DevOps

August 8, 2025

DEVOPS

Table of Contents

Get Started

Introduction

This section outlines ways to easily deploy Ping products with pre-defined configurations. After you have successfully deployed using this example, you can try other provided examples or move on to customizing the products to fit your needs and environment.

You will need to register for the Ping DevOps program in order to obtain trial licenses for evaluating or testing with our products.

After registering at the link above, you will be provided a username and key. These credentials provide a temporary license for your evaluation. See using your DevOps User and Key for instructions on use.

Finally, to manage PingOne resources using credentials other than your own, a PingOne Worker App is required. See this configuration page for more details on configuration.

Prerequisites

In order to use our resources, you will need the following components, software, or other information.

Product license

You must have a product license to run our images. You may either use an evaluation license or existing license.

Evaluation License

Generate an evaluation license obtained with a valid DevOps user key.

When you register for Ping Identity's DevOps program, you are issued credentials that automate the process of retrieving an evaluation product license.

DevOps User and Key

For more information about using your DevOps program user and key in various ways (including Kubernetes and with stand-alone containers) see this how-to guide: Using Your Devops User and Key

Evaluation License

Evaluation licenses are short-lived (30 days) and **must not** be used in production deployments.

Evaluation licenses can only be used with images published in the last 90 days. If you want to continue to use an image that was published more than 90 days ago, you must obtain a product license.

Existing License

If you possess a product license for the product, you can use it with supported versions of the image (including those over 90 days old mentioned above) by following these instructions to mount the product license.

> **Mount paths**
>
> The mount points and name of the license file vary by product. The link above provides the proper location and name for these files.

Local runtime environment

The initial example uses Kubernetes under Docker Desktop because it does not require a lot of configuration.

In order to try Ping products in a manner most similar to typical production installations, you should consider using a Kubernetes environment. Kind (**K**ubernetes **in D**ocker) provides a platform to get started with local Kubernetes development. Instructions for setting up a Kind cluster are here.

Other local Kubernetes environments include Rancher Desktop, Docker Desktop with Kubernetes enabled, and minikube.

> **Rancher Desktop**
>
> Rancher Desktop is compatible with Linux, MacOS, and Windows (using WSL). It also supports the docker container runtime, which provides support for running docker commands without installing individual docker components or Docker Desktop.

For running Docker Compose deployments of single products, any Docker Desktop installation or Linux system with Docker and `docker compose` installed can be used.

Applications / Utilities

- Helm cli
- kubectl
- Homebrew for package installation and management. Homebrew can be used to install k9s, kubectl, helm, and other programs.

```
/bin/bash -c "$(curl -fsSL +https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- pingctl

```
brew install pingidentity/tap/pingctl
```

Recommended Additional Utilities

- k9s

```
brew install derailed/k9s/k9s
```

- kubectx

```
brew install kubectx
```

- docker-compose

```
brew install docker-compose
```

Info

Installing docker-compose is only necessary to deploy Docker containers when using Docker with Rancher Desktop. It is included with the Docker Desktop installation.

See Rancher preferences to switch from containerd to dockerd (moby).

Configure the Environment

1. Open a terminal and create a local DevOps directory named `${HOME}/projects/devops`.

   Info

   ${HOME}/projects/devops is the parent directory for all examples referenced in our documentation.

2. Configure the environment as follows.

   ```
   pingctl config
   ```

   1. Respond to all configuration questions, accepting the defaults if uncertain. Settings for custom variables aren't needed initially but may be necessary for additional capabilities.
   2. All responses are captured in your local `~/.pingidentity/config` file. Allow the configuration script to source this file in your shell profile (for example, `~/.bash_profile` in a bash shell).

3. [Optional] Export configured pingctl variables as environment variables

   1. Modify your shell profile (for example, `~/.bash_profile` in a bash shell) so that the generated `source ~/.pingidentity/config` command is surrounded by `set -a` and `set +a` statements.

      ```
      set -a
      # Ping Identity - Added with 'pingctl config' on Fri Apr 22 13:57:04 MDT 2022
      test -f "${HOME}/.pingidentity/config" && source "${HOME}/.pingidentity/config"
      set +a
      ```

2. Verify configured variables are exported in your environment.
    1. Restart your shell or source your shell profile.
    2. Run `env | grep 'PING'`
4. To display your environment settings, run:

```
pingctl info
```

5. For more information on the options available for `pingctl` see Configuration & Environment Variables.

Environment and Configuration Variables

Configuration and environment variables allow users to cache secure and repetitive settings into a `pingctl` config file. The default location of the file is `~/.pingidentity/config`.

You can specify a given configuration item in one of three ways: the `pingctl` config file, the user's current environment variables, or through command line arguments. The order of priority (highest to lowest) is:

- Command-Line argument overrides (when available)
- `pingctl` config file
- Environment variable overrides

PingOne Variables

The standard **PingOne variables** used by `pingctl` are as follows:

Variable	Description
PINGONE_API_URL	PingOne API URL (i.e. api.pingone.com/v1)
PINGONE_AUTH_URL	PingOne Auth URL (i.e. auth.pingone.com, auth.pingone.eu, auth.pingone.asia)
PINGONE_ENVIRONMENT_ID	PingOne Environment ID GUID
PINGONE_WORKER_APP_CLIENT_ID	PingOne Worker App ID GUID with access to PingOne Environment
PINGONE_WORKER_APP_GRANT_TYPE	PingOne Worker App Grant Type to use. Should be one of authorization_code, implicit or client_credential
PINGONE_WORKER_APP_REDIRECT_URI	PingOne Worker App available redirect_uri. Defaults to http://localhost:8000
PINGONE_WORKER_APP_CLIENT_SECRET	PingOne Worker App Secret providing authentication to PingOne Worker App ID GUID

Legacy ping-devops Variables

Prior to the `pingctl` CLI tool, `ping-devops` was used to help with the management of docker, docker-console, and kustomize deployments (this utility has been deprecated). In configuring the legacy tool, several variables were used when deploying docker images into different environments.

The legacy variables still supported and managed by `pingctl` are as follows:

Variable	Description
PING_IDENTITY_ACCEPT_EULA	Specify `YES` or `NO` to accept Ping Identity EULA
PING_IDENTITY_DEVOPS_USER	Ping DevOps User
PING_IDENTITY_DEVOPS_KEY	Ping DevOps Key
PING_IDENTITY_DEVOPS_HOME	Home directory/path of your DevOps projects
PING_IDENTITY_DEVOPS_REGISTRY	Default Docker registry from which to pull images
PING_IDENTITY_DEVOPS_TAG	Default DevOps tag to use for deployments (i.e. 2205)

pingctl Variables

Additional variables honored by `pingctl` are:

Variable	Description
PINGCTL_CONFIG	Location of the `pingctl` configuration file. Set as an environment variable only. Default: `~/.pingidentity/config`
PINGCTL_DEFAULT_OUTPUT	Specifies default format of data returned. Command-Line arg `-o`. Default: `table`
PINGCTL_DEFAULT_POPULATION	Specifies default population to use for PingOne commands. Command-Line arg `-p`. Default: `Default`
PINGCTL_OUTPUT_COLUMNS_{resource_type}	Specify custom format of table csv data to be returned. Command-Line arg `-c`. See more detail below
PINGCTL_OUTPUT_SORT_{resource_type}	Specify column to use for sorting data. Command-Line arg `-s`. See more detail below

PINGCTL_OUTPUT_COLUMNS

There are two classes of variables provided by `PINGCTL_OUTPUT`:

- `PINGCTL_OUTPUT_COLUMNS_{resource}` - Specifies the columns to display whenever a `pingctl pingone get {resource}` command is used.

  Same as the `-c` option on the command-line (see pingctl pingone get command).

  Format of value should be constructed with `HeadingName:jsonName,HeadingName:jsonName`. The best way to understand is by looking at the example of the default `USERS` resource:

  Example PINGCTL_OUTPUT_COLUMNS_USERS setting and output

  `PINGCTL_OUTPUT_COLUMNS_USERS=LastName:name.family,FirstName:name.given`

  Setting the above will generate output similar to:

  ```
  $ pingctl pingone get users
  LastName     FirstName
  --------     ---------
  Badham       Antonik
  Agnès        Enterle
  --
  2 'USERS' returned
  ```

  Alternatively, you can use the `-c` option as a command-line argument:

  ```
  $ pingctl pingone get users -c "LastName:name.family,FirstName:name.given,Username:username"
  LastName     FirstName    Username
  --------     ---------    --------
  Badham       Antonik      antonik_adham
  Agnès        Enterle      enterle_agnès
  --
  2 'USERS' returned
  ```

PINGCTL_OUTPUT_SORT

- `PINGCTL_OUTPUT_SORT_{resource}` - specifies the column on which to sort.

  Same as the `-s` option on the command-line (see pingctl pingone get command).

  Format of the value should be constructed with `jsonName`. The name must be one of the entries in `PINGCTL_OUTPUT_COLUMNS_{resource}`.

Example PINGCTL_OUTPUT_SORT_USERS setting and output

```
PINGCTL_OUTPUT_SORT_USERS=name.family
```

Setting the above will generate output similar to the following (note that the LastName (name.family) is sorted):

```
$ pingctl pingone get users
LastName     FirstName
--------     ---------
Agnès        Enterle
Badham       Antonik
--
2 'USERS' returned
```

Alternatively, you can use the `-s` option as a command-line argument:

```
$ pingctl pingone get users -s "name.given"
LastName     FirstName    Username
--------     ---------    --------
Agnès        Enterle      enterle_agnès
Badham       Antonik      antonik_badham
--
2 'USERS' returned
```

Deploy an Example Stack

Video Demonstration

A video demonstration of this example is available here.

Versions Used

This example was written using Docker Desktop with Kubernetes enabled on the Mac platform using the Apple M4 chip. The Docker Desktop version used for this guide was `4.39.0 (184744)`, which includes Docker Engine `v28.0.1` and Kubernetes `v1.32.2`. The ingress-nginx controller version was `1.12.1`, deployed from Helm chart version `4.12.1`.

Kubernetes Services Kubernetes versus Server-Deployed Applications

If you are new to Kubernetes-based deployments, there is a distinct difference when running under Kubernetes compared to running applications on servers. In a server model, many applications typically run on the same server, and you can access any of them using the same host. For example, many on-premise deployments of PingFederate also include the PingDataConsole, hosted on the same server.
Under Kubernetes, however, each application that requires external access is associated with a `service`. A service is a fixed endpoint in the cluster that routes traffic to a given application. So, in this example, there are distinct service endpoints for PingFederate, PingDataConsole, and the other products.
In this demo, these service endpoints are load balanced using the Nginx ingress controller. By adding entries to the `/etc/hosts` file, you can access them using typical URL entries.

The Ping Identity Helm Getting Started page has instructions on getting your environment configured for using the Ping Helm charts.

For more examples, see Helm Chart Example Configurations.

For more information on Helm with Ping products, see Ping Identity DevOps Helm Charts.

What You Will Do

After using Git to clone the `pingidentity-devops-getting-started` repository, you will use Helm to deploy a sample stack to a Kubernetes cluster.

Prerequisites

- Register for the Ping DevOps program and install/configure `pingctl` with your User and Key
- Install Git
- Follow the instructions on the helm Getting Started page up through updating to the latest charts to ensure you have the latest version of our charts
- Access to a Kubernetes cluster. You can enable Kubernetes in Docker Desktop for a simple cluster, which was the cluster used for this guide (on the Mac platform).

Clone the `getting-started` repository

1. Clone the `pingidentity-devops-getting-started` repository to your local `${PING_IDENTITY_DEVOPS_HOME}` directory.

   The `${PING_IDENTITY_DEVOPS_HOME}` environment variable was set by running `pingctl config`.

   ```
   cd "${PING_IDENTITY_DEVOPS_HOME}"
   git clone \
     https://github.com/pingidentity/pingidentity-devops-getting-started.git
   ```

Deploy the example stack

1. Deploy the example stack of our product containers.

   Initial Deployment

   For this guide, avoid making changes to the `everything.yaml` file to ensure a successful first-time deployment.

   1. Create a namespace for running the stack in your Kubernetes cluster.

```
# Create the namespace
kubectl create ns pinghelm

# Set the kubectl context to the namespace
kubectl config set-context --current --namespace=pinghelm

# Confirm
kubectl config view --minify | grep namespace:
```

2. Deploy the ingress controller to Docker Desktop:

```
helm upgrade --install ingress-nginx ingress-nginx \
--repo https://kubernetes.github.io/ingress-nginx \
--namespace ingress-nginx --create-namespace
```

3. To wait for the Nginx ingress to reach a healthy state, run the following command. You can also observe the pod status using k9s or by running `kubectl get pods --namespace ingress-nginx`. You should see one controller pod running when the ingress controller is ready. This command should exit after no more than 90 seconds or so, depending on the speed of your computer:

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s
```

4. Create a secret in the namespace you will be using to run the example (pinghelm) using the `pingctl` utility. This secret will obtain an evaluation license based on your Ping DevOps username and key:

```
pingctl k8s generate devops-secret | kubectl apply -f -
```

5. This example will use the Helm release name `demo` and DNS domain suffix `*pingdemo.example` for accessing applications. Add all expected hosts to `/etc/hosts`:

```
echo '127.0.0.1 demo-pingaccess-admin.pingdemo.example demo-pingaccess-engine.pingdemo.example demo-
pingauthorize.pingdemo.example demo-pingauthorizepap.pingdemo.example demo-
pingdataconsole.pingdemo.example demo-pingdelegator.pingdemo.example demo-
pingdirectory.pingdemo.example demo-pingfederate-admin.pingdemo.example demo-pingfederate-
engine.pingdemo.example demo-pingcentral.pingdemo.example' | sudo tee -a /etc/hosts > /dev/null
```

6. To install the chart, go to your local `"${PING_IDENTITY_DEVOPS_HOME}"/pingidentity-devops-getting-started/30-helm` directory and run the command shown here. In this example, the release (deployment into Kubernetes by Helm) is called `demo`, forming the prefix for all objects created. The `ingress-demo.yaml` file configures the ingresses for the products to use the ***ping-local*** domain:

```
helm upgrade --install demo pingidentity/ping-devops -f everything.yaml -f ingress-demo.yaml
```

The latest product Docker images are automatically downloaded if they have not previously been pulled from Docker Hub.

Sample output:

```
Release "demo" does not exist. Installing it now.
NAME: demo
LAST DEPLOYED: Tue Apr  1 09:14:20 2025
NAMESPACE: pinghelm
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
#-------------------------------------------------------------------------------------
# Ping DevOps
#
# Description: Ping Identity helm charts - 03/03/2025
#-------------------------------------------------------------------------------------
#
#            Product          tag   typ  #  cpu R/L   mem R/L  Ing
#    --------------------- ------- --- -- --------- --------- ---
#    global               2502            0/0       0/0     √
#
#  √ pingaccess-admin     2502    sts  1    0/2     1Gi/4Gi   √
#  √ pingaccess-engine    2502    dep  1    0/2     1Gi/4Gi   √
#  √ pingauthorize        2502    dep  1    0/2    1.5G/4Gi   √
#    pingauthorizepap
#    pingcentral
#  √ pingdataconsole      2502    dep  1    0/2    .5Gi/2Gi   √
#    pingdatasync
#    pingdelegator
#  √ pingdirectory        2502    sts  1  50m/2     2Gi/8Gi   √
#    pingdirectoryproxy
#  √ pingfederate-admin   2502    dep  1    0/2     1Gi/4Gi   √
#  √ pingfederate-engine  2502    dep  1    0/2     1Gi/4Gi   √
#    pingintelligence
#
#    ldap-sdk-tools
#    pd-replication-timing
#    pingtoolkit
#
#-------------------------------------------------------------------------------------
# To see values info, simply set one of the following on your helm install/upgrade
#
#    --set help.values=all         # Provides all (i.e. .Values, .Release, .Chart, ...) yaml
#    --set help.values=global      # Provides global values
#    --set help.values={ image }   # Provides image values merged with global
#-------------------------------------------------------------------------------------
```

As you can see, PingAccess Admin and Engine, PingData Console, PingDirectory, PingAuthorize, and the PingFederate Admin and Engine are deployed from the provided `everything.yaml` values file.

It will take several minutes for all components to become operational.

7. To display the status of the deployed components, you can use k9s or issue the corresponding commands shown here:

   - Display the services (endpoints for connecting) by running `kubectl get service --selector=app.kubernetes.io/instance=demo`

     ```
     NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP
     PORT(S)                    AGE
     demo-pingaccess-admin           ClusterIP   10.105.30.25     <none>        9090/TCP,9000/
     TCP         6m33s
     demo-pingaccess-admin-cluster   ClusterIP   None             <none>
     <none>                     6m33s
     demo-pingaccess-engine          ClusterIP   10.100.1.136     <none>        3000/
     TCP                  6m33s
     demo-pingauthorize              ClusterIP   10.101.98.228    <none>        443/
     TCP                   6m33s
     demo-pingauthorize-cluster      ClusterIP   None             <none>        1636/
     TCP                  6m33s
     demo-pingdataconsole            ClusterIP   10.103.181.27    <none>        8443/
     TCP                  6m33s
     demo-pingdirectory              ClusterIP   10.106.174.162   <none>        443/TCP,389/TCP,636/
     TCP   6m33s
     demo-pingdirectory-cluster      ClusterIP   None             <none>        1636/
     TCP                  6m33s
     demo-pingfederate-admin         ClusterIP   10.96.52.217     <none>        9999/
     TCP                  6m33s
     demo-pingfederate-cluster       ClusterIP   None             <none>        7600/TCP,7700/
     TCP         6m33s
     demo-pingfederate-engine        ClusterIP   10.103.84.196    <none>        9031/
     TCP                  6m33s
     ```

   - To view the pods, run `kubectl get pods --selector=app.kubernetes.io/instance=demo` - you will need to run this at intervals until all pods have started (**Running** status):

     ```
     NAME                                      READY   STATUS    RESTARTS   AGE
     demo-pingaccess-admin-0                   1/1     Running   0          7m7s
     demo-pingaccess-engine-59cfb85b9d-7l6tz   1/1     Running   0          7m7s
     demo-pingauthorize-5696dd6b67-hsxnw       1/1     Running   0          7m7s
     demo-pingdataconsole-56b75f9ffb-wld5k     1/1     Running   0          7m7s
     demo-pingdirectory-0                      1/1     Running   0          7m7s
     demo-pingfederate-admin-67cdb47bb4-h88zr  1/1     Running   0          7m7s
     demo-pingfederate-engine-d9889b494-pdhv8  1/1     Running   0          7m7s
     ```

   - To see the ingresses you will use to access the product, run `kubectl get ingress`. If the ingress controller is configured properly, you should see `localhost` as the address as shown here:

```
NAME                         CLASS     HOSTS                                        ADDRESS
PORTS     AGE
demo-pingaccess-admin        nginx     demo-pingaccess-admin.pingdemo.example       localhost   80,
443   7m28s
demo-pingaccess-engine       nginx     demo-pingaccess-engine.pingdemo.example      localhost   80,
443   7m28s
demo-pingauthorize           nginx     demo-pingauthorize.pingdemo.example          localhost   80,
443   7m28s
demo-pingdataconsole         nginx     demo-pingdataconsole.pingdemo.example        localhost   80,
443   7m28s
demo-pingdirectory           nginx     demo-pingdirectory.pingdemo.example          localhost   80,
443   7m28s
demo-pingfederate-admin      nginx     demo-pingfederate-admin.pingdemo.example     localhost   80,
443   7m28s
demo-pingfederate-engine     nginx     demo-pingfederate-engine.pingdemo.example    localhost   80,
443   7m28s   `
```

Address must be localhost

If the ingress controller is working properly, the ingress definitions will all report the ADDRESS column as `localhost` as shown above. If you do not see this entry, then you will not be able to access the services later. This problem is due to a known error with Docker Desktop and the embedded virtual machine (VM) used on the Mac and Windows platform in combination with the ingress controller. To correct the problem, uninstall the chart as instructed at the bottom of this page and restart Docker Desktop. Afterward, you can re-run the helm command to install the Ping products as instructed above. The issue appears to be related to a stale networking configuration under the covers of Docker Desktop.

