.io/library/amster:7.1.4
```

3. Build the base image for AM:
   a. Unzip the AM `.zip` file.
   b. Change to the `openam/samples/docker` directory in the expanded `.zip` file output.
   c. Run the **`setup.sh`** script:

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

   d. Change to the `images/am-empty` directory.
   e. Build the `am-empty` Docker image:

```
$ docker build --tag am-empty:7.1.4 .

[+] Building 81.9s (31/31) FINISHED
 ⇒ [internal] load build definition from Dockerfile
0.0s
 ⇒ ⇒ transferring dockerfile: 3.55kB
0.0s
 ⇒ [internal] load .dockerignore
0.0s
 ⇒ ⇒ transferring context: 2B
0.0s
 ⇒ [internal] load metadata for
docker.io/library/tomcat:9-jdk11-adoptopenjdk-hotspot
```

```
1.6s
 ⇒ [internal] load build context
4.1s
 ⇒ ⇒ transferring context: 204.45MB
4.0s
 ⇒ [am-tomcat  1/13] FROM docker.io/library/tomcat:9-
jdk11-adoptopenjdk-
hotspot@sha256:1cac44feaa72ca0937995084a25512c43161c84bcb6
51ff4623977d96a31fa89
. . .
 ⇒ exporting to image
1.8s
 ⇒ ⇒ exporting layers
1.8s
 ⇒ ⇒ writing image
sha256:a95979051437a80dddef5829531a6d42a79ac55b9cd23b91617
811861b8e0a3d
0.0s
 ⇒ ⇒ naming to docker.io/library/am-empty:7.1.4
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

f. Change to the `../am-base` directory.

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

```
$ docker build --build-arg docker_tag=7.1.4 --tag my-
registry/am-base:7.1.4 .

[+] Building 129.7s (28/28) FINISHED
 ⇒ [internal] load build definition from Dockerfile
0.0s
 ⇒ ⇒ transferring dockerfile: 2.49kB
0.0s
 ⇒ [internal] load .dockerignore
0.0s
 ⇒ ⇒ transferring context: 2B
0.0s
 ⇒ [internal] load metadata for
docker.io/library/amster:7.1.4
0.0s

. . .
 ⇒ [am-base 7/7] COPY --chown=forgerock:root
```

```
scripts/import-pem-certs.sh /home/forgerock/
0.0s
 ⇒ exporting to image
0.3s
 ⇒ ⇒ exporting layers
0.3s
 ⇒ ⇒ writing image
sha256:0a32f2927ccffa54c044acbbe708bff58e97f4652c94223d526
11e4135b7f525
0.0s
 ⇒ ⇒ naming to my-registry/am-base:7.1.4
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

4. 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:7.1.4 my-registry/amster:7.1.4
```

5. Build the `am-config-upgrader` base image:

   a. Change to the `openam` directory in the expanded AM `.zip` file output.

   b. Unzip the `Config-Upgrader-7.1.4.zip` file.

   c. Change to the `amupgrade/samples/docker` directory in the expanded `Config-Upgrader-7.1.4.zip` file output.

   d. 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 .
   ```

   e. Create the base `am-config-upgrader` image:

   ```
   $ docker build . --tag my-registry/am-config-
   upgrader:7.1.4

   [+] Building 7.1s (9/9) FINISHED
    ⇒ [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 gcr.io/forgerock-io/java-
11:latest
0.3s
 ⇒ [internal] load build context
0.4s
 ⇒ ⇒ transferring context: 15.29MB
0.4s
 ⇒ CACHED [1/4] FROM gcr.io/forgerock-io/java-
11:latest@sha256:8828befbed5cb57cd1d0fa3aa01aceb849fe5a71a
a124871207d6c417bf1d8a9
0.0s
. . .
 ⇒ [4/4] COPY build/ /home/forgerock/
0.1s
 ⇒ exporting to image
0.2s
 ⇒ ⇒ exporting layers
0.2s
 ⇒ ⇒ writing image
sha256:98b35624f41081be4981abef8c6d22178ece19282543c7b6bf1
b5616abbc2721
0.0s
 ⇒ ⇒ naming to my-registry/am-config-upgrader:7.1.4
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

6. Build the base image for DS:

   a. Unzip the DS `.zip` file.

   b. Change to the opendj directory in the expanded `.zip` file output.

   c. 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
. . .
```

d. Build the `ds` base image:

```
$ docker build --tag my-registry/ds:7.1.7 .

[+] Building 2.1s (8/8) FINISHED
 ⇒ [internal] load build definition from Dockerfile
0.0s
 ⇒ ⇒ transferring dockerfile: 1.19kB
0.0s
 ⇒ [internal] load .dockerignore
0.0s
 ⇒ ⇒ transferring context: 2B
0.0s
 ⇒ [internal] load metadata for gcr.io/forgerock-io/java-
11:latest
0.3s
 ⇒ [internal] load build context
1.1s
 ⇒ ⇒ transferring context: 60.31MB
1.1s
 ⇒ CACHED [1/3] FROM gcr.io/forgerock-io/java-
11:latest@sha256:8828befbed5cb57cd1d0fa3aa01aceb849fe5a71a
a124871207d6c417bf1d8a9
0.0s
 ⇒ [2/3] COPY --chown=forgerock:root . /opt/opendj/
0.2s
 ⇒ [3/3] WORKDIR /opt/opendj
0.0s
 ⇒ exporting to image
0.3s
 ⇒ ⇒ exporting layers
0.3s
 ⇒ ⇒ writing image
sha256:d7e583ca5d632edcae2da90dbe3cafc336519f4b671ce1ab19e
38c09b06377d9
0.0s
```

```
 ⇒ ⇒ naming to my-registry/ds:7.1.7
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

7. Build the `ldif-importer` base image:
   a. Change to the `/path/to/forgeops/docker/7.0/ldif-importer` directory.
   b. Open the file, `Dockerfile`.
   c. Change the FROM statement—the first line in the file—to reference the `ds` base image you created in the previous step:

   ```
   FROM my-registry/ds:7.1.7
   ```

   d. Save and close the updated file.
   e. Create the base `ldif-importer` image:

   ```
   $ docker build . --tag my-registry/ldif-importer:7.1.7

   [+] Building 7.9s (10/10) FINISHED
    ⇒ [internal] load build definition from Dockerfile
   0.0s
    ⇒ ⇒ transferring dockerfile: 325B
   0.0s
    ⇒ [internal] load .dockerignore
   0.0s
    ⇒ ⇒ transferring context: 2B
   0.0s
    ⇒ [internal] load metadata for my-registry/ds:7.1.7
   0.0s
    ⇒ [internal] load build context
   0.1s
    ⇒ ⇒ transferring context: 3.08kB
   0.0s
    ⇒ [1/5] FROM my-registry/ds:7.1.7
   0.0s
    ⇒ [2/5] COPY debian-buster-sources.list
   /etc/apt/sources.list
   0.0s
    ⇒ [3/5] RUN apt-get update -y && apt-get install -y curl
   7.5s
    ⇒ [4/5] COPY --chown=forgerock:root start.sh /opt/opendj
   0.0s
    ⇒ [5/5] COPY --chown=forgerock:root ds-passwords.sh
   ```

```
/opt/opendj
0.0s
 ⇒ exporting to image
0.2s
 ⇒ ⇒ exporting layers
0.2s
 ⇒ ⇒ writing image
sha256:bdc5e75854c8015d0b5bd6d2a54c6c044cf0fd23ce6ea828b4a
e1a9d04517515
0.0s
 ⇒ ⇒ naming to my-registry/ldif-importer:7.1.7
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

8. Build the base image for IDM:
    a. Unzip the IDM `.zip` file.
    b. Change to the openidm directory in the expanded `.zip` file output.
    c. Build the `idm` base image:

```
$ docker build . --file bin/Custom.Dockerfile --tag my-
registry/idm:7.1.5

[+] Building 12.2s (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 gcr.io/forgerock-io/java-
11:latest
0.3s
 ⇒ [internal] load build context
8.1s
 ⇒ ⇒ transferring context: 314.59MB
8.1s
 ⇒ CACHED [1/4] FROM gcr.io/forgerock-io/java-
11:latest@sha256:8828befbed5cb57cd1d0fa3aa01aceb849fe5a71a
a124871207d6c417bf1d8a9
0.0s
```

```
 ⇒ [2/4] RUN apt-get update &&     apt-get install -y ttf-
dejavu
8.6s
 ⇒ [3/4] COPY --chown=forgerock:root . /opt/openidm
0.9s
 ⇒ [4/4] WORKDIR /opt/openidm
0.0s
 ⇒ exporting to image
2.3s
 ⇒ ⇒ exporting layers
2.3s
 ⇒ ⇒ writing image
sha256:f91dcb5e04c25a0362196d358452a52f75cc5ab0af5764f60e4
8e13ac4fbfd98
0.0s
 ⇒ ⇒ naming to my-registry/idm:7.1.5
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

9. (Optional) Build the base image for IG:

   a. Unzip the IG `.zip` file.

   b. Change to the `identity-gateway` directory in the expanded `.zip` file output.

   c. Build the `ig` base image:

```
$ docker build . --file docker/Dockerfile --tag my-
registry/ig:2023.11.0

[+] Building 3.3s (8/8) FINISHED
 ⇒ [internal] load build definition from Dockerfile
0.0s
 ⇒ ⇒ transferring dockerfile: 931B
0.0s
 ⇒ [internal] load .dockerignore
0.0s
 ⇒ ⇒ transferring context: 2B
0.0s
 ⇒ [internal] load metadata for gcr.io/forgerock-io/java-
11:latest
0.4s
 ⇒ [internal] load build context
1.4s
```

```
 ⇒ ⇒ transferring context: 77.56MB
1.4s
 ⇒ CACHED [1/3] FROM gcr.io/forgerock-io/java-
11:latest@sha256:8828befbed5cb57cd1d0fa3aa01aceb849fe5a71a
a124871207d6c417bf1d8a9
0.0s
 ⇒ [2/3] COPY --chown=forgerock:root . /opt/ig
0.3s
 ⇒ [3/3] RUN mkdir -p "/var/ig"     && chown -R
forgerock:root "/var/ig" "/opt/ig"     && chmod -R g+rwx
"/var/ig" "/opt/ig"
0.7s
 ⇒ exporting to image
0.4s
 ⇒ ⇒ exporting layers
0.4s
 ⇒ ⇒ writing image
sha256:02bf0ff34ee4b1a98ce3d6fbb21ac208b899d930634384018bf
aab1f9a300ac7
0.0s
 ⇒ ⇒ naming to my-registry/ig:2023.11.0
0.0s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

10. Run the **docker images** command to verify that you built the base images:

```
$ docker images

REPOSITORY                     TAG          IMAGE ID
CREATED        SIZE
my-registry/am-base            7.1.4        552073a1c000    1
hour ago     810MB
my-registry/am-config-upgrader 7.1.4        d115125b1c3f    1
hour ago     176MB
my-registry/amster             7.1.4        d9e1c735f415    1
hour ago     760MB
my-registry/ds                 7.1.7        ac8e8ab0fda6    1
hour ago     204MB
my-registry/idm                7.1.5        0cc1b7f70ce6    1
hour ago     486MB
my-registry/ig                 2023.11.0    9728c30c1829    1
hour ago     299MB
my-registry/ldif-importer      7.1.7        1ef5333c4230    1
```

```
hour ago     227MB
. . .
```

11. Push the new base Docker images to your Docker registry.

    See 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.

    However, some Docker registries have different requirements. For example, to push Docker images to Google Container Registry, you use Google Cloud SDK commands instead of using the **`docker push`** command.

    Push the following images:

    - *`my-registry`*`/am-base:7.1.4`
    - *`my-registry`*`/amster:7.1.4`
    - *`my-registry`*`/am-config-upgrader:7.1.4`
    - *`my-registry`*`/ds:7.1.7`
    - *`my-registry`*`/idm:7.1.5`
    - *`my-registry`*`/ldif-importer:7.1.7`

    If you're deploying your own IG base image, also push the *`my-registry`*`/ig:2023.11.0` image.

Developer Dockerfile Changes

After you've pushed your own base images to your Docker registry, update the Dockerfiles that your developers use when creating customized Docker images for the ForgeRock Identity Platform. The Dockerfiles can now reference your own base images instead of the evaluation-only images from ForgeRock.

To change developer Dockerfiles to use your base images:

1. Update the AM Dockerfile:
   a. Change to the `/path/to/forgeops/docker/7.0/am` directory.
   b. Open the file, `Dockerfile`, in that directory.
   c. Change the line:

      ```
      FROM gcr.io/forgerock-io/am-base:7.1.4
      ```

      to:

```
FROM my-registry/am-base:7.1.4
```

2. Make a similar change to the file, `/path/to/forgeops/docker/7.0/amster/Dockerfile`.
3. Make a similar change to the file, `/path/to/forgeops/docker/7.0/ds/cts/Dockerfile`.
4. Make a similar change to the file, `/path/to/forgeops/docker/7.0/ds/idrepo/Dockerfile`.
5. Make a similar change to the file, `/path/to/forgeops/docker/7.0/idm/Dockerfile`.
6. (Optional) Make a similar change to the file, `/path/to/forgeops/docker/7.0/ig/Dockerfile`.

You can now build customized Docker images for the ForgeRock Identity Platform based on your own Docker images and use them in production deployments.

IMPORTANT

Before you run Skaffold again, clear the Skaffold cache. Then, when you run Skaffold, set the `--no-prune` and `--cache-artifacts` options to `false`. Doing so triggers Skaffold to load the new images you just built instead of loading previously cached images. For example:

```
$ rm -rf $HOME/.skaffold/cache
$ skaffold run --no-prune=false --cache-artifacts=false
```

Occasionally, you need to pull the images even after clearing Skaffold's cache.

Deploy IG

IG is not deployed with the CDK or the CDM by default.

To deploy IG after you have deployed the CDK or the CDM:

1. Verify that the CDK or the CDM is up and running.
2. Set the active namespace in your local Kubernetes context to the namespace in which you have deployed the platform components.
3. Deploy IG:

```
$ /path/to/forgeops/bin/cdk install ig
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): ['ig']

secret/openig-secrets-env created
service/ig created
deployment.apps/ig created

Enjoy your deployment!
```

By default, the **cdk install** command uses the latest evaluation-only Docker images for release 7.1 of the platform, available from ForgeRock's public registry.

However, if you have built a custom IG image, the **cdk install** command uses your custom image.

4. Run the **kubectl get pods** command to check the status of the IG pod. Wait until the pod is ready before proceeding to the next step.
5. Verify that IG is running.

If you deployed IG on the CDK:

```
$ curl --insecure -L -X GET
https://dev.example.com/ig/openig/ping -v
Note: Unnecessary use of -X or --request, GET is already
inferred.
*   Trying . . .
* TCP_NODELAY set
. . .
> GET /ig/openig/ping HTTP/2
> Host: dev.example.com
> User-Agent: curl/7.64.1
> Accept: /
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
< HTTP/2 200
< date: Thu, 29 Jul 2021 21:07:44 GMT
<
* Connection #0 to host dev.example.com left intact
* Closing connection 0
```

If you deployed IG on the CDM:

```
$ curl --insecure -L -X GET
https://prod.iam.example.com/ig/openig/ping -v
. . .
```

6. Verify that the reverse proxy to the IDM pod is running.

   If you deployed IG on the CDK:

```
$ curl --insecure -L -X GET
https://dev.example.com/ig/openidm/info/ping -v
Note: Unnecessary use of -X or --request, GET is already
inferred.
*   Trying 192.168.99.155…
* TCP_NODELAY set
* Connected to dev.example.com (192.168.99.155) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/cert.pem
  CApath: none
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
. . .
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer
after upgrade: len=0
. . .
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
< HTTP/2 200
. . .
<
* Connection #0 to host dev.example.com left intact
{"_id":"","_rev":"","shortDesc":"OpenIDM
ready","state":"ACTIVE_READY"}* Closing connection 0
```

   If you deployed IG on the CDM:

```
$ curl --insecure -L -X GET
https://prod.iam.example.com/ig/openidm/info/ping -v
. . .
```

Custom IG Image

The IG configuration provided in the CDK canonical configuration profile is an example, and is not meant for use in production. Remove this configuration and replace it with your own routes before using IG in your environment.

See the IG Deployment Guide for configuring routes.

Prerequisites

Before starting to build and deploy your custom IG image, initialize a new configuration profile and set up your local environment to write Docker images:

1. Initialize a new configuration profile by copying the canonical CDK configuration:

```
$ cd /path/to/forgeops/config/7.0
$ cp -r cdk my-ig
```

2. Configure your environment to write to your Docker registry:

▼ *Minikube*

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

1. Run the **docker-env** command in your shell:

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

2. Stop Skaffold from pushing Docker images to a remote Docker registry:

```
$ skaffold config set --kube-context minikube local-
cluster true
set value local-cluster to true for context minikube
```

▼ *GKE shared cluster*

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. Set up a Docker credential helper:

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

3. Run the **kubectx** command to obtain the Kubernetes context.

4. Configure Skaffold with the Docker registry location you obtained from your cluster administrator and the Kubernetes context you obtained in Context for the Shared Cluster:

```
$ skaffold config set default-repo my-docker-registry --
kube-context my-kubernetes-context
```

▼ *EKS shared cluster*

Set up your local computer to push Docker images to Amazon ECR:

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
   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.
3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location and the Kubernetes context:

   ```
   $ skaffold config set default-repo my-docker-registry --kube-context my-kubernetes-context
   ```

▼ *AKS shared cluster*

Set up your local computer to push Docker images to Azure container registry:

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.
3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location you obtained from your cluster administrator and the Kubernetes context you obtained in Context for the Shared Cluster:

   ```
   $ skaffold config set default-repo my-docker-registry --kube-context my-kubernetes-context
   ```

Build a Custom IG Image and Deploy IG

1. Verify that the CDK is up and running.
2. Configure IG by creating, modifying, or deleting rules in `/path/to/forgeops/config/my-ig/ig/config/routes-service` directory.
3. Copy your customized IG configuration to the staging area:

```
$ cd /path/to/forgeops/config/7.0/my-ig
$ cp -r ./ig /path/to/forgeops/docker/7.0
```

4. Build a new IG image that includes your custom configuration:

```
$ /path/to/forgeops/bin/cdk build ig
Generating tags…
 - ig → ig:0a27bdfea
Checking cache…
 - ig: Not found. Building
Starting build…
Found [minikube] context, using local docker daemon.
Building [ig]…
Sending build context to Docker daemon  55.81kB
Step 1/5 : FROM us-docker.pkg.dev/forgeops-
public/images/ig:2023.11.0
 --→ ba6f8150204e
Step 2/5 : ARG CONFIG_PROFILE=cdk
. . .
Step 5/5 : COPY --chown=forgerock:root . /var/ig
 --→ c173995218a3
Successfully built c173995218a3
Successfully tagged ig:0a27bdfea

Updated the image_defaulter with your new image for ig:
"ig:c173995218a3c55dbca76fff08588153db0693a51ff0904e6adee34b71
63340a"
```

5. Uninstall the previously deployed IG from your CDK:

   a. Set the active namespace in your local Kubernetes context to the namespace in which you have deployed the IG.

   b. Delete IG:

   ```
   $ /path/to/forgeops/bin/cdk delete ig
   Uninstalling component(s): ['ig']
   OK to delete these components? [Y/N] y
   secret "openig-secrets-env" deleted
   service "ig" deleted
   deployment.apps "ig" deleted
   ```

6. Deploy your customized IG image:

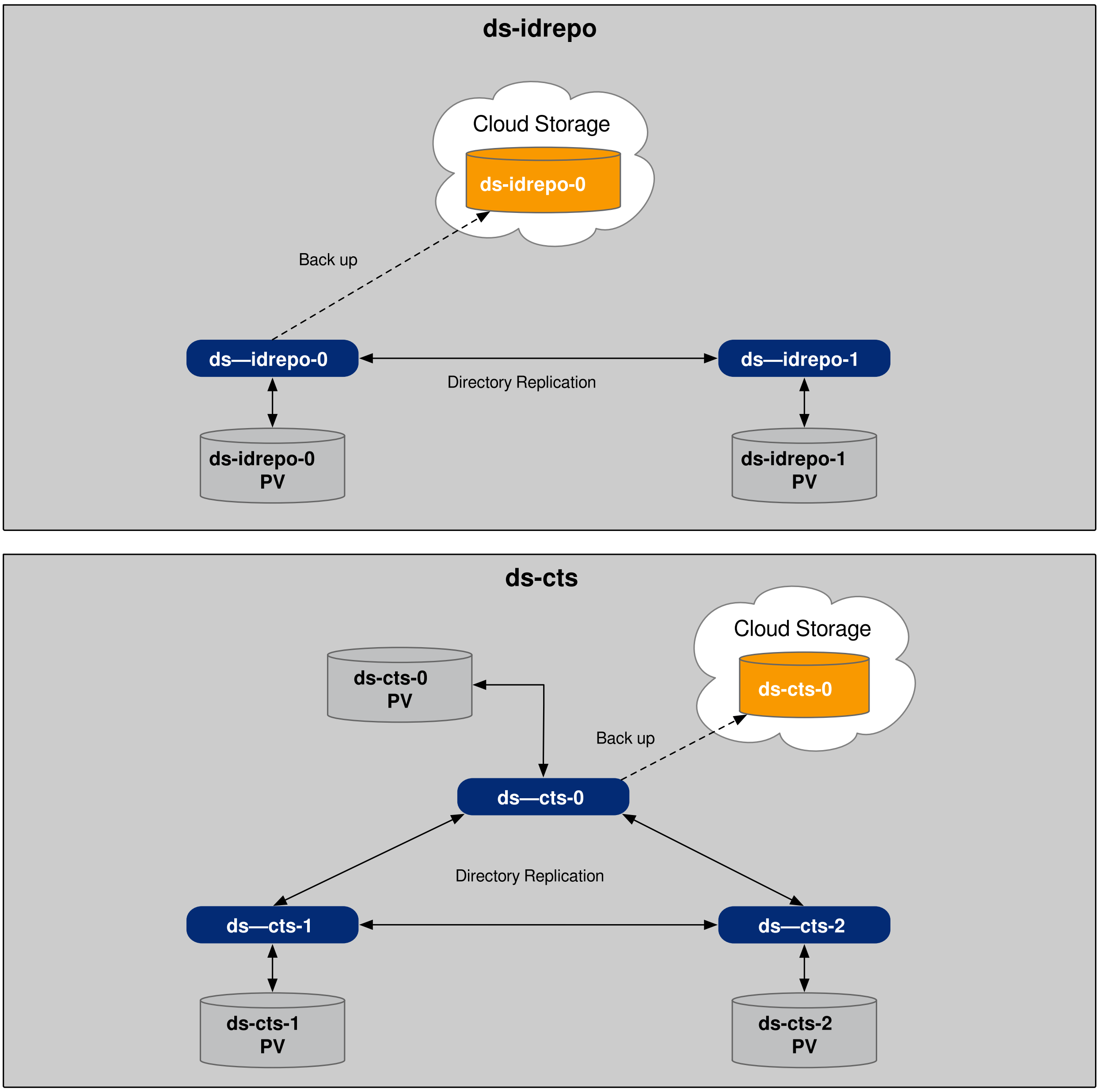
```
$ /path/to/forgeops/bin/cdk install ig
Checking secret-agent operator and related CRDs: secret-agent
```

```
CRD found in cluster.
Checking ds-operator and related CRDs: ds-operator CRD found
in cluster.