- To see everything tied to the helm release run `kubectl get all --selector=app.kubernetes.io/instance=demo`:

```
NAME                                           READY    STATUS     RESTARTS    AGE
pod/demo-pingaccess-admin-0                    1/1      Running    0           8m23s
pod/demo-pingaccess-engine-59cfb85b9d-7l6tz    1/1      Running    0           8m23s
pod/demo-pingauthorize-5696dd6b67-hsxnw        1/1      Running    0           8m23s
pod/demo-pingdataconsole-56b75f9ffb-wld5k      1/1      Running    0           8m23s
pod/demo-pingdirectory-0                       1/1      Running    0           8m23s
pod/demo-pingfederate-admin-67cdb47bb4-h88zr   1/1      Running    0           8m23s
pod/demo-pingfederate-engine-d9889b494-pdhv8   1/1      Running    0           8m23s
```

```
NAME                                 TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)                    AGE
service/demo-pingaccess-admin        ClusterIP   10.105.30.25     <none>        9090/TCP,9000/
TCP          8m23s
service/demo-pingaccess-admin-cluster ClusterIP   None             <none>
<none>                     8m23s
service/demo-pingaccess-engine       ClusterIP   10.100.1.136     <none>        3000/
TCP                   8m23s
service/demo-pingauthorize           ClusterIP   10.101.98.228    <none>        443/
TCP                    8m23s
service/demo-pingauthorize-cluster   ClusterIP   None             <none>        1636/
TCP                   8m23s
service/demo-pingdataconsole         ClusterIP   10.103.181.27    <none>        8443/
TCP                   8m23s
service/demo-pingdirectory           ClusterIP   10.106.174.162   <none>        443/TCP,389/
TCP,636/TCP    8m23s
service/demo-pingdirectory-cluster   ClusterIP   None             <none>        1636/
TCP                   8m23s
service/demo-pingfederate-admin      ClusterIP   10.96.52.217     <none>        9999/
TCP                   8m23s
service/demo-pingfederate-cluster    ClusterIP   None             <none>        7600/TCP,7700/
TCP          8m23s
service/demo-pingfederate-engine     ClusterIP   10.103.84.196    <none>        9031/
TCP                   8m23s
```

```
NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/demo-pingaccess-engine    1/1     1            1           8m23s
deployment.apps/demo-pingauthorize        1/1     1            1           8m23s
deployment.apps/demo-pingdataconsole      1/1     1            1           8m23s
deployment.apps/demo-pingfederate-admin   1/1     1            1           8m23s
deployment.apps/demo-pingfederate-engine  1/1     1            1           8m23s
```

```
NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/demo-pingaccess-engine-59cfb85b9d   1         1         1       8m23s
replicaset.apps/demo-pingauthorize-5696dd6b67       1         1         1       8m23s
replicaset.apps/demo-pingdataconsole-56b75f9ffb     1         1         1       8m23s
replicaset.apps/demo-pingfederate-admin-67cdb47bb4  1         1         1       8m23s
replicaset.apps/demo-pingfederate-engine-d9889b494  1         1         1       8m23s
```

```
NAME                                    READY   AGE
statefulset.apps/demo-pingaccess-admin  1/1     8m23s
statefulset.apps/demo-pingdirectory     1/1     8m23s
```

- To view logs, look at the logs for the deployment of the product in question. For example:

```
kubectl logs -f deployment/demo-pingfederate-admin
```

2. These are the URLs and credentials to sign on to the management consoles for the products.

Certificates

This example uses self-signed certificates that will have to be accepted in your browser or added to your keystore.

With the ingresses in place, you can access the products at these URLs:

Product	Connection Details
PingFederate	◦ URL: https://demo-pingfederate-admin.pingdemo.example/pingfederate/app ◦ Username: administrator ◦ Password: 2FederateM0re
PingDirectory	◦ URL: https://demo-pingdataconsole.pingdemo.example/console ◦ Server: ldaps://demo-pingdirectory-cluster:1636 ◦ Username: administrator ◦ Password: 2FederateM0re
PingAccess	◦ URL: https://demo-pingaccess-admin.pingdemo.example/ ◦ Username: administrator ◦ Password: 2FederateM0re
PingAuthorize	◦ URL: https://demo-pingdataconsole.pingdemo.example/console ◦ Server: ldaps://demo-pingauthorize-cluster:1636 ◦ Username: administrator ◦ Password: 2FederateM0re

3. When you are finished, you can remove the demonstration components by running the uninstall command for helm:

```
helm uninstall demo
```

Next Steps

Now that you have deployed a set of our product images using the provided chart, you can move on to deployments using configurations that more closely reflect use cases to be explored. Refer to the helm examples) page for other typical deployments.

Container logging

Maintaining logs in a containerized model is different from the typical server-deployed application. See this page for additional details.

Deployment Examples

Deployment Overview

This section assumes you have already deployed the full-stack server profile in Get Started.

In this section, you will find examples for using **Docker Compose** for running standalone product containers and **Helm/ Kubernetes** to deploy Ping products in typical combinations.

Kubernetes

Deploy a local Kubernetes Cluster

If you do not have access to a managed Kubernetes cluster you can deploy one on your local machine or or a virtual machine (VM). This document describes deploying a cluster with kind and also for minikube. Refer to the documentation of each product for additional information.

Caution

The instructions in this document are for testing and learning, and not intended for use in production.

Note

The processes outlined on this page will create either a Kubernetes in Docker (kind) or a minikube cluster. In both cases, the cluster you get is very similar in functionality to the Docker Desktop implementation of Kubernetes. However, a distinct advantage of both offerings is portability (not requiring Docker Desktop). As with the Deploy an Example Stack procedure, the files provided will enable and deploy an ingress controller for communicating with the services in the cluster from your local environment.

Caution

To use the both examples below, you will need to ensure the Kubernetes feature of Docker Desktop is turned off, as it will conflict.

Note

This note applies only if using Docker as a backing for either solution. kind uses Docker by default, and it is also an option for minikube. Docker on Linux is typically installed with root privileges and thus has access to the full resources of the machine. Docker Desktop for Mac and Windows provides a way to set the resources allocated to Docker. For this documentation, a Macbook Pro with the M4 chipset was configured to use 6 CPUs and 12 GB Memory. You can adjust these values as necessary for your needs.

Kind cluster

This section will cover the **kind** installation process. See the Minikube cluster section for minikube instructions.

Prerequisites

- docker
- kubectl
- ports 80 and 443 available on machine

> **Note**
>
> For this guide, the kind implementation of Kubernetes 1.32.2 is used. It is deployed using version 0.27.0 of kind.

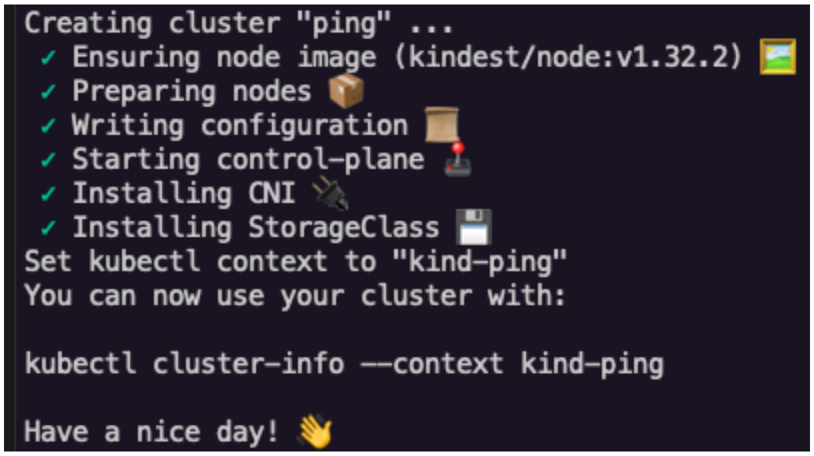
> **Note**
>
> At the time of the writing of this guide, Docker Desktop was version `4.39.0 (184744)`, running Docker Engine `28.0.1`.

Install and confirm the cluster

1. Install kind on your platform.
2. Use the provided sample kind.yaml file to create a kind cluster named `ping` with ingress support enabled. From the root of your copy of the repository code, run:

   `kind create cluster --config=./20-kubernetes/kind.yaml`

   Output:



```
Creating cluster "ping" ...
 ✓ Ensuring node image (kindest/node:v1.32.2) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-ping"
You can now use your cluster with:

kubectl cluster-info --context kind-ping

Have a nice day! 👋
```

3. Test cluster health by running the following commands:

```
kubectl cluster-info

# Output - port will vary
Kubernetes control plane is running at https://127.0.0.1:64129
CoreDNS is running at https://127.0.0.1:64129/api/v1/namespaces/kube-system/services/kube-dns:dns/
proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

------------------

kubectl version

< output clipped >
Server Version: v1.32.2

------------------

kubectl get nodes

NAME                 STATUS   ROLES           AGE    VERSION
ping-control-plane   Ready    control-plane   55m    v1.32.2
```

Enable ingress

1. Next, install the nginx-ingress-controller for `kind` (version 1.12.1 at the time of this writing). In the event the Github file is unavailable, a copy has been made to this repository here.

   To use the Github file:

   `kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/refs/heads/release-1.12/deploy/static/provider/kind/deploy.yaml`

   To use the local copy:

   `kubectl apply -f ./20-kubernetes/kind-nginx.yaml`

   Output:

```
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
```

2. To wait for the Nginx ingress to reach a healthy state, run the following command. You can also observe the pod status using k9s or by running `kubectl get pods --namespace ingress-nginx`. You should see one controller pod running when the ingress controller is ready. This command should exit after no more than 90 seconds or so, depending on the speed of your computer and internet connection to pull the image:

   `kubectl wait --namespace ingress-nginx \ --for=condition=ready pod \ --selector=app.kubernetes.io/component=controller \ --timeout=90s`

3. Verify nginx-ingress-controller is working:

   `curl localhost`

   Output:

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Our examples will use the Helm release name `myping` and DNS domain suffix `pingdemo.example` for accessing applications. You can add all expected hosts to `/etc/hosts`:

```
echo '127.0.0.1 myping-pingaccess-admin.pingdemo.example myping-pingaccess-engine.pingdemo.example myping-
pingauthorize.pingdemo.example myping-pingauthorizepap.pingdemo.example myping-pingdataconsole.pingdemo.example
myping-pingdelegator.pingdemo.example myping-pingdirectory.pingdemo.example myping-pingfederate-
admin.pingdemo.example myping-pingfederate-engine.pingdemo.example myping-pingcentral.pingdemo.example' | sudo tee -
a /etc/hosts > /dev/null
```

Setup is complete. This local Kubernetes environment should be ready to deploy our Helm examples.

Deploy the Example Stack

1. Create a namespace for running the stack in your Kubernetes cluster:

   ```
   # Create the namespace
   kubectl create ns pinghelm

   # Set the kubectl context to the namespace
   kubectl config set-context --current --namespace=pinghelm

   # Confirm
   kubectl config view --minify | grep namespace:
   ```

2. Create a secret in the namespace you will be using to run the example (pinghelm) using the `pingctl` utility. This secret will obtain an evaluation license based on your Ping DevOps username and key:

   `pingctl k8s generate devops-secret | kubectl apply -f -`

3. To install the chart, go to your local `"${PING_IDENTITY_DEVOPS_HOME}"/pingidentity-devops-getting-started/30-helm` directory and run the command shown here. In this example, the release (deployment into Kubernetes by Helm) is called `myping`, forming the prefix for all objects created. The `ingress-demo.yaml` file configures the ingresses for the products to use the ***ping-local*** domain:

   `helm upgrade --install myping pingidentity/ping-devops \ -f everything.yaml -f ingress-demo.yaml`

At this point, the flow will be the same as found in the Getting Started Example after the products are deployed using helm. The URLs will be prefaced with `myping` rather than `demo`.

Stop the cluster

When you are finished, you can remove the cluster by running the following command, which removes the cluster completely. You will be required to recreate the cluster and reinstall the ingress controller to use `kind` again.

`kind delete cluster --name ping`

Minikube cluster

In this section, a minikube installation with ingress is created. Minikube is simpler than kind overall to configure, but ends up needing one step to configured a tunnel to the cluster that must be managed. For this guide, the Docker driver will be used. As with `kind` above, Kubernetes in Docker Desktop must be disabled.

Prerequisites

- Container or virtual machine manager, such as: Docker, QEMU, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation
- kubectl

Note

At the time of the writing of this guide, minikube was version `1.35.0`, which installs Kubernetes version `1.32.0`.

Install and configure minikube

1. Install minikube for your platform. See the product Get Started! page for details.
2. Configure the minikube resources and virtualization driver. For example, the following options were used on an Apple Macbook Pro with Docker as the backing platform:

```
minikube config set cpus 6
minikube config set driver docker
minikube config set memory 12g
```

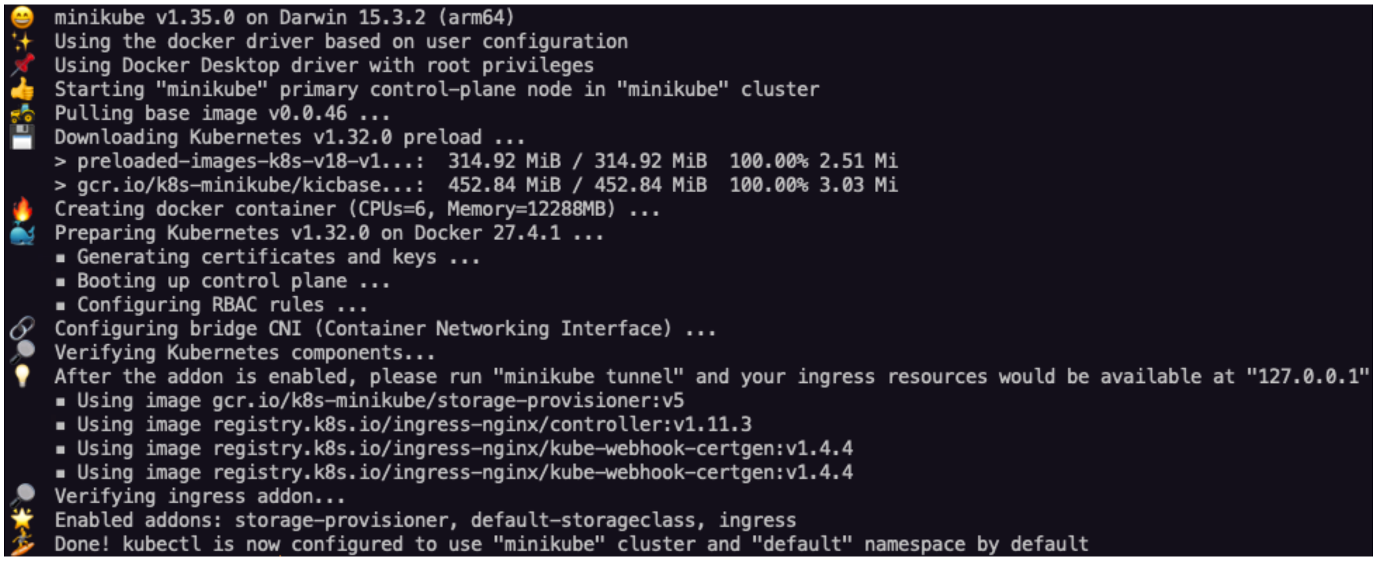
Note

See the documentation for more details on configuring minikube.

3. Start the cluster. Optionally you can include a profile flag (`--profile <name>`). Naming the cluster enables you to run multiple minikube clusters simultaneously. If you use a profile name, you will need to include it on other minikube commands.

`minikube start --addons=ingress --kubernetes-version=v1.32.0`

Output:



```
😄  minikube v1.35.0 on Darwin 15.3.2 (arm64)
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.46 ...
💾  Downloading Kubernetes v1.32.0 preload ...
    > preloaded-images-k8s-v18-v1...:  314.92 MiB / 314.92 MiB  100.00% 2.51 Mi
    > gcr.io/k8s-minikube/kicbase...:  452.84 MiB / 452.84 MiB  100.00% 3.03 Mi
🔥  Creating docker container (CPUs=6, Memory=12288MB) ...
🐳  Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
💡  After the addon is enabled, please run "minikube tunnel" and your ingress resources would be available at "127.0.0.1"
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.11.3
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.4
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.4
🔎  Verifying ingress addon...
🌟  Enabled addons: storage-provisioner, default-storageclass, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

4. Test cluster health by running the following commands:

```
kubectl cluster-info

# Output - Port will vary
Kubernetes control plane is running at https://127.0.0.1:51042
CoreDNS is running at https://127.0.0.1:51042/api/v1/namespaces/kube-system/services/kube-dns:dns/
proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

------------------

kubectl version

< output clipped >
Server Version: v1.32.0

------------------

kubectl get nodes

NAME       STATUS   ROLES           AGE    VERSION
minikube   Ready    control-plane   6m2s   v1.32.0
```

Confirm ingress

1. Confirm ingress is operational:

```
kubectl get po -n ingress-nginx

NAME                                        READY   STATUS      RESTARTS    AGE
ingress-nginx-admission-create-hnmhx        0/1     Completed   0           6m14s
ingress-nginx-admission-patch-c4mct         0/1     Completed   1           6m14s
ingress-nginx-controller-56d7c84fd4-qqffn   1/1     Running     0           6m14s
```

2. Deploy a test application

   Use the following YAML file to create a Pod, Service and Ingress:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-web-pod
  labels:
    role: webserver
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: example-svc
spec:
  selector:
    role: webserver
  ports:
    - protocol: TCP
      port: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  namespace: default
  annotations:
    spec.ingressClassName: nginx
spec:
  rules:
    - host: example.k8s.local
      http:
        paths:
          - backend:
              service:
                name: example-svc
                port:
                  number: 80
            path: /
            pathType: Prefix
```

```
kubectl apply -f test.yaml

pod/example-web-pod created
service/example-svc created
ingress.networking.k8s.io/example-ingress created
```

3. Add an alias to the application to `/etc/hosts`.

`echo '127.0.0.1 example.k8s.local' | sudo tee -a /etc/hosts > /dev/null`

4. Start a tunnel. This command will tie up the terminal:

```
minikube tunnel

✅  Tunnel successfully started

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

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

Open a browser to http://example.k8s.local. You should see the Nginx landing page.

5. Clean up tests

`kubectl delete -f test.yaml`

Our examples will use the Helm release name `myping` and DNS domain suffix `pingdemo.example` for accessing applications. You can add all expected hosts to `/etc/hosts`:

```
echo '127.0.0.1 myping-pingaccess-admin.pingdemo.example myping-pingaccess-engine.pingdemo.example myping-
pingauthorize.pingdemo.example myping-pingauthorizepap.pingdemo.example myping-pingdataconsole.pingdemo.example
myping-pingdelegator.pingdemo.example myping-pingdirectory.pingdemo.example myping-pingfederate-
admin.pingdemo.example myping-pingfederate-engine.pingdemo.example myping-pingcentral.pingdemo.example' | sudo tee -
a /etc/hosts > /dev/null
```

Setup is complete. This local Kubernetes environment should be ready to deploy our Helm examples

Optional features

Dashboard

Minikube provides other add-ons that enhance your experience when working with your cluster. One such add-on is the Dashboard, which can also provide metrics as follows:

`minikube addons enable metrics-server minikube dashboard`

Multiple nodes

If you have enough system resources, you can create a multi-node cluster.

For example, to start a 3-node cluster:

`minikube start --nodes 3`

Caution

Keep in mind that each node will receive the RAM/CPU/Disk configured for minikube. Using the example configuration provided above, a 3-node cluster would need 36GB of RAM and 18 CPUs.

Stop the cluster

When you are finished, you can stop the cluster by running the following command. Stopping retains the configuration and state of the cluster (namespaces, deployments, and so on) that will be restored when starting the cluster again.

```
minikube stop
```

You can also pause and unpause the cluster:

```
minikube pause minikube unpause
```

Alternatively, you can delete the minikube environment, which will do a reset and recreate everything the next time it is started.