Installing component(s): ['ig']

secret/openig-secrets-env created
service/ig created
deployment.apps/ig created

Enjoy your deployment!
```

7. Run the **`kubectl get pods`** command to check the status of the IG pod. Wait until the IG pod is ready before proceeding to the next step.
8. Verify that your IG routes work.

CDM Backup and Restore

The backup topology of DS in the CDM:



CDM directory architecture

Five DS instances are deployed using Kubernetes stateful sets (two `idrepo` and three `cts` backends).

Backup architecture

Before you can back up CDM data, you must set up a cloud storage container, and then configure a Kubernetes secret with the container's location and credentials in the CDM deployment.

DS data backups are stored in Google Cloud Service, Amazon S3, or Azure Blob Storage.

All backups are incremental from the previous backup. The first backup created is a full backup.

DS server instances use cryptographic keys to sign and verify the integrity of backup files, and to encrypt data. Server instances protect these keys in a deployment by encrypting them with the shared master key. For portability, servers store the encrypted keys in the backup files.

Backup scheduling

Backups must be scheduled using the **`schedule-backups.sh`** script in the `/path/to/forgeops/bin` path on your local computer.

By default, CTS and identity repository directory instances are backed up every hour.

The backup schedule can be customized separately for CTS and identity repository DS instances based on your recovery objectives.

By default, the backups are scheduled from the first (or `-0`) pod of each DS instance. You can customize and schedule backups from any pod of a DS instance. You can also schedule backups from multiple pods.

Restore

You can initialize new DS instances with data from a backup when you deploy CDM.

You can restore existing `cts` or `idrepo` DS instances from a backup.

Restoring a new or existing DS instance from a backup requires that the instance being restored has the same shared master key as the instance that performed the backup. To ensure that the master key is shared, implement cloud secret management when you deploy the CDM.

Google Cloud

Set up a Google Cloud storage bucket for the DS data backup, and configure the `forgeops` artifacts with the location and credentials for the bucket:

1. Create a Google Cloud service account with sufficient privileges to write objects in a Google Cloud Storage (GCS) bucket. For example, Storage Object Creator.
2. Add a key to the service account, and download the JSON file that contains the new key.
3. Configure a multi-region GCS bucket for storing DS backups:
    a. Create a new bucket, or identify an existing bucket to use.
    b. Note the bucket's Link for gsutil value.
    c. Grant permissions on the bucket to the service account you created in step 1.
4. Make sure that your current Kubernetes context references the CDM cluster and the `prod` namespace.
5. Create secrets that contain credentials to write to cloud storage. The DS pods will use these when performing backups.

For `my-sa-credential.json`, specify the JSON file that contains the service account's key:

a. Create the `cloud-storage-credentials-cts` secret:

```
$ kubectl create secret generic cloud-storage-credentials-cts \
 --from-file=GOOGLE_CREDENTIALS_JSON=/path/to/my-sa-credential.json \
 --dry-run --output yaml | kubectl apply --filename -
```

b. Create the `cloud-storage-credentials-idrepo` secret:

```
$ kubectl create secret generic cloud-storage-credentials-idrepo \
 --from-file=GOOGLE_CREDENTIALS_JSON=/path/to/my-sa-credential.json \
 --dry-run --output yaml | kubectl apply --filename -
```

6. Set the backup location in the configuration of the running CDM instance:

   a. Get the `platform-config` configmap:

   ```
   $ kubectl get configmap platform-config --output yaml > my-config.yaml
   ```

   b. In the output file from the preceding step, set the `DSBACKUP_DIRECTORY` parameter to the Link for gsutil of the DS data backup bucket:

   For example: `DSBACKUP_DIRECTORY "gs://my-backup-bucket"`

   c. Apply the change to the running CDM:

   ```
   $ kubectl apply --filename my-config.yaml
   ```

7. Apply the same change to your local Kustomization overlay file to ensure that the backup location is configured correctly the next time you deploy the CDM:

   a. Change to the `/path/to/forgeops/kustomize/base/kustomizeConfig` directory.

   b. Edit the `kustomization.yaml` file and set the `DSBACKUP_DIRECTORY` parameter to the location of the backup bucket.

   For example: `DSBACKUP_DIRECTORY "gs://my-backup-bucket"`

8. Restart the pods that perform backups, so that DS can obtain the backup location and the credentials needed to write to the backup location:

```
$ kubectl delete pods ds-cts-0
$ kubectl delete pods ds-idrepo-0
```

Now you are ready to schedule backups.

AWS

Set up an S3 bucket for the DS data backup, and configure the `forgeops` artifacts with the location and credentials for the S3 bucket:

1. Create or identify an existing S3 bucket for storing the DS data backup, and note the S3 link of the bucket.
2. Make sure that your current Kubernetes context references the CDM cluster and the `prod` namespace.
3. Create secrets that contain credentials to write to cloud storage. The DS pods will use these when performing backups:
   a. Create the `cloud-storage-credentials-cts` secret:

   ```
   $ kubectl create secret generic cloud-storage-credentials-
   cts \
    --from-literal=AWS_ACCESS_KEY_ID=my-access-key \
    --from-literal=AWS_SECRET_ACCESS_KEY=my-secret-access-key
   \
    --dry-run --output yaml | kubectl apply --filename -
   ```

   b. Create the `cloud-storage-credentials-idrepo` secret:

   ```
   $ kubectl create secret generic cloud-storage-credentials-
   idrepo \
    --from-literal=AWS_ACCESS_KEY_ID=my-access-key \
    --from-literal=AWS_SECRET_ACCESS_KEY=my-secret-access-key
   \
    --dry-run --output yaml | kubectl apply --filename -
   ```

4. Set the backup location in the configuration of the running CDM instance:
   a. Get the `platform-config` configmap:

   ```
   $ kubectl get configmap platform-config --output yaml >
   my-config.yaml
   ```

   b. In the output file from the preceding step, set the `DSBACKUP_DIRECTORY` parameter to the S3 link of the DS data backup bucket:

For example: `DSBACKUP_DIRECTORY s3://my-backup-bucket`

c. Apply the change to the running CDM instance:

```
$ kubectl apply --filename my-config.yaml
```

5. Apply the same change to your local Kustomization overlay file to ensure that the backup location is configured correctly the next time you deploy the CDM:

   a. Change to the `/path/to/forgeops/kustomize/base/kustomizeConfig` directory.

   b. Edit the `kustomization.yaml` file and set the `DSBACKUP_DIRECTORY` parameter to the S3 link of the DS data backup bucket.

   For example: `DSBACKUP_DIRECTORY s3://my-backup-bucket`

6. Restart the pods that perform backups, so that DS can obtain the backup location and the credentials needed to write to the backup location:

```
$ kubectl delete pods ds-cts-0
$ kubectl delete pods ds-idrepo-0
```

Now you are ready to schedule backups.

Azure

Set up an Azure Blob Storage container for the DS data backup, and configure the `forgeops` artifacts with the location and credentials for the container:

1. Create or identify an existing Azure Blob Storage container for the DS data backup. For more information on how to create and use Azure Blob Storage, see Quickstart: Create, download, and list blobs with Azure CLI.

2. Make sure that your current Kubernetes context references the CDM cluster and the `prod` namespace.

3. Create secrets that contain credentials to write to cloud storage. The DS pods will use these when performing backups:

   a. Get the name and access key of the Azure storage account that contains your storage container.

   b. Create the `cloud-storage-credentials-cts` secret:

```
$ kubectl create secret generic cloud-storage-credentials-
cts \
 --from-literal=AZURE_ACCOUNT_NAME=my-storage-account-name
\
 --from-literal=AZURE_ACCOUNT_KEY=my-storage-account-
```

```
access-key \
 --dry-run --output yaml | kubectl apply --filename -
```

c. Create the `cloud-storage-credentials-idrepo` secret:

```
$ kubectl create secret generic cloud-storage-credentials-
idrepo \
 --from-literal=AZURE_ACCOUNT_NAME=my-storage-account-name
\
 --from-literal=AZURE_ACCOUNT_KEY=my-storage-account-
access-key \
 --dry-run --output yaml | kubectl apply --filename -
```

4. Set the backup location in the configuration of the running CDM instance:

   a. Get the `platform-config` configmap:

   ```
   $ kubectl get configmap platform-config --output yaml >
   my-config.yaml
   ```

   b. In the output file from the preceding step, set the `DSBACKUP_DIRECTORY` parameter to the string `az://`, followed by the name of the storage container:

   For example: `DSBACKUP_DIRECTORY az://my-storage-container`

   c. Apply the change to the running CDM:

   ```
   $ kubectl apply --filename my-config.yaml
   ```

5. Apply the same change to your local Kustomization overlay file to ensure that the backup location is configured correctly the next time you deploy the CDM:

   a. Change to the `/path/to/forgeops/kustomize/base/kustomizeConfig` directory.

   b. Edit the `kustomization.yaml` file and set the `DSBACKUP_DIRECTORY` parameter to the string `az://`, followed by the name of the storage container.

   For example: `DSBACKUP_DIRECTORY az://my-storage-container`

6. Restart the pods that perform backups, so that DS can obtain the backup location and the credentials needed to write to the backup location:

   ```
   $ kubectl delete pods ds-cts-0
   $ kubectl delete pods ds-idrepo-0
   ```

Now you are ready to schedule backups.

DS Backup Scheduling

To schedule DS backups:

1. Make sure that you've set up cloud storage for your cloud provider platform:
   - Google Cloud
   - AWS
   - Azure
2. Make sure that you've implemented cloud secret management. DS instances on which you restore data must have the same master key as DS instances on which you perform backups. If you don't share the master key, you won't be able to restore from your backups.
3. Decide whether you would prefer to use the default backup schedule or a customized backup schedule.
4. Schedule backups:
   a. If you want to use the default backup schedule, run the **`schedule-backups.sh`** script as described in Default Backup Schedule.
   b. If you want to use a customized backup schedule, edit files, and then run the **`schedule-backups.sh`** script as described in Customized Backup Schedule.

Default Backup Schedule

The default backup schedule creates incremental backups of the `idrepo` instance at the beginning of every hour, and the `cts` instance at 10 minutes past every hour.

Run the **`schedule-backups.sh`** script to start backing up `cts` and `idrepo` instances using the default schedule:

```
$ /path/to/forgeops/bin/schedule-backups.sh my-namespace
```

In the CDM deployment, DS is deployed to the `prod` namespace. So you would specify `prod` as the namespace in the above command. If you have deployed DS in another namespace, you must specify the corresponding namespace.

Customized Backup Schedule

You can customize the backup schedule for `cts` and `idrepo` instances separately. You can also schedule backups from any DS pod.

For example, suppose you wanted to make these customizations to the default backup schedule:

- Back up the `ds-idrepo-1` directory instance (instead of the `ds-idrepo-0` instance).
- Back up the `idrepo` directory at the start of every hour, and at the thirtieth minute of every hour (instead of once an hour at the start of the hour).
- Back up the `cts` directory at 20 minutes after the hour (instead of at 10 minutes after the hour).

To customize the schedules for the `idrepo` and `cts` instances, and to schedule backups from the `ds-idrepo-1` pod instead of the `ds-idrepo-0` pod:

1. Revise the set of instances to be backed up in the configuration of the running CDM instance:
   a. Get the `platform-config` configmap:

      ```
      $ kubectl get configmap platform-config --output yaml >
      my-config.yaml
      ```

   b. In the output file from the preceding step, set the `DSBACKUP_HOSTS` parameter to the revised set of instances to be backed up:

      For example: `DSBACKUP_HOSTS "ds-idrepo-1,ds-cts-0"`
   c. Apply the change to the running CDM:

      ```
      $ kubectl apply --filename my-config.yaml
      ```

2. Apply the same change to your local Kustomization overlay file to ensure that the set of backup instances is configured correctly the next time you deploy the CDM:
   a. Change to the `/path/to/forgeops/kustomize/base/kustomizeConfig` directory.
   b. Edit the `kustomization.yaml` file and set the `DSBACKUP_HOSTS` parameter to the revised set of backup instances.

      For example: `DSBACKUP_HOSTS "ds-idrepo-1,ds-cts-0"`
3. Restart the pods that perform backups, so that DS can obtain the revised set of backup instances:

   ```
   $ kubectl delete pods ds-idrepo-1
   $ kubectl delete pods ds-cts-0
   ```

4. Change the frequency of `idrepo` backups and the starting time of `cts` backups:
   a. Open the `/path/to/forgeops/bin/schedule-backups.sh` script.
   b. To back up the `idrepo` directory at the start of every hour, and at the thirtieth minute of every hour, change the line:

```
BACKUP_SCHEDULE_IDREPO="0 * * * *"
```

to:

```
BACKUP_SCHEDULE_IDREPO="*/30 * * * *"
```

c. To back up the `cts` directory at 20 minutes after the hour, change the line:

```
BACKUP_SCHEDULE_CTS="10 * * * *"
```

to:

```
BACKUP_SCHEDULE_CTS="20 * * * *"
```

d. Save your changes.

5. Run the **schedule-backups.sh** script.

```
$ /path/to/forgeops/bin/schedule-backups.sh prod
```

The **schedule-backups.sh** script stops any previously scheduled backup jobs before initiating the new schedule.

CDM Restore

Before you attempt to restore data from backups, make sure that you've implemented cloud secret management. DS instances on which you restore data must have the same master key as DS instances on which you perform backups. If you don't share the master key, you won't be able to restore from your backups.

This page covers three options to restore data from backups:

- New CDM Using DS Backup
- Restore All DS Directories
- Restore One DS Directory

New CDM Using DS Backup

Creating new instances from previously backed up DS data is useful when a system disaster occurs, or when directory services are lost. It is also useful when you want to port test environment data to a production deployment.

To create new DS instances with data from a previous backup:

1. Make sure that your current Kubernetes context references the CDM cluster and the `prod` namespace.
2. Update the YAML file used by the CDM to create Kubernetes secrets that contain your cloud storage credentials:

   ▼ *On Google Cloud*

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --from-file=GOOGLE_CREDENTIALS_JSON=/path/to/my-sa-credential.json \
    --dry-run --output yaml >
   /path/to/forgeops/kustomize/base/7.0/ds/base/cloud-storage-credentials.yaml
   ```

   In this example, specify the path and file name of the JSON file that contains the Google service account key for *my-sa-credential.json*.

   ▼ *On AWS*

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --from-literal=AWS_ACCESS_KEY_ID=my-access-key \
    --from-literal=AWS_SECRET_ACCESS_KEY=my-secret-access-key \
    --dry-run --output yaml >
   /path/to/forgeops/kustomize/base/7.0/ds/base/cloud-storage-credentials.yaml
   ```

   ▼ *On Azure*

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --from-literal=AZURE_ACCOUNT_NAME=my-storage-account-name \
    --from-literal=AZURE_ACCOUNT_KEY=my-storage-account-access-key \
    --dry-run --output yaml >
   /path/to/forgeops/kustomize/base/7.0/ds/base/cloud-storage-credentials.yaml
   ```

3. Configure the backup bucket location and enable the automatic restore capability:
   a. Change to the `/path/to/forgeops/kustomize/base/kustomizeConfig` directory.
   b. Open the `kustomization.yaml` file.
   c. Set the `DSBACKUP_DIRECTORY` parameter to the location of the backup bucket. For example:

      ▼ *On Google Cloud*

      `DSBACKUP_DIRECTORY "gs://my-backup-bucket"`

▼ On AWS

```
DSBACKUP_DIRECTORY "s3://my-backup-bucket"
```

▼ On Azure

```
DSBACKUP_DIRECTORY "az://my-backup-bucket"
```

   d. Set the `AUTORESTORE_FROM_DSBACKUP` parameter to `"true"`.

4. Deploy the platform.
5. Remove your credentials from the YAML file that the CDM uses to create Kubernetes secrets:

   ```
   $ kubectl create secret generic cloud-storage-credentials \
    --dry-run --output yaml >
   /path/to/forgeops/kustomize/base/7.0/ds/base/cloud-storage-
   credentials.yaml
   ```

When the platform is deployed, new DS pods are created, and the data is automatically restored from the most recent backup available in the cloud storage location you have configured.

To verify that the data has been restored:

- Use the IDM UI or platform UI.
- Review the logs for the DS pods' `initialize` container. For example:

  ```
  $ kubectl logs --container initialize ds-idrepo-0
  ```

Restore All DS Directories

To restore all the DS directories in your CDM deployment from backup:

1. Delete all the PVCs attached to DS pods using the **`kubectl delete pvc`** command.
2. Because PVCs might not get deleted immediately when the pods to which they're attached are running, stop the DS pods.

   Using separate terminal windows, stop every DS pod using the **`kubectl delete pod`** command. This deletes the pods and their attached PVCs.

   Kubernetes automatically restarts the DS pods after you delete them. The automatic restore feature of CDM recreates the PVCs as the pods restart by retrieving backup data from cloud storage and restoring the DS directories from the latest backup.

3. After the DS pods have come up, restart IDM pods to reconnect IDM to the restored PVCs:
    a. List all the pods in the `prod` namespace.
    b. Delete all the pods running IDM.

Restore One DS Directory

In a CDM deployment that has automatic restore enabled, you can recover a failed DS pod if the latest backup is within the replication purge delay:

1. Delete the PVC attached to the failed DS pod using the **`kubectl delete pvc`** command.
2. Because the PVC might not get deleted immediately if the attached pod is running, stop the failed DS pod.

    In another terminal window, stop the failed DS pod using the **`kubectl delete pod`** command. This deletes the pod and its attached PVC.

    Kubernetes automatically restarts the DS pod after you delete it. The automatic restore feature of CDM recreates the PVC as the pod restarts by retrieving backup data from cloud storage and restoring the DS directory from the latest backup.
3. If the DS instance that you restored was the `ds-idrepo` instance, restart IDM pods to reconnect IDM to the restored PVC:
    a. List all the pods in the `prod` namespace.
    b. Delete all the pods running IDM.

For information about how to manually restore DS where the latest available backup is older than the replication purge delay, see the Restore section in the DS documentation.

Best Practices for Restoring Directories

- Use a backup that is newer than the last replication purge.
- When you restore a DS replica using backups that are older than the purge delay, that replica will no longer be able to participate in replication.

    Reinitialize the replica to restore the replication topology.
- If the available backups are older than the purge delay, then initialize the DS replica from an up-to-date master instance. For more information on how to initialize a replica, see Manual Initialization in the DS documentation.

CDM Monitoring

The CDM uses Prometheus to monitor ForgeRock 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.

Monitoring Pods

The following Prometheus and Grafana pods from the `prometheus-operator` project run in the `monitoring` namespace:

Pod	Description
`alertmanager-prometheus-operator-kube-p-alertmanager-0`	Handles Prometheus alerts by grouping them together, filtering them, and then routing them to a receiver, such as a Slack channel.
`prometheus-operator-kube-state-metrics-...`	Generates Prometheus metrics for cluster node resources, such as CPU, memory, and disk usage. One pod is deployed for each CDM node.

Pod	Description
`prometheus-operator-prometheus-node-exporter-...`	Generates Prometheus metrics for Kubernetes objects, such as deployments and nodes.
`prometheus-operator-grafana-...`	Provides the Grafana service.
`prometheus-prometheus-operator-kube-p-prometheus-0`	Provides the Prometheus service.
`prometheus-operator-kube-p-operator-...`	Runs the Prometheus operator.

See the prometheus-operator Helm chart README file for more information about the pods in the preceding table.

Custom Grafana Dashboards

In addition to the pods from the `prometheus-operator` project, the CDM includes a set of Grafana dashboards. The `import-dashboards-...` pod from the `forgeops` repository runs after Grafana starts up. This pod imports Grafana dashboards for the ForgeRock Identity Platform and terminates after importing has completed.

You can customize, export and import Grafana dashboards using the Grafana UI or HTTP API.

For information about importing custom Grafana dashboards, see the Import Custom Grafana Dashboards section of the Prometheus and Grafana Deployment README file in the `forgeops` repository.

Alerts

CDM alerts are defined in the fr-alerts.yaml file in the `forgeops` repository.

To configure additional alerts, see the Configure Alerting Rules section of the Prometheus and Grafana Deployment README file in the `forgeops` repository.

CDM Security

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

Secret Agent

Kubernetes operator that generates secrets and provides cloud secret management.

Secure Communications

Secure HTTP, secure LDAP, and certificate management.

IP Address Restriction

Access restriction by incoming IP address, enforced by the NGINX ingress 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.

Secret Agent Operator

The open source Secret Agent operator randomly generates all secrets for AM, IDM, and DS services running in the CDK and the CDM. The Secret Agent operator runs as a Kubernetes deployment that must be available before AM, IDM, and DS are deployed. In addition to generating secrets, the operator integrates with Google Cloud Secret Manager, AWS Secrets Manager, and Azure Key Vault to manage secrets, providing cloud backup and retrieval for secrets.

Secret Generation

By default, the operator examines your namespace to determine whether it contains all the secrets required for ForgeRock Identity Platform deployment. If any of the required secrets are not present, the operator generates them.

See the Secret Agent project README for information about:

- Importing your own secrets
- Secret Agent naming conventions
- Modifying the Secret Agent configuration

Cloud Secret Management

Configuring the Secret Agent operator to integrate with a cloud secret manager, such as Google Cloud Secret Manager, AWS Secret Manager, or Azure Key Vault, changes the operator's behavior:

- First, the operator examines your namespace to determine whether it contains all the secrets required for ForgeRock Identity Platform deployment.
- If any of the required secrets are not in your namespace, the operator checks to see if the missing secrets are available in the cloud secret manager:
    - If any of the secrets missing from your namespace are available in the cloud secret manager, the operator gets them from the cloud secret manager and adds them to your namespace.
    - If missing secrets are not available in the cloud secret manager, the Secret Agent operator generates them.

See the Secret Agent project README for information about how to integrate the Secret Agent operator with these cloud secret managers:

- Google Cloud Secret Manager
- AWS Secret Manager
- Azure Key Vault

CAUTION

Before deploying the ForgeRock Identity Platform in production, you *must* configure the Secret Agent operator to support cloud secret management. If you do not do so, you run the risk of not being able to access your directory data. Because directory data is encrypted by secrets that have been generated by the operator, loss of your secrets will result in data loss.

Administration Password Changes

The CDM uses six administration passwords:

- The AM and IDM administration user, `amadmin`
- The AM CTS service account, `uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`

- The shared identity repository service account, `uid=am-identity-bind-account,ou=admins,ou=identities`
- The DS root user, `uid=admin`

Some organizations have a requirement to change administration passwords from time to time. Follow these steps if you need to change the CDM administration passwords:

1. Ensure that you have configured Cloud Secret Management in your deployment.

   Cloud secret management is required when deploying the platform in production on Google Cloud, AWS, or Azure.

2. Change the `amadmin` user's password:

   a. Run the **`print-secrets`** command from the `bin` directory in your `forgeops` repository clone. Note the current password for the `amadmin` user.

   b. Delete the entry that contains the `amadmin` user's password from the cloud secret manager:

      ▼ *Google Cloud*

      List the secrets managed by the cloud secret manager, locate the URI for the secret that contains the `AM-PASSWORDS-AMADMIN-CLEAR` password, and delete it. For example:

      ```
      $ gcloud secrets list --uri
      $ gcloud secrets delete \
       https://secretmanager.googleapis.com/. . ./prod-am-env-
      secrets-AM-PASSWORDS-AMADMIN-CLEAR
      ```

      ▼ *AWS*

      List the secrets managed by the cloud secret manager, locate the ARN for the secret that contains the `AM-PASSWORDS-AMADMIN-CLEAR` password, and delete it. For example:

      ```
      $ aws secretsmanager list-secrets --region=my-region
      $ aws secretsmanager delete-secret --region=my-region \
       --force-delete-without-recovery \
       --secret-id arn:aws:secretsmanager:. . .:prod-am-env-
      secrets-AM-PASSWORDS-AMADMIN-CLEAR-c3KfsL
      ```

      ▼ *Azure*

      Soft delete the secret that contains the `AM-PASSWORDS-AMADMIN-CLEAR` password from Azure Key Vault. For example:

```
$ az keyvault secret delete --vault-name my-key-vault --
name prod-am-env-secrets-AM-PASSWORDS-AMADMIN-CLEAR
```

Purge the soft deleted secret from Azure Key Vault. For example:

```
$ az keyvault secret purge --vault-name my-key-vault --
name prod-am-env-secrets-AM-PASSWORDS-AMADMIN-CLEAR
```

c. Delete the Kubernetes secret that contains the `amadmin` user's password from the `prod` namespace:

```
$ kubens prod
$ kubectl patch secrets am-env-secrets --type=json \
 --patch='[{"op":"remove", "path":
"/data/AM_PASSWORDS_AMADMIN_CLEAR"}]'
```

d. Restart AM by deleting all active AM pods: list all the pods in the `prod` namespace, and then delete all the pods running AM.

e. After AM comes up, run the **`print-secrets`** command again to get the current administration passwords.