```
minikube delete
```

Deploy a robust local Kubernetes Cluster

Note

A video demonstrating the manual process outlined on this page is available here. An updated video using the ansible playbooks is planned.

In some cases, a single-node cluster is insufficient for more complex testing scenarios. If you do not have access to a managed Kubernetes cluster and want something more similar to what you would find in a production environment, this guide can help.

This document describes deploying a multi-node cluster using ansible and the kubeadm utility, running under virtual machines. When completed, the cluster will consist of:

- Three nodes, consisting of a master with two worker nodes (to conserve resources, the master will also be configured to run workloads)
- (At the time of writing) Kubernetes 1.32.3 using the `containerd` runtime (no Docker installed)
- (Optional but recommended) Load balancer
- Block storage support for PVC/PV needed by some Ping products
- (Optional) Ingress controller (ingress-nginx)
- (Optional) Istio service mesh
- (Optional) Supplementary tools for tracing and monitoring with Istio

Warning

While the result is a full Kubernetes installation, the instructions in this document only create an environment sufficient for testing and learning. The cluster is not intended for use in production environments.

Note

The ansible playbooks for this guide assume the ARM chip set as found on the Apple M-series chip. If you are running on an Intel-based processor, you will have to adjust some file packages and names accordingly.

Prerequisites

In order to complete this guide, you will need:

- 64 GB of RAM (32 GB might be enough if you have an SSD to handle some memory swapping and reduce the RAM on the VMs to 12 GB)
- At least 150 GB of free disk
- Modern processor with multiple cores
- Ansible-playbook CLI tool. You can use brew by running `brew install ansible` or see the ansible site for instructions on how to install and configure this application.
- Virtualization solution. For this guide, VMware Fusion is used, but other means of creating and running a VM (Virtualbox, KVM) can be adapted.
- Access to Ubuntu Server 24.04.2 LTS installation media
- **A working knowledge of Kubernetes**, such as knowing how to port-forward, install objects using YAML files, and so on. Details on performing some actions will be omitted, and it is assumed the reader will know what to do.
- Patience

Virtual machines

First, create 3 VMs as described here, using a default installation of Ubuntu 24.04. For this guide, the user created was `ubuntu`. You can use any name with a password of your choice.

- 4 vCPU
- 16 GB RAM
- 2 disks: 80 GB disk for the primary / 60 GB as secondary
- Attached to a network that allows internet access (bridged or NAT), using a fixed IP address

Note

192.168.163.0/24 was the IP space used in this guide; adjust to your environment accordingly.

VM	Hostname	IP address
Master node	k8smaster	192.168.163.70
Worker	k8snode01	192.168.163.71
Worker	k8snode02	192.168.163.72

Preliminary Operating System setup

Perform these actions on all three VMs.

Install the Operating System

Install the operating system as default, using the first disk (80 GB) as the installation target. For this guide, the installation disk was formatted to use the entire disk as the root partition, without LVM support.

Create snapshot 'base-os'

Halt each VM by running `sudo shutdown -h now`.

Create a snapshot of each VM, naming it **base-os**. This snapshot provides a rollback point in case issues arise later. You will use snapshots at several other key points for the same purpose. After installation is complete, these intermediate snapshots can be removed.

Power up each VM set after taking the snapshots.

Prepare for using Ansible

Note

This block of commands is executed on the host.

```
# Add the IP addresses to the local hosts file for convenience
sudo tee -a /etc/hosts >/dev/null <\<-EOF
192.168.163.70 k8smaster
192.168.163.71 k8snode01
192.168.163.72 k8snode02
EOF

# Copy the SSH key you will use to access the VMs from your host machine to each VM.
# See https://www.ssh.com/academy/ssh/keygen for instructions on generating an SSH key
# For this guide, the ed25519 algorithm was used
# Adjust the key name accordingly in the ssh-copy-id command

export TARGET_MACHINES=("k8smaster" "k8snode01" "k8snode02")

for machine in "${TARGET_MACHINES[@]}"; do
   echo "Copying key to $machine:"
   ssh-copy-id -i ~/.ssh/localvms ubuntu@"$machine"
   echo "======================"
   echo "Confirming access. You should not be prompted
   for a password and will be shown the hostname:"
   ssh -i ~/.ssh/localvms ubuntu@"$machine" 'hostname'
   echo
done
```

Sample output:

```
Copying key to k8smaster:
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/Users/davidross/.ssh/localvms.pub"
The authenticity of host 'k8smaster (192.168.163.70)' can't be established.
ED25519 key fingerprint is SHA256:qud9m1FRgwzJuwKcEsVVUbZ4bltYmiyKNj5e330ZQCA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ubuntu@k8smaster's password:

Number of key(s) added:        1

Now try logging into the machine, with:   "ssh 'ubuntu@k8smaster'"

======================
Confirming access. You should not be prompted for a password and will be shown the hostname:
k8smaster

<The output above is repeated for each node.>
```

After installation and reboot, perform the basic configuration needed for ansible support on the VMs. The primary change required is to allow `sudo` commands without requiring a password.

Note

Run these commands on each VM.

```
# Modify /etc/sudoers to allow the ubuntu user to sudo without a password
# This configuration grants the user full root access with no password
# DO NOT DO THIS IN PRODUCTION!

echo "ubuntu ALL=(ALL) NOPASSWD: ALL" | sudo tee -a /etc/sudoers
```

Configure Ansible Playbooks for your environment

At this point, you are ready to modify the ansible playbooks for creating your cluster.

1. If you have not already done so, clone the `pingidentity-devops-getting-started` repository to your local `${PING_IDENTITY_DEVOPS_HOME}` directory.

   Note

   The `${PING_IDENTITY_DEVOPS_HOME}` environment variable was set by running `pingctl config`.

   ```
   cd "$\{PING_IDENTITY_DEVOPS_HOME}"
   git clone \
   https://github.com/pingidentity/pingidentity-devops-getting-started.git
   ```

2. Navigate to the directory with the ansible scripts:

```
cd "$\{PING_IDENTITY_DEVOPS_HOME}"/pingidentity-devops-getting-started/99-helper-scripts/ansible
```

3. Modify the `inventory.ini`, `ansible.cfg`, `install_kubernetes.yaml` and `install_list.yaml` files accordingly to suit your environment.

    1. The **inventory.ini** will need modification for your IP addresses, private key file, and user if one other than `ubuntu` was used:

    ```
    [kubernetes_master]
    k8smaster ansible_host=192.168.163.70

    [kubernetes_nodes]
    k8snode01 ansible_host=192.168.163.71
    k8snode02 ansible_host=192.168.163.72

    [all:vars]
    ansible_user=ubuntu
    ansible_ssh_private_key_file=/Users/davidross/.ssh/localvms
    ansible_python_interpreter=/usr/bin/python3
    ```

    2. The **ansible.cfg** file should not need any modification:

    ```
    [defaults]
    inventory = inventory.ini
    host_key_checking = False
    ```

    3. The **install_kubernetes.yaml** file will need the following changes to lines 11-13 if your IP address differs from this example:

    ```
    k8smaster_ip: "192.168.163.70"
    k8snode01_ip: "192.168.163.71"
    k8snode02_ip: "192.168.163.72"
    ```

    4. Finally, update the **install_list.yaml** file. By default, no additional components are installed other than block storage, which is needed for some Ping products. To install other optional components, set the value to ***True***. Note that helm is required to install the Ingress controller, and adding K9s, metallb and ingress will provide additional tools for the most production-like implementation:

    ```
    ---
    helm: False
    k9s: False
    metallb: False
    storage: True
    ingress: False
    istio: False
    istioaddons: False
    ...
    ```

Run the first playbook

With the changes above, you are now ready to run the playbooks. First, run the **install_kubernetes.yaml** playbook to install Kubernetes.

```
ansible-playbook install_kubernetes.yaml -i inventory.ini
```

Note

The playbook was designed to to be idempotent so you can run it multiple times if needed. An alternative is to reset to the **base-os** snapshot, update the `sudoers` file and run it again.

Sample output

```
PLAY [Install Kubernetes on all VMs] ************************************************************************

TASK [Gathering Facts] ************************************************************************
ok: [k8smaster]
ok: [k8snode01]
ok: [k8snode02]

TASK [Gather architecture info] ************************************************************************
ok: [k8smaster]
ok: [k8snode01]
ok: [k8snode02]

TASK [Update package cache] ************************************************************************
ok: [k8smaster]
ok: [k8snode02]
ok: [k8snode01]

TASK [Upgrade all packages] ************************************************************************
changed: [k8snode02]
changed: [k8snode01]
changed: [k8smaster]

TASK [Add host file entries] ************************************************************************
changed: [k8snode02]
changed: [k8snode01]
changed: [k8smaster]

TASK [Add the br_netfilter kernel module and configure for load at boot]
************************************************************************
changed: [k8snode02]
changed: [k8snode01]
changed: [k8smaster]

TASK [Add the overlay kernel module and configure for load at boot]
************************************************************************
changed: [k8snode01]
changed: [k8snode02]
changed: [k8smaster]

TASK [Creating kernel modules file to load at boot]
************************************************************************
changed: [k8snode02]
changed: [k8smaster]
changed: [k8snode01]

TASK [Disable swap in fstab by commenting it out]
************************************************************************
changed: [k8snode01]
changed: [k8snode02] changed: [k8smaster]

TASK [Disable swap] ************************************************************************
changed: [k8snode02]
changed: [k8snode01]
changed: [k8smaster]

TASK [Enable IP forwarding for iptables] ************************************************************************
changed: [k8smaster]
changed: [k8snode01]
changed: [k8snode02]
```

```
TASK [Update sysctl parameters without reboot - bridge (ipv4)]
************************************************************************
changed: [k8snode02]
changed: [k8snode01]
changed: [k8smaster]

TASK [Update sysctl parameters without reboot - bridge (ipv6)]
************************************************************************
changed: [k8smaster]
changed: [k8snode01]
changed: [k8snode02]

TASK [Update sysctl parameters without reboot - IPforward]
************************************************************************
changed: [k8smaster]
changed: [k8snode01]
changed: [k8snode02]

TASK [Install containerd] ************************************************************************
changed: [k8snode02] \=> (item=containerd)
changed: [k8snode01] \=> (item=containerd)

changed: [k8smaster] \=> (item=containerd) TASK [Add Kubernetes APT key]
************************************************************************
changed: [k8smaster]
changed: [k8snode02]
changed: [k8snode01]

TASK [Create directory for containerd configuration file]
************************************************************************
changed: [k8snode02]
changed: [k8snode01]
changed: [k8smaster]

TASK [Check if containerd toml configuration file exists]
************************************************************************
ok: [k8smaster]
ok: [k8snode01]
ok: [k8snode02]

TASK [Create containerd configuration file if it does not exist]
************************************************************************
changed: [k8snode01]
changed: [k8smaster]
changed: [k8snode02]

TASK [Read config.toml file]
************************************************************************
ok: [k8snode01]
ok: [k8snode02]
ok: [k8smaster]

TASK [Check if correct containerd.runtimes.runc line exists]
************************************************************************
sk: [k8smaster]
ok: [k8snode01]
ok: [k8snode02]

TASK [Error out if the incorrect or missing containerd.runtimes.runc line does not exist]
************************************************************************
skipping: [k8smaster]
skipping: [k8snode01]
```

```
skipping: [k8snode02]

TASK [Set SystemdCgroup line in file to true if it is currently false]
*****************************************************************************
changed: [k8smaster]
changed: [k8snode01]
changed: [k8snode02]

TASK [Restart containerd service] *****************************************************************************
changed: [k8snode02]
changed: [k8smaster]
changed: [k8snode01]

TASK [Install prerequisites for Kubernetes] *****************************************************************************
changed: [k8snode01] \=> (item=apt-transport-https)
changed: [k8smaster] \=> (item=apt-transport-https)
changed: [k8snode02] \=> (item=apt-transport-https)
ok: [k8smaster] \=> (item=ca-certificates)
ok: [k8snode01] \=> (item=ca-certificates)
ok: [k8snode02] \=> (item=ca-certificates)
ok: [k8snode02] \=> (item=curl)
ok: [k8smaster] \=> (item=curl)
ok: [k8snode01] \=> (item=curl)
changed: [k8smaster] \=> (item=gnupg2)
changed: [k8snode02] \=> (item=gnupg2)
changed: [k8snode01] \=> (item=gnupg2)
ok: [k8snode02] \=> (item=software-properties-common)
ok: [k8smaster] \=> (item=software-properties-common)
ok: [k8snode01] \=> (item=software-properties-common)
changed: [k8snode01] \=> (item=bzip2)
changed: [k8snode02] \=> (item=bzip2)
changed: [k8smaster] \=> (item=bzip2)
ok: [k8snode02] \=> (item=tar)
ok: [k8snode01] \=> (item=tar)
ok: [k8smaster] \=> (item=tar)
ok: [k8snode02] \=> (item=vim)
ok: [k8smaster] \=> (item=vim)
ok: [k8snode01] \=> (item=vim)
ok: [k8snode02] \=> (item=git)
ok: [k8snode01] \=> (item=git)
ok: [k8smaster] \=> (item=git)
ok: [k8snode02] \=> (item=wget)
ok: [k8smaster] \=> (item=wget)
ok: [k8snode01] \=> (item=wget)
ok: [k8smaster] \=> (item=net-tools)
ok: [k8snode02] \=> (item=net-tools)
ok: [k8snode01] \=> (item=net-tools)
ok: [k8snode02] \=> (item=lvm2)
ok: [k8smaster] \=> (item=lvm2)
ok: [k8snode01] \=> (item=lvm2)

TASK [Get Kubernetes package signing key] *****************************************************************************
changed: [k8smaster]
changed: [k8snode01]
changed: [k8snode02]

TASK [Add Kubernetes APT repository] *****************************************************************************
hanged: [k8snode01]
changed: [k8smaster]
changed: [k8snode02]

TASK [Update package cache] *****************************************************************************
```

```
ok: [k8smaster]
ok: [k8snode02]
ok: [k8snode01]

TASK [Install Kubernetes components] ************************************************************************
changed: [k8smaster] \=> (item=kubelet)
changed: [k8snode02] \=> (item=kubelet)
changed: [k8snode01] \=> (item=kubelet)
changed: [k8smaster] \=> (item=kubeadm)
changed: [k8snode01] \=> (item=kubeadm)
changed: [k8snode02] \=> (item=kubeadm)
changed: [k8smaster] \=> (item=kubectl)
changed: [k8snode01] \=> (item=kubectl)
changed: [k8snode02] \=> (item=kubectl)

TASK [Hold Kubernetes packages at current version]
************************************************************************
changed: [k8smaster]
changed: [k8snode01]
changed: [k8snode02]

TASK [Run kubeadm reset to ensure fresh start each time.]
************************************************************************
changed: [k8smaster]
changed: [k8snode02]
changed: [k8snode01]

TASK [Remove any files from a previous installation attempt]
************************************************************************
skipping: [k8snode01] \=> (item=/home/ubuntu/.kube)
skipping: [k8snode01]
skipping: [k8snode02] \=> (item=/home/ubuntu/.kube)
skipping: [k8snode02]
ok: [k8smaster] \=> (item=/home/ubuntu/.kube)

TASK [Initialize Kubernetes master] ************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Check if k8s installation file exists] ************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
ok: [k8smaster]

TASK [Fail if K8s installed file does not exist]
************************************************************************
skipping: [k8smaster]
skipping: [k8snode01]
skipping: [k8snode02]

TASK [Create .kube directory] ************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Copy kubeconfig to user's home directory]
************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]
```

```
TASK [Install Pod network] ************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Remove taint from master node] ************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Retrieve join command from master and run it on the nodes]
************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Join worker nodes to the cluster] ************************************************************************
skipping: [k8snode01] \=> (item=k8snode01)
skipping: [k8snode01] \=> (item=k8snode02)
skipping: [k8snode01]
skipping: [k8snode02] \=> (item=k8snode01)
skipping: [k8snode02] \=> (item=k8snode02)
skipping: [k8snode02]
changed: [k8smaster \-> k8snode01(192.168.163.71)] \=> (item=k8snode01)
changed: [k8smaster \-> k8snode02(192.168.163.72)] \=> (item=k8snode02)

TASK [Pause for 5 seconds] ************************************************************************
Pausing for 5 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [k8smaster]

TASK [Confirm flannel pods are ready] ************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Run confirmation command by listing nodes]
************************************************************************
changed: [k8smaster]
changed: [k8snode02 \-> k8smaster(192.168.163.70)]
changed: [k8snode01 \-> k8smaster(192.168.163.70)]

TASK [Nodes in the cluster] ************************************************************************
ok: [k8smaster] \=> {
    "nodes_command_output.stdout_lines": [
       "NAME STATUS ROLES AGE VERSION",
       "k8smaster Ready control-plane 36s v1.32.3",
       "k8snode01 Ready +++<none>+++25s v1.32.3",
       "k8snode02 Ready +++<none>+++24s v1.32.3"
    ]
}
ok: [k8snode01] \=> {
    "nodes_command_output.stdout_lines": [
        "NAME STATUS ROLES AGE VERSION",
        "k8smaster Ready control-plane 36s v1.32.3",
        "k8snode01 Ready +++<none>+++25s v1.32.3",
        "k8snode02 Ready +++<none>+++24s v1.32.3"
    ]
}
ok: [k8snode02] \=> {
    "nodes_command_output.stdout_lines": [
        "NAME STATUS ROLES AGE VERSION",
        "k8smaster Ready control-plane 36s v1.32.3",
        "k8snode01 Ready +++<none>+++25s v1.32.3",
        "k8snode02 Ready +++<none>+++24s v1.32.3"
    ]
}

TASK [Provide cluster connection information]
************************************************************************
skipping: [k8snode01]
skipping: [k8snode02]
changed: [k8smaster]

TASK [Cluster information for your .kube/config file]
************************************************************************
ok: [k8smaster] \=> {
    "msg": [
        "apiVersion: v1",
        "clusters:",
        "- cluster:",
        "    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0l...",
        "    server: https://192.168.163.70:6443",
        "  name: kubernetes",
        "contexts:",
        "- context:",
        "    cluster: kubernetes",
        "    user: kubernetes-admin",
        "  name: kubernetes-admin@kubernetes",
        "current-context: kubernetes-admin@kubernetes",
        "kind: Config",
        "preferences: {}",
        "users:",
        "- name: kubernetes-admin",
        "  user:",
        "    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLVENDQWhHZ0F3SUJBZ0lJVnZqM2x...",
        "    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBM0R4WkU..." ] }

skipping: [k8snode01]
skipping: [k8snode02]

PLAY RECAP *****************************************************************************************
k8smaster      : ok=45   changed=33   unreachable=0   failed=0   skipped=2    rescued=0   ignored=0
k8snode01      : ok=32   changed=24   unreachable=0   failed=0   skipped=14   rescued=0   ignored=0
k8snode02.     : ok=32   changed=24   unreachable=0   failed=0   skipped=14   rescued=0   ignored=0
```

(Optional but recommended) Create snapshot 'k8s-installed'

Halt each VM by running `sudo shutdown -h now`.

Create a snapshot of each VM, naming it **k8s-installed**.

Power up each VM set after taking the snapshot.

Run the components playbook

Now that Kubernetes is installed, modify the `install_list.yaml` file accordingly to enable or disable the components that you want to install. After making changes, run the **install_others.yaml** playbook.

`ansible-playbook install_others.yaml -i inventory.ini`

Note

The playbook was designed to to be idempotent so you can run it multiple times if needed. An alternative is to reset to the **k8s-installed** snapshot and run it again.

Note

After K9s is installed on the master, you can launch it in an SSH session and monitor the progress of the pods being instantiated while the playbook is running.

Sample output with everything enabled other than Istio and the Istio-addons

```
PLAY [Install Other Components in the Cluster] *********************************

TASK [Gathering Facts] *********************************************************
ok: [k8smaster]

TASK [Get helm installation file] **********************************************
changed: [k8smaster]

TASK [Extract helm] ************************************************************
changed: [k8smaster]

TASK [Remove helm tarball and extracted folder] ********************************
changed: [k8smaster] => (item=/home/ubuntu/linux-amd64)
changed: [k8smaster] => (item=/home/ubuntu/helm-v3.12.1-linux-amd64.tar.gz)

TASK [Get K9s installation file] ***********************************************
changed: [k8smaster]

TASK [Extract k9s] *************************************************************
changed: [k8smaster]

TASK [Remove k9s tarball] ******************************************************
changed: [k8smaster]

TASK [Get latest MetalLB version] **********************************************
changed: [k8smaster]

TASK [Get MetalLB installer] ***************************************************
changed: [k8smaster]

TASK [Apply MetalLB file] ******************************************************
changed: [k8smaster]

TASK [Pause for 10 seconds] ****************************************************
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [k8smaster]

TASK [Wait for MetalLB controller and speaker pods to be ready] ****************
changed: [k8smaster] => (item=app=metallb)
changed: [k8smaster] => (item=component=speaker)

TASK [Creating MetalLB configuration file] *************************************
changed: [k8smaster]

TASK [Configure MetalLB] *******************************************************
changed: [k8smaster]

TASK [Remove MetalLB installation yaml files] **********************************
changed: [k8smaster] => (item=/home/ubuntu/ipaddress_pool_metal.yaml)
changed: [k8smaster] => (item=/home/ubuntu/metallb-native.yaml)

TASK [Install CertManager prerequisite] ****************************************
changed: [k8smaster]

TASK [Check if rook directory exists] ******************************************
ok: [k8smaster]

TASK [Remove directory] ********************************************************
skipping: [k8smaster]
```

```
TASK [Clone Rook repository] ***************************************************
changed: [k8smaster]

TASK [Install Rook controller] *************************************************
changed: [k8smaster]

TASK [Wait for Rook controller pod to be ready] ********************************
changed: [k8smaster]

TASK [Install Rook components] *************************************************
changed: [k8smaster]

TASK [Pause for 3 1/2 minutes - wait for Rook components to get started] *******
Pausing for 210 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [k8smaster]

TASK [Confirm Rook cluster pods are ready] *************************************
changed: [k8smaster] => (item=app=csi-cephfsplugin)
changed: [k8smaster] => (item=app=csi-cephfsplugin-provisioner)
changed: [k8smaster] => (item=app=csi-rbdplugin)
changed: [k8smaster] => (item=app=rook-ceph-mgr)
changed: [k8smaster] => (item=app=rook-ceph-mon)
changed: [k8smaster] => (item=app=rook-ceph-crashcollector)
changed: [k8smaster] => (item=app=csi-rbdplugin-provisioner)
changed: [k8smaster] => (item=app=rook-ceph-osd)

TASK [Creating Block Storage class] ********************************************
changed: [k8smaster]

TASK [Create block ceph storage class] *****************************************
changed: [k8smaster]

TASK [Creating script to patch storage class (globbing and substitution hack)] ***
changed: [k8smaster]

TASK [Set storage class as default] ********************************************
changed: [k8smaster]

TASK [Remove Rook installation files] ******************************************
changed: [k8smaster] => (item=/home/ubuntu/rook)
changed: [k8smaster] => (item=/home/ubuntu/sc-ceph-block.yaml)
changed: [k8smaster] => (item=/home/ubuntu/patchsc.yaml)

TASK [Install Ingress Nginx] ***************************************************
changed: [k8smaster]

TASK [Pause for 5 seconds] *****************************************************
Pausing for 5 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [k8smaster]

TASK [Confirm ingress controller pod is ready] *********************************
changed: [k8smaster]

TASK [Get Ingress service components for confirmation] *************************
changed: [k8smaster]

TASK [Ingress controller information] ******************************************
ok: [k8smaster] => {
    "msg": [
        "NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)                      AGE",
        "ingress-nginx-controller             LoadBalancer   10.101.36.117    192.168.163.151   80:30865/TCP,
443:31410/TCP   31s",
        "ingress-nginx-controller-admission   ClusterIP      10.103.154.247   <none>            443/
TCP                      31s"
    ]
}

TASK [Get 'istioctl' installation file] ****************************************
skipping: [k8smaster]

TASK [Extract istioctl] ********************************************************
skipping: [k8smaster]

TASK [Install istio] ***********************************************************
skipping: [k8smaster]

TASK [Pause for 5 seconds] *****************************************************
skipping: [k8smaster]

TASK [Confirm Istio pods are ready] ********************************************
skipping: [k8smaster] => (item=app=istiod)
skipping: [k8smaster] => (item=app=istio-ingressgateway)
skipping: [k8smaster] => (item=app=istio-egressgateway)
skipping: [k8smaster]

TASK [Install Istio add-ons] ***************************************************
skipping: [k8smaster]

TASK [Confirm Istio additional pods are ready] *********************************
skipping: [k8smaster] => (item=app=grafana)
skipping: [k8smaster] => (item=app=jaeger)
skipping: [k8smaster] => (item=app=kiali)
skipping: [k8smaster] => (item=app=prometheus)
skipping: [k8smaster] => (item=app.kubernetes.io/name=loki)
skipping: [k8smaster]

TASK [Creating patch file for services] ****************************************
skipping: [k8smaster]

TASK [Patch istio add-on services to use load balancer] ************************
skipping: [k8smaster] => (item=grafana)
skipping: [k8smaster] => (item=kiali)
skipping: [k8smaster] => (item=tracing)
skipping: [k8smaster] => (item=prometheus)
skipping: [k8smaster]

TASK [Remove istio tarball and extracted folder] *******************************
skipping: [k8smaster] => (item=/home/ubuntu/istio-1.18.0)
skipping: [k8smaster] => (item=/home/ubuntu/istio-1.18.0-linux-amd64.tar.gz)
skipping: [k8smaster] => (item=/home/ubuntu/patch-service.yaml)
skipping: [k8smaster]

PLAY RECAP *********************************************************************
k8smaster       : ok=33   changed=27   unreachable=0    failed=0    skipped=11   rescued
```

Snapshot 'k8sComplete'

Shut down the VMs, and snapshot each one. See the helper script in this repository located at `99-helper-scripts/manageCluster.sh` that can be used for automating things under VMware. At this time, your cluster is ready for use.

Resources & References

This guide was built on the work of others. As with many how-to documents, we have contributed our skills and knowledge to pull everything together and fill in the gaps that were experienced. However, we want to acknowledge at least some of the many sources where we found inspiration, guidance, fixes for errors, and sanity when a step was missed. Not shown here are dozens of places where we went in exploring different options for pieces we did not use or install in the end. **The Ping DevOps Integrations team**

- How to Deploy MetalLB on Kubernetes - ComputingForGeeks
- How To: Ubuntu / Debian Linux Regenerate OpenSSH Host Keys - nixCraft
- Block Storage Overview - Rook Ceph Documentation
- Toolbox - Rook Ceph Documentation
- Quickstart - Rook Ceph Documentation
- How To Install Kubernetes on Ubuntu 24.04 LTS
- Containerd Github
- etcd-io Github issue #13670

Deploy a Local Openshift Cluster

Caution

The instructions in this document are for testing and learning, and not intended for use in production.

Note

Openshift is a licensed product following the open source model where "upstream" versions are available under community licensing, but the official release supported by Red Hat requires a subscription in order to access the software for installation. This guide assumes the user has access to such a license. Red Hat provides a free Developer account that allows a participant to obtain a 60-day license of the Openshift product at no charge.

Note

A video demonstration of the process outlined on this page is available here.

Some customers are using Openshift as their platform for running Ping containerized applications. If this is the case, access to an Openshift cluster is assumed. Even in those cases, there are times when a local implementation of Openshift for development and testing is convenient.

Note

For this guide, the Apple MacBook Pro M4 platform is used, and the release of the *Red Hat Openshift Local* offering is version 2.49.0, which installs Openshift 4.18.2.

The Openshift Local offering is used in this guide.

Prerequisites

- Entitlement for Openshift code. If you have registered for the Red Hat developer program, you can obtain your entitlement for the free trial from the portal after logging in.
- kubectl
- Openshift client (oc)
- ports 80, 443 and 6443 available on machine. If you have Docker Desktop installed, you must either disable Kubernetes or stop Docker in order for the installation to work.
- **sudo** privileges on your hosting environment

Configuration and Setup

1. Install the Red Hat Openshift Local binary for your platform. When you download the installer, also download the pull secret.
2. With the `crc` utility installed, configure settings. Provide as much RAM and CPU as you can, depending on your system. In this example, 20 GB of RAM, 9 vCPUs, and a disk size of 80GB are set. The consent-telemetry is optional, depending on whether you consent to usage data metrics being sent to Red Hat.

```
# Version information
crc version

# Output
CRC version: 2.49.0+e843be
OpenShift version: 4.18.2
MicroShift version: 4.18.2

# Set configuration
crc config set memory 20480
crc config set cpus 9
crc config set consent-telemetry no
crc config set disk-size 80
crc config set pull-secret-file <path>/pull-secret.txt

# Confirm
crc config view

- consent-telemetry                     : no
- cpus                                  : 9
- disk-size                             : 80
- memory                                : 20480
- pull-secret-file                      : <path>/pull-secret.txt
```

3. Set up your local machine for running Red Hat Openshift Local by running **crc setup**:

```
crc setup

# Output
INFO Using bundle path /Users/davidross/.crc/cache/crc_vfkit_4.18.2_arm64.crcbundle
INFO Checking if running macOS version >= 13.x
INFO Checking if running as non-root
INFO Checking if crc-admin-helper executable is cached
INFO Checking if running on a supported CPU architecture
INFO Checking if crc executable symlink exists
INFO Checking minimum RAM requirements
INFO Check if Podman binary exists in: /Users/davidross/.crc/bin/oc
INFO Checking if running emulated on Apple silicon
INFO Checking if vfkit is installed
INFO Checking if CRC bundle is extracted in '$HOME/.crc'
INFO Checking if /Users/davidross/.crc/cache/crc_vfkit_4.18.2_arm64.crcbundle exists
INFO Getting bundle for the CRC executable
INFO Downloading bundle: /Users/davidross/.crc/cache/crc_vfkit_4.18.2_arm64.crcbundle...
5.35 GiB / 5.35 GiB [---------------------------------------] 100.00% 38.49 MiB/s
INFO Uncompressing /Users/davidross/.crc/cache/crc_vfkit_4.18.2_arm64.crcbundle
crc.img:  31.00 GiB / 31.00 GiB [--------------------------------------] 100.00%
oc:  138.71 MiB / 138.71 MiB [-----------------------------------------] 100.00%
INFO Checking if old launchd config for tray and/or daemon exists
INFO Checking if crc daemon plist file is present and loaded
INFO Adding crc daemon plist file and loading it
INFO Checking SSH port availability
Your system is correctly setup for using CRC. Use 'crc start' to start the instance
```

4. Start the Red Hat Openshift Local instance:

```
crc start

# Output
INFO Using bundle path /Users/davidross/.crc/cache/crc_vfkit_4.18.2_arm64.crcbundle
INFO Checking if running macOS version >= 13.x
INFO Checking if running as non-root
INFO Checking if crc-admin-helper executable is cached
INFO Checking if running on a supported CPU architecture
INFO Checking if crc executable symlink exists
INFO Checking minimum RAM requirements
INFO Check if Podman binary exists in: /Users/davidross/.crc/bin/oc
INFO Checking if running emulated on Apple silicon
INFO Checking if vfkit is installed
INFO Checking if old launchd config for tray and/or daemon exists
INFO Checking if crc daemon plist file is present and loaded
INFO Checking SSH port availability
INFO Loading bundle: crc_vfkit_4.18.2_arm64...
INFO Starting CRC VM for openshift 4.18.2...
INFO CRC instance is running with IP 127.0.0.1
INFO CRC VM is running
INFO Updating authorized keys...
INFO Resizing /dev/vda4 filesystem
INFO Configuring shared directories
INFO Check internal and public DNS query...
INFO Check DNS query from host...
INFO Verifying validity of the kubelet certificates...
INFO Starting kubelet service
INFO Waiting for kube-apiserver availability... [takes around 2min]
INFO Adding user's pull secret to the cluster...
INFO Updating SSH key to machine config resource...
INFO Waiting until the user's pull secret is written to the instance disk...
INFO Changing the password for the kubeadmin user
INFO Updating cluster ID...
INFO Updating root CA cert to admin-kubeconfig-client-ca configmap...
INFO Starting openshift instance... [waiting for the cluster to stabilize]
INFO Operator authentication is progressing
INFO Operator console is progressing
INFO All operators are available. Ensuring stability...
INFO Operators are stable (2/3)...
INFO Operators are stable (3/3)...
INFO Adding crc-admin and crc-developer contexts to kubeconfig...
Started the OpenShift cluster.

The server is accessible via web console at:
  https://console-openshift-console.apps-crc.testing

Log in as administrator:
  Username: kubeadmin
  Password: <password>

Log in as user:
  Username: developer
  Password: <password>

Use the 'oc' command line interface:
  $ eval $(crc oc-env)
  $ oc login -u developer https://api.crc.testing:6443
```

Depending on the speed of your system, this will take 5 to 15 minutes. There is a 10 minute timeout on checking the stability of operators deployed by Openshift. It might be the case that the tool reports these have not reached full stability in that window, particularly if using an Apple MacBook Pro with an Intel chip. In the writing of this guide, no issues were found using Openshift deployed in this manner, even if the error occurs. Each time the steps in this guide were tested, everything eventually reached a healthy status, even if not in the window expected on some occasions.

Setup is complete. This local environment should be ready to deploy our Helm examples.

Stop the Red Hat Openshift Local instance

When not working with the environment, you can stop the instance by running the following command. All settings, projects and objects created will be retained and available when it is started again.

`crc stop`

Run `crc start` again to launch the instance.

Delete the Red Hat Openshift Local instance

You can also remove the instance by running the following command. If you take this action, the embedded VM instance, all objects and projects created will be lost. A new instance will be deployed the next time you run `crc start`.

`crc delete`

Note

Deleting the instance does not delete the configuration settings for Red Hat Openshift Local (RAM, CPU, disk, and so on, created or modified when running the `crc config set` command). If you want to completely remove all configuration, you can delete the $HOME/.crc folder and its contents. Also, you will need to edit `/etc/hosts` and remove the following aliases to the 127.0.0.1 IP: `api.crc.testing canary-openshift-ingress-canary.apps-crc.testing console-openshift-console.apps-crc.testing default-route-openshift-image-registry.apps-crc.testing downloads-openshift-console.apps-crc.testing host.crc.testing oauth-openshift.apps-crc.testing`

Deploy Ping DevOps Charts using Helm

Helm Charts Repo

To use Ping Identity Helm charts for deployment to Kubernetes, go to the Getting Started page for the Helm chart repository to configure your system to run Helm. Afterward, continue on this page for examples illustrating how to deploy various scenarios from the charts.

Note

If you want to be notified when a new version of the chart is released, see the **Orchestration/Helm/Kubernetes** section of the FAQ page for instructions on following the GitHub repository for our chart.

Note

If you want to run these examples on the local kind cluster created as outlined on the Deploy a local Kubernetes cluster page with ingresses, see this page for ingress configuration instructions. Otherwise, you can port-forward to product services.

Helm Chart Example Configurations

The following table contains example configurations and instructions to run and configure Ping products using the Ping Devops Helm Chart.

Config	Description	`.yaml`	Notes
Everything	Example with most products integrated together	everything.yaml	
Ingress	Expose an application outside of the cluster	ingress.yaml	Update line 7 with your domain
RBAC	Enable RBAC for workloads	rbac.yaml	
Vault	Example vault values section	vault.yaml	

Vault Keystores	Example vault values for keystores	vault-keystores.yaml	
PingAccess	PingAccess Admin Console & Engine	pingaccess-cluster.yaml	
PingAccess & PingFederate Integration	PA & PF Admin Console & Engine	pingaccess-pingfederate-integration.yaml	
PingFederate	PingFederate Admin Console & Engine	pingfederate-cluster.yaml	
PingFederate	Upgrade PingFederate	See .yaml files in pingfederate-upgrade	
PingDirectory	PingDirectory Instance	pingdirectory.yaml	
PingDirectory Upgrade	PingDirectory Upgrade with partition	See .yaml files in pingdirectory-upgrade-partition	
PingDirectory Backup and Sidecar	PingDirectory with periodic backup and sidecar	pingdirectory-periodic-backup.yaml	
PingDirectory Archive Backup to S3 (Demo Only)	Archive PingDirectory backup to S3	Sample files in s3-sidecar	
PingDirectory Scale Down	Scale Down a PingDirectory StatefulSet	See .yaml files in pingdirectory-scale-down	
PingAuthorize and PingDirectory	PingAuthorize with PAP and PingDirectory	pingauthorize-pingdirectory.yaml	
Entry Balancing	PingDirectory and PingDirectoryProxy entry balancing	See .yaml files in entry-balancing	
PingCentral	PingCentral	pingcentral.yaml	
PingCentral with MySQL	PingCentral with external MySQL deployment	pingcentral-external-mysql-db.yaml	
Simple Sync	PingDataSync and PingDirectory	simple-sync.yaml	
PingDataSync Failover	PingDataSync and PingDirectory with failover	pingdatasync-failover.yaml	
Cluster Metrics	Example values using various open source tools	See .yaml files in cluster-metrics	

PingDataConsole SSO with PingOne	Sign into PingDataConsole with PingOne	pingdataconsole-pingone-sso.yaml	
Using CSI Volumes	Mount secrets with CSI volumes	csi-secrets-volume.yaml	
Splunk logging sidecar	Forward product logs to splunk	See files in the splunk folder	
ImagePullSecrets (individual)	Provide secret for private registry authentication	image-pull-secrets-individual.yaml	Replace stubs with your values
ImagePullSecrets (global)	Provide global secret for private registry authentication	image-pull-secrets-global.yaml	Replace stubs with your values

To Deploy

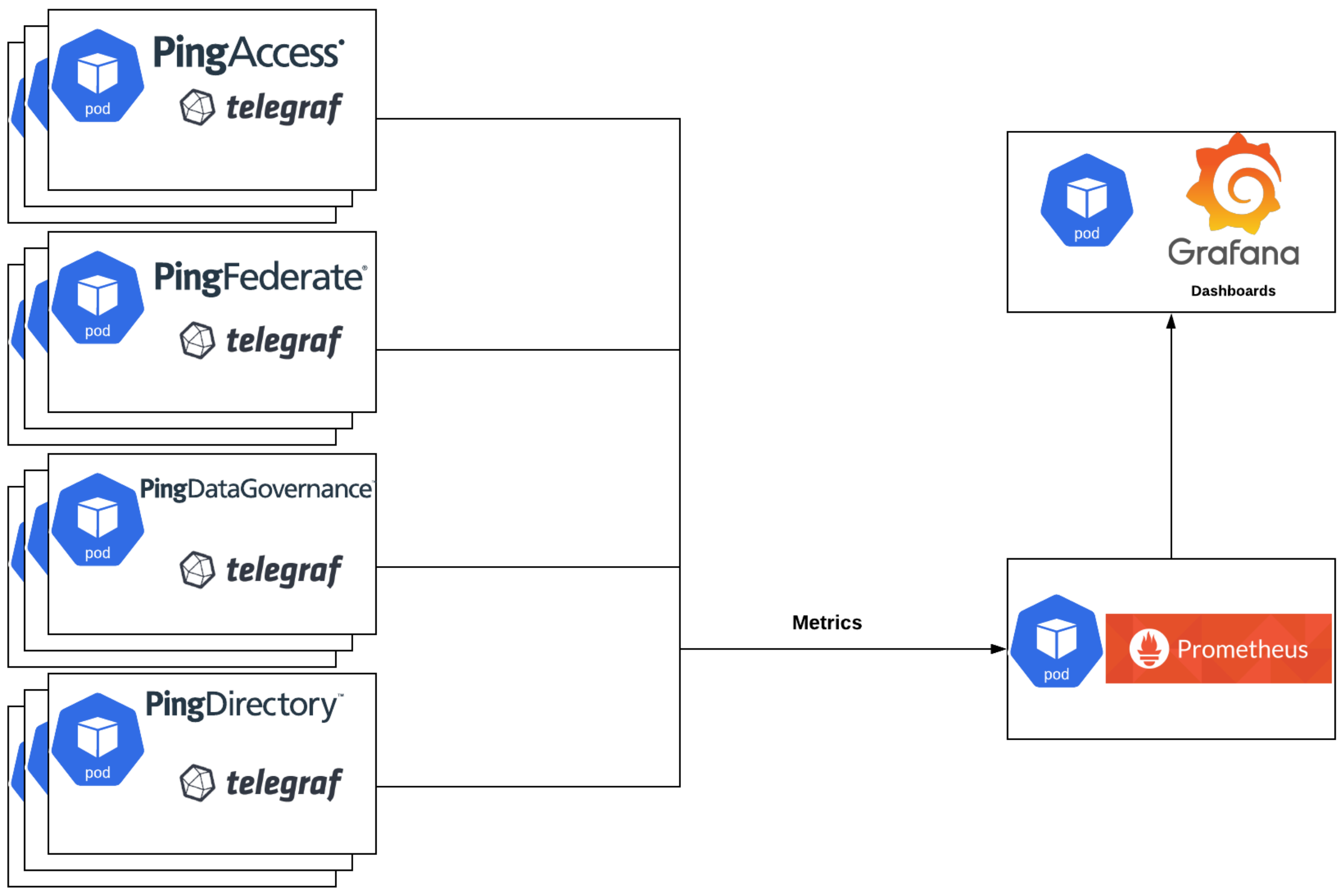
```
helm upgrade --install myping pingidentity/ping-devops \
    -f <HTTP link to yaml>
```

Uninstall

```
helm uninstall myping
```

Other Kubernetes Information

Deploy a Kubernetes Cluster Metrics Stack



This document demonstrates the process of deploying and using a sample open-source monitoring stack in a Kubernetes cluster.

Caution

The resulting environment is not production-ready. It is only intended to show how Ping software can produce metrics for consumption by a popular open-source monitoring system. This example stack is not maintained or directly supported by Ping.

Stack Components

Open Source Tools

- kube-prometheus-stack, which includes:
  - Prometheus - Metrics collection and storage
  - Grafana - Metrics visualization in Dashboards
  - telegraf-operator - Metrics exposure and formatting

Grafana Dashboard - JSON file to import for dashboard definition

Ping-provided values.yaml - Values relevant to exposing metrics for Ping Identity software

Prerequisites

- Familiarity with the prerequisites for the base Helm examples
- Working knowledge of Prometheus, Grafana, and Telegraf

Deploy the Stack

In the `pingidentity-devops-getting-started/30-helm/cluster-metrics directory` of this repository, edit the `01-prometheus-values.yaml` as needed. This file provides configurations beyond the default kube-prometheus-stack. In this sample deployment, the monitoring stack is granted read access to the entire cluster and is deployed into the `metrics` namespace.

Caution

Altering these settings or making the deployment production-ready is beyond the scope of this document. The full set of optional values can be found on the Github repository for the Prometheus chart.

There are numerous lines that have `##CHANGEME`. These lines should be considered for configuration options to meet your needs.

After updating the file, deploy the `kube-prometheus-stack`. The path to the configuration file assumes you are in the root folder of a local copy of the Getting Started repository:

```
kubectl create namespace metrics

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

helm upgrade --install metrics prometheus-community/kube-prometheus-stack -f 30-helm/cluster-metrics/01-prometheus-
values.yaml -n metrics --version 57.2.0
```

Deploy `telegraf-operator`:

```
helm repo add influxdata https://helm.influxdata.com/

helm upgrade --install telegraf influxdata/telegraf-operator -n metrics --version 1.3.11 -f 30-helm/cluster-metrics/
02-telegraf-values.yaml
```

Telegraf operator makes it very easy to add monitoring sidecars to your deployments. All you need to do is add annotations, which are shown in `30-helm/cluster-metrics/03-ping-with-metrics-values.yaml`

These values can be copied to your ping-devops `values.yaml` manually, or the file can be referenced at the end of your helm install command. For example:

`helm upgrade --install ping-metrics pingidentity/ping-devops -f my-values.yaml -f 30-helm/cluster-metrics/03-ping-with-metrics-values.yaml`

After the Ping software is healthy and producing metrics, there should be sidecars on Ping pods.