   Verify that the `amadmin` user's password has changed by comparing its previous value to its current value.

f. Verify that you can log in to the platform UI using the new password.

3. Change the CTS service account's password:

   a. Change to the `bin` directory in your `forgeops` repository clone.

   b. Run the **`print-secrets`** command. Note the current password for the identity repository service account.

   c. Delete the entry that contains this account's password from the cloud secret manager:

▼ *Google Cloud*

List the secrets managed by the cloud secret manager, locate the URI for the secret that contains the `AM_STORES_CTS_PASSWORD` password, and delete it. For example:

```
$ gcloud secrets list --uri
$ gcloud secrets delete \
 https://secretmanager.googleapis.com/. . ./prod-ds-env-
secrets-AM_STORES_CTS_PASSWORD
```

▼ *AWS*

List the secrets managed by the cloud secret manager, locate the ARN for the secret that contains the `AM_STORES_CTS_PASSWORD` password, and delete it. For example:

```
$ aws secretsmanager list-secrets --region=my-region
$ aws secretsmanager delete-secret --region=my-region \
 --force-delete-without-recovery \
 --secret-id arn:aws:secretsmanager:. . .:prod-ds-env-
secrets-AM_STORES_CTS_PASSWORD-1d4432
```

▼ *Azure*

Soft delete the secret that contains the `AM_STORES_CTS_PASSWORD` password from Azure Key Vault. For example:

```
$ az keyvault secret delete --vault-name my-key-vault --
name prod-ds-env-secrets-AM_STORES_CTS_PASSWORD
```

Purge the deleted secret from Azure Key Vault. For example:

```
$ az keyvault secret purge --vault-name my-key-vault --
name prod-ds-env-secrets-AM_STORES_CTS_PASSWORD
```

d. Delete the Kubernetes secret that contains the service account's password from the `prod` namespace:

```
$ kubens prod
$ kubectl patch secrets ds-env-secrets --type=json \
 --patch='[{"op":"remove", "path":
"/data/AM_STORES_CTS_PASSWORD"}]'
```

e. Redeploy the platform:

```
$ cd /path/to/forgeops
$ skaffold delete --profile small; skaffold run --profile
small
```

f. After the platform comes up, run the **`print-secrets`** command again to get the current administration passwords.

Verify that the CTS service account's password has changed by comparing its previous value to its current value.

4. Change the identity repository service account's password:

a. Change to the `bin` directory in your `forgeops` repository clone.

b. Run the **print-secrets** command. Note the current password for the the identity repository service account.

c. Delete the entry that contains this account's password from the cloud secret manager:

▼ *Google Cloud*

List the secrets managed by the cloud secret manager, locate the URI for the secret that contains the `AM_STORES_USER_PASSWORD` password, and delete it. For example:

```
$ gcloud secrets list --uri
$ gcloud secrets delete \
 https://secretmanager.googleapis.com/. . ./prod-ds-env-
secrets-AM_STORES_USER_PASSWORD
```

▼ *AWS*

List the secrets managed by the cloud secret manager, locate the ARN for the secret that contains the `AM_STORES_USER_PASSWORD` password, and delete it. For example:

```
$ aws secretsmanager list-secrets --region=my-region
$ aws secretsmanager delete-secret --region=my-region \
 --force-delete-without-recovery \
 --secret-id arn:aws:secretsmanager:. . .:prod-ds-env-
secrets-AM_STORES_USER_PASSWORD-1d4432
```

▼ *Azure*

Soft delete the secret that contains the `AM_STORES_USER_PASSWORD` password from Azure Key Vault. For example:

```
$ az keyvault secret delete --vault-name my-key-vault --
name prod-ds-env-secrets-AM_STORES_USER_PASSWORD
```

Purge the deleted secret from Azure Key Vault. For example:

```
$ az keyvault secret purge --vault-name my-key-vault --
name prod-ds-env-secrets-AM_STORES_USER_PASSWORD
```

d. Delete the Kubernetes secret that contains the service account's password from the `prod` namespace:

```
$ kubens prod
$ kubectl patch secrets ds-env-secrets --type=json \
```

```
 --patch='[{"op":"remove", "path":
"/data/AM_STORES_USER_PASSWORD"}]'
```

e. Redeploy the platform:

```
$ cd /path/to/forgeops
$ skaffold delete --profile small; skaffold run --profile
small
```

f. After the platform comes up, run the **print-secrets** command again to get the current administration passwords.

   Verify that the identity repository service account's password has changed by comparing its previous value to its current value.

5. Change the DS root user's password:

   a. Change to the `bin` directory in your `forgeops` repository clone.

   b. Run the **print-secrets** command. Note the current password for the `uid=admin` account.

   c. Delete the entry that contains this account's password from the cloud secret manager:

   ▼ *Google Cloud*

   List the secrets managed by the cloud secret manager, locate the URI for the secret that contains the `dirmanager-pw` password, and delete it. For example:

```
$ gcloud secrets list --uri
$ gcloud secrets delete \
 https://secretmanager.googleapis.com/. . ./prod-ds-
passwords-dirmanager-pw
```

   ▼ *AWS*

   List the secrets managed by the cloud secret manager, locate the ARN for the secret that contains the `dirmanager-pw` password, and delete it. For example:

```
$ aws secretsmanager list-secrets --region=my-region
$ aws secretsmanager delete-secret --region=my-region \
 --force-delete-without-recovery \
 --secret-id arn:aws:secretsmanager:. . .:prod-ds-
passwords-dirmanager-pw-2eeaa0
```

   ▼ *Azure*

Soft delete the secret that contains the `dirmanager-pw` password from Azure Key Vault. For example:

```
$ az keyvault secret delete --vault-name my-key-vault --
name prod-ds-passwords-dirmanager-pw
```

Purge the deleted secret from Azure Key Vault. For example:

```
$ az keyvault secret purge --vault-name my-key-vault --
name prod-ds-passwords-dirmanager-pw
```

d. Delete the Kubernetes secret that contains the service account's password from the `prod` namespace:

```
$ kubens prod
$ kubectl patch secrets ds-passwords --type=json \
 --patch='[{"op":"remove", "path":
"/data/dirmanager.pw"}]'
```

e. Redeploy the platform:

```
$ cd /path/to/forgeops
$ skaffold delete --profile small; skaffold run --profile
small
```

f. After the platform comes up, run the **`print-secrets`** command again to get the current administration passwords.

Verify that the password for the `uid=admin` account has changed by comparing its previous value to its current value.

Secure HTTP and Secure LDAP

The CDK and CDM enable secure communication with ForgeRock Identity Platform services using a TLS-enabled ingress controller. Incoming requests and outgoing responses are encrypted. TLS is terminated at the ingress controller.

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

You can configure communication with ForgeRock Identity Platform services other than directory services using one of the following options:

- **Over HTTPS using a self-signed certificate.** Communication is encrypted, but users will receive warnings about insecure communication from some browsers.

- **Over HTTPS using a certificate with a trust chain that starts at a trusted root certificate**. Communication is encrypted, and users will not receive warnings from their browsers.
- **Over HTTPS using a dynamically obtained certificate from Let's Encrypt**. Communication is encrypted and users will not receive warnings from their browsers. A `cert-manager` pod installed in your Kubernetes cluster calls Let's Encrypt to obtain a certificate, and then automatically installs a Kubernetes secret.

You install a Helm chart from the cert-manager project to provision certificates. By default, the pod issues a self-signed certificate. You can also configure the pod to issue a certificate with a trust chain that begins at a trusted root certificate, or to dynamically obtain a certificate from Let's Encrypt.

Certificate Management Automation

In the CDM, certificate management is provided by the cert-manager add-on. The certificate manager deployed in CDM generates a self-signed certificate to secure CDM communication.

In your own deployment, you can specify a different certificate issuer or DNS challenge provider by changing values in the ingress.yaml file.

For more information about configuring certificate management, see the cert-manager documentation.

Access Restriction by IP Address

When installing the ingress controller in production environments, you should consider configuring a CIDR block in the Helm chart for the ingress controller so that you restrict access to worker nodes from a specific IP address or a range of IP addresses.

To specify a range of IP addresses allowed to access resources controlled by the ingress controller, specify the `--set controller.service.loadBalancerSourceRanges=`*`your IP range`* option when you install your ingress controller.

For example:

```
$ helm install --namespace nginx --name nginx \
 --set rbac.create=true \
 --set controller.publishService.enabled=true \
 --set controller.stats.enabled=true \
 --set controller.service.externalTrafficPolicy=Local \
 --set controller.service.type=LoadBalancer \
 --set controller.image.tag="0.21.0" \
 --set
```

```
controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="nlb" \
 --set controller.service.loadBalancerSourceRanges="{81.0.0.0/8,3.56.113.4/32}" \
 stable/nginx-ingress
```

Network Policies

Kubernetes network policies let you specify specify how pods are allowed to communicate with other pods, namespaces, and IP addresses.

Network Policies Example

The `forgeops` repository contains an example with six network policies for the ForgeRock Identity Platform. These network policies are in the netpolicies.yaml file, part of a Kustomize base named `security`.

Customize this example to meet your security needs, or use it to help you better understand how network policies can make Kubernetes deployments more secure.

Deploy the Example

The `forgeops` repository's `skaffold.yaml` file contains a Skaffold profile named `security` that references the Kustomize `security` base. To deploy the platform with the example network policies, run:

```
$ cd /path/to/forgeops
$ skaffold run --profile security
```

About the Example Network Policies

All the example policies have the value `Ingress` in the `spec.policyTypes` key:

```
spec:
  policyTypes:
  - Ingress
```

Network policies with this policy type are called *ingress policies*, because they limit ingress traffic in a deployment.

`deny-all` Policy

By default, if no network policies exist in a namespace, then all ingress and egress traffic is allowed to and from pods in that namespace.

The `deny-all` policy modifies the default network policy for ingress. If a pod isn't selected by another network policy in the namespace, ingress is *not* allowed.

For information about how Kubernetes controls pod ingress when pods are selected by multiple network policies in a namespace, see the Kubernetes documentation.

`ds-idrepo-ldap` Policy

The `ds-idrepo-ldap` policy limits access to `ds-idrepo` pods. Access can only be requested over port 1389, 1636, or 8080, and must come from an `am`, `idm`, or `amster` pod.

This part of the network policy specifies that access must be requested over port 1389, 1636, or 8080:

```
ingress:
- from:
  . . .
  ports:
  - protocol: TCP
    port: 1389
  - protocol: TCP
    port: 1636
  - protocol: TCP
    port: 8080
```

This part of the network policy specifies that access must be from an `am`, `idm`, or `amster` pod:

```
ingress:
- from:
  - podSelector:
      matchExpressions:
      - key: app
        operator: In
        values:
        - am
        - idm
        - amster
```

Understanding the example network policies and how to customize them requires some knowledge about labels defined in CDM deployments. For example, `am` pods are defined with a label, `app`, that has the value `am`. You'll find this label in `/path/to/forgeops/kustomize/base/am/kustomization.yaml` file:

```
commonLabels:
  app.kubernetes.io/name: am
  app.kubernetes.io/instance: am
  app.kubernetes.io/component: am
  app.kubernetes.io/part-of: forgerock
  tier: middle
  app: am
```

`ds-cts-ldap` Policy

The `ds-cts-ldap` policy limits access to `ds-cts` pods. Access can only be requested over port 1389, 1636, or 8080, and must come from an `am` or `amster` pod.

`ds-replication` Policy

`ds` pods in CDM deployments are labeled with `tier: ds`; they're said to reside in the `ds` tier of the deployment.

The `ds-replication` policy limits access to the pods on the `ds` tier. This policy specifies that access to `ds` tier pods over port 8989 can only come from other pods in the same tier.

Note that port 8989 is the default DS replication port. This network policy ensures that only DS pods can access the replication port.

`backend-http-access` Policy

The `backend-http-access` policy limits access to the pods in the `middle` tier, which contains the `am`, `idm`, and `ig` pods. Access can only be requested over port 8080.

`front-end-http-access` Policy

The `front-end-http-access` policy limits access to the pods in the `ui` tier: the `login-ui`, `admin-ui`, and `end-user-ui` pods. Access can only be requested over port 8080.

Note that users send HTTPS requests for the ForgeRock UIs to the ingress controller over port 443. The ingress controller terminates TLS, and then forwards requests to the UI pods over port 8080.

Cluster Access for Multiple AWS Users

It's common for team members to share the use of a cluster. For team members to share a cluster, the cluster owner must grant access to each user:

1. Get the ARNs and names of users who need access to your cluster.
2. Set the Kubernetes context to your Amazon EKS cluster.

3. Edit the authorization configuration map for the cluster using the **kubectl edit** command:

```
$ kubectl edit -n kube-system configmap/aws-auth
```

4. Under the `mapRoles` section, insert the `mapUser` section. An example is shown here with the following parameters:
   - The user ARN is `arn:aws:iam::`*`012345678901`*`:`*`user/new.user`*.
   - The user name registered in AWS is *`new.user`*.

```
… mapUsers: |
    - userarn: arn:aws:iam::012345678901:user/new.user
      username: new.user
      groups:
        - system:masters
…
```

5. For each additional user, insert the `- userarn:` entry in the `mapUsers:` section:

```
… mapUsers: |
    - userarn: arn:aws:iam::012345678901:user/new.user
      username: new.user
      groups:
        - system:masters
    - userarn: arn:aws:iam::901234567890:user/second.user
      username: second.user
      groups:
        - system:masters
…
```

6. Save the 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

About CDM Benchmarking

CDM Benchmarks provides instructions for running lightweight benchmarks to give you a means for validating your own CDM deployment.

The Cloud Deployment Team runs the same benchmark tests. Our results are available upon request from ForgeRock. To get them, contact your ForgeRock sales representative.

We conduct our tests using the configurations specified for small, medium, and large CDM clusters. We create our clusters using the techniques described in the CDM documentation.

Next, we create test users:

- 1,000,000 test users for a small cluster.
- 10,000,000 test users for a medium cluster.
- 100,000,000 test users for a large cluster.

Finally, we run tests that measure authentication rates and OAuth 2.0 authorization code flow performance.

If you follow the same method of deploying the CDM and running benchmarks, the results you obtain should be similar to ForgeRock's results. However, factors beyond the scope of the CDM, or a failure to use our documented sizing and configuration, may affect your benchmark test results. These factors might include (but are not limited to): updates to cloud platform SDKs; changes to third-party software required for Kubernetes; changes you have made to sizing or configuration to suit your business needs.

The CDM is designed to:

- Conform to DevOps best practices
- Facilitate continuous integration and continuous deployment
- Scale and deploy on any Kubernetes environment in the cloud

If you require higher performance than the benchmarks reported here, you can scale your deployment horizontally and vertically. Vertically scaling ForgeRock Identity Platform works particularly well in the cloud. For more information about scaling your deployment, contact your qualified ForgeRock partner or technical consultant.

Next Step

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

Third-Party Software

The Cloud Deployment Team used Gradle 6.8.3 to benchmark the CDM. Before you start running benchmarks, install this version of Gradle in your local environment.

Earlier and later versions will *probably* work. If you want to try using another version, it is your responsibility to validate it.

In addition to Gradle, you'll need all the third-party software required to deploy the CDM:

- GKE
- EKS
- AKS

Next Step

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

Test User Generation

Running the Authentication Rate and OAuth 2.0 Authorization Code Flow benchmarks requires a set of test users. This page provides instructions for generating a set of test users suitable for these two lightweight AM benchmarks. Note that these test users are not necessarily suitable for other benchmarks or load tests, and that they can't be used with IDM.

For Small and Medium Clusters

To generate test users for lightweight AM benchmarks for small and medium clusters, to provision the CDM userstores, and to prime the directory servers:

1. Make sure your Kubernetes context is set to the cluster in which the CDM is deployed, and that `prod` is the current namespace.
2. Obtain the password for the directory superuser, `uid=admin`:

   ```
   $ cd /path/to/forgeops/bin
   $ ./print-secrets dsadmin
   ```

   Make a note of this password. You'll need it for subsequent steps in this procedure.
3. Change to the directory that contains the source for the `dsutil` Docker container:

   ```
   $ cd /path/to/forgeops/docker/7.0/ds/dsutil
   ```

   You'll generate test users from a pod you create from the `dsutil` container.
4. Build and push the `dsutil` Docker container to your container registry, and then run the container.

   The *my-registry* parameter varies, depending on the location of your registry:

   ```
   $ docker build --tag=my-registry/dsutil .
   $ docker push my-registry/dsutil
   $ kubectl run -it dsutil --image=my-registry/dsutil --
   restart=Never -- bash
   ```

   The **kubectl run** command creates the `dsutil` pod, and leaves you in a shell that lets you run commands in the pod.
5. Generate the test users—1,000,000 users for a small CDM cluster, and 10,000,000 for a medium cluster:

   Run these substeps from the `dsutil` pod's shell:

   a. Make an LDIF file that has the number of user entries for your cluster size:

      For example, for a small cluster:

      ```
      $ /opt/opendj/bin/makeldif -o data/entries.ldif \
       -c numusers=1000000 config/MakeLDIF/ds-idrepo.template
      Processed 1000 entries
      Processed 2000 entries
      Processed 3000 entries
      ```

```
. . .
Processed 1000000 entries
LDIF processing complete. 1000003 entries written
```

When the Cloud Deployment Team ran the **makeldif** script, it took approximately:

- 30 seconds to run on a small cluster.
- 4 minutes to run on a medium cluster.

b. Create the user entries in the directory:

```
$ /opt/opendj/bin/ldapmodify \
 -h ds-idrepo-0.ds-idrepo -p 1389 --useStartTls --trustAll
\
 -D "uid=admin" -w directory-superuser-password --
noPropertiesFile \
 --no-prompt --continueOnError --numConnections 10
data/entries.ldif
```

`ADD operation successful` messages appear as user entries are added to the directory.

When the Cloud Deployment Team ran the **ldapmodify** command, it took approximately:

- 15 minutes to run on a small cluster.
- 2 hours 35 minutes to run on a medium cluster.

6. Prime the directory servers:

   a. Open a new terminal window or tab.

   Use this new terminal window—not the one running the `dsutil` pod's shell—for the remaining substeps in this step.

   b. Prime the directory server running in the `ds-idrepo-0` pod:

      i. Start a shell that lets you run commands in the `ds-idrepo-0` pod:

      ```
      $ kubectl exec ds-idrepo-0 -it -- bash
      ```

      ii. Run the following command:

      ```
      $ ldapsearch -D "uid=admin" -w directory-superuser-
      password \
       -p 1636 -b "ou=identities"  uid=user.*  | grep dn: |
      ```

```
wc -l
10000000
```

iii. Exit from the `id-dsrepo-0` pod's shell:

```
$ exit
```

c. Prime the directory server running in the `ds-idrepo-1` pod.

For Large Clusters

Here are some very general steps you can follow if you want to generate test users for benchmarking or load testing a large cluster:

1. Install DS in a VM in the cloud.
2. Run the `makeldif` and `ldapmodify` commands, as described above.
3. Back up your directory.
4. Upload the backup files to cloud storage.
5. Restore a CDM `idrepo` pod from your backup, following the steps outlined in CDM Restore.

Next Step

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

Authentication Rate

The `AMRestAuthNSim.scala` simulation tests authentication rates using the REST API. It measures the throughput and response times of an AM server performing REST authentications when AM is configured to use CTS-based sessions.

To run the simulation:

1. Make sure the userstore is provisioned, and the Directory Services cache is primed.

   See Test User Generation.

2. Set environment variables that specify the host on which to run the test, the number of concurrent threads to spawn when running the test, the duration of the test (in seconds), the first part of the user ID, and the user password, and the number of users for the test:

```
$ export TARGET_HOST=prod.iam.example.com
$ export CONCURRENCY=100
$ export DURATION=60
$ export USER_PREFIX=user.
$ export USER_PASSWORD=T35tr0ck123
$ export USER_POOL=n-users
```

where *n-users* is `1000000` for a small cluster, `10000000` for a medium cluster, and `100000000` for a large cluster.

3. Configure AM for CTS-based sessions:
   a. Log in to the platform console as the `amadmin` user. For details, see AM Services.
   b. Go to the native AM console.
   c. Select the top level realm.
   d. Select Properties.
   e. Make sure the Use Client-based Sessions option is disabled.

      If it's not disabled, disable it, and then select Save Changes.
4. Change to the `/path/to/forgeops/docker/gatling` directory.
5. Run the simulation:

```
$ gradle clean; gradle gatlingRun-am.AMRestAuthNSim
```

   When the simulation is complete, the name of a file containing the test results appears near the end of the output.
6. Open the file containing the test results in a browser to review the results.

Next Step

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

OAuth 2.0 Authorization Code Flow

The `AMAccessTokenSim.scala` simulation tests OAuth 2.0 authorization code flow performance. It measures the throughput and response time of an AM server performing authentication, authorization, and session token management when AM is configured to use client-based sessions, and OAuth 2.0 is configured to use client-based tokens. In this test, one transaction includes all three operations.

To run the simulation:

1. Make sure the userstore is provisioned, and the Directory Services cache is primed.

   See Test User Generation.

2. Set environment variables that specify the host on which to run the test, the number of concurrent threads to spawn when running the test, the duration of the test (in seconds), the first part of the user ID, and the user password, and the number of users for the test:

   ```
   $ export TARGET_HOST=prod.iam.example.com
   $ export CONCURRENCY=100
   $ export DURATION=60
   $ export USER_PREFIX=user.
   $ export USER_PASSWORD=T35tr0ck123
   $ export USER_POOL=n-users
   ```

   where *n-users* is `1000000` for a small cluster, `10000000` for a medium cluster, and `100000000` for a large cluster.

3. Configure AM for CTS-based sessions:

   a. Log in to the platform console as the `amadmin` user. For details, see AM Services.

   b. Go to the native AM console.

   c. Select the top level realm.

   d. Select Properties.

   e. Make sure the Use Client-based Sessions option is disabled.

      If it’s not disabled, disable it, and then select Save Changes.

4. Configure AM for CTS-based OAuth2 tokens:

   a. Select Realms > Top Level Realm.

   b. Select Services > OAuth2 Provider.

   c. Make sure the Use Client-based Access & Refresh Tokens option is disabled.

      If it’s not disabled, disable it, and then select Save Changes.

5. Change to the `/path/to/forgeops/docker/gatling` directory.

6. Run the simulation:

   ```
   $ gradle clean; gradle gatlingRun-am.AMAccessTokenSim
   ```

When the simulation is complete, the name of a file containing the test results appears near the end of the output.

7. Open the file containing the test results in a browser to review the results.

Congratulations!

You've successfully run the CDM lightweight benchmark tests.

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

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. Simplify your deployment. Install ForgeRock Identity Platform components separately, instead of installing all the components with a single command. Then, make sure each component works correctly before installing the next component: For the CDK, use staged deployment. For the CDM, use each individual component's Skaffold profile.
All the pods have started, but you can't reach the services running in them.	Make sure you don't have any ingress issues.
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.
Skaffold doesn't run as expected.	If Skaffold cannot push a Docker image, review your push setup. For other problems with Skaffold, you can try increasing Skaffold's logging verbosity.

Problem	Troubleshooting Technique
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 DS tools in the debug tools container.
The `kubectl` command requires too much typing.	Enable kubectl tab autocompletion.

DS Diagnostic Tools

The debug tools container, named `ds-util`, 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/7.0/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
```

Staged CDK Installation

By default, the **cdk install** command installs the entire ForgeRock Identity Platform in the CDK's namespace.

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 CDK is deployed is set in your Kubernetes context.
2. Install the `base` and `ds` components first. Other components have dependencies on these two components:
    a. Install the platform `base` component:

```
$ cd /path/to/forgeops/bin
$ ./cdk install base --fqdn dev.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/secretagentc
onfigurations.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/directoryser
vices.directory.forgerock.io created
. . .