```
NAME                                                READY   STATUS
ping-metrics-pingaccess-admin-0                     1/1     Running
ping-metrics-pingaccess-engine-68464d8cc8-mhlsv     2/2     Running
ping-metrics-pingdataconsole-559786c98f-8wsrm       1/1     Running
ping-metrics-pingdirectory-0                        2/2     Running
ping-metrics-pingfederate-admin-64fdb4b975-2xdjl    1/1     Running
ping-metrics-pingfederate-engine-64c5f896c7-fn99v   2/2     Running
```

Note the `2/2` indicator for pods with sidecars.

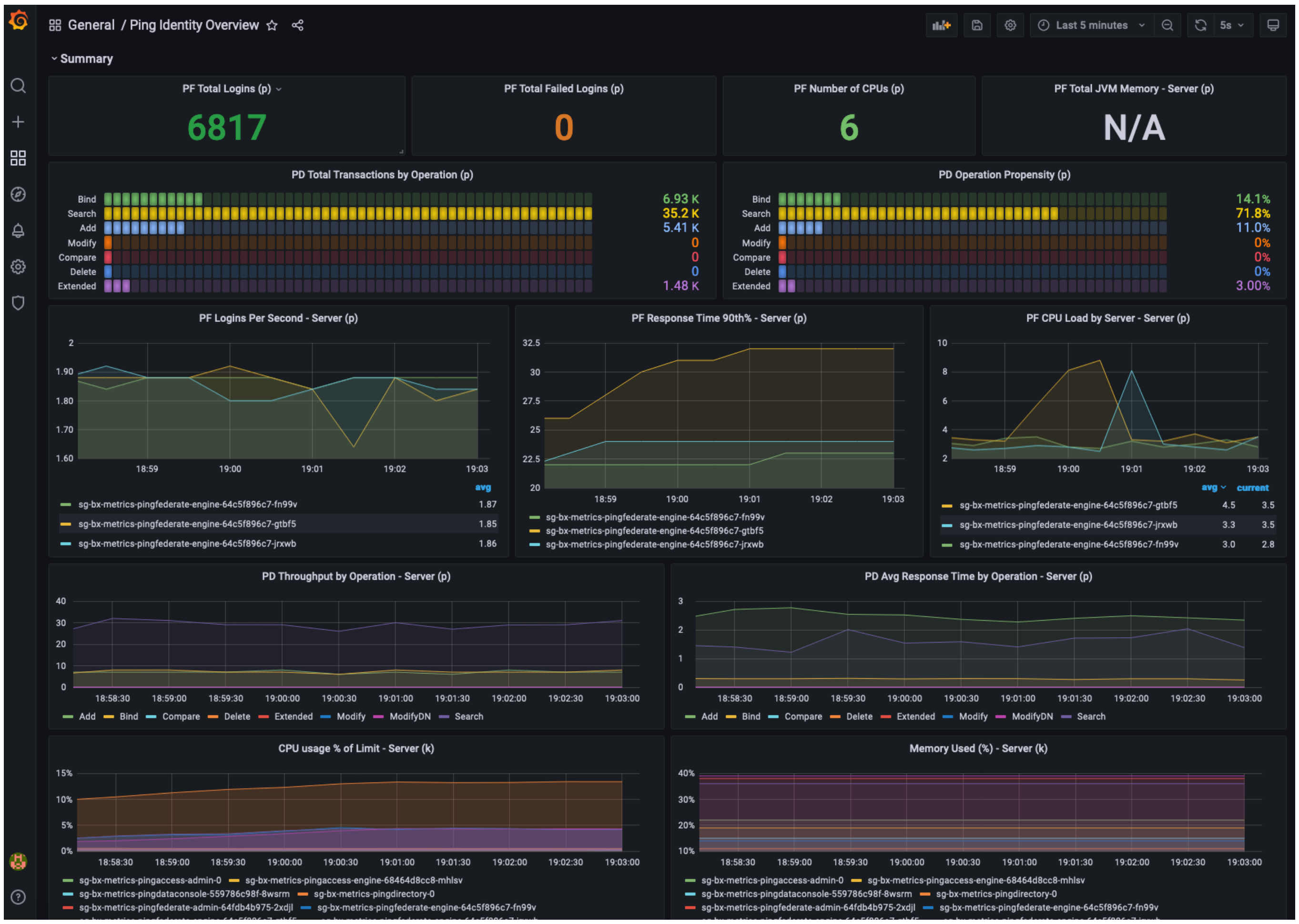
View Metrics

Browse to Grafana using the Ingress URL or by running a `kubectl port-forward` command. For example: `kubectl port-forward svc/metrics-grafana --namespace metrics 9000:80`.

In your browser, navigate to `http://localhost:9000` and log in with the user `admin` and the password set in `01-prometheus-values.yaml`.

Finally, import the `04-ping-overview-dashboard.json` using the **New** button at the top right of the Dashboard landing page in Grafana.

The `Ping Identity Overview` dashboard will have a dropdown for namespace at the top. Select the namespace running Ping products to see something similar to this example:


General / Ping Identity Overview
Last 5 minutes
5s
Summary
PF Total Logins (p)
6817
PF Total Failed Logins (p)
0
PF Number of CPUs (p)
6
PF Total JVM Memory - Server (p)
N/A
PD Total Transactions by Operation (p)
Bind 6.93 K
Search 35.2 K
Add 5.41 K
Modify 0
Compare 0
Delete 0
Extended 1.48 K
PD Operation Propensity (p)
Bind 14.1%
Search 71.8%
Add 11.0%
Modify 0%
Compare 0%
Delete 0%
Extended 3.00%
PF Logins Per Second - Server (p)
sg-bx-metrics-pingfederate-engine-64c5f896c7-fn99v 1.87
sg-bx-metrics-pingfederate-engine-64c5f896c7-gtbf5 1.85
sg-bx-metrics-pingfederate-engine-64c5f896c7-jrxwb 1.86
PF Response Time 90th% - Server (p)
PF CPU Load by Server - Server (p)
avg current
sg-bx-metrics-pingfederate-engine-64c5f896c7-gtbf5 4.5 3.5
sg-bx-metrics-pingfederate-engine-64c5f896c7-jrxwb 3.3 3.5
sg-bx-metrics-pingfederate-engine-64c5f896c7-fn99v 3.0 2.8
PD Throughput by Operation - Server (p)
PD Avg Response Time by Operation - Server (p)
Add Bind Compare Delete Extended Modify ModifyDN Search
CPU usage % of Limit - Server (k)
Memory Used (%) - Server (k)
sg-bx-metrics-pingaccess-admin-0
sg-bx-metrics-pingaccess-engine-68464d8cc8-mhlsv
sg-bx-metrics-pingdataconsole-559786c98f-8wsrm
sg-bx-metrics-pingdirectory-0
sg-bx-metrics-pingfederate-admin-64fdb4b975-2xdjl
sg-bx-metrics-pingfederate-engine-64c5f896c7-fn99v

Any of the panels can be edited, or new ones created to fit your needs.

HorizontalPodAutoscaler

If you use the `autoscaling/v2` API version, you can configure a HorizontalPodAutoscaler to scale based on a custom metric not built in to Kubernetes or any Kubernetes component. If you are using our Helm Charts, you can pass the custom metrics under `global.cluster.autoscalingMetricsTemplate`. The example code here will scale on a requests-per-second threshold of 10,000:

```
- type: Pods
  pods:
   metric:
     name: custom-metric
   target:
     type: AverageValue
     averageValue: 10000m
- type: Resource
  resource:
    name: cpu
    target:
      type: Utilization
      averageUtilization: 50
- type: Object
  object:
    metric:
      name: requests-per-second
    describedObject:
      apiVersion: networking.k8s.io/v1
      kind: Ingress
      name: main-route
    current:
      value: 10k
```

In addition, you can define the behaviors for scaling up and down under `global.cluster.autoscaling.behavior`.

```
scaleDown:
  stabilizationWindowSeconds: 300
  policies:
  - type: Percent
    value: 100
    periodSeconds: 15
scaleUp:
  stabilizationWindowSeconds: 0
  policies:
  - type: Percent
    value: 100
    periodSeconds: 15
  - type: Pods
    value: 4
    periodSeconds: 15
  selectPolicy: Max
```

For more information on custom HPA metrics please visit Kubernetes

Using a Utility Sidecar

Why Use a Sidecar

When running containerized software, each individual container should represent one process. This model allows containers to be minimal, more easily secured, and to be configured with the proper resource allocations accurately.

There are common situations where running commands and tools on a pod running Ping Identity software is useful. These situations include collecting a support archive, exporting data, or running a backup. However, because many of these processes might introduce unexpected contention for CPU and memory resources, executing such commands inside the container running the actual server process can be risky. The container for the product is sized for the server process without consideration for auxiliary processes that might be executed at the same time.

To avoid these issues, one practice is to use a utility sidecar for tools that need to run alongside the main server process. This sidecar runs as a separate container on the pod. However, in the Kubernetes model, all containers in the same pod can share a process namespace (if enabled), and can also be configured to share a persistent volume. This co-location allows any required processes to run in the sidecar without competing with the main server process for the same resources.

The major downside of running a utility sidecar is that it must always be running because new containers cannot be attached to existing pods. The sidecar can be configured with minimal memory and CPU resources, but will continue to run even when it is not actively in use.

Caution

You cannot remove a sidecar from a running StatefulSet without rolling all the pods.

How to Deploy a Sidecar

If you are using the Ping Identity Helm charts, you can update your custom values.yaml file to enable a sidecar for any product. For example:

```
pingdirectory:
  enabled: true
  workload:
    # Share process namespace for sidecar to get a view inside the main container
    shareProcessNamespace: true
  # Share /tmp so sidecar can see Java processes. Don't keep /tmp around between restarts though.
  volumes:
  - name: temp
    emptyDir: {}
  volumeMounts:
  - name: temp
    mountPath: /tmp
  # Backups, restores, and other CLI tools should be run from the sidecar to prevent interfering
  # with the main PingDirectory container process.
  utilitySidecar:
    enabled: true
```

These values will add a sidecar container using the same image as the main server container, configured with minimal resources and waiting in an endless loop. The generated yaml will look like the following example and could be applied directly outside of Helm:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app.kubernetes.io/name: pingdirectory
  name: sidecar-pingdirectory
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: pingdirectory
  serviceName: sidecar-pingdirectory-cluster
  template:
    metadata:
      labels:
        app.kubernetes.io/name: pingdirectory
    spec:
      containers:
      - envFrom:
        - secretRef:
            name: devops-secret
            optional: true
        image: pingidentity/pingdirectory:latest
        livenessProbe:
          exec:
            command:
            - /opt/liveness.sh
          failureThreshold: 4
          initialDelaySeconds: 30
          periodSeconds: 30
          successThreshold: 1
          timeoutSeconds: 5
        name: pingdirectory
        ports:
        - containerPort: 1443
          name: https
        - containerPort: 1389
          name: ldap
        - containerPort: 1636
          name: ldaps
        readinessProbe:
          exec:
            command:
            - /opt/readiness.sh
          failureThreshold: 4
          initialDelaySeconds: 30
          periodSeconds: 30
          successThreshold: 1
          timeoutSeconds: 5
        resources:
          limits:
            cpu: 2
            memory: 8Gi
          requests:
            cpu: 50m
            memory: 2Gi
        startupProbe:
          exec:
            command:
            - /opt/liveness.sh
          failureThreshold: 180
          periodSeconds: 10
          timeoutSeconds: 5
        volumeMounts:
        - mountPath: /tmp
          name: temp
        - mountPath: /opt/out
          name: out-dir
      - args:
        - -f
        - /dev/null
        command:
        - tail
        image: pingidentity/pingdirectory:latest
        name: utility-sidecar
        resources:
          limits:
            cpu: "1"
            memory: 2Gi
          requests:
            cpu: "0"
            memory: 128Mi
        volumeMounts:
        - mountPath: /opt/out
          name: out-dir
        - mountPath: /tmp
          name: temp
      securityContext:
        fsGroup: 0
        runAsGroup: 0
        runAsUser: 9031
      shareProcessNamespace: true
      terminationGracePeriodSeconds: 300
      volumes:
      - emptyDir: {}
        name: temp
      - name: out-dir
        persistentVolumeClaim:
          claimName: out-dir
  volumeClaimTemplates:
  - metadata:
      name: out-dir
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 8Gi
      storageClassName: null
```

Deploying PingFederate Across Multiple Kubernetes Clusters

This section will discuss deploying a single PingFederate cluster that spans across multiple Kubernetes clusters.

Deploying PingFederate in multiple regions should not imply that spanning a single PingFederate cluster across multiple Kubernetes clusters is necessary or optimal. This scenario makes sense when you have:

- Traffic that can cross between regions at any time. For example, *west* and *east* and users may be routed to either location.
- Configuration that needs to be the same in multiple regions **and** there is no reliable automation to ensure this is the case

If all configuration changes are delivered via a pipeline, and traffic will not cross regions, having separate PingFederate clusters can work.

Note

The set of pre-requisites required for AWS Kubernetes multi-clustering to be successful is found here.

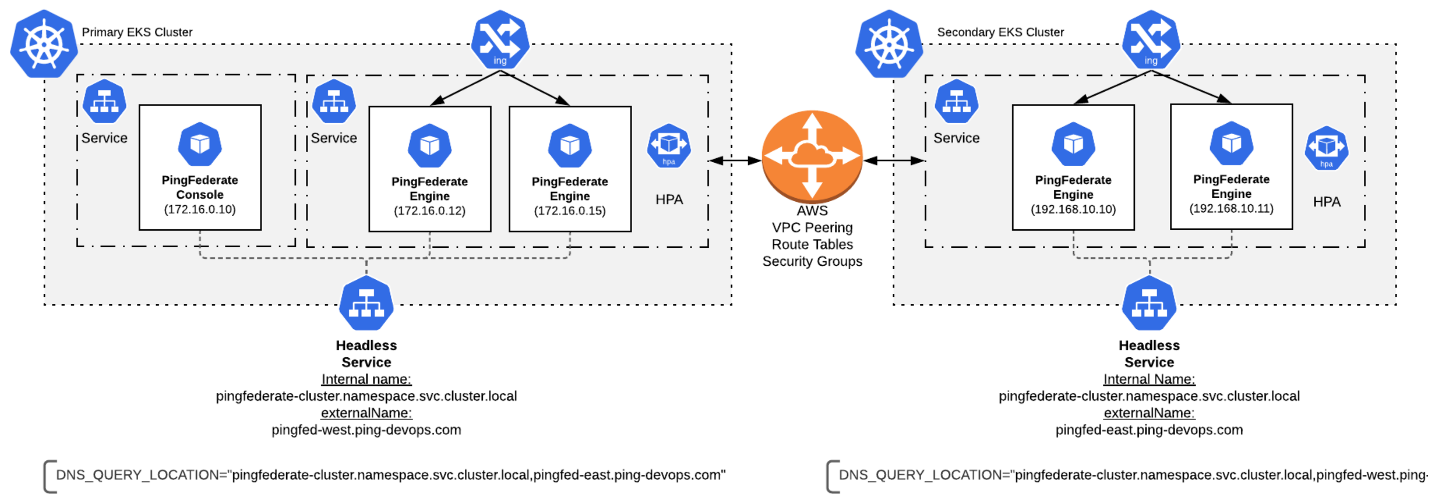
Static engine lists, which may be used to extend traditional, on-premise PingFederate clusters is out of scope in this document.

Prerequisites

- Two Kubernetes clusters created with the following requirements:
  - VPC IPs selected from RFC1918 CIDR blocks
  - The two cluster VPCs peered together
  - All appropriate routing tables modified in both clusters to send cross cluster traffic to the VPC peer connection
  - Security groups on both clusters to allow traffic for ports 7600 and 7700 in both directions
  - Verification that a pod in one cluster can connect to a pod in the second cluster on ports 7600 and 7700 (directly to the back-end IP assigned to the pod, not through an exposed service)
  - externalDNS enabled

    See example "AWS configuration" instructions here
- Helm client installed

Overview


Primary EKS Cluster
Service
PingFederate Console (172.16.0.10)
Service
PingFederate Engine (172.16.0.12)
PingFederate Engine (172.16.0.15)
HPA
AWS
VPC Peering
Route Tables
Security Groups
Secondary EKS Cluster
Service
PingFederate Engine (192.168.10.10)
PingFederate Engine (192.168.10.11)
HPA
Headless Service
Internal name:
pingfederate-cluster.namespace.svc.cluster.local
externalName:
pingfed-west.ping-devops.com
Headless Service
Internal Name:
pingfederate-cluster.namespace.svc.cluster.local
externalName:
pingfed-east.ping-devops.com
DNS_QUERY_LOCATION="pingfederate-cluster.namespace.svc.cluster.local,pingfed-east.ping-devops.com"
DNS_QUERY_LOCATION="pingfederate-cluster.namespace.svc.cluster.local,pingfed-west.ping-

The PingFederate Docker image default `instance/server/default/conf/tcp.xml` file points to DNS_PING. After you have two peered Kubernetes clusters, spanning a PingFederate cluster across the two becomes easy. A single PingFederate cluster uses DNS_PING to query a local headless service. In this example we use externalDNS to give an externalName to the headless service. The `externalDNS` feature from the Kubernetes special interest group (SIG) creates a corresponding record on AWS Route53 and constantly updates it with container IP addresses of the backend PF engines.

> **Warning**
>
> If you are unable to use externalDNS, another way to expose the headless service across clusters is needed. HAProxy may be a viable option to explore and is beyond the scope of this document.

What You Will Do

- Clone the example files from the `getting-started` Repository
- Edit the externalName of the pingfederate-cluster service and the DNS_QUERY_LOCATION variable as needed

  Search the files for `# CHANGEME` comments to find where these changes need to be made.
- Deploy the clusters
- Cleanup

Example deployment

Clone this `getting-started` repository to get the Helm values yaml for the exercise. The files are located under the folder `30-helm/multi-region/pingfederate`.