Waiting for ds-operator…
customresourcedefinition.apiextensions.k8s.io/directoryser
vices.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://dev.example.com/platform
https://dev.example.com/admin
https://dev.example.com/am
https://dev.example.com/enduser

Enjoy your deployment!
```

b. After you've installed the base component, install the ds component:

```
$ ./cdk install ds
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!
```

3. Install the other ForgeRock Identity Platform components. You can either install all the other components by using the **`cdk install apps`** command, or install them separately:

   a. Install AM:

   ```
   $ ./cdk install am
   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!
   ```

   b. Install Amster:

   ```
   $ ./cdk install amster
   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!
   ```

   c. Install IDM:

   ```
   $ ./cdk install idm
   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!
```

4. Install the user interface components. You can either install all the applications by using the **cdk install ui** command, or install them separately:

   a. Install the administration UI:

```
$ ./cdk install admin-ui
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!
```

   b. Install the login UI:

```
$ ./cdk install login-ui
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!
```

   c. Install the end user UI:

```
$ ./cdk install end-user-ui
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!
```

5. 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.

Multiple Component Installation

You can specify multiple components with a single **`cdk install`** command. For example, to install the `base`, `ds`, `am`, and `amster` components:

```
$ ./cdk install base ds am amster
```

`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.

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 ForgeRock 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
  - Your Skaffold configuration
  - 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

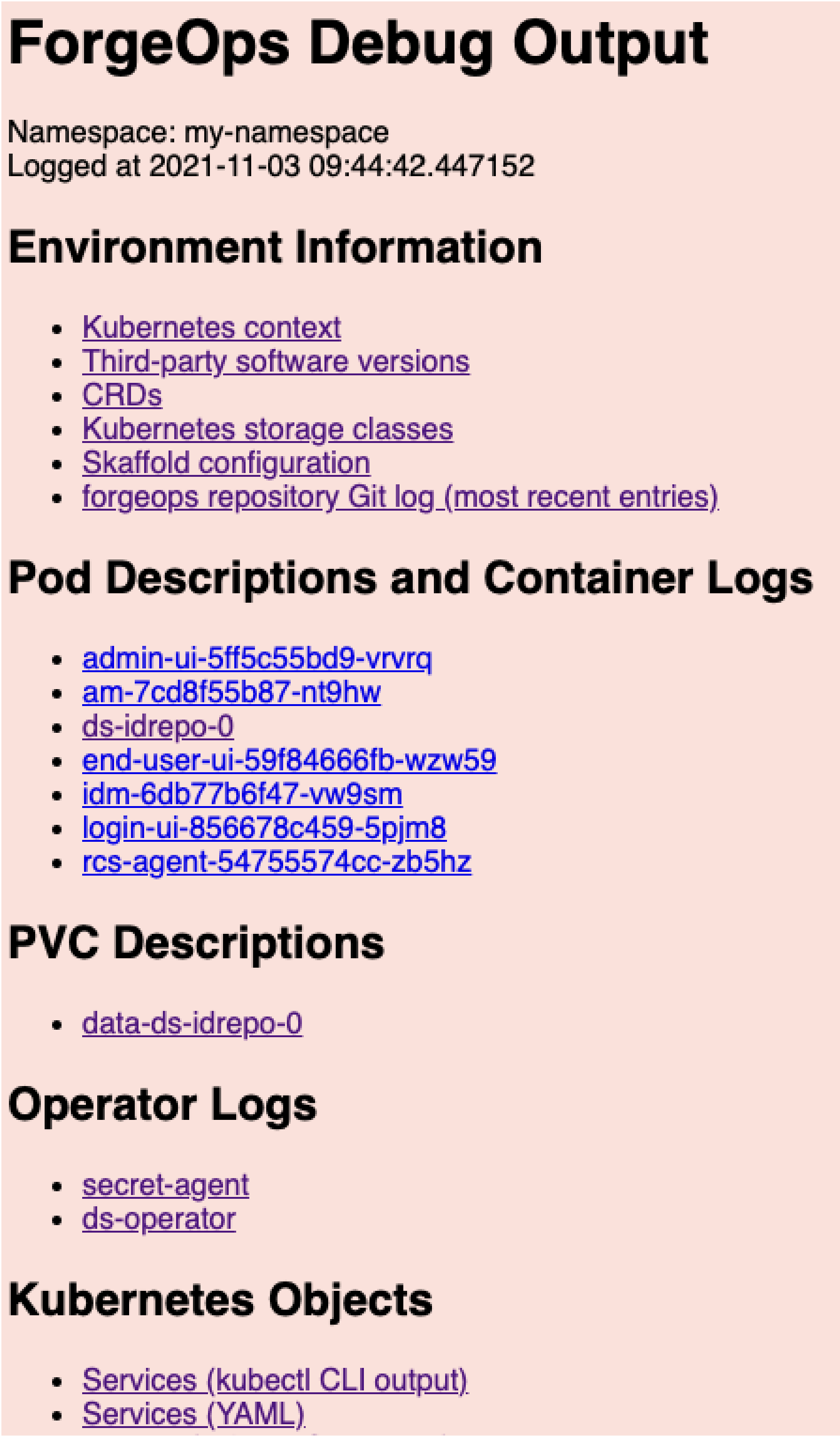
Suppose you installed the CDK, but noticed that one of the CDK pods had an `ImagePullBackOff` error at startup. Here's an example of how you might use pod descriptions and container logs to troubleshoot the problem:

1. Make sure the namespace in which the CDK is deployed is set in your Kubernetes context.
2. Make sure you've checked out the release/7.1-20240223 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 ForgeRock Identity Platform pod in your namespace:



ForgeOps Debug Output

Namespace: my-namespace
Logged at 2021-11-03 09:44:42.447152

Environment Information

- Kubernetes context
- Third-party software versions
- CRDs
- Kubernetes storage classes
- Skaffold configuration
- forgeops repository Git log (most recent entries)

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
- rcs-agent-54755574cc-zb5hz

PVC Descriptions

- data-ds-idrepo-0

Operator Logs

- secret-agent
- ds-operator

Kubernetes Objects

- Services (kubectl CLI output)
- Services (YAML)

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.

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.

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.

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/7.0
$ kustomize build all
2020/10/02 11:07:53 well-defined vars that were never replaced:
DOMAIN,DSBACKUP_DIRECTORY,DSBACKUP_HOSTS,NAMESPACE,SUBDOMAIN,AUTOR
ESTORE_FROM_DSBACKUP
apiVersion: v1
data:
  IDM_ENVCONFIG_DIRS: /opt/openidm/resolver
  LOGGING_PROPERTIES: /var/run/openidm/logging/logging.properties
  OPENIDM_ANONYMOUS_PASSWORD: anonymous
  OPENIDM_CLUSTER_REMOVE_OFFLINE_NODE_STATE: "true"
  OPENIDM_CONFIG_REPO_ENABLED: "false"
  PROJECT_HOME: /opt/openidm
kind: ConfigMap
metadata:
  labels:
    app: idm
    app.kubernetes.io/name: forgerock
    component: idm
    tier: middle
    vendor: forgerock
  name: idm
  namespace: prod
---
apiVersion: v1
data:
  logging.properties: |
    # Properties file that configures the operation of the JDK
    # logging facility.
    # The system will look for this configuration file, first
using
    # a System property specified at startup:
```

```
    #
    # >java -
Djava.util.logging.config.file=myLoggingConfigFilePath
    #
. . .
```

Minikube Hardware Resources

Cluster Configuration

The **`cluster-up`** 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.

Skaffold Troubleshooting

Push Setup

If the `skaffold run` command fails because it does not have permissions to push a Docker image, it may be trying to push to the Docker hub. The reported image name will be something like `docker.io/am`.

When running on Minikube, Skaffold assumes that a push is not required, because it can `docker build` directly to the Docker machine. If it is attempting to push to Docker Hub, it is because Skaffold thinks it is not running on Minikube. Make sure your Minikube context is named `minikube`.

An alternate solution is to modify the Docker build in `skaffold.yaml` and set the value of the `local.push` key to `false`. For more information, see the Skaffold documentation.

Logging Verbosity

Skaffold provides different levels of debug logging information. When you encounter issues deploying the platform with Skaffold, you can set the logging verbosity to display more messages. The additional messages might help you identify problems.

For example:

```
$ cd /path/to/forgeops
$ skaffold dev -v debug
INFO[0000] starting gRPC server on port 50051
INFO[0000] starting gRPC HTTP server on port 50052
INFO[0000] Skaffold &{Version:v0.38.0
ConfigVersion:skaffold/v1beta14 GitVersion:
GitCommit:1012d7339d0055ab93d7f88e95b7a89292ce77f6
GitTreeState:clean BuildDate:2020-09-13T02:16:09Z GoVersion:go1.13
Compiler:gc Platform:darwin/amd64}
DEBU[0000] config version (skaffold/v1beta12) out of date:
upgrading to latest (skaffold/v1beta14)
DEBU[0000] found config for context "minikube"
DEBU[0000] Defaulting build type to local build
DEBU[0000] validating yamltags of struct SkaffoldConfig
DEBU[0000] validating yamltags of struct Metadata
. . .
```

Technology Previews

DS Operator

WARNING

WARNING

The DS operator is currently in technology preview status for production deployments. Do not use the operator in production deployments of the platform.

The DS operator is supported in developer and demonstration deployments. The **`cdk install ds`** command checks to see whether the DS operator is present in your cluster, and installs it if it's not.

The DS operator uses the Kubernetes operator design pattern to let you easily deploy and manage DS instances running in a Kubernetes cluster. After you install the `ds-operator` custom resource definition (CRD) in a cluster, you can use it to create DS instances, scale them, and manage backup and restore.

To deploy the platform by using the DS operator, install the CDK (*not* the legacy CDK). During installation, progress messages indicate that the **cdk** command installs the DS operator:

```
Checking ds-operator and related CRDs: ds-operator CRD not found.
Installing ds-operator.
namespace/fr-system created
customresourcedefinition.apiextensions.k8s.io/directoryservices.di
rectory.forgerock.io created
role.rbac.authorization.k8s.io/ds-operator-leader-election-role
created
clusterrole.rbac.authorization.k8s.io/ds-operator-
directoryservice-editor-role created
clusterrole.rbac.authorization.k8s.io/ds-operator-manager-role
created
rolebinding.rbac.authorization.k8s.io/ds-operator-leader-election-
rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/ds-operator-
directoryservice-editor-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/ds-operator-manager-
rolebinding created
deployment.apps/ds-operator-ds-operator created

Waiting for ds-operator…
customresourcedefinition.apiextensions.k8s.io/directoryservices.di
rectory.forgerock.io condition met
deployment.apps/ds-operator-ds-operator condition met
pod/ds-operator-ds-operator-f974dd8fc-z4vx8 condition met
. . .
Deploying ds.yaml. This is includes all directory resources.
directoryservice.directory.forgerock.io/ds-idrepo created
. . .
```

After you get the CDK up and running, you can explore the DS operator. Some things you might try:

- Show the operator's status:

```
$ kubectl describe directoryservice

Name:         ds-idrepo
Namespace:    my-namespace
Labels:       app.kubernetes.io/component=directory
              app.kubernetes.io/instance=ds-idrepo
              app.kubernetes.io/name=ds
              app.kubernetes.io/part-of=forgerock
Annotations:  <none>
API Version:  directory.forgerock.io/v1alpha1
Kind:         DirectoryService
Metadata:
  Creation Timestamp:  2021-05-18T23:18:27Z
  Generation:          2
  Managed Fields:
    API Version:  directory.forgerock.io/v1alpha1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:
        f:labels:
          .:
          f:app.kubernetes.io/component:
          f:app.kubernetes.io/instance:
          f:app.kubernetes.io/name:
          f:app.kubernetes.io/part-of:
      f:spec:
        .:
        f:image:
        f:keystores:
          .:
          f:keyStoreSecretName:
        f:passwords:
          .:
          f:uid=admin:
            .:
            f:key:
            f:secretName:
```

```
            f:uid=am-config,ou=admins,ou=am-config:
. . .
```

- Scale one or both of the DS pods. For example:

  ```
  $ kubectl scale directoryservice/ds-idrepo --replicas=2

  directoryservice.directory.forgerock.io/ds-idrepo scaled
  ```

- Modify DS properties:

  ```
  $ kubectl edit directoryservice/ds-idrepo
  ```

  For example, you could modify the backup properties (under `spec`) to enable backups, or to change the backup interval.

- Take a volume snapshot. For more information, see Volume Snapshots (Preview) in the DS operator README.

Legacy Features

Cloud Developer's Kit Documentation

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

The CDK is a minimal sample deployment of the ForgeRock Identity Platform. If you have access to a cluster on Google Cloud, EKS, or AKS, you can install the CDK in a namespace on your cluster. But even if you don't have access to a cloud-based cluster, you can deploy the CDK locally in a standalone environment called Minikube, and when you're done, you'll have a local Kubernetes cluster with the platform installed on it.

CDK Checklist

- ❑ Become familiar with the CDK
- ❑ Understand CDK architecture (Minikube|Shared Cluster)
- ❑ Set up your local environment (Minikube|Shared Cluster)
- ❑ Deploy the platform
- ❑ Access platform UIs and APIs

❑ Develop custom Docker images

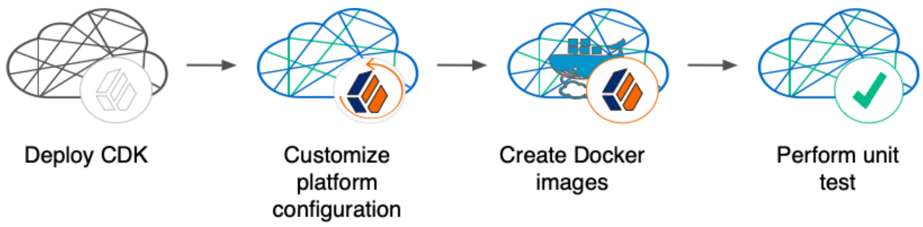
About the Cloud Developer's Kit

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

The CDK is a minimal sample deployment for development purposes. It includes fully integrated AM, IDM, and DS installations, and randomly generated secrets. Developers deploy the CDK, and then access AM's and IDM's GUI consoles and REST APIs to configure the platform and build customized Docker images for the platform.

This documentation describes how to use the CDK to stand up the platform in your developer environment, then create and test customized Docker images containing your custom AM and IDM configurations:



Customizing the platform using the CDK is one of the major activities required before deploying the platform in production. To better understand how this activity fits in to the overall deployment process, see Configure the Platform.

Containerization

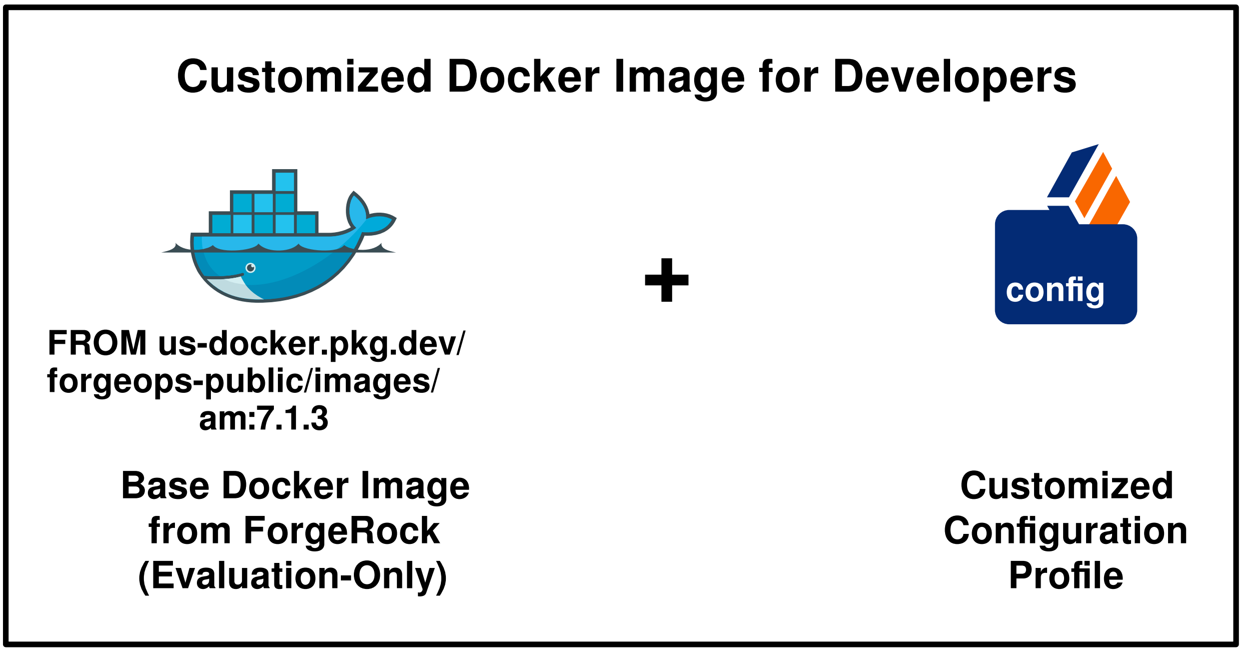
The CDK uses Docker for containerization. The CDK leverages the following Docker capabilities:

- **File-Based Representation of Containers**. Docker *images* contain a file system and run-time configuration information. Docker *containers* are running instances of Docker images.
- **Modularization**. Docker images are based on other Docker images. For example, an AM image is based on a Tomcat image that is itself based on an OpenJDK JRE image. In this example, the AM container has AM software, Tomcat software, and the OpenJDK JRE.
- **Collaboration**. Public and private Docker registries let users collaborate by providing cloud-based access to Docker images. Continuing with the example, the public Docker registry at https://hub.docker.com/ has Docker images for Tomcat and the OpenJDK JRE that any user can download. You build Docker images for the ForgeRock Identity Platform based on the Tomcat and OpenJDK JRE images in the

public Docker registry. You can then push the Docker images to a private Docker registry that other users in your organization can access.

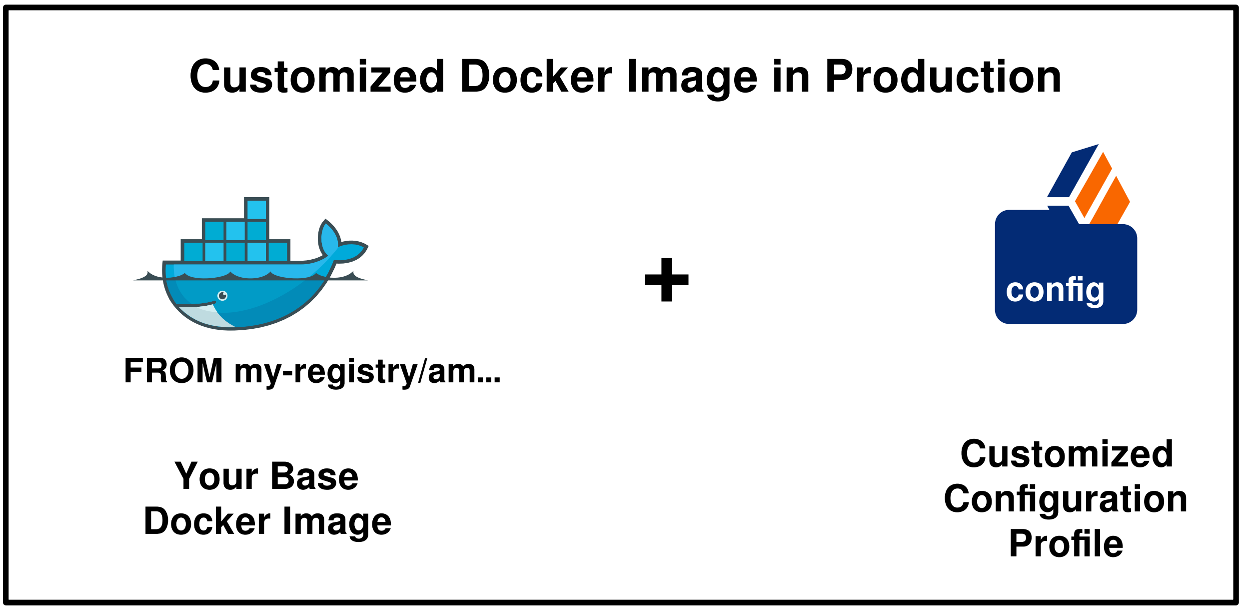
ForgeRock provides a set of unsupported, evaluation-only base images for the ForgeRock Identity Platform. These images are available in ForgeRock's public Docker registry.

Developers working with the CDK use the base images from ForgeRock to build customized Docker images for a fully-configured ForgeRock Identity Platform deployment:



Users working with the CDM also use the base images from ForgeRock to perform proof-of-concept deployments.

Except for several Docker images that implement user interface elements, the base images from ForgeRock are evaluation-only. *They are unsupported for production use.* Because of this, you must build your own base images before you deploy in production:



For information about how to build base images for deploying the ForgeRock Identity Platform in production, see Base Docker Images.