After cloning:

1. Update the first uncommented line under any `## CHANGEME` comment in the files. The changes will indicate the Kubernetes namespace and the externalName of the pingfederate-cluster service.
2. Deploy the first cluster (the example here uses kubectx to set the kubectl context))

```
kubectx west
helm upgrade --install example pingidentity/ping-devops -f base.yaml -f 01-layer-west.yaml
```

3. Deploy the second cluster

```
kubectx east
helm upgrade --install example pingidentity/ping-devops -f base.yaml -f 01-layer-east.yaml
```

4. Switch back to the first cluster, and simulate a regional failure by removing the PingFederate cluster entirely:

```
kubectx west
helm uninstall example
```

5. Switch back to the second cluster and switch failover to active

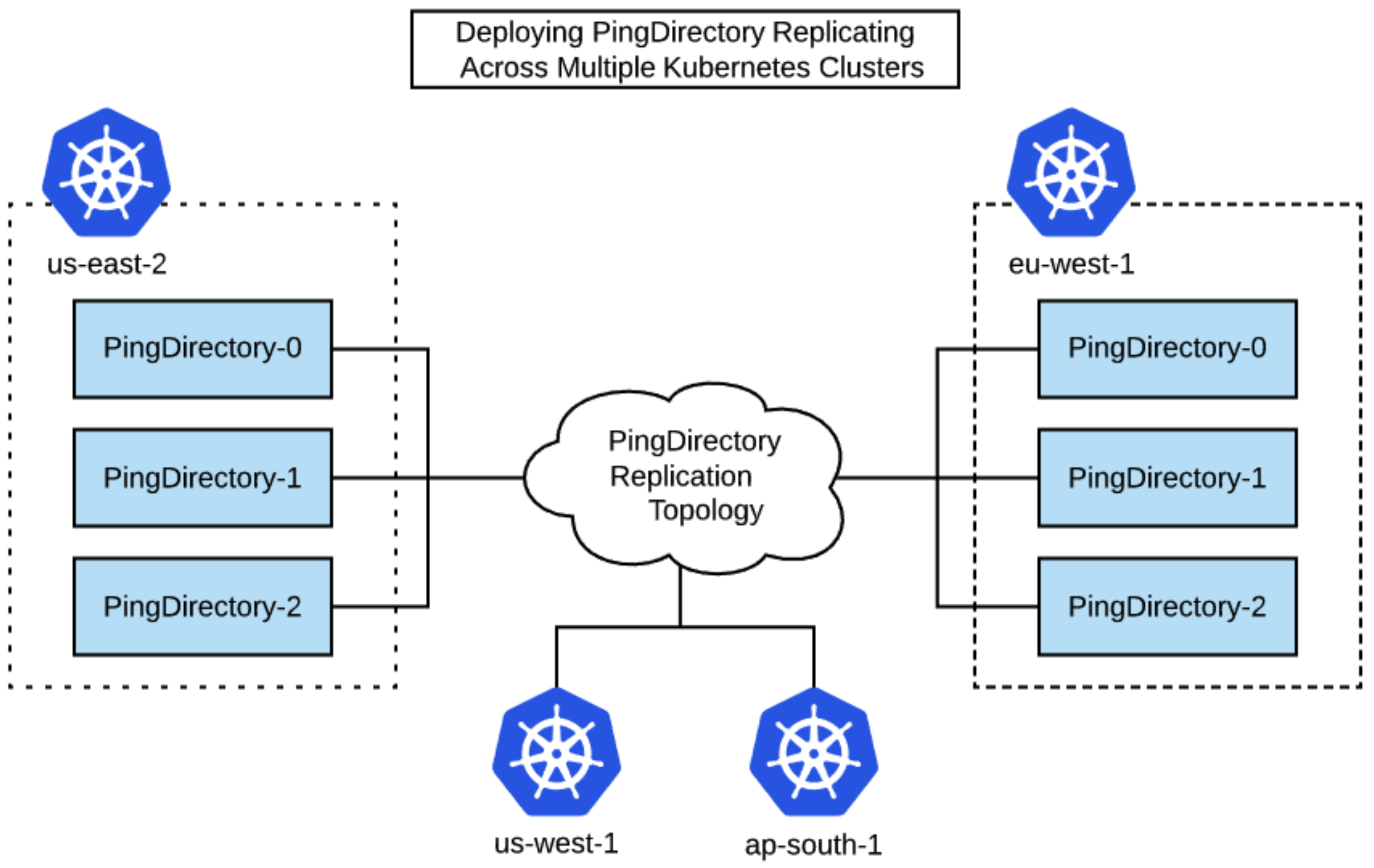
```
kubectx east
helm upgrade --install example pingidentity/ping-devops -f base.yaml -f 02-layer-east.yaml
```

Cleanup

```
kubectx east
helm uninstall example
kubectx west
helm uninstall example
```

Deploy PingDirectory Across Multiple Kubernetes Clusters

This example shows how to deploy PingDirectory containers that replicate across multiple Kubernetes clusters.



Overview

Implementing a replicated PingDirectory topology across multiple Kubernetes clusters is desired for highly available active/active deployments as well as active/partial-active scenarios where a hot backup is expected.

PingDirectory Docker images abstract much of the complexity of replication initialization scripts, even across clusters. With this simplification, the focus shifts to providing accessible DNS hostnames across clusters and environment variables to build ordinal hostnames for each PingDirectory instance.

What You Will Do

1. PingDirectory Host Naming - Set the variables needed to create your hostnames
2. Cluster Startup Walkthrough - A description of what happens when a PingDirectory cluster starts
3. Deploy the Helm Example - Deploy an example set of servers across multiple Kubernetes clusters

Details within each Kubernetes cluster are hidden from outside the cluster, which means that external access to each pod in the cluster is required. The PingDirectory images will set up access to each of the pods using load balancers from an external host to allow each pod to communicate over the LDAP and replication protocols.

PingDirectory Host Naming

The most important aspect of a successful PingDirectory cross-cluster deployment is assigning accessible and logical DNS hostnames. The rules for this setup include:

1. Each PingDirectory pod needs its own hostname available in DNS
2. Hostnames need to include the ordinal representing the instance in the statefulset
3. All hostnames must be accessible to all directory instances

These rules still leave plenty of room for flexibility, particularly when accounting for the cluster-native DNS names Kubernetes creates.

Single Cluster Example with Multiple Namespaces

For example, you can simulate a multi-cluster environment in a single Kubernetes cluster by using separate namespaces and creating a separate ClusterIP service for each directory environment. You would end up with something similar to this example:

Primary Cluster

Pod	Service Name	Namespace	Hostname
pingdirectory-0	pingdirectory-0	primary	pingdirectory-0.primary
pingdirectory-1	pingdirectory-1	primary	pingdirectory-1.primary
pingdirectory-2	pingdirectory-2	primary	pingdirectory-2.primary

Secondary Cluster

Pod	Service Name	Namespace	Hostname
pingdirectory-0	pingdirectory-0	secondary	pingdirectory-0.secondary
pingdirectory-1	pingdirectory-1	secondary	pingdirectory-1.secondary
pingdirectory-2	pingdirectory-2	secondary	pingdirectory-2.secondary

Production Example with External DNS Names

An example from a production environment with external hostnames might appear more like this example:

us-west cluster

Pod	Service Name	DNS/Hostname
pingdirectory-0	pingdirectory-0	pingdirectory-0-us-west.ping-devops.com
pingdirectory-1	pingdirectory-1	pingdirectory-1-us-west.ping-devops.com
pingdirectory-2	pingdirectory-2	pingdirectory-2-us-west.ping-devops.com

us-east cluster

Pod	Service Name	DNS/Hostname
pingdirectory-0	pingdirectory-0	pingdirectory-0-us-east.ping-devops.com
pingdirectory-1	pingdirectory-1	pingdirectory-1-us-east.ping-devops.com
pingdirectory-2	pingdirectory-2	pingdirectory-2-us-east.ping-devops.com

Variables to Create Hostnames

To provide flexibility on how each PingDirectory instance will find other instances, a full DNS hostname is broken into multiple variables.

Variable	Description
`K8S_POD_HOSTNAME_PREFIX`	The string used as the prefix for all host names. Defaults to the name of the `StatefulSet`.
`K8S_POD_HOSTNAME_SUFFIX`	The string used as the suffix for all pod host names. Defaults to `K8S_CLUSTER`.
`K8S_SEED_HOSTNAME_SUFFIX`	The string used as the suffix for all seed host names. Defaults to `K8S_SEED_CLUSTER` (discussed later).

With these variables, a full hostname is created in this manner:

```
${K8S_POD_HOSTNAME_PREFIX}<instance-ordinal>${K8S_SEED_HOSTNAME_SUFFIX}
```

Previous Hostname Example Breakdown

Hostname	K8S_POD_HOSTNAME_PREFIX	K8S_POD_HOSTNAME_SUFFIX	K8S_SEED_HOSTNAME_SUFFIX
pingdirectory-0.primary	`pingdirectory-`	`.primary`	`.primary`
pingdirectory-2-us-west.ping-devops.com	`pingdirectory-`	`-us-west.ping-devops.com`	`-us-west.ping-devops.com`

Environment Variables

Variable	Required	Description
`K8S_CLUSTERS`	**	The total list of Kubernetes clusters to which the StatefulSet will replicate.
`K8S_CLUSTER`	**	The Kubernetes cluster to which the StatefulSet will be deployed.
`K8S_SEED_CLUSTER`	**	The Kubernetes cluster to which the seed server is deployed.
`K8S_NUM_REPLICAS`		The number of replicas that make up the StatefulSet.
`K8S_POD_HOSTNAME_PREFIX`		The string used as the prefix for all host names. Defaults to `StatefulSet`.
`K8S_POD_HOSTNAME_SUFFIX`		The string used as the suffix for all pod host names. Defaults to `K8S_CLUSTER`.
`K8S_SEED_HOSTNAME_PREFIX`		The string used as the suffix for all seed host names. Defaults to `K8S_SEED_CLUSTER`.
`K8S_INCREMENT_PORTS`		`true` or `false`. If `true`, the port for each pod will be incremented by 1.

An example set of the YAML configuration for these environment variables is as follows:

Primary

```
K8S_STATEFUL_SET_NAME=pingdirectory
K8S_STATEFUL_SET_SERVICE_NAME=pingdirectory

K8S_CLUSTERS=us-east-2 eu-west-1
K8S_CLUSTER=us-east-2
K8S_SEED_CLUSTER=us-east-2
K8S_NUM_REPLICAS=3

K8S_POD_HOSTNAME_PREFIX=pd-
K8S_POD_HOSTNAME_SUFFIX=.us-cluster.ping-devops.com
K8S_SEED_HOSTNAME_SUFFIX=.us-cluster.ping-devops.com

K8S_INCREMENT_PORTS=true
LDAPS_PORT=8600
REPLICATION_PORT=8700
```

These environment variable settings map out like this:

Seed	Instance	Hostname	LDAP	REPL
	CLUSTER:us-east-2			
**	pingdirectory-0.us-east-2	pd-0.us-cluster.ping-devops.com	8600	8700
	pingdirectory-1.us-east-2	pd-1.us-cluster.ping-devops.com	8601	8701
	pingdirectory-2.us-east-2	pd-2.us-cluster.ping-devops.com	8602	8702

Secondary

```
K8S_STATEFUL_SET_NAME=pingdirectory
K8S_STATEFUL_SET_SERVICE_NAME=pingdirectory

K8S_CLUSTERS=us-east-2 eu-west-1
K8S_CLUSTER=eu-west-1
K8S_SEED_CLUSTER=us-east-2
K8S_NUM_REPLICAS=3

K8S_POD_HOSTNAME_PREFIX=pd-
K8S_POD_HOSTNAME_SUFFIX=.eu-cluster.ping-devops.com
K8S_SEED_HOSTNAME_SUFFIX=.us-cluster.ping-devops.com

K8S_INCREMENT_PORTS=true
LDAPS_PORT=8600
REPLICATION_PORT=8700
```

Seed	Instance	Hostname	LDAP	REPL
	CLUSTER:eu-west-1			
	pingdirectory-0.eu-west-1	pd-0.eu-cluster.ping-devops.com	8600	8700
	pingdirectory-1.eu-west-1	pd-1.eu-cluster.ping-devops.com	8601	8701
	pingdirectory-2.eu-west-1	pd-2.eu-cluster.ping-devops.com	8602	8702

Cluster Startup Walkthrough

By now you can see that there are *many* variables that have been described. These variables exist to provide flexibility to accommodate various infrastructure constraints. For example, in some environments you cannot use the same port for each instance, so we must accommodate incrementing ports.

Continuing, it is helpful to know what happens when a cluster starts in order to understand why the initial creation of a cluster must be very prescriptive.

1. The first pod must start on its own and become healthy. This startup is critical to prevent replication islands. The very first time the very first pod starts, we call it "GENESIS". All other pods are dependent on this `SEED_POD` in the `SEED_CLUSTER` starting correctly by itself. The entire purpose of defining `SEED_POD` and `SEED_CLUSTER` variables is avoid multiple genesis scenarios.

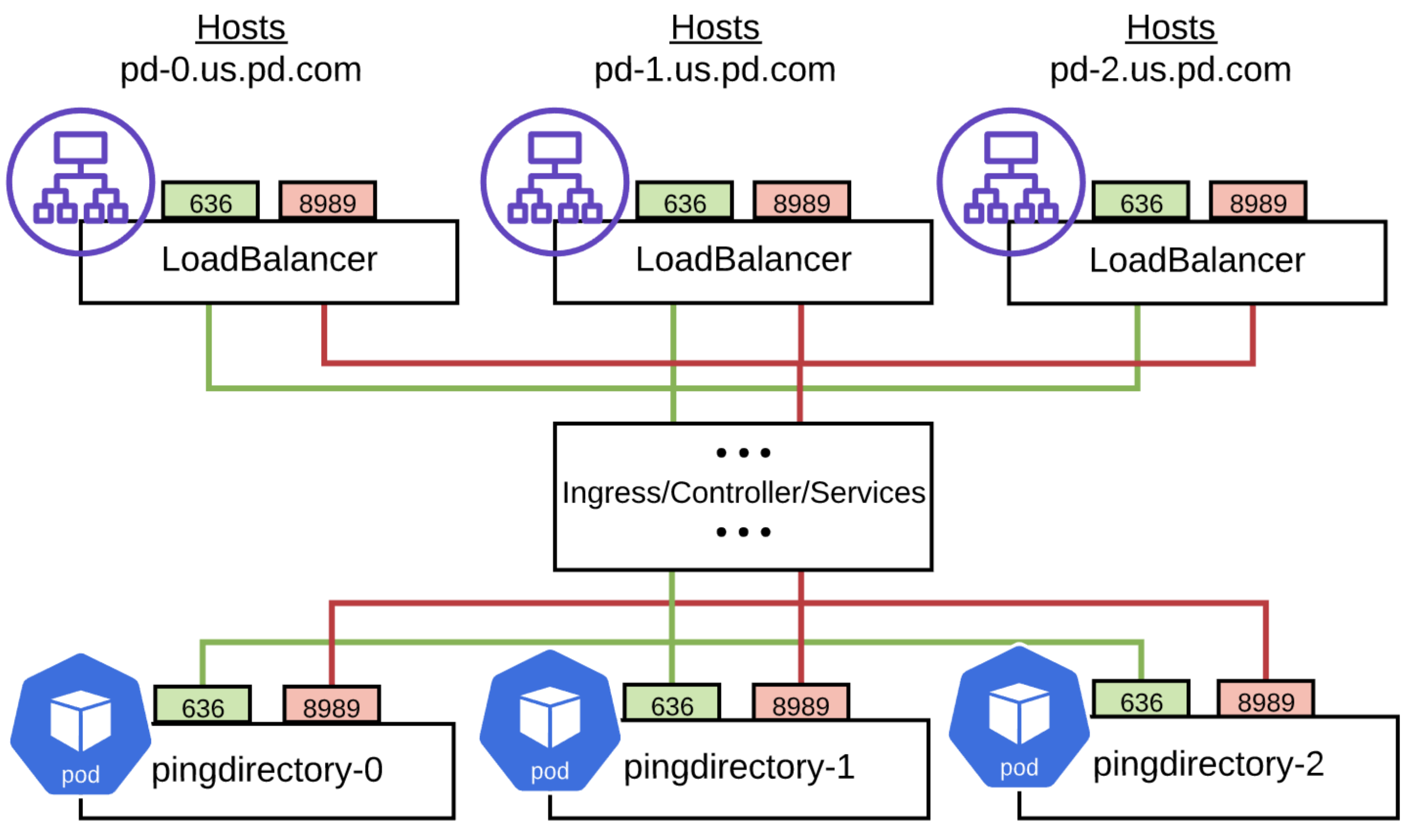
2. After the first pod is healthy, it begins querying DNS for combinations of hostnames at their LDAPS port to find another PingDirectory instance.

In our first cluster, this would be the hostname of pingdirectory-1, but it could also be pingdirectory-0 of another cluster. After the query returns successful, creation of the replication topology automatically begins. From this point onward, the order in which instances start is less important.

Deploying Across Multiple Regions with Multiple Load Balancers

If infrastructure constraints prevent you from using Peered Clusters, an alternate option is to deploy with a separate LoadBalancer service for each PingDirectory pod.

The following diagram shows how you can use muliple load balancers.



Advantages:

- Use the same well-known port, such as 1636/8989
- Separate IP addresses per instance

Disadvantages:

- DNS management
  - Separate hostname required per pod

This method is supported in our Helm charts with the `pingdirectory.services.loadBalancerServicePerPod` field.

Deploy the Helm Example

Clone this `getting-started` repository to get the Helm values .yaml files for the exercise. There are two multi-region examples. For peered clusters, the example files are under the folder `30-helm/multi-region/pingdirectory`. For deploying with multiple load balancers, the example files are under the folder `30-helm/multi-region/pingdirectory-loadbalancer-per-pod`. After cloning:

1. Modify any external hostnames in the sample files as necessary - see the lines under `## CHANGEME` comments.
2. Deploy the first set of pods (the example here uses kubectx to set the kubectl context).

```
kubectx west
helm upgrade --install example pingidentity/ping-devops -f 01-west.yaml
```

3. Wait for the example-pingdirectory pods to be running and ready.
4. Deploy the second set of pods.

```
kubectx east
helm upgrade --install example pingidentity/ping-devops -f 02-east.yaml
```

5. Wait for all example-pingdirectory pods to be running and ready.
6. Verify that pods are replicating.

```
kubectx west
kubectl exec example-pingdirectory-0 -- dsreplication status --showAll
```

Cleanup

```
kubectx west
helm uninstall example
kubectl delete pvc --selector=app.kubernetes.io/instance=example
kubectx east
helm uninstall example
kubectl delete pvc --selector=app.kubernetes.io/instance=example
```

Deploy PingDirectoryProxy and PingDirectory with automatic backend discovery

Since version 8.3 of PingDirectoryProxy, proxy servers can use automatic server discovery to determine the backend PingDirectory servers, rather than adding those servers individually to the configuration. This page describes how to use this feature with the PingDirectory and PingDirectoryProxy Docker images and the ping-devops Helm chart

The directory and proxy Docker images added support for this feature as of the 2310 release, and the ping-devops Helm chart added support in release `0.9.20`.

Configuring the proxy instance to join the directory topology

The first step of enabling automatic server discovery is to have the proxy server(s) join the topology of replicating directory servers. To enable this, the proxy Docker image supports the following variables:

- `JOIN_PD_TOPOLOGY` : Set to `true` to add the proxy instance to a directory topology
- `PINGDIRECTORY_HOSTNAME` : The hostname of the directory server to connect with when joining the topology
- `PINGDIRECTORY_LDAPS_PORT` : The LDAPS port of the directory server to connect with when joining the topology

If all three of these variables are set, the proxy server will join the designated topology after the server starts up.

Waiting for the directory topology to be ready before starting

The designated directory server must be running for the proxy server to join the topology. To ensure directory is running before proxy attempts to join, a `wait-for` can be used.

For example, using the ping-devops Helm chart, the following values yaml instructs proxy to wait until the second `pingdirectory` pod is running before starting and attempting to join the topology. "releasename" can be replaced with the Helm release name.

```
initContainers:
  wait-for-pd:
    name: wait-for-pd
    image: pingidentity/pingtoolkit:2309
    command: ['sh', '-c', 'echo "Waiting for PingDirectory..." && wait-for releasename-
pingdirectory-1.releasename-pingdirectory-cluster:1636 -t 300 -- echo "PingDirectory running"']

pingdirectory:
  container:
    replicaCount: 2
  enabled: true
  envs:
    SERVER_PROFILE_URL: https://github.com/pingidentity/pingidentity-server-profiles.git
    SERVER_PROFILE_PATH: baseline/pingdirectory
    LOAD_BALANCING_ALGORITHM_NAMES:dc_example_dc_com-fewest-operations;dc_example_dc_com-failover

pingdirectoryproxy:
  includeInitContainers:
  - wait-for-pd
  container:
    replicaCount: 1
  enabled: true
  envs:
    SERVER_PROFILE_URL: https://github.com/pingidentity/pingidentity-server-profiles.git
    SERVER_PROFILE_PATH: pingdirectoryproxy-automatic-server-discovery
    JOIN_PD_TOPOLOGY: "true"
    PINGDIRECTORY_HOSTNAME: releasename-pingdirectory-0.releasename-pingdirectory-cluster
    PINGDIRECTORY_LDAPS_PORT: "1636"
```

Configuring automatic server discovery on proxy using a server profile

The proxy server must also be configured via `dsconfig` to enable automatic server discovery. For an example, see the automatic server discovery server profile.

Setting load balancing algorithm names on the directory instances

To associate directory servers with the load balancing algorithms configured on the proxy server, the `load-balancing-algorithm-name` property must be set. This can be done with the `LOAD_BALANCING_ALGORITHM_NAMES` environment variable in the directory Docker image. When using multiple algorithm names, separate them with a `;`. See the above yaml snippet for an example.