Orchestration

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

- Single-node deployments suitable for proofs of concept and development:
  - Minikube
- Cloud-based Kubernetes orchestration frameworks. These are suitable for both 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 (Minikube|Shared Cluster)
- ❏ Set up your local environment (Minikube|Shared Cluster)
- ❏ Deploy the platform
- ❏ Access platform UIs and APIs
- ❏ Develop custom Docker images

CDK Architecture: Minikube

IMPORTANT

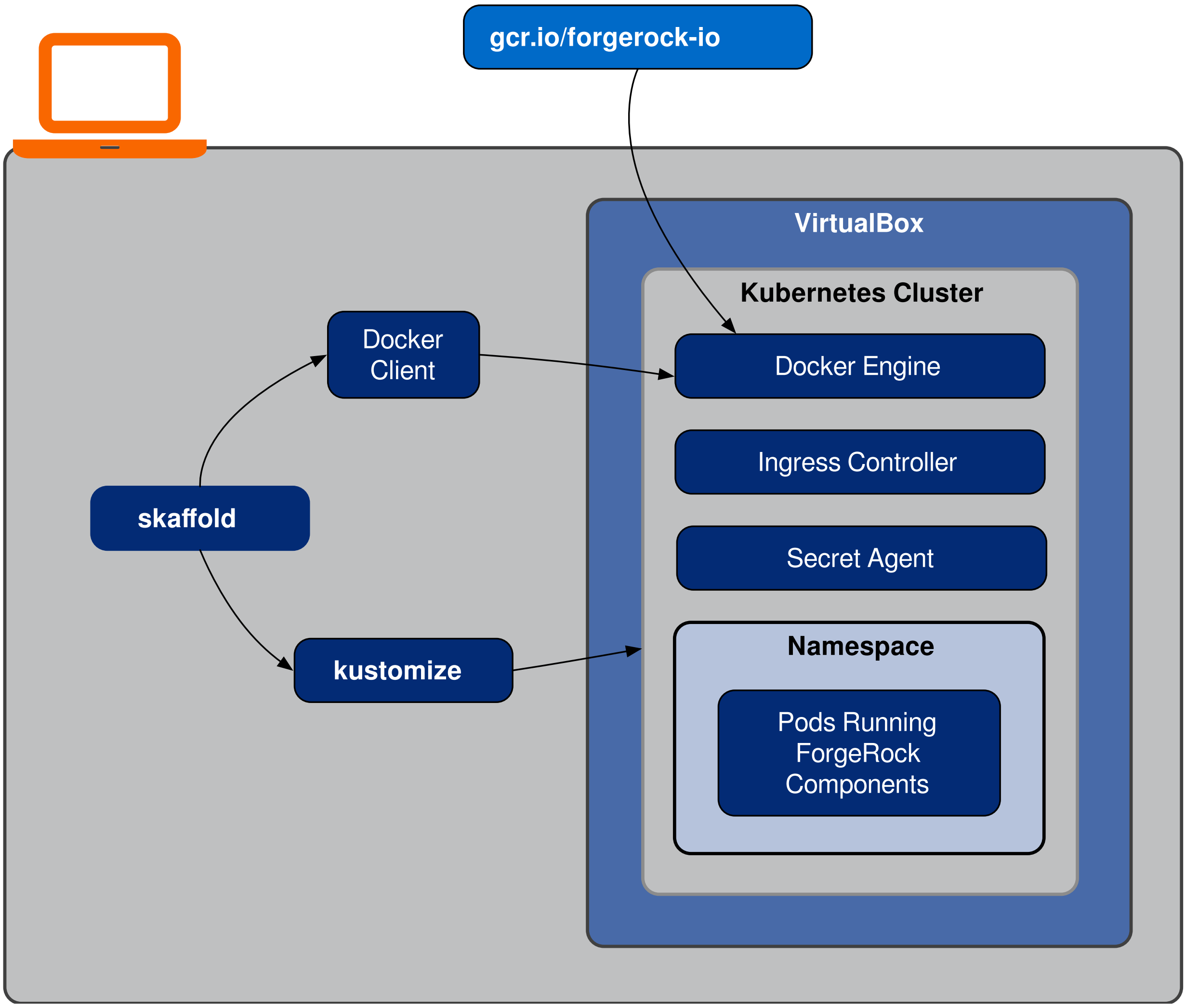
This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

The CDK uses Skaffold to trigger Docker image builds and Kubernetes orchestration. Here's what Skaffold does:

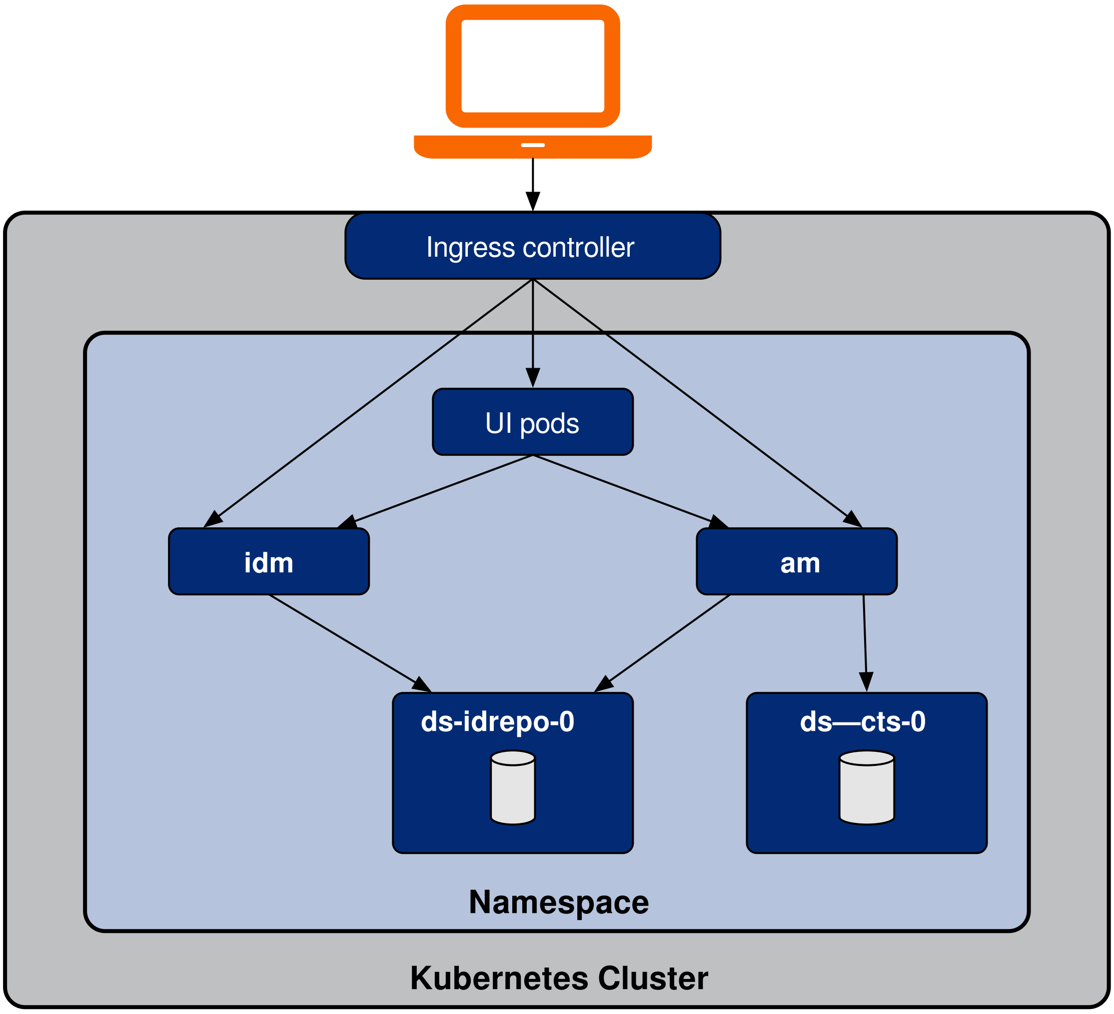
1. Calls the Docker client on the local computer to build and tag their customized Docker images for the ForgeRock Identity Platform. The customized images are based on Docker images in ForgeRock's public Docker registry, `gcr.io/forgerock-io`.
2. Pushes the Docker images to the Docker engine that's part of the Minikube VM.

3. Calls Kustomize to orchestrate the ForgeRock Identity Platform in your namespace. Kustomize uses the Docker images that Skaffold pushed to your Docker registry.

The following diagram illustrates how the CDK uses Skaffold to build and orchestrate Docker images on Minikube:



After deploying the ForgeRock Identity Platform, you'll see the following pods running in your namespace:



am

The `am` pod runs AM.

When AM starts, it obtains its configuration from the `/home/forgerock/openam/config` directory [14].

After the `am` pod has started, an Amster job is triggered. This job populates AM's run-time data.

ds-cts-0

The `ds-cts-0` pod runs the directory service used by the AM Core Token Service.

ds-idrepo-0

The `ds-idrepo-0` pod runs the following directory services:

- Identity repository shared by AM and IDM
- IDM repository
- AM application and policy store

idm

The `idm` pod runs IDM.

When IDM starts, it obtains its configuration from the `/opt/openidm/conf` directory [15].

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`

In addition to these pods, you'll see that two jobs that load data into the environment have run to completion:

- The `amster` job, which loads application data, such as OAuth 2.0 client definitions, to the `idrepo` DS instance.
- The `ldif-importer` job, which sets passwords for the DS `idrepo and cts` instances.

The CDK also requires two other services that are external to your namespace:

- Minikube's ingress controller plugin, for providing external access to services in the Minikube cluster.
- ForgeRock's Secret Agent operator, for generating and managing Kubernetes secrets.

Next Step

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube | Shared Cluster)
- ❑ Set up your local environment (Minikube | Shared Cluster)
- ❑ Deploy the platform
- ❑ Access platform UIs and APIs
- ❑ Develop custom Docker images

CDK Architecture: Shared Cloud Cluster

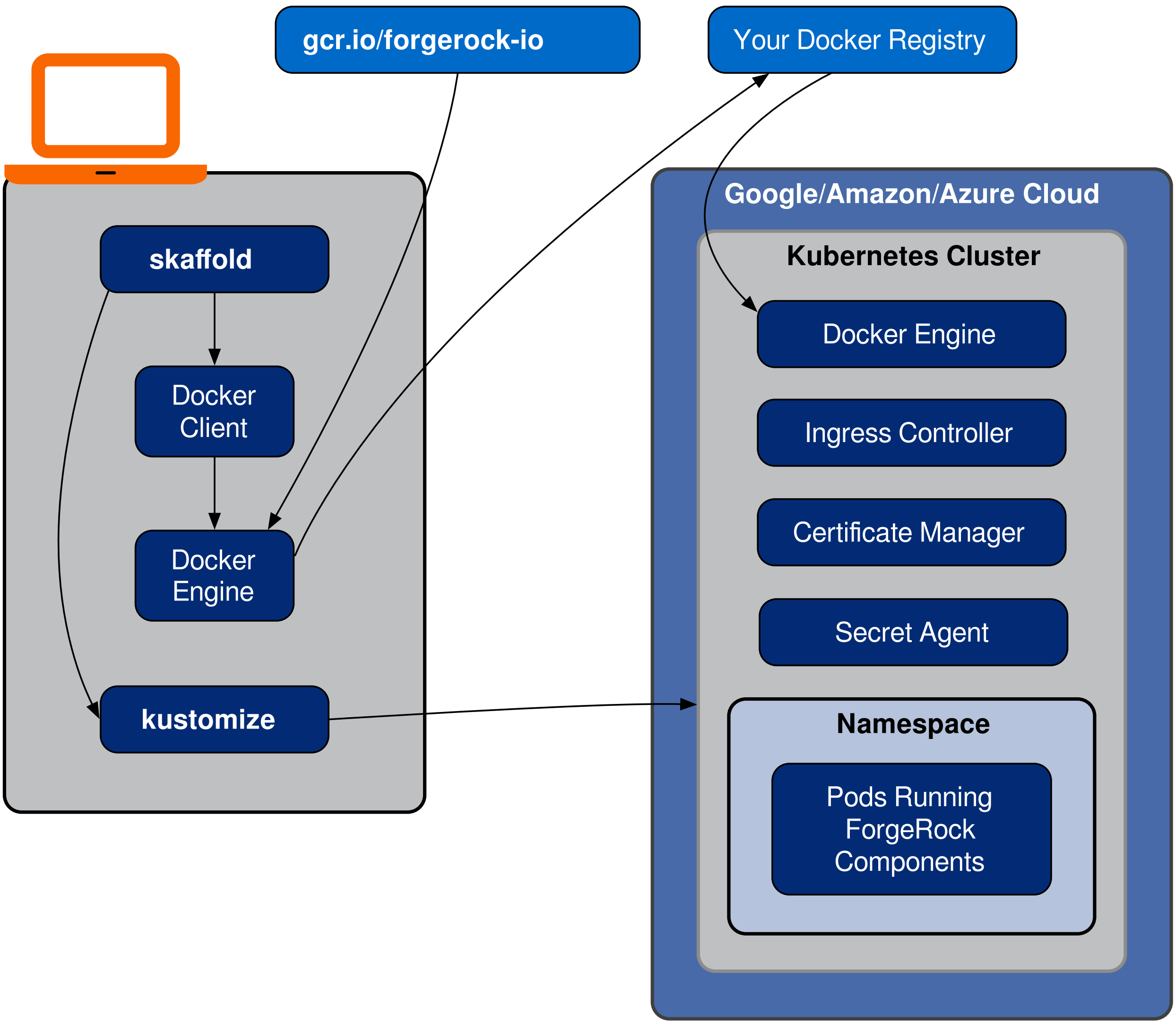
> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

A shared cluster lets multiple developers deploy and configure the ForgeRock Identity Platform on a single cloud-based Kubernetes cluster. A Kubernetes administrator sets up the shared cluster, then provides details to the developers so that they can access the cluster. Each developer then works in their own isolated environment within the cluster, called a *namespace*.

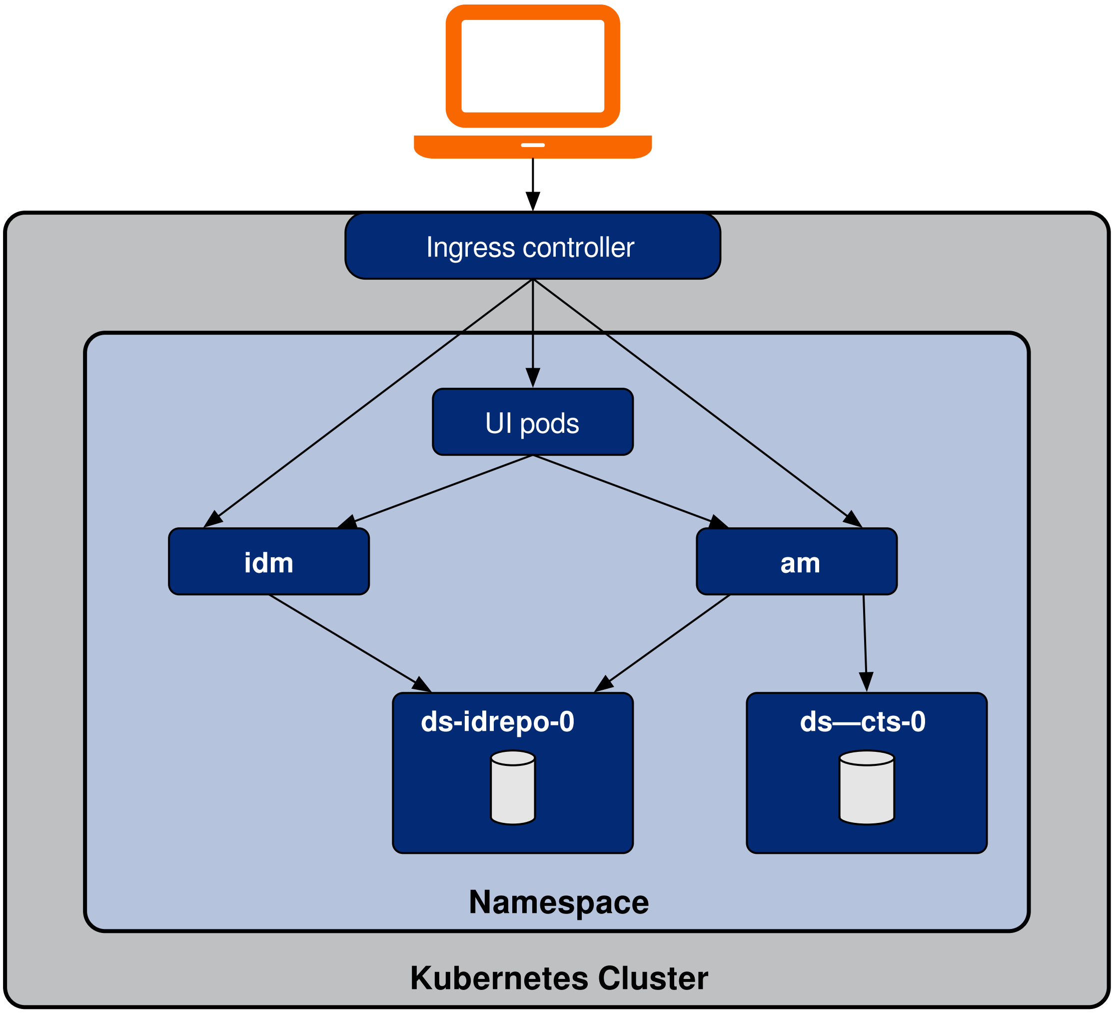
The CDK uses Skaffold to trigger Docker image builds and Kubernetes orchestration. Here's what Skaffold does:

1. Calls the Docker client on the local computer to build and tag their customized Docker images for the ForgeRock Identity Platform. The customized images are based on Docker images in ForgeRock's public Docker registry, `gcr.io/forgerock-io`.
2. Pushes the Docker images to a Docker registry accessible to the shared cluster.
3. Calls Kustomize to orchestrate the ForgeRock Identity Platform in your namespace. Kustomize uses the Docker images that Skaffold pushed to your Docker registry.

The following diagram illustrates how the CDK uses Skaffold to build Docker images locally, push them to a shared registry, and orchestrate them in a shared cluster:



After deploying the ForgeRock Identity Platform, you'll see the following pods running in your namespace:



am

The `am` pod runs AM.

When AM starts, it obtains its configuration from the `/home/forgerock/openam/config` directory [16].

After the `am` pod has started, an Amster job is triggered. This job populates AM's run-time data.

ds-cts-0

The `ds-cts-0` pod runs the directory service used by the AM Core Token Service.

ds-idrepo-0

The `ds-idrepo-0` pod runs the following directory services:

- Identity repository shared by AM and IDM
- IDM repository
- AM application and policy store

idm

The `idm` pod runs IDM.

When IDM starts, it obtains its configuration from the `/opt/openidm/conf` directory [17].

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`

In addition to these pods, you'll see that two jobs that load data into the environment have run to completion:

- The `amster` job, which loads application data, such as OAuth 2.0 client definitions, to the `idrepo` DS instance.
- The `ldif-importer` job, which sets passwords for the DS `idrepo` and `cts` instances.

The CDK also requires three other services that are external to your namespace. These services must be installed by your cluster administrator before you attempt to install the CDK in your shared cluster:

- An NGINX ingress controller, for providing external access to services in the shared cluster.
- Certificate manager, for obtaining and installing security certificates.
- ForgeRock's Secret Agent operator, for generating and managing Kubernetes secrets.

Next Step

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ❑ Set up your local environment (Minikube|Shared Cluster)
- ❑ Deploy the platform
- ❑ Access platform UIs and APIs
- ❑ Develop custom Docker images

Environment Setup: Minikube

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before deploying the CDK, you must set up your local environment with a Minikube cluster.

▼ Windows users

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: Ubuntu 19.10 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.

Environment Setup Checklist

- ❑ Get the forgeops repository
- ❑ Install third-party software
- ❑ Create the Minikube VM
- ❑ Create a Kubernetes namespace
- ❑ Optionally install a TLS certificate
- ❑ Set up hostname resolution
- ❑ Use Minikube's Docker engine

After you've completed all of these environment setup tasks, you'll be ready to deploy the ForgeRock Identity Platform on your new Minikube cluster.

`forgeops` Repository

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

```
$ cd forgeops
$ git checkout release/7.1-20240223
```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❏ Install third-party software
- ❏ Create the Minikube VM
- ❏ Create a Kubernetes namespace
- ❏ Optionally install a TLS certificate
- ❏ Set up hostname resolution
- ❏ Use Minikube's Docker engine

Third-Party Software

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before installing the CDK, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the following table have been validated for building custom Docker images for the ForgeRock Identity Platform. Earlier and later versions will *probably* work. If you want to try using versions that are not in the tables, it is your responsibility to validate them.

Install all of the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
VirtualBox	6.1.32	`virtualbox` (cask)[1]
Minikube	1.25.2	`minikube`
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❏ Create the Minikube VM
- ❏ Create a Kubernetes namespace
- ❏ Optionally install a TLS certificate
- ❏ Set up hostname resolution
- ❏ Use Minikube's Docker engine

Minikube Virtual Machine

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Minikube is a tool that runs a single-node Kubernetes cluster in a virtual machine.

The following configuration has been validated for building custom Docker images for the ForgeRock Identity Platform using Minikube:

- Kubernetes version: stable version. See the Minikube CLI documentation.
- Memory: 10 GB or more.
- Disk space: 40 GB or more.

To set up Minikube:

1. Use the **minikube start** command to create a Minikube VM. In this example, the Minikube VM is created with a Kubernetes cluster suitable for building custom Docker images for the ForgeRock Identity Platform:

```
$ minikube start --memory=12288 --cpus=3 --disk-size=40g --
cni=true --vm=true \
 --driver=virtualbox --bootstrapper kubeadm --kubernetes-
version=stable
?  minikube v1.23.2 on Darwin 11.5.1
✨  Using the virtualbox driver based on user configuration
?  Downloading VM boot image …
    > minikube-v1.23.1.iso.sha256: 65 B / 65 B [-------------]
100.00% ? p/s 0s
    > minikube-v1.23.1.iso: 225.22 MiB / 225.22 MiB [ 100.00%
4.00 MiB p/s 1m2s
?  Starting control plane node minikube in cluster minikube
?  Creating virtualbox VM (CPUs=3, Memory=12288MB,
Disk=40960MB) …
?  Preparing Kubernetes on Docker 20.10.6 …
    ▪ Generating certificates and keys …
    ▪ Booting up control plane …
    ▪ Configuring RBAC rules …
?  Configuring CNI (Container Networking Interface) …
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
?  Enabled addons: default-storageclass, storage-provisioner
?  Verifying Kubernetes components. . .
?  Done! kubectl is now configured to use "minikube" by
default
```

2. Run the following command to enable the ingress controller built into Minikube:

```
$ minikube addons enable ingress
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
    ▪ Using image docker.io/jettech/kube-webhook-
certgen:v1.5.1
    ▪ Using image docker.io/jettech/kube-webhook-
certgen:v1.5.1
? Verifying ingress addon…
? The 'ingress' addon is enabled
```

3. Install the Secret Agent operator:

```
$ kubectl apply -f https://github.com/ForgeRock/secret-
agent/releases/latest/download/secret-agent.yaml
namespace/secret-agent-system created
customresourcedefinition.apiextensions.k8s.io/secretagentconfi
gurations.secret-agent.secrets.forgerock.io created
serviceaccount/secret-agent-manager-service-account created
role.rbac.authorization.k8s.io/secret-agent-leader-election-
role created
clusterrole.rbac.authorization.k8s.io/secret-agent-manager-
role created
rolebinding.rbac.authorization.k8s.io/secret-agent-leader-
election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/secret-agent-
manager-rolebinding created
service/secret-agent-webhook-service created
deployment.apps/secret-agent-controller-manager created
Warning: admissionregistration.k8s.io/v1beta1
MutatingWebhookConfiguration is deprecated in v1.16+,
unavailable in v1.22+; use admissionregistration.k8s.io/v1
MutatingWebhookConfiguration
mutatingwebhookconfiguration.admissionregistration.k8s.io/secr
et-agent-mutating-webhook-configuration created
Warning: admissionregistration.k8s.io/v1beta1
ValidatingWebhookConfiguration is deprecated in v1.16+,
unavailable in v1.22+; use admissionregistration.k8s.io/v1
ValidatingWebhookConfiguration
validatingwebhookconfiguration.admissionregistration.k8s.io/se
cret-agent-validating-webhook-configuration created
```

Next Step

✓ Get the forgeops repository

- ✓ Install third-party software
- ✓ Create the Minikube VM
- ❑ Create a Kubernetes namespace
- ❑ Optionally install a TLS certificate
- ❑ Set up hostname resolution
- ❑ Use Minikube's Docker engine

Namespace

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Create a namespace in your new cluster.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the default namespace. Deploying to a non-default namespace lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your active namespace:

```
$ kubens my-namespace
Context "minikube" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Create the Minikube VM
- ✓ Create a Kubernetes namespace
- ❑ Optionally install a TLS certificate
- ❑ Set up hostname resolution

- ❑ Use Minikube's Docker engine

Hostname Resolution

IMPORTANT

> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

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

1. Run the **`minikube ip`** command to get the Minikube ingress controller's IP address:

   ```
   $ minikube ip
   192.168.99.100
   ```

2. Add an entry to the `/etc/hosts` file to resolve the deployment FQDN. For example:

   ```
   minikube-ip-address my-namespace.iam.example.com
   ```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Create the Minikube VM
- ✓ Create a Kubernetes namespace
- ✓ Optionally install a TLS certificate
- ✓ Set up hostname resolution
- ❑ Use Minikube's Docker engine

TLS Certificate (Optional)

IMPORTANT

> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

This page covers several options you can use to encrypt HTTP communications over TLS in CDK deployments.

Self-Signed Certificate

By default, Minikube's ingress controller plugin is configured with a self-signed certificate. This is the simplest encryption option—you don't have to make any changes to the CDK to get encryption.

However, when you access one of the ForgeRock web applications from your browser, you'll get a "Not Secure" message from your browser. You'll need to bypass the message.

Certificate From a Certificate Authority (CA)

If you have a certificate from a CA, you can use the certificate for TLS encryption. Install the certificate and your private key in a Kubernetes secret in your namespace. Minikube's ingress controller plugin gets the certificate from the secret, and then uses it to encrypt communications.

To use a certficate from a CA in a CDK deployment on Minikube:

1. Obtain the certificate:
   - Make sure that the certificate is PEM-encoded.
   - A best practice is to include the entire trust chain in your `.pem` file.
   - Make sure that the certificate's host name works with the FQDN you'll use when you deploy the platform.

     In the CDK, the deployment FQDN is set to `*my-namespace*.iam.example.com` by default. But you'll want to use a certificate with your own domain, not with `example.com`. Because of this:

     - When you deploy the CDK. you'll need to change the deployment FQDN to use your own domain name. For example, change `*my-namespace*.iam.example.com` to `*my-namespace*.iam.*my-domain*.com`.
     - Your certificate should also specify this host name. If you're using a wildcard certificate that could be used by multiple developers, the certificate's host name should be `*.iam.*my-domain*.com`.
     - Note that you *can* deploy the CDK with a deployment FQDN that does not include the subdomain `iam`. You might want to do this if you have a wildcard certificate for `*.*my-domain*.com`. However, ForgeRock recommends that you use a subdomain, such as `iam`, in your deployment FQDN when possible. Using a subdomain can help you avoid FQDN collisions, provide simpler routing, and simplify DNS.
2. Create a secret named `sslcert` in your namespace that contains the certificate. For example:

```
$ kubectl create secret tls sslcert --cert=/path/to/my-cert.crt --key=/path/to/my-key.key
```

Certificate Generated by the mkcert Utility

If you don't have a certificate from a CA, you can use the mkcert utility to generate a locally trusted certificate. In many cases, it's acceptable to use such certificates for development purposes.

To use a certificate generated by the mkcert utility in a CDK deployment on Minikube:

1. If you don't have mkcert software installed locally, install it. Firefox users also need to install certutil software. See the mkcert installation instructions for more information.
2. If you haven't ever done so, run the **`mkcert -install`** command to create a local certificate authority (CA) and install it in your system root store. Restart your browser after creating the local CA.
3. Create a wildcard certificate for the `iam.example.com` domain:

   ```
   $ cd
   $ mkcert "*.iam.example.com"
   ```

4. Create a secret named `sslcert` in your namespace that contains the wildcard certificate. For example:

   ```
   $ kubectl create secret tls sslcert --cert=./_wildcard.iam.example.com.pem --key=./_wildcard.iam.example.com-key.pem
   ```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Create the Minikube VM
- ✓ Create a Kubernetes namespace
- ✓ Optionally install a TLS certificate
- ❏ Set up hostname resolution
- ❏ Use Minikube's Docker engine

Minikube's Docker Engine

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

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 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:

1. Run the **docker-env** command in your shell:

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

2. Stop Skaffold from pushing Docker images to a remote Docker registry [18]:

```
$ skaffold config set --kube-context minikube local-cluster
true
set value local-cluster to true for context minikube
```

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.

Next Step

You've completed all the setup tasks for Minikube. Now you're ready to deploy the platform in the Minikube cluster:

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ✓ Set up your local environment (Minikube|Shared Cluster)
- ❏ Deploy the platform
- ❏ Access platform UIs and APIs
- ❏ Develop custom Docker images

Environment Setup: Shared Cloud Cluster

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before deploying the CDK, you must set up your local environment to communicate with the shared cluster.

▼ Windows users

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: Ubuntu 19.10 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.

Environment Setup Checklists

Perform the tasks in the checklist *for your cloud provider only.*

GKE Shared Cluster

- ❏ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared GKE cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

EKS Shared Cluster

- ❏ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared EKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

AKS Shared Cluster

- ❏ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared AKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

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

`forgeops` Repository

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

```
$ cd forgeops
$ git checkout release/7.1-20240223
```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❑ Install third-party software
- ❑ Get details about the shared GKE cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Third-Party Software

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before installing the CDK, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the table below have been validated for building custom Docker images for the ForgeRock Identity Platform. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`

Software	Version	Homebrew package
Kubernetes client (**kubectl**)	1.23.5	kubectl
Kubernetes context switcher (**kubectx**)	0.9.4	kubectx
Kustomize	4.5.3	kustomize
Google Cloud SDK	378.0.0	google-cloud-sdk (cask) [1]
Docker Desktop[2]	4.6.1	docker (cask)[1]
Skaffold	2.0.1	skaffold

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❑ Get details about the shared GKE cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Cluster Details

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

You'll need to get some information about the cluster from your cluster administrator. You'll provide this information as you perform various tasks to access the cluster.

1. Obtain the following cluster details:
   - The name of the Google Cloud project that contains the cluster.
   - The cluster name.
   - The Google Cloud zone in which the cluster resides.
   - The IP address of your cluster's ingress controller.

- The location of the Docker registry from which your cluster will obtain images for the ForgeRock Identity Platform.

2. Verify that the Secret Agent operator is installed in the cluster.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Context for the Shared Cluster

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Kubernetes uses contexts to access Kubernetes clusters. Before you can access the shared cluster, you must create a context on your local computer if it's not already present.

To create a context for the shared cluster:

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted:
    - If the current context references the shared cluster, there is nothing further to do. Proceed to Namespace.
    - If the context of the shared cluster is present in the **kubectx** command output, set the context as follows:

      ```
      $ kubectx my-context
      Switched to context "my-context".
      ```

      After you have set the context, proceed to Namespace.
    - If the context of the shared cluster is not present in the **kubectx** command output, continue to the next step.
2. Configure the Google Cloud SDK standard component to use your Google account. Run the following command:

```
$ gcloud auth login
```

3. A browser window prompts you to log in to Google. Log in using your Google account.

   A second screen requests several permissions. Select Allow.

   A third screen should appear with the heading, "You are now authenticated with the Google Cloud SDK!"

4. Return to the terminal window and run the following command. Use the cluster name, zone, and project name you obtained from your cluster administrator:

```
$ gcloud container clusters \
 get-credentials cluster-name --zone google-zone --project
google-project
Fetching cluster endpoint and auth data.
kubeconfig entry generated for cluster-name.
```

5. Run the **kubectx** command again and verify that the context for our Kubernetes cluster is now the current context.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ✓ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Namespace

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Create a namespace in the shared cluster. Namespaces let you isolate your deployments from other developers' deployments.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the `default` namespace. Deploying to a non-default namespace

lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Hostname Resolution

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

You might need to set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace.

Take the following actions:

1. Determine whether DNS resolves the hostname, `my-namespace.iam.example.com`.
2. If DNS does not resolve the hostname, add an entry to the `/etc/hosts` file similar to the following:

```
ingress-ip-address my-namespace.iam.example.com
```

For *ingress-ip-address*, specify the IP address of your cluster's ingress controller that you obtained from your cluster administrator.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared GKE cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ✓ Set up hostname resolution
- ❏ Prepare to push Docker images

docker push Setup

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

In the environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your GKE cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

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. Set up a Docker credential helper:

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

3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location you obtained from your cluster administrator and the Kubernetes context you obtained in Context for the Shared Cluster:

```
$ skaffold config set default-repo my-docker-registry -k my-kubernetes-context
```

Next Step

You've completed all the setup tasks required before deploying the ForgeRock Identity Platform in a shared GKE cluster. Now you're ready to deploy the platform in your namespace on the shared cluster:

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ✓ Set up your local environment (Minikube|Shared Cluster)
- ❏ Deploy the platform
- ❏ Access platform UIs and APIs
- ❏ Develop custom Docker images

`forgeops` Repository

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

```
$ cd forgeops
$ git checkout release/7.1-20240223
```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared EKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Third-Party Software

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before installing the CDK, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the table below have been validated for building custom Docker images for the ForgeRock Identity Platform. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`

Software	Version	Homebrew package
Kustomize	4.5.3	`kustomize`
Amazon AWS Command Line Interface	2.8.9	`awscli`
AWS IAM Authenticator for Kubernetes	0.5.5	`aws-iam-authenticator`
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❑ Get details about the shared EKS cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Cluster Details

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

You'll need to get some information about the cluster from your cluster administrator. You'll provide this information as you perform various tasks to access the cluster.

1. Obtain the following cluster details:
   - Your AWS access key ID.
   - Your AWS secret access key.
   - The AWS region in which the cluster resides.
   - The cluster name.
   - The IP address of your cluster's ingress controller.
   - The location of the Docker registry from which your cluster will obtain images for the ForgeRock Identity Platform.

2. Verify that the Secret Agent operator is installed in the cluster.

Next Step

- ✔ Get the forgeops repository
- ✔ Install third-party software
- ✔ Get details about the shared EKS cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Context for the Shared Cluster

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Kubernetes uses contexts to access Kubernetes clusters. Before you can access the shared cluster, you must create a context on your local computer if it's not already present.

To create a context for the shared cluster:

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted:
   - If the current context references the shared cluster, there is nothing further to do. Proceed to Namespace.
   - If the context of the shared cluster is present in the **kubectx** command output, set the context as follows:

     ```
     $ kubectx my-context
     Switched to context "my-context".
     ```

     After you have set the context, proceed to Namespace.
   - If the context of the shared cluster is not present in the **kubectx** command output, continue to the next step.
2. Run the **aws configure** command. This command logs you in to AWS and sets the AWS region. Use the access key ID, secret access key, and region you obtained from your cluster administrator. You do not need to specify a value for the default output format:

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

3. Run the following command. Use the cluster name you obtained from your cluster administrator:

```
$ aws eks update-kubeconfig --name my-cluster
Added new context arn:aws:eks:us-east-
1:813759318741:cluster/my-cluster
to /Users/my-user-name/.kube/config
```

4. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

In Amazon EKS environments, the cluster owner must grant access to a user before the user can access cluster resources. For details about how the cluster owner can grant you access to the cluster, refer the cluster owner to Cluster Access for Multiple AWS Users.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared EKS cluster
- ✓ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Namespace

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Create a namespace in the shared cluster. Namespaces let you isolate your deployments from other developers' deployments.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the `default` namespace. Deploying to a non-default namespace

lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared EKS cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Hostname Resolution

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

You might need to set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace.

Take the following actions:

1. Determine whether DNS resolves the hostname, `my-namespace.iam.example.com`.
2. If DNS does not resolve the hostname, add an entry to the `/etc/hosts` file similar to the following:

```
ingress-ip-address my-namespace.iam.example.com
```

For *ingress-ip-address*, specify the IP address of your cluster's ingress controller that you obtained from your cluster administrator.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared EKS cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ✓ Set up hostname resolution
- ❏ Prepare to push Docker images

`docker push` Setup

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

In the environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your EKS cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

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
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.[19]

3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location and the Kubernetes context:

```
$ skaffold config set default-repo my-docker-registry -k my-
kubernetes-context
```

Next Step

You've completed all the setup tasks required before deploying the ForgeRock Identity Platform in a shared EKS cluster. Now you're ready to deploy the platform in your namespace on the shared cluster:

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ✓ Set up your local environment (Minikube|Shared Cluster)
- ❑ Deploy the platform
- ❑ Access platform UIs and APIs
- ❑ Develop custom Docker images

forgeops Repository

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository and check out the `release/7.1-20240223` branch:

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

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

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

2. Check out the `release/7.1-20240223` branch:

```
$ cd forgeops
$ git checkout release/7.1-20240223
```

Depending on your organization's repository strategy, you might need to clone the repository from a fork, instead of cloning ForgeRock's master repository. You might also need to create a working branch from the `release/7.1-20240223` branch. For more information, see Repository Updates.

Next Step

- ✓ Get the forgeops repository
- ❏ Install third-party software
- ❏ Get details about the shared AKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Third-Party Software

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Before installing the CDK, you must obtain non-ForgeRock software and install it on your local computer.

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

The versions listed in the table below have been validated for building custom Docker images for the ForgeRock Identity Platform. Earlier and later versions will *probably* work. If you want to try using versions that are not in the table, it is your responsibility to validate them.

Install the following third-party software:

Software	Version	Homebrew package
Python 3	3.9.9	`python`
Kubernetes client (**`kubectl`**)	1.23.5	`kubectl`
Kubernetes context switcher (**`kubectx`**)	0.9.4	`kubectx`
Kustomize	4.5.3	`kustomize`
Azure Command Line Interface	2.42.0	`azure-cli`
Docker Desktop[2]	4.6.1	`docker` (cask)[1]
Skaffold	2.0.1	`skaffold`

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ❑ Get details about the shared AKS cluster
- ❑ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Cluster Details

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

You'll need to get some information about the cluster from your cluster administrator. You'll provide this information as you perform various tasks to access the cluster.

1. Obtain the following cluster details:
   - The ID of the Azure subscription that contains the cluster. Be sure to obtain the hexadecimal subscription ID, not the subscription name.
   - The name of the resource group that contains the cluster.
   - The cluster name.

   - The IP address of your cluster's ingress controller.
   - The location of the Docker registry from which your cluster will obtain images for the ForgeRock Identity Platform.
2. Verify that the Secret Agent operator is installed in the cluster.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ❏ Create a Kubernetes context
- ❏ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Context for the Shared Cluster

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Kubernetes uses contexts to access Kubernetes clusters. Before you can access the shared cluster, you must create a context on your local computer if it's not already present.

To create a context for the shared cluster:

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted:
   - If the current context references the shared cluster, there is nothing further to do. Proceed to Namespace.
   - If the context of the shared cluster is present in the **kubectx** command output, set the context as follows:

     ```
     $ kubectx my-context
     Switched to context "my-context".
     ```

     After you have set the context, proceed to Namespace.
   - If the context of the shared cluster is not present in the **kubectx** command output, continue to the next step.

2. Configure the Azure CLI to use your Microsoft Azure. Run the following command:

```
$ az login
```

3. A browser window prompts you to log in to Azure. Log in using your Microsoft account.

   A second screen should appear with the message, "You have logged into Microsoft Azure!"

4. Return to the terminal window and run the following command. Use the resource group, cluster name, and subscription ID you obtained from your cluster administrator:

```
$ az aks get-credentials \
 --resource-group my-fr-resource-group \
 --name my-fr-cluster \
 --subscription your subscription ID \
 --overwrite-existing
```

5. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ✓ Create a Kubernetes context
- ❑ Create a Kubernetes namespace
- ❑ Set up hostname resolution
- ❑ Prepare to push Docker images

Namespace

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Create a namespace in the shared cluster. Namespaces let you isolate your deployments from other developers' deployments.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the default namespace. Deploying to a non-default namespace lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

To create a namespace:

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ❏ Set up hostname resolution
- ❏ Prepare to push Docker images

Hostname Resolution

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

You might need to set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace.

Take the following actions:

1. Determine whether DNS resolves the hostname, `my-namespace.iam.example.com`.

2. If DNS does not resolve the hostname, add an entry to the `/etc/hosts` file similar to the following:

```
ingress-ip-address my-namespace.iam.example.com
```

For *`ingress-ip-address`*, specify the IP address of your cluster's ingress controller that you obtained from your cluster administrator.

Next Step

- ✓ Get the forgeops repository
- ✓ Install third-party software
- ✓ Get details about the shared AKS cluster
- ✓ Create a Kubernetes context
- ✓ Create a Kubernetes namespace
- ✓ Set up hostname resolution
- ❑ Prepare to push Docker images

`docker push` *Setup*

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

In the environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your AKS cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to be able to push the Docker images:

- Docker must be running on your local computer.
- Your local computer needs credentials that let Skaffold push the images to the Docker registry available to your cluster.
- Skaffold needs to know the location of the Docker registry.

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.

3. Run the **kubectx** command to obtain the Kubernetes context.
4. Configure Skaffold with the Docker registry location you obtained from your cluster administrator and the Kubernetes context you obtained in Context for the Shared Cluster:

```
$ skaffold config set default-repo my-docker-registry -k my-kubernetes-context
```

Next Step

You've completed all the setup tasks required before deploying the ForgeRock Identity Platform in a shared AKS cluster. Now you're ready to deploy the platform in your namespace on the shared cluster:

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ✓ Set up your local environment (Minikube|Shared Cluster)
- ❏ Deploy the platform
- ❏ Access platform UIs and APIs
- ❏ Develop custom Docker images

CDK Deployment

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

After you've set up your development environment, your next step is to deploy the platform.

To deploy the ForgeRock Identity Platform in your namespace:

1. Change the deployment namespace for the `all` environment from the `default` namespace to your namespace:
   a. Change to the directory containing the `all` environment:

   ```
   $ cd /path/to/forgeops/kustomize/overlay/7.0/all
   ```

   b. Open the `kustomization.yaml` file.
   c. Modify two lines in the file so that the platform is deployed in your namespace:

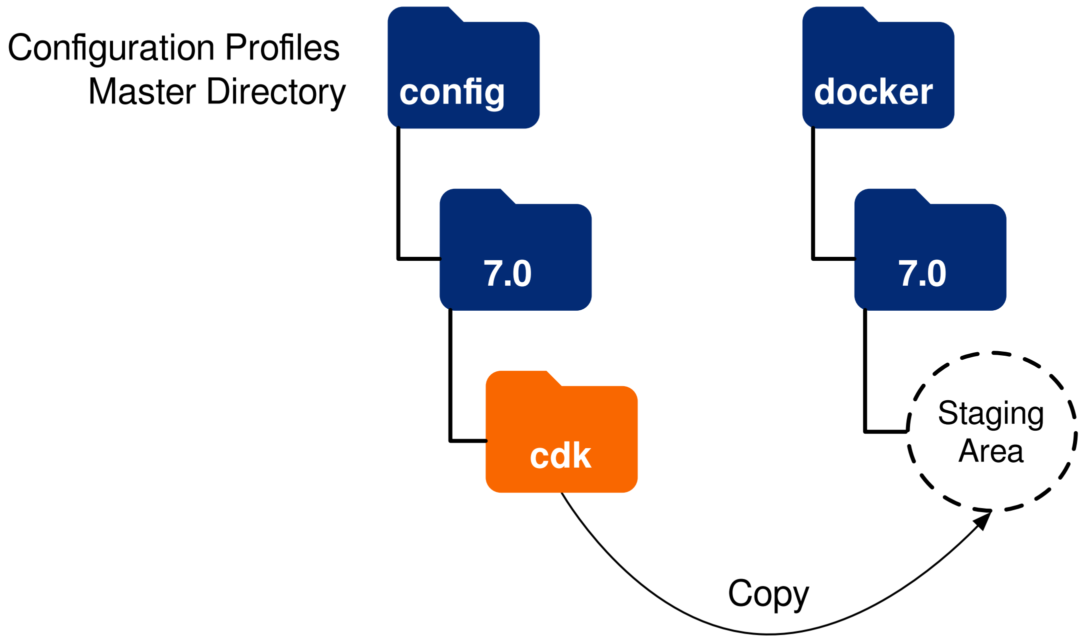
Original Text	Revised Text
`namespace: default`	`namespace: my-namespace`
`FQDN: "default.iam.example.com"`	`FQDN: "my-namespace.iam.example.com"`

d. Save the updated `kustomization.yaml` file.

2. Initialize the staging area for configuration profiles with the canonical CDK configuration profile for the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops/bin
$ ./config.sh init --profile cdk
Removing docker/7.0/am/config/
Removing docker/7.0/amster/config/
Removing docker/7.0/idm/conf/
Removing docker/7.0/idm/ui/
Removing docker/7.0/ig/config/
Copying /Users/me/Repositories/forgeops/config/7.0/cdk/idm.
Copying /Users/me/Repositories/forgeops/config/7.0/cdk/am.
Copying /Users/me/Repositories/forgeops/config/7.0/cdk/ig.
Copying /Users/me/Repositories/forgeops/config/7.0/cdk/amster.
Completed
```

The **`config.sh init`** command copies the canonical CDK configuration profile from the master directory for configuration profiles to the staging area:



For more information about the management of ForgeRock Identity Platform configuration profiles in the `forgeops` repository, see Configuration Profiles.

3. Configure secrets for the ForgeRock Identity Platform:

a. Make sure that context is set to your namespace:

```
$ kubens my-namespace
```

b. Deploy the secrets:

```
$ cd /path/to/forgeops/kustomize/base/secrets
$ kubectl apply --filename secret_agent_config.yaml
```

c. Verify that all the ForgeRock Identity Platform secrets have been created:

```
$ kubectl get sac
NAME            STATUS      NUMSECRETS   NUMK8SSECRETS
forgerock-sac   Completed   14           14
```

When the `forgerock-sac` entry reaches `Completed` status, all the secrets have been created.

4. Run Skaffold to build Docker images and deploy the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops
$ skaffold run
Generating tags. . .
 - am → am:. . .
 - amster → amster:. . .
 - idm → idm:. . .
 - ds-cts → ds-cts:. . .
. . .
```

5. 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.

Next Step

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ✓ Set up your local environment (Minikube|Shared Cluster)
- ✓ Deploy the platform
- ❏ Access platform UIs and APIs
- ❏ Develop custom Docker images

UI and API Access

IMPORTANT

IMPORTANT

> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

Now that you've deployed the ForgeRock 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 ForgeRock Identity Platform's administrative consoles and REST APIs.

You access AM and IDM services through the Kubernetes ingress controller. Access components using their normal interfaces:

- For AM, the console and REST APIs.
- For IDM, the Admin UI 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 console:

1. Make sure that your namespace is the current namespace:

   ```
   $ kubens my-namespace
   ```

2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./print-secrets amadmin
   179rd8en9rffa82rcf1qap1z0gv1hcej
   ```

3. Open a new window or tab in a web browser.
4. Go to https://*my-namespace*.iam.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 ForgeRock Identity Platform UI appears in the browser.

6. Select Native Consoles > Access Management.

   The AM console 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://my-
namespace.iam.example.com/am/json/realms/root/authenticate"
{
    "tokenId":"AQIC5wM2. . .TU3OQ*",
    "successUrl":"/am/console",
    "realm":"/"
}
```

IDM Services

To access the IDM Admin UI:

1. Make sure that your namespace is the current namespace:

```
$ kubens my-namespace
```

2. Obtain the `amadmin` user's password:

```
$ cd /path/to/forgeops/bin
$ ./print-secrets amadmin
vr58qt11ihoa31zfbjsdxxrqryfw0s31
```

3. Open a new window or tab in a web browser.
4. Go to https://*my-namespace*.iam.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 ForgeRock 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 **`print-secrets`** 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:

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

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

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

```
$ curl \
 --dump-header - \
```

```
 --insecure \
 --request GET \
 --Cookie "iPlanetDirectoryPro= AQIC5wM. . .TU3OQ* " \
 "https://my-
namespace.iam.example.com/am/oauth2/realms/root/authorize?
redirect_uri=https://my-
namespace.iam.example.com/platform/appAuthHelperRedirect.h
tml&client_id=idm-admin-
ui&scope=openid%20fr:idm:*&response_type=code&state=abc123
"
HTTP/2 302
server: nginx/1.17.10
date: Tue, 21 Jul 2020 16:54:20 GMT
content-length: 0
location: https://my-
namespace.iam.example.com/platform/appAuthHelperRedirect.h
tml
 ?code= 3cItL9G52DIiBdfXRngv2_dAaYM &iss=http://my-
namespace.iam.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
strict-transport-security: max-age=15724800;
includeSubDomains
```

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

```
$ curl --request POST \
 --insecure \
 --data "grant_type=authorization_code" \
 --data "code= 3cItL9G52DIiBdfXRngv2_dAaYM " \
 --data "client_id=idm-admin-ui" \
 --data "redirect_uri=https://my-
namespace.iam.example.com/platform/appAuthHelperRedirect.h
tml" \
 "https://my-
namespace.iam.example.com/am/oauth2/realms/root/access_tok
en"
```

```
{
 "access_token":" oPzGzGFY1SeP2RkI-ZqaRQC1cDg ",
 "scope":"openid fr:idm:*",
 "id_token":"eyJ0eXAiOiJKV
  . . .
  sO4HYqlQ",
 "token_type":"Bearer",
 "expires_in":239
}
```

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

   The following example command provides information about the IDM configuration:

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

Directory Services

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 **print-secrets dsadmin** command.

Next Step

- ✓ Become familiar with the CDK
- ✓ Understand CDK architecture (Minikube|Shared Cluster)
- ✓ Set up your local environment (Minikube|Shared Cluster)
- ✓ Deploy the platform
- ✓ Access platform UIs and APIs
- ❏ Develop custom Docker images

Custom Docker Image Development

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

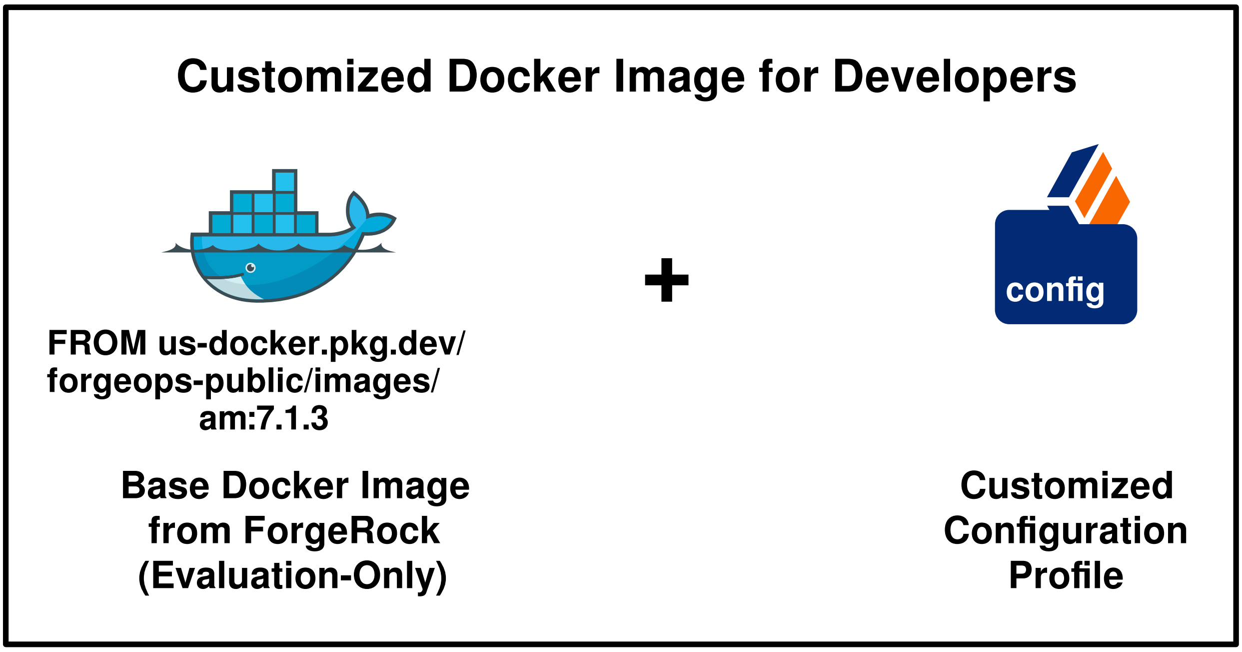
Before you can develop custom Docker images, you must have deployed the ForgeRock Identity Platform and learned how to access its administration GUIs and REST APIs. Now you're ready to configure the platform to meet your needs. As you configure the platform, you can decide at any point to build new custom Docker images that will incorporate the configuration changes you've made.

Repeat this process as you develop custom Docker images:

- Access AM and IDM running in the CDK, and customize them using their GUIs and REST APIs.
- Export your customizations from the CDK to a Git repository on your local computer.
- Rebuild the Docker images for the platform with your new customizations.
- Redeploy the platform on the CDK.

Before you build customized Docker images for the platform, be sure you're familiar with the types of data used by the platform. This conceptual information helps you understand which type of data is included in custom Docker images.

To develop customized Docker images, start with base images and a canonical configuration profile from ForgeRock. Then, build up a configuration profile, customizing the platform to meet your needs. The configuration profile is integrated into the customized Docker image:



Before you deploy the platform in production, you'll need to stop using ForgeRock's evaluation-only base images, and start using base images you build yourself. Building your own base images is covered in Base Docker Images.

Types of Configuration

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

The ForgeRock Identity Platform uses three types of configuration: static configuration, dynamic configuration, and identities.

Static Configuration

Static configuration consists of properties and settings used by the ForgeRock 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 ForgeRock 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 and IDM Images for more detailed steps.

In ForgeRock 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 ForgeRock Identity Platform. Examples of dynamic configuration include AM access policies, AM OAuth 2.0 client definitions, and IDM relationships.

In ForgeRock Identity Platform deployments, dynamic configuration is *not immutable*. You can change dynamic configuration 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.

> NOTE
>
> The CDK and CDM run an Amster job when they start. The job loads dynamic configuration required by the CDK and CDM into the AM application and policy store.
>
> The required dynamic configuration is stored in the `amster` Docker image. You can include additional dynamic configuration in this Docker image as a convenient way of loading your own applications and policies during CDK and CDM startup.
>
> See xref:legacy/cdk/develop/am-idm.adoc for detailed steps to build the `amster` Docker image.

Identities

Identities are another type of dynamic configuration. They can be modified at any time, including when the platform is running in production. Identities are never incorporated into Docker images for the platform.

As with AM and IDM dynamic configuration, you'll need to devise a strategy to manage identities that lets you:

- Extract sample user identities that can be used by developers.
- Back up and restore user identities.

Configuration Profiles

A ForgeRock 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.
- AM dynamic configuration to be loaded into the AM application and policy store when the CDK and CDM start up.

Configuration profiles reside in two locations in the `forgeops` repository:

- **The master directory**. Holds a canonical configuration profile for the CDK and user-customized configuration profiles. User-customized configuration profiles in this directory are considered to be the *source of truth* for ForgeRock Identity Platform deployments.

  The master directory for configuration profiles is located at the path `/path/to/forgeops/config/7.0`. Use Git to manage the configuration profiles in this directory.

- **The staging area**. Holds a single configuration profile. You copy a profile from the master directory to the staging area before building a customized Docker image for the ForgeRock Identity Platform.

  The staging area is located in subdirectories of the path, `/path/to/forgeops/docker/7.0`. Configuration profiles copied to the staging area are transient and are not managed with Git.

The **`config.sh`** script lets you copy configuration profiles between the master directory and the staging area. You run this script before you build customized Docker images for the platform. The script lets you specify which configuration profile to copy to the staging area. The **`skaffold run`** command uses the profile that's been copied to the staging area when it builds a Docker image.

AM and IDM Images

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

AM Images

AM uses two Docker images, `am` and `amster`:

- The `am` image contains your custom AM configuration.
- The `amster` image contains your custom AM run-time data.

With AM up and running, you can iteratively update the `am` Docker image:

- Customize AM's configuration and run-time data using the console and the REST APIs.
- Capture changes to the AM configuration by synchronizing them from the AM service running on Kubernetes back to the staging area and the master directory for configuration profiles in your `forgeops` repository clone.
- Run Skaffold to detect the changes, rebuild the `am` Docker image, and restart AM. You can then test changes you've made to the AM configuration based on the updated Docker image.

You can also iteratively update the `amster` image:

- Capture changes to AM run-time data by synchronizing the changes from the AM service running on Kubernetes back to the staging area and the master directory for configuration profiles in your `forgeops` repository clone.

  AM run-time data includes:

  - OAuth 2.0 clients
  - OpenID Connect 1.0 clients
  - IG, Web, Java, and SOAP STS agents
  - Policies
  - SAML v2.0 circles of trust and entities
- Run Skaffold to detect the changes and rebuild the `amster` Docker image.

`am` Image

The `am` Docker image contains the AM configuration.

Perform the following steps iteratively when developing a customized `am` Docker image:

1. Perform version control activities on your `forgeops` repository clone:
   a. Run the **`git status`** command.

b. Review the state of the working directory and staging area.

c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

2. Make sure that context is set to your namespace:

```
$ kubens my-namespace
```

3. Modify the AM configuration using the AM console or the REST APIs.

   For information about how to access the AM Admin UI or REST APIs, see AM Services.

   See 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 to your `forgeops` repository clone:

```
$ cd /path/to/forgeops/bin
$ ./config.sh export --component am
Exporting AM configuration..
. . .
```

   The **config.sh export** command exports the modified parts of the AM configuration from the running ForgeRock Identity Platform to the `docker/7.0/am/config` directory.

5. List the changed files using the **config.sh diff -c am** command:

```
$ ./config.sh diff --component am
diff  -u --recursive config/7.0/cdk/am docker/7.0/am
Only in
docker/7.0/am/config/services/realm/root/authenticationtreesse
rvice/1.0/organizationconfig/default: my-test-tree.json
Only in docker/7.0/am: logback.xml
. . .
```

6. Save the exported configuration to your profile:

```
$ ./config.sh save --component am --profile my-profile
Saving AM configuration..
```

   For more information about the management of ForgeRock Identity Platform configurations in the `forgeops` repository, see Configuration Profiles.

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

a. Run the **git status** command.

b. Review the state of the working directory and staging area.

c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

8. Delete the existing deployment:

```
$ cd /path/to/forgeops/
$ skaffold delete
Cleaning up…
 - configmap "idm" deleted
 - configmap "idm-logging-properties" deleted
 - configmap "platform-config" deleted
. . .
```

9. Redeploy with changes using the **skaffold run** command:

```
$ skaffold run
```

Skaffold builds a new `am` Docker image and redeploys AM.

10. To validate that AM has the expected configuration, obtain the new password for `amadmin` user, start the console, and verify that your configuration changes are present.

`amster` Image

The `amster` Docker image contains AM run-time data.

Perform the following steps iteratively when developing a customized `amster` Docker image:

1. Perform version control activities on your `forgeops` repository clone:
   a. Run the **git status** command.
   b. Review the state of the working directory and staging area.
   c. (Optional) Run the **git commit** command to commit changes to files that have been modified.
2. Modify AM run-time data using the AM console or the REST APIs.

   For information about how to access the AM console or REST APIs, see AM Services.

   AM run-time data includes:
   - OAuth 2.0 clients
   - OpenID Connect 1.0 clients
   - IG, Web, Java, and SOAP STS agents

- Policies
- SAML v2.0 circles of trust and entities

3. Make sure that context is set to your namespace:

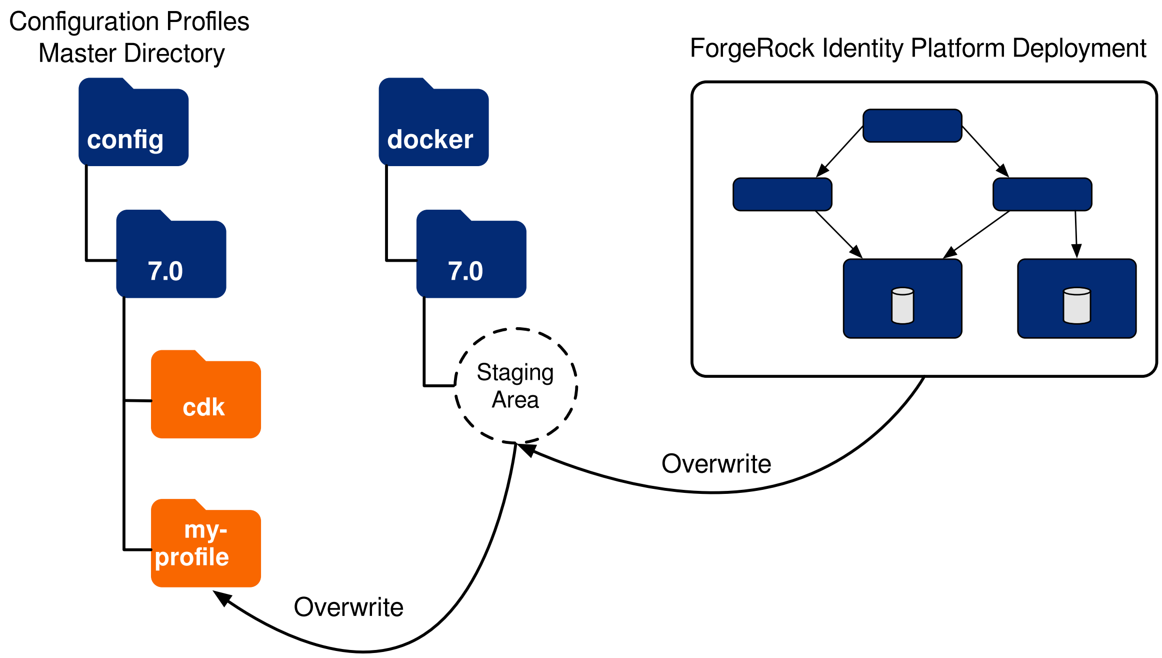
```
$ kubens my-namespace
```

4. Synchronize the changes you made to the AM configuration to your configuration profile in your `forgeops` repository clone:

```
$ cd /path/to/forgeops/bin
$ ./config.sh sync --profile my-profile --component amster
/Users/. . ./forgeops/bin/amster export
docker/7.0/amster/config
Cleaning up any previous amster jobs…
starting the amster job
kustomize build /Users/. .
./forgeops/bin/../kustomize/base/amster-export | kubectl
apply -f -
job.batch/amster created
kubectl  get pod -l app=amster --output=jsonpath=
{.items[0].metadata.name}
Waiting for pod amster-95v45
kubectl  wait --for=condition=ready pod amster-95v45 --
timeout=90s
kubectl  cp -c pause amster-95v45:/var/tmp/amster/realms
docker/7.0/amster/config/realms
tar: Removing leading `/' from member names
kubectl  delete job amster
job.batch "amster" deleted

Saving Amster configuration

* APPLYING FIXES *
Adding back amsterVersion placeholder …
Adding back FQDN placeholder …
Removing 'userpassword-encrypted' fields …

Adding back password placeholder with defaults in these files:

idm-provisioning.json
idm-resource-server.json
resource-server.json
oauth2.json
ig-agent.json
```

```
The above fixes have been made to the Amster files.
If you have exported new files that should contain commons
placeholders or passwords, please update the rules in this
script.
```

The **config.sh sync** command exports the modified AM configuration profile from the running ForgeRock Identity Platform to the staging area. Then, it saves the configuration profile as *my-profile* in the master directory for configuration profiles:



For more information about the management of ForgeRock Identity Platform configuration profiles in the `forgeops` repository, see Configuration Profiles.

5. Examine each JSON file that was written to your configuration profile.

   If any of the files contain hard-coded host names or passwords, replace them with configuration expressions. AM resolves configuration expressions when it starts up.

   See Property Value Substitution for important information about configuring values that vary at run-time, such as passwords and host names.

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

   a. Run the **git status** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

7. Make sure that context is set to your namespace:

```
$ kubens my-namespace
```

8. Reinitialize the staging area with your configuration profile:

```
$ cd /path/to/forgeops/bin
$ ./config.sh init --profile my-profile
Removing docker/7.0/am/config/
Removing docker/7.0/amster/config/
Removing docker/7.0/idm/conf/
Removing docker/7.0/idm/ui/
Removing docker/7.0/ig/config/
Copying /path/to/forgeops/config/7.0/my-profile/idm.
Copying /path/to/forgeops/config/7.0/my-profile/am.
Copying /path/to/forgeops/config/7.0/my-profile/ig.
Copying /path/to/forgeops/config/7.0/my-profile/amster.
Completed
```

9. (Optional) If you have customized DS data in the `idrepo` directory, take a backup of those changes, so you can restore your DS data after redeploying your customized `amster` image.
10. Shut down your ForgeRock Identity Platform deployment, and remove the PVCs. See CDK Shutdown and Removal for details.
11. Redeploy the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops
$ skaffold run
```

12. (Optional) If needed, restore any user identity data that you have customized in your environment.
13. (Optional) Suppose you have sample AM run-time data that you want to use for testing, but you don't want to include the sample data in the `amster` Docker image.
14. Make sure that context is set to your namespace:

```
$ kubens my-namespace
```

   - You can import the sample data to your running CDK deployment:

   ```
   $ cd /path/to/forgeops/bin
   $ ./amster import /path/to/run-time-data
   Cleaning up amster components
   job.batch "amster" deleted
   Packing and uploading configs
   configmap/amster-import created
   Deploying amster
   job.batch/amster created
   ```

```
Waiting for amster job to complete. This can take several
minutes.
. . .
Amster output ***
java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
am> :load amster-scripts/import.amster
Importing directory /opt/amster/config
Imported
/opt/amster/config/realms/root/OAuth2Clients/MyClient.json
Import completed successfully
import done
Cleaning up amster components
job.batch "amster" deleted
configmap "amster-import" deleted
```

In this example, `/path/to/run-time-data` is a directory that contains JSON files with run-time AM data. JSON files in all of this path's subdirectories are imported into AM.

- Be sure to delete the sample AM data before you export the Amster component. If you do not delete the sample data, it will be incorporated into the `amster` Docker image the next time you build the image.

15. To validate that AM has the expected changes to run-time data, start the console and verify that your changes are present.

IDM Image

With IDM up and running, you can iteratively:

- Customize IDM's configuration using the Admin UI and the REST APIs.
- Capture your configuration changes by synchronizing them from the IDM service running on Kubernetes back to the staging area and the master directory for configuration profiles in your `forgeops` repository clone.
- Run Skaffold to detect the changes, rebuild the `idm` Docker image, and restart IDM. You can then test changes you've made to the IDM configuration based on the updated Docker image.

`idm` Image

The `idm` Docker image contains the IDM configuration.

Perform the following steps iteratively when developing a customized `idm` Docker image:

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

   a. Run the **`git status`** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **`git commit`** command to commit changes to files that have been modified.

2. 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, see IDM Services.

   See Property Value Substitution for important information about configuring values that vary at run-time, such as passwords and host names.

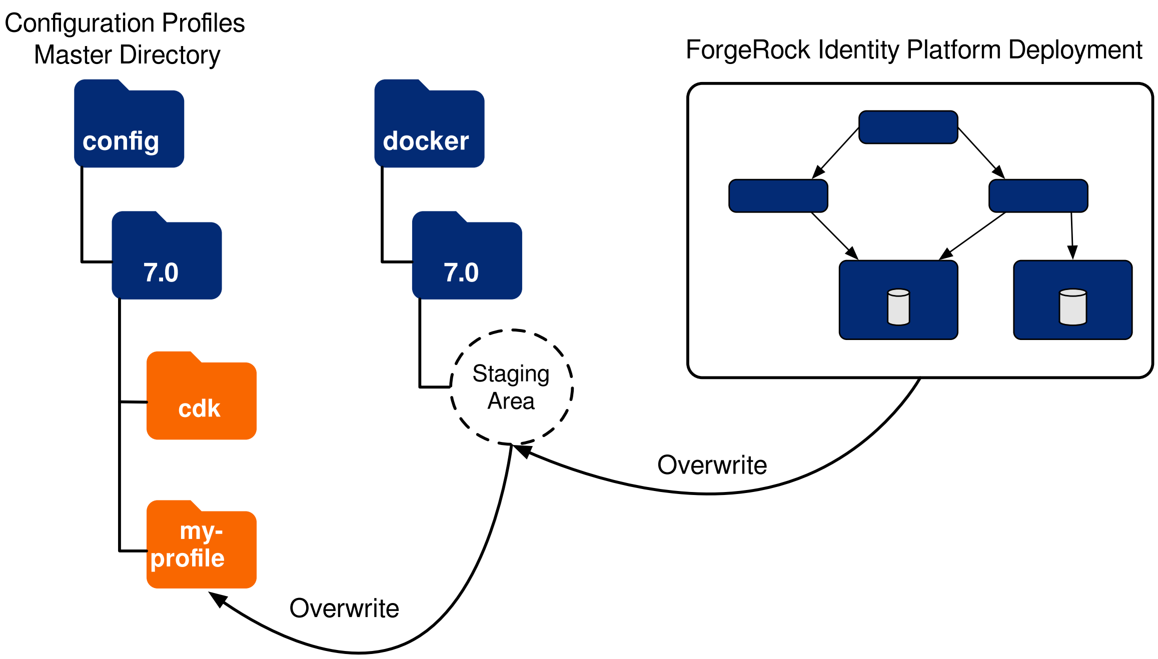
3. Make sure that context is set to your namespace:

```
$ kubens my-namespace
```

4. Synchronize the changes you made to the IDM configuration to your `forgeops` repository clone:

```
$ cd /path/to/forgeops/bin
$ ./config.sh sync --profile my-profile --component idm
tar: Removing leading '/' from member names
```

   The **`config.sh sync`** command exports the modified IDM configuration from the running ForgeRock Identity Platform to the staging area. Then, it saves the configuration profile as *`my-profile`* in the master directory for configuration profiles:



For more information about the management of ForgeRock Identity Platform configurations in the `forgeops` repository, see Configuration Profiles.

5. Execute the **`skaffold run`** command:

```
$ cd /path/to/forgeops
$ skaffold run
```

   Skaffold builds a new `idm` Docker image and redeploys IDM.

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

   a. Run the **`git status`** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **`git commit`** command to commit changes to files that have been modified.

7. To validate that IDM has the expected configuration, start the Admin UI, and verify that your configuration changes are present.

Property Value Substitution

> IMPORTANT
>
> This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

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

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

How Property Value Substitution Works

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

1. Search the `/path/to/forgeops/config/7.0/cdk` directory for the string `&{`.
2. Locate this line in your search results:

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

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

Configuration expressions take their values from environment variables as follows:

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

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

AM and IDM Differences

There are several subtle but important differences between the AM and IDM implementations of configuration expressions:

- **Canonical configuration profile.**

  AM: Contains configuration expressions for usernames, passwords, FQDNs, and the URL access protocol.

  IDM: Contains configuration expressions for usernames, passwords, FQDNs, the URL access protocol, and additional properties.
- **Administration console handling of configuration expressions.**

  AM: The console is not aware of configuration expressions. Values specified as configuration expressions in configuration profiles are displayed as run-time values in the console. You cannot specify property values as configuration expressions in the console.

IDM: The Admin UI is aware of configuration expressions. Values specified as configuration expressions in configuration profiles are displayed as configuration expressions in the Admin UI. You can specify property values as configuration expressions in the Admin UI.

- **Export configuration behavior.**

  AM: Configuration expressions are reinserted into the AM configuration, overriding hard-coded property values you might have set using the console.

  IDM: Configuration expressions are exported to the IDM configuration profile.

- **Replacing a hard-coded property value with a configuration expression**.

  AM: Edit the AM configuration manually, replacing a property value with a configuration expression. Then, open the `/path/to/forgeops/docker/7.0/am-config-upgrader/rules/placeholders.groovy` file and add a rule that preserves the new configuration expression.

  IDM: Change the property's value to a configuration expression in the Admin UI. When the configuration is exported, the exported configuration contains the configuration expression.

- **Replacing a configuration expression with a hard-coded property value**.

  AM: Edit the AM configuration manually, replacing a configuration expression with a hard-coded property value. Then, open the `/path/to/forgeops/docker/7.0/am-config-upgrader/rules/placeholders.groovy` file and remove the rule that preserved the configuration expression you replaced.

  IDM: Hard-code the property's value in the IDM Admin UI. When the configuration is exported, the exported configuration contains the hard-coded property value.

- **Configuration expressions' run-time values**.

  AM: Configuration expressions get their values from environment variables.

  IDM: Configuration expressions can get their values from a variety of sources: environment variables, Java system properies, and configuration files.

CDK Shutdown and Removal

IMPORTANT

IMPORTANT

This documentation describes the legacy CDK implementation, which will be deprecated in an upcoming release. We strongly recommend that you transition to the current CDK implementation as soon as possible.

When you're done working with your ForgeRock Identity Platform deployment, shut it down and remove it from your namespace:

1. Go to the terminal window where you started Skaffold.
2. Run the **skaffold delete** command to shut down your deployment and remove it from your namespace:

```
$ cd /path/to/forgeops
$ skaffold delete
Cleaning up…
. . .
```

3. Delete DS persistent volume claims (PVCs) from your namespace:

```
$ kubectl delete pvc --all
persistentvolumeclaim "data-ds-cts-0" deleted
persistentvolumeclaim "data-ds-idrepo-0" deleted
```

ForgeOps 7.1 Release Notes

Get an email when there's an update to ForgeOps 7.1. Go to the Notifications page in your Backstage profile and select ForgeOps 7.1 Changes in the Documentation Digests section.

Or subscribe to the ForgeOps 7.1 RSS feed.

Important information for this ForgeOps release:

Validated Kubernetes versions for deploying ForgeRock Identity Platform 7.1	Link
Validated NGINX ingress versions for deploying ForgeRock Identity Platform 7.1	Link
Limitations when deploying ForgeRock Identity Platform 7.1 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 May 12, 2021	Link

2024

March 4, 2024

Highlights

Version `release/7.1-20240223` Docker images are now available from ForgeRock

Evaluation-only Docker images are now available for version `release/7.1-20240223` of the following ForgeRock Identity Platform components:

- ForgeRock Access Management: 7.1.4
- ForgeRock Directory Services: 7.1.7
- ForgeRock Identity Management: 7.1.5
- ForgeRock Identity Gateway: 2023.11.0

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

For more information about changes to the ForgeRock 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.