Removing the proxy server from the topology on pod shutdown

By default the proxy server will rejoin the topology automatically on restarts. In the `ping-devops` Helm chart, proxy does not use a persistent volume, so it will fully restart and rejoin the topology during each startup.

Another option, which allows for scaling down the number of proxy servers, is adding a `preStop` hook to remove the proxy server from the topology. In general this can cause slowness because it will run whenever a pod stops, but it ensures that scaling down the number of proxies does not leave outdated servers in the topology registry. For example:

```
pingdirectoryproxy:
  container:
    # Add the preStop hook to run the remove-defunct-server tool
    lifecycle:
      preStop:
        exec:
          command:
          - /opt/staging/hooks/90-shutdown-sequence.sh
```

Automatic server discovery when directory and proxy pods are split across multiple clusters

When deploying directory pods across multiple Kubernetes clusters, some additional configuration needs to be added to allow proxy to join the directory topology and enable automatic server discovery.

Essentially, the proxy workload will need to have similar variables and network access as the directory workload (see the directory multi-cluster doc linked above). In addition, proxy will need the right variables set to join the topology and the right wait-for logic to wait for the other servers to be ready before starting and joining the topology.

See here for a complete Helm example.

Deploy a PingAccess Cluster with PingIdentity Helm Charts Without a Server Profile

Caution

The instructions in this document are for testing and learning and are not intended for use in production.

Purpose

Create and deploy a PingAccess Cluster using PingIdentity Helm Charts, without having to create a custom server profile. This process will allow you to quickly bring up the PingAccess UI and conduct any tests you need.

Prerequisites

- kubectl
- Access to a Kubernetes cluster

Steps

1. Confirm that your kuberenetes context and namespace are set correctly

   ```
   # Display kuberenetes context
   kubectx

   # Display namespace
   kubens -c
   ```

   If these values are not set or are incorrect, you can set them with the following commands. If you do not yet have a namespace, or do not have access to a kubernetes cluster, refer to Deploy an Example Stack.

   ```
   # Display kuberenetes context
   kubectx <context>

   # Display namespace
   kubens <namespace>
   ```

2. Confirm that there are no conflicting persistent volumes.

   ```
   #List any persistent volumes
   kubectl get pvc
   ```

   If you see a persistent volume with a name that resembles `out-dir-demo-pingaccess-admin-0`, then delete it before deploying your cluster.

   ```
   #Delete name_of_pvc persistent volume
   kubectl delete pvc out-dir-demo-pingaccess-admin-0
   ```

   Warning

   This functionality has only been implemented for Sprint tags of 2211 or later. Therefore, it will not work for all earlier tags.

3. Create a YAML file similar to the one shown here. Make sure to replace `insert domain name here` with your domain name.

```
global:
envs:
    PING_IDENTITY_ACCEPT_EULA: "YES"
ingress:
    enabled: true
    addReleaseNameToHost: prepend
    defaultDomain: "insert domain name here"
    defaultTlsSecret:
    annotations:
        nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
        kubernetes.io/ingress.class: "nginx-public"

#############################################################
# pingaccess-admin values
#############################################################
pingaccess-admin:
enabled: true
privateCert:
    generate: true
envs:
    PING_IDENTITY_PASSWORD: "2FederateM0re!"

#############################################################
# pingaccess-engine values
#############################################################
pingaccess-engine:
enabled: true
container:
    replicaCount: 1
envs:
    PING_IDENTITY_PASSWORD: "2FederateM0re!"
```

4. Create the default PingAccess cluster. Make sure that you fill in the "PATH" to your new values.yaml file. This deployment may take a few minutes to become healthy.

```
helm upgrade --install demo pingidentity/ping-devops -f <path-to-yaml>/values.yaml
```

5. To display the status of the deployed components, you can use k9s or issue the corresponding commands shown here:

   - Display the services (endpoints for connecting) by running `kubectl get service --selector=app.kubernetes.io/instance=demo`

```
NAME                             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
demo-pingaccess-admin            ClusterIP   172.20.221.233  <none>        9090/TCP,9000/TCP
37s
demo-pingaccess-admin-cluster    ClusterIP   None            <none>        <none>
37s
demo-pingaccess-engine           ClusterIP   172.20.126.86   <none>        3000/TCP
37s
```

◦ To view the pods, run `kubectl get pods --selector=app.kubernetes.io/instance=demo` - you will need to run this at intervals until all pods have started (**Running** status):

```
NAME                                    READY   STATUS        RESTARTS   AGE
demo-pingaccess-admin-0                 1/1     Running   0          28m
demo-pingaccess-engine-6b977b9498-298jw   1/1     Running   0          28m
```

◦ To see the ingresses you will use to access the product, run `kubectl get ingress`. If the ingress controller is configured properly, the URL you will see under demo-pingaccess-admin HOST (`demo-pingaccess-admin.<domain-name>`) will be the URL you use to access the PingAccess management console.

```
NAME                     CLASS    HOSTS
ADDRESS                                                                 PORTS     AGE
demo-pingaccess-admin    <none>   demo-pingaccess-admin.<domain-name>
adab69408130011eab1cd028479a4fe3-532fea1b3272797d.elb.us-east-2.amazonaws.com   80, 443   2m1s
demo-pingaccess-engine   <none>   demo-pingaccess-engine.<domain-name>
adab69408130011eab1cd028479a4fe3-532fea1b3272797d.elb.us-east-2.amazonaws.com   80, 443   2m1s
```

◦ To see everything tied to the helm release run `kubectl get all --selector=app.kubernetes.io/instance=demo` :

```
NAME                                        READY   STATUS    RESTARTS   AGE
pod/demo-pingaccess-admin-0                 1/1     Running   0          29m
pod/demo-pingaccess-engine-6b977b9498-298jw   1/1     Running   0          29m

NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)             AGE
service/demo-pingaccess-admin         ClusterIP   172.20.221.233   <none>        9090/TCP,
9000/TCP   29m
service/demo-pingaccess-admin-cluster   ClusterIP   None             <none>
<none>              29m
service/demo-pingaccess-engine        ClusterIP   172.20.126.86    <none>        3000/
TCP            29m

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/demo-pingaccess-engine   1/1     1            1           29m

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/demo-pingaccess-engine-6b977b9498   1         1         1       29m

NAME                                     READY   AGE
statefulset.apps/demo-pingaccess-admin   1/1     29m
```

◦ To view logs, look at the logs for the deployment of the product in question. For example:

```
#Admin pod logs
kubectl logs demo-pingaccess-admin-0

#Engine pod logs
kubectl logs demo-pingaccess-engine-6b977b9498
```

6. Below are the credentials and URL to sign on to the PingAccess management console after the cluster is up and healthy.

> **Note**
>
> This example uses self-signed certificates that will have to be accepted in your browser or added to your keystore.

With the ingress in place, you can access the product at the URL seen below, using the domain-name you set in your values.yaml file.

Product	Connection Details
PingAccess	◦ URL: https://demo-pingaccess-admin.(domain-name) ◦ Username: Administrator ◦ Password: 2FederateM0re!

7. When you are finished, you can remove the demonstration components by running the uninstall command for helm:

`helm uninstall demo`

8. Finally make sure to prune the persistent volume created in the deployment of your PingAccess cluster, by running the delete pvc command for kubectl:

```
#Delete name_of_pvc persistent volume
kubectl delete pvc out-dir-demo-pingaccess-admin-0
```

Environment considerations

Network File System (NFS) constraints

All PingData products use the `manage-extension` tool for installing extensions. Due to how the tool operates, it can lead to issues when the deployment involves NFS.

If your deployment uses NFS, rather than using the `manage-extensions` tool, unzip the extension manually and add it to the appropriate directory.

The following example script, called `181-install-extensions.sh.post`, loops through the extensions to unzip and then removes them from the server profile.

```
#!/usr/bin/env sh
# Loop through extensions to unzip, then remove them from the server profile
PROFILE_EXTENSIONS_DIR="${PD_PROFILE}/server-sdk-extensions"
if test -d "${PROFILE_EXTENSIONS_DIR}"; then
  find "${PROFILE_EXTENSIONS_DIR}" -type f -name '*.zip' -print > /tmp/_extensionList
  while IFS= read -r _extensionFile; do
      echo "Installing extension: ${_extensionFile}"
      unzip -q "${_extensionFile}" -d /opt/out/instance/extensions/
      rm "${_extensionFile}"
  done < /tmp/_extensionList
  rm -f /tmp/_extensionList
fi
```

PingDirectory inotify watch limit requirement

When using inotify with PingDirectory, you must set a watch limit on the host system. This value cannot be set from a docker container, and the value read within a docker container is always the host value.

For more information, see Set file system event monitoring (inotify) in the PingDirectory documentation.

Operating Patterns

This page discusses how to have a successful first day and beyond.

After you are comfortable with the deployment examples in the getting-started repository, you can shift your focus to managing ongoing operations of the products that are relevant to you. Since it is not feasible to cover every operating scenario, this section will focus on guidance to identify an operating pattern suitable for your organization.

The PingFederate application is used as an example of performing this assessment with some example patterns.

PingFederate Configuration Management

PingFederate has a variety of operating patterns. These patterns typically involve a trade-off between ease of implementation and mitigation of deployment risks.

To simplify the moving parts, PingFederate configuration can be categorized into three patterns:

1) Infrastructure Configuration

Examples of managed components:

- Resource allocation (CPU/Memory/Storage)
- Client Ingress (Access and Hostnames)
- Image Version
- Exposed Ports
- Environment Variable Definitions

- Secrets Definitions

Orchestration

- These items are defined in the release values.yaml file and any changes here triggers an update.

2) Server Configuration

This pattern can be oversimplified to *everything outside of* the `/instance/server/default/data` folder or `/instance/bulk-config/data.json`.

Examples of managed components:

- `*.properties` files
- Integration Kits
- HTML templates
- log formatting (log4j2.xml)

Orchestration

These items are stored in the Server Profile and any change *should* trigger an update. It is up to the implementer to ensure that happens. Triggering an update can be done by adding a non-functional variable in `values.yaml` to track the current profile "version". Example: `SERVER_PROFILE_VERSION: v1.1`

3) Application Configuration (App Config)

Application configuration can be managed via any combination of the following, according to customer internal configuration management requirements:

- The PingFederate Terraform provider - a way to declare the "end state" of configuration of a PingFederate server. Terraform can be used to identify and correct configuration drift in an environment ad-hoc or on a schedule. Configuration managed through Terraform uses the PingFederate administration API and requires the server to have successfully started. Configuration changes typically require replication to the engine nodes, but do not require a restart of the PingFederate service to take effect.
- The `/instance/server/default/data` folder in the server profile - a way to declare the initial start-up configuration of PingFederate admin and engine nodes by providing a foundational filesystem structure. The configuration is pulled from Git and applied during container start-up, while changes to this configuration typically requires servers to be restarted to take effect. This configuration method is typically used when deploying new adapter JAR files or deployments to the PingFederate server's built-in Jetty container.
- The `/instance/bulk-config/data.json` file in the server profile - a way to declare the initial start-up configuration of the PingFederate service by providing a foundational configuration package. The configuration is pulled from Git and applied during container start-up, while changes to this configuration typically requires a server restart to take effect.

Managed components

This category is the core PingFederate configuration. This pattern incorporates changes that are typically made through the UI or Admin APIs.

Orchestration

Depending on your operating pattern, changes here may be delivered through a rolling update or by configuration replication.

Using the PingFederate Terraform provider

Terraform updates should trigger server replication to the engine nodes at the end of the PingFederate configuration pipeline.

The admin server should use a persistent volume so it can recover the same admin configuration as before if the pod is restarted. If the clustered engine pods are restarted, they'll refresh their configuration from the admin server during startup. See the below section for details on how to configure a persistent volume.

The Terraform configuration should be managed in a repository separate from infrastructure and server profile configuration. Changes made to the PingFederate server via Terraform require replication to PingFederate engine nodes as the final step of configuration and don't require rolling restarts to the PingFederate deployment for changes to take effect.

For more information on using the PingFederate Terraform provider, see the getting started guide.

PingFederate Data Mount

In the most common pattern, a user would attach a persistent volume (PV) to `/opt/out/instance/server/default/data` *only* on the PingFederate Admin Console.

This model is intended to be used when PingFederate Administrators need to deliver configuration through the UI in *each environment, including production*. Another reason for this use case may be if SP connections are allowed to be created by app developers using the Admin API. In both of these scenarios, the defining factor is that there are mutations in the production Admin console that are not being tracked in any other way, such as through source control, and therefore must be persisted.

Attributes of this pattern:

- App Config is persisted in each SDLC environment (e.g. Dev, QA, Prod)
- App Config promotion is done manually or via the Admin API
- App Config is replicated from Admin Console to Engines
- Server Config is maintained and delivered via the server profile
- Server profile *does not* include App Config
- Server profile ***must not*** have `instance/bulk-config/data.json` or `/instance/server/default/data`
- Backups are taken regularly to provide recovery in case of PV loss or corruption

Data Mount Helm Example

Helm values relevant to this configuration may look like:

```
pingfederate-admin:
  enabled: true
  container:
    replicaCount: 1
  envs:
    SERVER_PROFILE_URL: <insert your server profile URL here>
    SERVER_PROFILE_PATH: <insert your server profile path here>
    SERVER_PROFILE_VERSION: <server profile version>
  workload:
    type: StatefulSet
    statefulSet:
      persistentvolume:
        enabled: true
        volumes:
          out-dir:
            NOTE THIS PVC DEFINITION
            mountPath: /opt/out/instance/server/default/data
            persistentVolumeClaim:
              accessModes:
              - ReadWriteOnce
              storageClassName:
              resources:
                requests:
                  storage: 8Gi

pingfederate-engine:
  enabled: true
  envs:
    SERVER_PROFILE_URL: <insert your server profile URL here>
    SERVER_PROFILE_PATH: <insert your server profile path here>
    SERVER_PROFILE_VERSION: <server profile version>
  container:
    replicaCount: 3
  workload:
    type: Deployment
    deployment:
      strategy:
        type: RollingUpdate
        rollingUpdate:
          maxSurge: 1
          maxUnavailable: 0
```

The key aspect here is `pingfederate-admin.workload.statefulset.persistentvolume.volumes.out-dir.mountPath=/opt/out/instance/server/default/data`. This location is where all UI configuration (App Config) is stored as files. As this location is the `mountPath`, PingFederate administrators have the freedom to deliver any files *not* used in `/opt/out/instance/server/default/data` via a Server Profile.

For example, adding a new IDP adapter requires a restart of the service in order for the adapter to be identified and available to App Config. The steps in this case would be:

1. Add the adapter at `<server profile URL>/<server profile path>/pingfederate/instance/server/default/deploy/idp-adapter-name-1.jar`

2. Update `SERVER_PROFILE_VERSION: <current version>` -> `SERVER_PROFILE_VERSION: <new version>` on both the admin and engine deployments (for example, v1.1 -> v1.2)
3. Run `helm upgrade --install myping pingidentity/ping-devops -f /path/to/values.yaml`

If the release already exists, the variable change signifies that the definition has mutated, and therefore must be redeployed. The admin pod will be deleted and recreated while the engines will surge and roll one by one.

Reference links:

- K8s - Performing a Rolling Update
- K8s - Update a deployment

Data Mount Pros and Cons

Values with this approach

- Managing App Config is more familiar to PingFederate administrators with traditional experience
- Fewer parts to consider when building a CI/CD pipeline because there is no configuration export and templating needed
- Ability to have configurations different in each environment

Cautions with this approach

- There is more room for user configuration error and possible outages because configurations are not promoted with automated testing

Kubernetes deployments for cloud platforms

We currently have instructions for typical configuration of Kubernetes for use with Ping products on these platforms:

- Amazon Web Services (AWS) Elastic Kubernetes Service (EKS)
- Microsoft Azure Kubernetes Service (AKS)

Each hosting platform supports and manages Kubernetes differently.

Before you begin

You must:

- Complete Get Started to set up your DevOps environment and run a test deployment of the products.
- Create a Kubernetes cluster on one of these platforms:
  - Amazon EKS
  - Microsoft AKS

- Create a Kubernetes secret using your DevOps credentials. For more information, see *For Kubernetes* in Using Your DevOps User and Key.

AWS EKS

See Peering VPCs for multi-region EKS deployments.

AKS

See Azure Kubernetes Service.

Peering VPCs for multi-region EKS deployments

Preparing AWS EKS for Multi-Region Deployments

In this guide you will deploy two Kubernetes clusters, each in a different Amazon Web Services (AWS) region. An AWS virtual private cloud (VPC) is assigned and dedicated to each cluster. You will also add communication between these clusters, using a transit gateway. Throughout this document, "VPC" is synonymous with "cluster".

Prerequisites

Before you begin, you must have

- AWS account permissions to create clusters

Create the multi-region clusters

1. Create VPCs.
   - Sign on to the AWS console and navigate to the **VPC** service.
   - Toggle to the `eu-west-1` region.
   - Select **Your VPCs** (under Virtual Private Cloud) and click **Create VPC**
   - Add a name tag, such as `demo-vpc-eu-west-1`
   - Add a IPv4 CIDR, such as `10.0.0.0/16`
   - Click **Create VPC**.

     > *Make note of the `VpcId` and `IPv4 CIDR` values for the `eu-west-1` and `us-east-1` VPCs for use in subsequent steps.*

   - Repeat this step in `us-east-1` region.

2. Create the transit gateway for each region on which your deployment is being hosted. Toggle to the `eu-west-1` region.

   - Navigate to the **Transit gateways** section and click **Create transit gateway**.
   - Add a name tag such as `demo-tgw-eu-west-1`.
   - Add a unique Amazon side Autonomous System Number for each region (ex. 64512 or 64513).
   - Disable both the `Default route table association` and `Default route table propagation`.

   > **Note**
   >
   > If you enable the options above, the default association route table and propagation route table will be created. This action may not suit more complex routing needs; see below for details on how to manually set the associations/propagation route tables.

   - Click **Create transit gateway**.
   - Repeat this step in `us-east-1` region.

3. Create the transit gateway peering connection attachment. Toggle to the `eu-west-1` region.

   - Navigate to the **Transit gateway attachments** section and click **Create transit gateway attachments**.
   - Add a name tag such as `demo-peering-attachment-us-east-1`.

   > **Note**
   >
   > This name refers to the region to which it is peering, not the region in which it is being created.

   - Select the Transit gateway id that you just made in the `eu-west-1` region.
   - Change **Attachment type** to `Peering Connection`.
   - For **Region** select `us-east-1`.
   - For **Transit gateway (accepter)** add the Transit gateway id that you just made in the `us-east-1` region.
   - Click **Create transit gateway attachment** .

4. Accept transit gateway peering attachment.

   - After the transit gateway peering connection shows `pending acceptance` as its **State**, toggle to the `us-east-1` region and select **Transit gateway attachments**.
   - You should see the attachment you just made. Select **Actions** and click accept.
   - Add a name to this attachment such as `demo-peering-attachment-eu-west-1`.

   > **Note**
   >
   > This name refers to the region to which it is peering, not the region in which it is being created.

5. Attach VPCs to the transit gateways in each region. Toggle to the `eu-west-1` region.
   - Navigate to the **Transit gateway attachments** section and click **Create transit gateway attachments**.
   - Add a name tag such as `demo-vpc-eu-west-1`.
   - Select the Transit gateway id that you just made in the `eu-west-1` region.
   - Select the Vpc Id that you made note of in step 3 for the `eu-west-1` region.
   - Click **Create transit gateway attachment**.
   - Repeat this step in `us-east-1` region.
6. Accept transit gateway VPC attachments. Toggle to the `eu-west-1` region.
   - Navigate to the **Transit gateway attachments** section and click **Create transit gateway attachments**.
   - You should see the vpc attachment you just made. Select **Actions** and click accept.

     > **Note**
     >
     > If you are using different accounts to create transit gateways and their attachments, the name tag will not be visible here. In this situation, you should add an attachment name now, such as `demo-vpc-eu-west-1`.

   - Repeat this step in `us-east-1` region.
7. Add routes to vpc route table. Toggle to the `eu-west-1` region.
   - Navigate to the **Route tables** section and select the route table for the Vpc Id you created.
   - Select **Routes** in the bottom third of the page.
   - Select **Edit routes** then click **Add route**.
   - Add a destination that is more broad than the local one that is present. For example if the local destination is `10.0.0.0/16` add `10.0.0.0/8`.
   - For the **Target** select `Transit Gateway` and add then add the Transit gateway id that you created in this region.
   - Click **Save changes**.
   - Repeat this step in `us-east-1` region.
8. Configure the transit gateway route tables. Toggle to the `eu-west-1` region.
   - Navigate to the **Transit gateway route tables** section and click **Create transit gateway route table**.
   - Add a name tag such as `demo-eu-west-1-route-table`.
   - Select the Transit gateway id that you created in this region.
   - Click **Create transit gateway route table**.
   - Repeat this step in `us-east-1` region.