Changes

The `export.sh` command updated

The **`export.sh`** command is updated to copy the configuration version service to the export folder so the config upgrader can read it.

2023

October 13, 2023

Changes

CDM deployments on Amazon EKS should now use Kubernetes version 1.27

When you create an Amazon EKS cluster for deploying version 7.1 of the platform, use Kubernetes version 1.27.

October 9, 2023

Changes

CDM deployments on Amazon EKS should now use Kubernetes version 1.25

When you create an Amazon EKS cluster for deploying version 7.1 of the platform, use Kubernetes version 1.25.

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.

March 3, 2023

Changes

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.

2022

December 6, 2022

Changes

CDM deployments on EKS should now use Kubernetes version 1.22

When you create an EKS cluster for deploying version 7.1 of the platform, use Kubernetes version 1.22.

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

When you deploy the NGINX Ingress Controller in your CDM cluster, use version 1.4.0[20] or higher.

November 11, 2022

Highlights

New evaluation-only Docker images are now available from ForgeRock

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

- ForgeRock Access Management: 7.1.4
- ForgeRock Directory Services: 7.1.7

ForgeRock Identity Management and ForgeRock Identity Gateway Docker images remain at version 7.1.5.

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

For more information about changes to the ForgeRock 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.

New convention for forgeops repository branch names

forgeops repository branch names now consist of the major and minor release numbers of ForgeRock Identity Platform components, followed by the release date.

Changes

The `bin/prometheus-deploy.sh` script is temporarily unavailable

There's an outstanding issue (CLOUD-4064) logged against the **`bin/prometheus-deploy.sh`** script. Do not attempt to run this script until this issue has been resolved.

May 19, 2022

Changes

The RCS Agent has been removed from the CDM and CDK deployments

The RCS Agent is no longer available in the CDM and CDK deployments.

March 21, 2022

Highlights

Version 7.1.2 evaluation-only Docker images are now available from ForgeRock

Evaluation-only Docker images are now available for version 7.1.2 of ForgeRock Identity Platform components.

This documentation has been updated to refer to version 7.1.2 Docker images instead of version 7.1.1 Docker images.

For more information about changes to the ForgeRock Identity Platform for version 7.1.2, refer to the Release Notes for platform components at https://backstage.forgerock.com/docs.

To upgrade from version 7.1.1 of the ForgeRock Identity Platform to version 7.1.2, you'll need to rebuild your custom Docker images. Refer to Base Docker Images for instructions.

March 4, 2022

Changes

The stable version of Kubernetes is now supported on Minikube clusters

You can now use the stable Kubernetes version when creating Minikube clusters that run the CDK.

Previously, the NGINX ingress configuration required the use of Kubernetes version 1.21 on Minikube. The ingress configuration has been updated, allowing the use of newer Kubernetes versions.

January 10, 2022

Highlights

Version 7.1.1 evaluation-only Docker images are now available from ForgeRock

Evaluation-only Docker images are now available for version 7.1.1 of ForgeRock Identity Platform components.

This documentation has been updated to refer to version 7.1.1 Docker images instead of version 7.1.0 Docker images.

For more information about changes to the ForgeRock Identity Platform for version 7.1.1, refer to the Release Notes for platform components at https://backstage.forgerock.com/docs.

To upgrade from version 7.1.0 of the ForgeRock Identity Platform to version 7.1.1, you'll need to rebuild your custom Docker images. Refer to Base Docker Images for instructions.

2021

November 11, 2021

Changes

Limitation on IDM workflow support in the CDK and CDM

The Release Notes now document the limitation that the CDK and CDM are not preconfigured to support IDM's workflow engine.

Note that this limitation has existed since version 7.0 of the platform, when the CDK and CDM starting using DS as the IDM repository.

October 6, 2021

Changes

Use the new `cluster/minikube/cluster-up` utility to create a Minikube cluster

The new **`cluster/minikube/cluster-up`** utility lets you create a Minikube cluster that's configured for running the CDK.

The Minikube Cluster page now includes an example of how to run this utility.

September 28, 2021

Changes

Use Kubernetes version 1.21 with Minikube deployments

When you create a Minikube cluster for deploying version 7.1 of the platform, use Kubernetes version 1.21.

Newer versions of Kubernetes are currently incompatible with version 7.1 of the platform.

Enhanced `debug-logs` utility

The **`bin/debug-logs.sh`** script, which gathers information needed to help troubleshoot problems, has been replaced with a new utility, named **`bin/debug-logs`**.

In addition to the pod descriptions and container logs provided by the **`bin/debug-logs.sh`** script, the new utility provides information about PVCs, various Kubernetes objects, logs for the Secret Agent and DS operators, and other diagnostic information.

August 10, 2021

Changes

New recommendation: deploy AM without subrealms

It's now recommended that, when you deploy AM on Kubernetes, use a single root realm without any subrealms. For more information, see the section on AM limitations in the Release Notes.

Deprecated

Dynamic AM configuration in the `amster` Docker image

Adding dynamic AM configuration to the `amster` Docker image is deprecated.

Instead, import and export dynamic configuration in and out of the CDK and CDM using utilities such as:

- The **`bin/amster`** command in the `forgeops` repository
- ForgeRock Identity Platform REST APIs
- IDM reconciliation

Documentation Updates

IG how-to

A new how-to that provides instructions for deploying IG, and for creating a custom IG image, is now available.

NGINX ingress version page

A new page provides information about which version of the NGINX version to use with ForgeRock Identity Platform 7.1.

Supported Skaffold profiles listed explicitly

The list of supported and unsupported Skaffold profiles is now explicitly listed in the Repository Reference.

The unsupported Skaffold profiles are for ForgeRock internal use only. Do not use any of the unsupported profiles or their associated Kustomize bases and overlays.

July 12, 2021

Highlights

New CDK technology released from technology preview status

The new way of deploying the CDK has moved from technology preview status to evolving status.

The documentation for the new way of deploying the CDK, previously in the Technology Previews menu, can now be found here.

DS operator supported for use with the CDK

The DS operator is now supported for use with demonstration and developer deployments that use the CDK.

The DS operator remains in technology preview status for production deployments. Do not use the operator in production deployments of the ForgeRock Identity Platform.

Changes

New `amster` command

Use the new **`amster import`** command instead of the **`config.sh import`** command to import sample AM run-time data to the CDK.

Statement on `forgeops` repository feature evolution

The new feature evolution page has been added to these release notes to clarify the meaning of feature statuses, such as technology preview, evolving, legacy, deprecated, and removed.

Deprecated

Previous CDK technology

The former way of deploying the CDK is now deprecated.

The documentation for the the former way of deploying the CDK, previously in the Cloud Developer's Kit (CDK) menu, can be found here.

Removed

Cloud Deployment Quickstart (CDQ)]

The CDQ has been removed from the `forgeops` repository.

May 12, 2021

This major new release of the `forgeops` repository supports ForgeRock Identity Platform 7.1. In addition to enabling new features in the platform, this release adds usability and security enhancements.

Highlights

New CDK technology preview

A first look at a new way to deploy the CDK, and to use the CDK to develop custom Docker images for the ForgeRock Identity Platform with it:

- The new way of deploying the CDK is generally simpler and faster.
- The new CDK deployment uses a single DS pod— `ds-idrepo-0`. Functionality provided by the DS CTS pod in previous CDK versions is now merged into the ID repo pod. Deployment with a single DS pod is simpler, faster, and requires less resources than earlier versions. For example, the memory requirement for Minikube deployments decreases from 12GB to 10GB.
- The new **`cdk install`** command lets developers deploy the CDK one component at a time. It's still possible to deploy the entire CDK with a single **`cdk install`** command, but you can also deploy individual CDK components one at a time, review the results, and then deploy the next component. Deploying the platform one component at a time can make troubleshooting simpler if you run into a problem.

  For a list of CDK components you can install one at a time, run the **`cdk install -h`** command.

- The new **`cdk install`** command is idempotent. The command checks the installation status of a component before it attempts to install it. For example, if you run the **`cdk install`** command, and the ForgeRock UI pods are already installed and available, the installer won't attempt to install the UI a second time unless you've specified different Docker images for running it, or modified the Kustomize files that orchestrate it.
- The new **`cdk build`** command lets you build custom Docker images for the ForgeRock Identity Platform.
- The new image defaulter gives developers fine-grain control over which Docker images are deployed with the CDK. The deployed Docker image no longer needs to be the last image that you built.
- The CDK incorporates the DS operator, simplifying directory deployment. Note that the DS operator remains in technology preview status for CDM deployments.
- The **`cdk install`** command incorporates Secret Agent and DS operator installation. Separate commands are no longer required to install these CDK components.

You'll find the documentation for the new technology CDK here.

DS operator technology preview

The DS operator uses the Kubernetes operator design pattern to let you easily deploy and manage DS instances running in a Kubernetes cluster. After you install the `ds-operator` custom resource definition (CRD) in a cluster, you can use it to create DS instances, scale them, and manage backup and restore.

The DS operator is offered as a technology preview. Do not use it production deployments of the ForgeRock Identity Platform.

For more information, refer to DS Operator.

New RCS Agent pod in the CDM

The CDM now includes an RCS Agent pod. The RCS Agent is a reliable websocket proxy between remote connector servers and the IDM instances in the CDM.

For more information, refer to CDM Architecture.

Cloud Deployment Quickstart (CDQ)

The CDQ is a very quick, single-command deployment of the ForgeRock Identity Platform on a Kubernetes cluster. The CDQ has very limited capabilities.

New Secret Agent operator

The new Secret Agent operator provides secret generation and management services for ForgeRock Identity Platform deployments on Kubernetes. The new Secret Agent operator replaces the deprecated `forgeops-secret` job, which previously was invoked when you deployed the platform using Skaffold.

By default, the operator examines your namespace to determine whether it contains all the secrets required for ForgeRock Identity Platform deployment. If any of the required secrets are not present, the operator generates them. Configuration options that let you change this default behavior are available.

In addition to secret generation, the new operator also integrates with Google Cloud Secret Manager, AWS Secrets Manager, and Azure Key Vault, providing cloud backup and retrieval for secrets.

For more information about secret generation options and secret management, refer to the Secret Agent project README.

New cluster provisioning scripts

This release of the `forgeops` repository introduces the `cluster-up.sh` and `cluster-down.sh` scripts, which you use to create and delete CDM clusters. These scripts replace the Pulumi scripts previously in the repository.

The new scripts are designed to be lightweight, and easy to use and modify. For GKE and AKS, the scripts call the cloud providers' SDKs. For EKS, the scripts call the eksctl CLI.

Instructions for creating clusters using the new scripts are available in the CDM Cookbooks for GKE, EKS, and AKS.

The deprecated Pulumi scripts are still available in the `forgeops` repository, in the `/path/to/forgeops/cluster/pulumi-deprecated` directory. They are no longer being maintained or upgraded. You can still use them with Pulumi 2.7.1 before you move to the new scripts.

Small, medium, and large CDM cluster sizing

This release restores the ability to create sized CDM clusters. Before 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

Changes

Release branch

Version 7.1.0 of the `forgeops` repository is available in the `release/7.1.0` branch.

Previously, release tags were used for `forgeops` repository releases.

Several Docker images from ForgeRock are supported in production deployments

The Docker images that implement UI elements in the ForgeRock Identity Platform are now supported for use in production deployments. For more information, see Base Docker Images.

Previously, users were required to build all the Docker images for the platform for use in their production deployments.

Third-Party Kubernetes support changes

The section, Third-Party Kubernetes Services in the Statement of Support has been revised.

Secure LDAP

Inbound communication to DS instances now occurs over secure LDAP (LDAPS). Previously, communication was over LDAP connections.

IDM is now a Kubernetes deployment

Previously, IDM was deployed as a stateful set.

Python 3 is now on the list of required third-party software

The `bin` directory in the `forgeops` repository now contains scripts written in Python 3.

Python 3 has been added to the list of third-party software that you need to install before using the `forgeops` repository. Note that Homebrew users can install Python 3 using the command, `brew install python`.

Python scripts

Some of the functionality available in bash scripts is replaced by the identical functionality in Python scripts. No functionality has been removed with these script changes:

- **`clean.sh`** - Use the **`cdk delete`** Python script instead.
- **`ds-operator.sh`** - Use the **`ds-operator`** Python script instead.
- **`print-secrets.sh`** - Use the **`print-secrets`** Python script instead.
- **`secret-agent.sh`** - Use the **`secret-agent`** Python script instead.

Secrets are not created automatically when you install the platform on the CDM

A new step to configure the Secret Agent and create secrets is required when deploying the CDM.

The new step—running the **`kubectl apply`** command—has been added to the Secret Agent Operator sections in the CDM Cookbooks for GKE, EKS, and AKS.

Previously, this was done automatically by the **`skaffold run`** command.

Note that Skaffold still automates secret creation when you deploy the CDK.

Volume snapshots technology preview

Support for volume snapshots has been added to the DS operator technology preview. For more information, see Snapshots.

Configuration expressions in the AM configuration are preserved when the configuration is exported

Configuration expressions used in an AM configuration profile are now preserved in that profile after you export a configuration from the CDK to a `forgeops` repository clone.

For more information, see About Property Value Substitution in the CDK documentation.

CDK and CDM deployment verified on newer Kubernetes versions

CDK and CDM deployments are now verified on newer Kubernetes versions. For more information, see Recommended Kubernetes Versions.

The Secret Agent operator lets you change individual administration passwords

The Secret Agent operator now supports changing individual administration passwords. If periodic password changes are a requirement for your organization, you can change individual administration passwords as needed.

CDM deployments no longer create a third `ds-idrepo` replica

The `ds-idrepo-2` replica is no longer deployed as part of the CDM.

IDM did not use this replica, and removing the replica improved replication performance for the CDM, and lowered the cost of the deployment.

CDM backups are now taken from the `-0` DS instances by default

CDM backups are now taken from the `ds-idrepo-0` and `ds-cts-0` DS instances by default.

In previous versions, backups were taken from the `ds-idrepo-2` and `ds-cts-2` DS instances by default.

For more information, see CDM Backup and Restore.

Regions for CDM cluster creation no longer default

With this change, you must explicitly configure a region when you run one of the CDM cluster creation scripts. For details, see the environment setup sections for Google Cloud, AWS, and Azure.

Previously, CDM clusters were created in specific regions by default.

Long form command-line options for the `ingress-controller-deploy.sh` command

Long form command-line options are now available for the `ingress-controller-deploy.sh` command. To see the available options, run `/path/to/forgeops/bin/ingress-controller-deploy.sh --help`.

How to eliminate the need to accept a self-signed certificate on Minikube deployments

The CDK documentation now includes an optional step for adding a secret to Minikube deployments. The secret contains a TLS certificate issued by an external certificate authority (CA), or by a local CA that you create using the mkcert utility. Users who access ForgeRock web-based applications on deployments that have this type of secret do not need to accept a self-signed certificate.

See TLS Certificate.

All main AM run-time data types supported when exporting configuration data

The `export` and `sync` options of the `config.sh` command let you export AM run-time data from a running CDK instance to a configuration profile stored in a local clone of the `forgeops` repository. With this release, the `export` and `sync` options can now export all of these types of run-time data:

- OAuth 2.0 clients
- OpenID Connect 1.0 clients
- IG, Web, Java, and SOAP STS agents
- Policies
- SAML v2.0 circles of trust and entities

In previous releases, only OAuth 2.0 clients and IG agents were exported.

Performance benchmark changes

Two benchmarks are available for ForgeRock Identity Platform version 7:

- An authentication rate benchmark, which measures authentication performed with AM REST API calls to an AM server configured to use CTS-based (stateful) sessions.

- An OAuth 2.0 authorization code flow benchmark, which measures the throughput and response time of an AM server performing authentication, authorization, and session token management. AM is configured to use client-based (stateful) sessions for this benchmark.

Contact your ForgeRock sales representative to obtain our results for benchmarks for these ForgeRock Identity Platform version 7.

Small and medium clusters now use a single node pool

For simpler deployments, small and medium CDM clusters now use a single node pool for all pods instead of using a second node pool for DS pods.

Large CDM clusters continue to use two node pools.

Task maps and checklists in the documentation

The CDK and CDM documentation has been improved! New checklists help you navigate through set up and deployment activities:

- CDK deployment checklist
- Minikube setup checklist
- Shared cloud cluster setup checklist
- CDM deployment checklist
- GKE environment setup checklist
- EKS environment setup checklist
- AKS environment setup checklist

Task maps are provided with each set up and deployment activity. They help you determine where you are in the deployment process, and indicate the next step you'll perform.

Minikube `cni=true` option

ForgeRock now recommends that you start Minikube with the `cni=true` option. Starting Minikube with this option circumvents Minikube issue 1568, which required users to run the Minikube VM in promiscuous mode.

In Minikube Cluster:

- The step to create the Minikube VM has been modified to use the `cni=true` option.
- The instruction to circumvent Minikube issue 1568 by placing the Minikube VM in promiscuous mode has been removed.

Deprecated

DevOps artifacts for deploying ForgeRock Identity Platform 7.0

The DevOps artifacts for deploying ForgeRock Identity Platform 7.0 are deprecated. You should migrate to version 7.1 as soon as you're able to.

The DevOps artifacts for deploying version 7.0 of the platform have been removed from the `master` branch of the `forgeops` repository. You can still get them from the `2020.08.07-ZucchiniRicotta.1` release tag of the repository.

`forgeops-secret` job

The `forgeops-secret` job is deprecated. Use the new Secret Agent operator to obtain similar functionality, and for storing and retrieving secrets in Google Cloud Secret Manager, AWS Secrets Manager, and Azure Key Vault.

Cluster provisioning using Pulumi

The scripts that provision CDM clusters using Pulumi are deprecated. Use the new cluster provisioning and removal scripts to obtain similar functionality.

The Pulumi scripts are still available in the `/path/to/forgeops/cluster/pulumi-deprecated` directory to help you as you transition to the new cluster provisioning scripts. You should move to the new scripts as quickly as possible, because the Pulumi scripts will be removed from the `forgeops` repository in a future release.

Support From ForgeRock

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

ForgeRock DevOps Support

ForgeRock has developed artifacts in the forgeops Git repository for the purpose of deploying the ForgeRock Identity Platform in the cloud. The companion DevOps documentation provides examples, including the ForgeRock Cloud Developer's Kit (CDK) and the ForgeRock 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 located at link:https://www.forgerock.com/terms.

Commercial Support

ForgeRock provides commercial support for the following DevOps resources:

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

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

- ForgeRock Access Management Release Notes
- ForgeRock Identity Management Release Notes
- ForgeRock Directory Services Release Notes
- ForgeRock Identity Gateway Release Notes

Support Limitations

ForgeRock provides no commercial support for the following:

- Artifacts other than Dockerfiles, Kustomize bases, Kustomize overlays, and Skaffold YAML configuration files 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.
- 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 Feedback

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, Kustomize overlays, or Skaffold YAML configuration files 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 Logs and Other Diagnostics.

- Description of the environment, including the following information:
  - Environment type: Minikube, GKE, EKS, or AKS.
  - Software versions of supporting components:
    - Oracle VirtualBox (Minikube environments only).
    - Docker client (all environments).
    - Minikube (all environments).
    - **`kubectl`** command (all environments).
    - Kustomize (all environments).
    - Skaffold (all environments).
    - Google Cloud SDK (GKE environments only).
    - Amazon AWS Command Line Interface (EKS environments only).
    - Azure Command Line Interface (AKS environments only).
  - `forgeops` repository branch.
  - Any patches or other software that might be affecting the problem.

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.

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 (AKS) 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: *The `cloud-controller-manager`* section in the Kubernetes Concepts documentation

Cloud Developer's Kit (CDK)

The developer artifacts in the `forgeops` Git repository, together with the ForgeRock 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 ForgeRock Identity Platform architecture, designed to be easy to deploy and easy to replicate. The ForgeRock Cloud Deployment Team has developed Kustomize bases and overlays, Skaffold configuration files, 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 cluster master and multiple worker machines called nodes. The Kubernetes objects that represent your containerized applications all run on top of a cluster.

Source: *Cluster Architecture* in the Google Kubernetes Engine (GKE) documentation

cluster master

A cluster master schedules, runs, scales, and upgrades the workloads on all nodes of the cluster. The cluster master also manages network and storage resources for workloads.

Source: *Cluster master in the Google Kubernetes Engine (GKE) docuementation*

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

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 Platform 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 section* 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: *Docker daemon* section in the Docker Overview documentation

Docker Engine

The Docker Engine is a client-server application with these components:

- A server, which is a type of long-running program called a daemon process (the `dockerd` command)
- A REST API, which specifies interfaces that programs can use to talk to the daemon and tell it what to do

- A command-line interface (CLI) client (the `docker` command)

  Source: *Docker Engine* section in the Docker Overview 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* section in the Docker Reference 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: *Namespaces* 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: *Repositories* in the Docker Overview 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 overview* 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: *kubelets* 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: *Kubernetes components* in the Kubernetes Concepts documentation

Kubernetes
: Kubernetes is an open source platform designed to automate deploying, scaling, and operating application containers.

  Source: [*What is Kubernetes?* in the Kubernetes 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: [*Cloud computing terms* 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: *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

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

End of Consolidated File

1. The Linux version of Homebrew does not support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew to install software in several cases. You'll need to refer to the software's documentation for information about how to install the software on a Linux system.
2. Install Docker Desktop on macOS. On Linux computers, install Docker CE instead. For more information, see the Docker documentation.

3. If your cluster's context is not `minikube`, replace `minikube` with the actual context name in the **`skaffold config set`** command.
4. You can automate logging into ECR every 12 hours by using the `cron` utility.
5. Occasionally, Skaffold has issues with cached images. To work around a caching problem, remove Skaffold's cache by running the **`rm -rf $HOME/.skaffold/cache`** command. If removing the cache still does not resolve the problem, use the **`docker pull`** command to manually pull the images.
6. Occasionally, Skaffold has issues with cached images. To work around a caching problem, remove Skaffold's cache by running the **`rm -rf $HOME/.skaffold/cache`** command. If removing the cache still does not resolve the problem, use the **`docker pull`** command to manually pull the images.
7. On GKE, the node pool shown in the diagram as `Primary` is named `default-pool`.
8. The cluster creation script adds a set of required labels to clusters created by ForgeRock employees. The first time you run the script, it prompts you to specify whether you're a ForgeRock employee or not, so that it can add these labels if appropriate. You should not receive this prompt during subsequent executions of the script.
9. The cluster creation script adds a set of required labels to clusters created by ForgeRock employees. The first time you run the script, it prompts you to specify whether you're a ForgeRock employee or not, so that it can add these labels if appropriate. You should not receive this prompt during subsequent executions of the script.
10. You can automate logging into ECR every 12 hours by using the **`cron`** utility.
11. The cluster creation script adds a set of required labels to clusters created by ForgeRock employees. The first time you run the script, it prompts you to specify whether you're a ForgeRock employee or not, so that it can add these labels if appropriate. You should not receive this prompt during subsequent executions of the script.
12. You can automate logging in to ACR by using the **`cron`** utility.
13. The CDM and the CDK both use the CDK canonical configuration profile.
14. When you build the `am` Docker image, the AM configuration files are copied from the `/path/to/forgeops/docker/7.0/am/config` directory to the `/home/forgerock/openam/config` directory.
15. When you build the `idm` Docker image, the IDM configuration files are copied from the `/path/to/forgeops/docker/7.0/idm/conf` directory to the `/opt/openidm/conf` directory.
16. When you build the `am` Docker image, the AM configuration files are copied from the `/path/to/forgeops/docker/7.0/am/config` directory to the `/home/forgerock/openam/config` directory.
17. When you build the `idm` Docker image, the IDM configuration files are copied from the `/path/to/forgeops/docker/7.0/idm/conf` directory to the `/opt/openidm/conf` directory.
18. If your cluster's context is not `minikube`, replace `minikube` with the actual context name in the **`skaffold config set`** command.
19. You can automate logging into ECR every 12 hours by using the `cron` utility.
20. NGINX Ingress Controller Helm chart version 4.3.0 installs NGINX Ingress Controller version 1.4.0.