9. Associate the transit gateway. Toggle to the `eu-west-1` region.
   - After the transit gateway route table has been successfully created, select that route table and click **Associations** then **Create association**.
   - Choose the VPC attachment for this region and click **Create association**.
10. Add static routes to the transit gateway. Toggle to the `eu-west-1` region.
    - Select that route table that you just created an association for and click **Routes** then **Create static route**.
    - Add the `IPv4 CIDR` for the remote VPC that you made note of in step 3 for the `us-east-1` region.
    - Select the transit gateway peering connection attachment.
    - Click **Create static route**.
    - Repeat this step in `us-east-1` region.
11. Create a blackout static route to ensure the transit gateway drops any other network traffic. Toggle to the `eu-west-1` region.
    - Select **Create static route**
    - Add 10.0.0.0/8 as the CIDR
    - Select Blackhole
    - Click **Create static route**.
    - Repeat this step in `us-east-1` region.

At this point you should have a system of connected VPCs on the `us-east-1` `eu-west-1` regions. You can now deploy EC2 instances to these VPCs and communicate between them.

Azure Kubernetes Service

Preparing Azure Kubernetes Service

This directory contains scripts and deployment files to help with the deployment, management, and scaling of Ping Identity DevOps Docker images to Microsoft Azure Kubernetes Service (AKS).

Prerequisites

Before you begin, you must:

- Set up your DevOps environment and run a test deployment of the products. For more information, see Get Started.
- Create a Kubernetes cluster on AKS.

- Create a Kubernetes secret using your DevOps credentials. See the *For Kubernetes* topic in Using your DevOps user and key.
- Download and install the Azure CLI.

We also highly recommend you are familiar with the information in these AKS articles:

- Azure Kubernetes Service

Deploying our fullstack example in AKS

1. Create an Azure Resource Group to put all resources into by entering:

```
az group create \
   --name ping-devops-rg \
   --location westus
```

2. Create a two-node Azure AKS cluster by entering the following.

```
az aks create \
   --resource-group ping-devops-rg \
   --name ping-devops-cluster \
   --node-count 2 \
   --enable-addons monitoring \
   --ssh-key-value ~/.ssh/id_rsa.pub
```

   You need a public certificate by default in ~/.ssh/id_rsa.pub.

3. Import the AKS Credentials into `.kube/config` by entering:

```
az aks get-credentials \
   --resource-group ping-devops-rg \
   --name ping-devops-cluster
```

4. At this point, the cluster should be ready for helm deployments.
5. To clean up the Azure Resource Group and all associated resources, including the AKS cluster created, enter the following command:

   Caution

   This will remove everything you created that is associated with this resource group.

```
az group delete \
  --name ping-devops-rg
```

Sizing Kubernetes clusters

When creating your Kubernetes cluster, size the nodes appropriately.

Kubernetes cluster capacity

When determining the capacity of your cluster, there are many ways to approach sizing the nodes. For example, if you calculated a cluster sizing of 16 CPU and 64GB RAM, you could break down the node sizing into these options:

1. 2 nodes: 8 CPU / 32 GB RAM
2. 4 nodes: 4 CPU / 16 GB RAM
3. 8 nodes: 2 CPU / 8 GB RAM
4. 16 nodes: 1 CPU / 4 GB RAM

To understand which sizing option to select, consider the associated pros and cons.

> **Note**
>
> You can generally assume that instance cost per CPU/RAM is linear among the major cloud platforms.

Option 1: fewer, larger nodes

Pros

- If you have applications that are CPU or RAM intensive, having larger nodes can ensure your application has sufficient resources.

Cons

- High availability is difficult to achieve with a minimal set of nodes. If your application has 50 pods across two nodes (25 pods per node) and a node goes down, you lose 50% of your service.
- Scaling: When autoscaling the cluster, the increment size becomes larger which could result in provisioning more hardware than needed.

Option 2: more, smaller nodes

Pros

- High availability is easier to maintain. If you have 50 instances with two pods per node (25 nodes) and one node goes down, you only reduce your service capacity by 4%.

Cons

- More system overhead to manage all of the nodes.
- Possible under-utilization as the nodes might be too small to add additional services.

Guidance

For production deployments where high availability is paramount, creating a cluster with more nodes running fewer pods per node is preferable to ensure the health of your deployed service.

> **Note**
>
> For some applications, you can decide to size one pod per node.

To determine the physical instance type needed, multiply the desired resources for each service by the number of pods per node, plus additional capacity for system overhead. Follow product guidelines to determine system requirements.

Example service using 3 pods per node

- Each pod is typically deployed with 2 CPU and 4GB RAM which when multiplied by 3 yields:
  - Minimum node requirement: 6 CPU 12 GB RAM
- Add 10% for system overhead

For these requirements in Amazon Web Services (AWS), a `c5.2xlarge` type (8 CPU / 16 GB RAM) might be the instance type selected.

To determine the base number of nodes required, divide the number of pods by 3 to determine your minimum cluster size. Further, you must ensure that you add definitions for cluster horizontal auto-scaling so the cluster scales in or out as needed.

Ingress on Local Kind Cluster

If you have deployed the local kind cluster as outlined on the Deploy a local Kubernetes Cluster page, follow these instructions to use an ingress for accessing your products.

Prerequisites

- A kind cluster deployed with ingress enabled. For this guide, the cluster name `ping` is assumed
- The hostname aliases have been appended to the `/etc/hosts` file
- You have created the secret for your DevOps user and key for retrieving licenses

Assumptions

With the `/etc/hosts` file entries created from the page linked above, the release in helm **must** be `myping` for the hostnames to work with the configuration here. Consider the first entry as an example:

```
127.0.0.1 myping-pingaccess-admin.pingdemo.example ...
```

When using our charts, the release name provided to helm is prepended - that is what provides the `myping-` portion of the hostname in the file. The `pingdemo.example` domain suffix is provided through the ingress definitions as shown later on this page. So, the structure is:

```
<helm-release-name>-<ping-product-service>.<domain-name-from-ingress>
```

If you use a release name other than `myping` or a domain other than `pingdemo.example` you will need to update the aliases in `/etc/hosts` / accordingly.

Instructions

There is a file under the `30-helm` directory of this repository named `ingress.yaml`. Modify this file for use with a local cluster:

- Replace `insert domain name here` with your domain name (pingdemo.example in this guide)
- Edit line 11, removing the `-public` suffix for the class

The file should look as shown here:

```
global:
  envs:
    PING_IDENTITY_ACCEPT_EULA: "YES"
  ingress:
    enabled: true
    addReleaseNameToHost: prepend
    defaultDomain: "pingdemo.example"
    defaultTlsSecret:
    annotations:
      nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
      kubernetes.io/ingress.class: "nginx"
```

Create ingresses

When deploying using helm and one of the example YAML files in the 30-helm directory, also pass in the ingress.yaml file to include the ingress definitions as part of the overall deployment.

For example, to use the `everything.yaml` and include ingresses, you would run the following command from the 30-helm directory (after updating the ingress.yaml file):

```
helm upgrade --install myping pingidentity/ping-devops -f everything.yaml -f ingress.yaml
```

After everything has started, you will see the same pods and services as in the getting started example. In addition, you will see ingress definitions in the same namespace as the products:

```
# List ingress definitions
kubectl get ingress

# Output
NAME                         CLASS    HOSTS                                         ADDRESS     PORTS     AGE
myping-pingaccess-admin      <none>   myping-pingaccess-admin.pingdemo.example        localhost   80, 443
47m
myping-pingaccess-engine     <none>   myping-pingaccess-engine.pingdemo.example       localhost   80, 443
47m
myping-pingauthorize         <none>   myping-pingauthorize.pingdemo.example           localhost   80, 443
47m
myping-pingdataconsole       <none>   myping-pingdataconsole.pingdemo.example         localhost   80, 443
47m
myping-pingdirectory         <none>   myping-pingdirectory.pingdemo.example           localhost   80, 443
47m
myping-pingfederate-admin    <none>   myping-pingfederate-admin.pingdemo.example      localhost   80, 443
47m
myping-pingfederate-engine   <none>   myping-pingfederate-engine.pingdemo.example     localhost   80, 443
47m
```

The HOSTS column reflects the entries added to the `/etc/hosts` file.

To access a given service, enter the HOSTS entry in your browser (you will have to accept the self-signed certificate). For example, to view the Ping Federate console, you would access https://myping-pingfederate-admin.pingdemo.example/. For the Ping Data console, https://myping-pingdataconsole.pingdemo.example and so on.

Here are the credentials and URLs. This table is similar to the getting started example but reflects the release name used on this page:

Product	Connection Details
PingFederate	• URL: https://myping-pingfederate-admin.pingdemo.example/pingfederate/app • Username: administrator • Password: 2FederateM0re
PingDirectory	• URL: https://myping-pingdataconsole.pingdemo.example/console • Server: ldaps://myping-pingdirectory-cluster:1636 • Username: administrator • Password: 2FederateM0re
PingAccess	• URL: https://myping-pingaccess-admin.pingdemo.example • Username: administrator • Password: 2FederateM0re
PingAuthorize	• URL: https://myping-pingdataconsole.pingdemo.example/console • Server: ldaps://myping-pingauthorize-cluster:1636 • Username: administrator • Password: 2FederateM0re

Cleaning up

Since the ingresses are deployed as part of the overall release, deleting the release will also remove the ingress definitions (leaving the ingress controller intact).

The ingress controller will be removed when the cluster is deleted. If you only want to remove the ingress controller, you can either:

- Run `kubectl delete -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/1.23/deploy.yaml` if you installed the controller from Github

  OR

- Run `kubectl delete -f ./20-kubernetes/kind-nginx.yaml` if you used the local copy to install the controller.

Restoring a Multi-Region PingDirectory Deployment After Seed Cluster Failure

The PingDirectory hook scripts rely on the seed server of the seed cluster being available when running in a multi-region environment. This dependency leads to the question of what can be done if the seed region's servers, along with their persistent volumes, are lost. This page describes manual steps that can be taken to restore the PingDirectory topology in this case.

Note

These steps are only needed if the seed region is lost *including its persistent volumes*.

In this document, the non-seed region is referred to as the "surviving" region.

Starting point

Assume the seed region and all of its persistent volumes have been lost. The surviving region will still replicate among itself, but it will not be able to reach the seed region servers even after they are restarted, due to changes to the server certificates caused by the restart.

Running the `status` command on one of the surviving pods, you will see in the output that there are configured servers in the topology that are not reachable:

```
          --- Mirrored Subtrees ---
Base DN               : Write Master                   : Configured      : Outbound        :
Inbound         : Failed Write
                      :                                : Peers           : Connections     :
Connections     : Operations
----------------------:--------------------------------:-----------------:-----------------:-----------------:-------
cn=Cluster,cn=config  : N/A (single instance topology) : 0               : 0               :
0               : 0
cn=Topology,cn=config : No Master (data read-only)     : 3               : 1               :
1               : 0
```

You will also see administrative alerts in the output indicating that the mirrored subtree manager cannot establish a connection with the servers in the seed region.

Overview

The key steps to restore the topology in this case are:

1. Force a server in the surviving region to act as master of the topology
2. Remove the unreachable servers from the topology
3. Undo forcing a server to act as master of the topology
4. Ensure any seed region pods are in single-server topologies
5. Use `dsreplication enable` to add the servers from the refreshed seed region to the topology of the surviving region
6. Use `dsreplication initialize-all` from a server of the surviving region to update the data across the regions

Force a server to act as master of the topology

The PingDirectory topology will see that it cannot connect with half the servers and will switch to read-only mode. To allow the changes we need to make to the topology to fix this, exec into one of the pods in the surviving region.

`kubectl exec -ti example-pingdirectory-0 sh`

Run the following command to force this pod as master:

`dsconfig set-global-configuration-prop --set force-as-master-for-mirrored-data:true --no-prompt`

Remove the unreachable servers from the topology

Now we must tell the surviving pods that the original seed region pods no longer exist, and that they must be removed from the topology. These commands may take a long time to run, as the `remove-defunct-server` tool will keep trying to connect for up to ten minutes depending on the state of the seed region.

`remove-defunct-server --ignoreOnline --serverInstanceName example-pingdirectory-1.west --bindDN [bind dn] --bindPassword [bind password]`

In the above command, replace the `--serverInstanceName` argument with the instance name of one of the seed region pods. Repeat the command for each seed region pod's instance name.

This step may differ depending on the state of the seed region. If the seed region is wiped out and is still not available, then you may be prompted during the `remove-defunct-server` process whether you want to retry connecting to a server that was from the failed seed region. Enter "no" and continue if prompted.

If the seed region has been restored and the servers are up by the time you are running this command, then you will likely see the ten minute timeout described above. This situation occurs because the servers are available on the same hostnames as before, but their inter-server certificates have changed during the restart. The new certificates mean SSL connections will not be possible, leading to the connection timeout.

Undo forcing a server to act as master of the topology

At this point the pods in the surviving region should now be the only pods in that region's topology - they should no longer be attempting to contact any pod from the failed seed region.

```
  --- Mirrored Subtrees ---
Base DN               : Write Master                     : Configured      : Outbound        :
Inbound         : Failed Write
                      :                                  : Peers           : Connections     :
Connections     : Operations
----------------------:----------------------------------:-----------------:-----------------:-----------------:-------
cn=Cluster,cn=config  : N/A (single instance topology) : 0               : 0               :
0               : 0
cn=Topology,cn=config : example-pingdirectory-0.east   : 1               : 1               :
1               : 0
```

Exec into the pod that was forced as master in the first step. Run this command to undo the previous change:

`dsconfig set-global-configuration-prop --set force-as-master-for-mirrored-data:false --no-prompt`

Remove the seed servers from their own topology after restart if necessary

If the seed region was completely wiped out and unavailable during the earlier `remove-defunct-server` step, this step will be necessary. When the seed region comes up again, it will join its servers together in a new topology containing only the seed pods, as it is unaware of the other region.

It is not possible to merge two existing topologies containing more than one server each. We need to split up the restarted seed region pods into individual single-server topologies so that we can add them to the topology of the surviving region.

Exec into one of the seed region pods:

`kubectl exec -ti example-pingdirectory-0 sh`

Use `remove-defunct-server` to split up each server, starting with the highest pod ordinal and working down until ordinal `1`. After this is done, all seed region pods will be in separate single-server topologies, and we can then add them to the existing topology of the surviving region.

`remove-defunct-server --ignoreOnline --serverInstanceName example-pingdirectory-1.west --bindDN [bind dn] --bindPassword [bind password]`

Add the servers from the refreshed seed region to the topology of the surviving region

At this point the servers in the seed region should be in their own single-server topologies, and the servers in the surviving region should be in a topology containing only the pods in that region.

Now we can re-enable replication between the regions. Run the following command once for each pod in the seed region, updating the `--host1` or `--host2` argument each time to point to the server being enabled in that run. The command can be run from a shell on any pod.

```
dsreplication enable \
    --trustAll \
    --host1 example-pingdirectory-0.example.east.example.com \
    --port1 "${LDAPS_PORT}" \
    --useSSL1 \
    --replicationPort1 "${REPLICATION_PORT}" \
    --bindDN1 "${ROOT_USER_DN}" \
    --bindPasswordFile1 "${ROOT_USER_PASSWORD_FILE}" \
    --host2 example-pingdirectory-0.example.west.example.com \
    --port2 "${LDAPS_PORT}" \
    --useSSL2 \
    --replicationPort2 "${REPLICATION_PORT}" \
    --bindDN2 "${ROOT_USER_DN}" \
    --bindPasswordFile2 "${ROOT_USER_PASSWORD_FILE}" \
    --adminUID "${ADMIN_USER_NAME}" \
    --adminPasswordFile "${ADMIN_USER_PASSWORD_FILE}" \
    --no-prompt --ignoreWarnings \
    --baseDN dc=example,dc=com \
    --noSchemaReplication
```

Run dsreplication initialize-all from a server of the surviving region

All of the pods are again in a topology together. Now we need to initialize the seed region with the data from the surviving region. Run the following command, targeting a server in the surviving region with the `--hostname` argument (this indicates which server the data is coming from, so we want to use a server in the surviving region):

```
dsreplication initialize-all \
    --hostname example-pingdirectory-0.example.east.example.com \
    --port 7700 --useSSL \
    --baseDN dc=example,dc=com --adminUID admin \
    --adminPasswordFile /tmp/pw --no-prompt
```

Now we can see from `dsreplication status --showAll` that all the pods are replicating and have matching generation IDs:

```
          --- Replication Status for dc=example,dc=com: Enabled ---
Server                                                                               : Location : Entries :
Conflict Entries : Backlog (1) : Rate (2) : A.O.B.C. (3) : Generation ID : Server ID : Replica ID
-------------------------------------------------------------------------------------:----------:---------:----------
example-pingdirectory-0.east (example-pingdirectory-0.example.east.example.com:7700) : east     : 2038    :
0                : 0           : 0        : 0 seconds    : 4105471824    : 19064     : 32073
example-pingdirectory-1.east (example-pingdirectory-1.example.east.example.com:7700) : east     : 2038    :
0                : 0           : 0        : 0 seconds    : 4105471824    : 4444      : 18281
example-pingdirectory-0.west (example-pingdirectory-0.example.west.example.com:7700) : west     : 2038    :
0                : 0           : 0        : 0 seconds    : 4105471824    : 28554     : 13185
example-pingdirectory-1.west (example-pingdirectory-1.example.west.example.com:7700) : west     : 2038    :
0                : 0           : 0        : 0 seconds    : 4105471824    : 2590      : 4761
```

And from `status` we can see all the inbound and outbound connections are functioning as expected:

```
          --- Mirrored Subtrees ---
Base DN                : Write Master                     : Configured       : Outbound         :
Inbound         : Failed Write
                       :                                  : Peers            : Connections      :
Connections     : Operations
-----------------------:----------------------------------:------------------:------------------:------------------:-------
cn=Cluster,cn=config   : N/A (single instance topology) : 0                : 0                :
0               : 0
cn=Topology,cn=config  : example-pingdirectory-0.east     : 3                : 3                :
3               : 0
```

Docker Compose

Docker Compose

Single product examples only

> **Caution**
>
> Docker Compose was used by Ping in the past for basic orchestration and examples. We are no longer maintaining multi-product or clustering docker compose examples. All of those files have been removed from this repository. The only examples remaining are for deploying individual products. For orchestration of multiple products, clustering, and other use cases, see Deploy Ping DevOps Charts using Helm.

Example docker compose files to deploy standalone instances of PingAccess, PingCentral, PingDirectory or PingFederate are in the Github repository. Refer to the comments in each provided file for instructions on accessing the product after running `docker compose up` from the directory of the product in which you are interested.

DevOps Support Policy

This DevOps Support Policy is an extension of the Ping Identity Support Policy.

This Ping Identity Corporation ("Ping Identity") DevOps Support Policy (this "Policy") encompasses all support obligations that Ping Identity has toward you as Ping Identity's Customer ("Customer").

Note

The support policy for Ping Identity product Docker images is found at Docker Image Support Policy.

Included in Support

- Providing base images for Ping Identity products to Customers
- Providing documentation and basic examples for Helm deployments using Ping Identity's Helm charts
- Providing the Customer with DevOps tooling, such as `config_export` and `pingctl`
- Providing the Customer direction in using Server Profile to achieve the following:
  - Deployment
  - Customization
  - Saving Configuration
  - Layering
  - Environment Substitution
  - Private Github repos

Supported orchestration tools

Tool	Description
Kubernetes	Also known as K8s, Kubernetes is an open-source system for automating deployment, scaling, and management of containerized software.
Helm charts	Helm is the easiest way to deploy Ping Identity software images in a Kubernetes environment.
Docker images	Docker images are maintained by Ping Identity and are a collection of preconfigured environments for Ping Identity products.
GitHub repositories	These repositories provide all of the components to build Docker images for your own development, testing and deployments.

Resources

Ping Identity customers can create a case in the Ping Identity Support Portal.

Non-Ping Identity customers can use the PingDevOps Community.

License

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/
```

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

    1. You must give any other recipients of the Work or Derivative Works a copy of this License; and

    2. You must cause any modified files to carry prominent notices stating that You changed the files; and

    3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

    4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

        You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

Disclaimer

Disclaimer Of Warranties

Third-Party Software

Product Images

Ping Identity Docker images bundle various third-party tools to enable product functionality. For more details, click on the links below:

- OpenJDK: GNU General Public License version 2.0
- OpenSSH: Based on BSD licensing
- Git: GNU General Public License version 2.0
- Gettext: GNU General Public License version 2.0
- Curl: Based on MIT/X license
- ca-certificates: GNU General Public License version 2.0
- Jq: MIT licensing
- Gnupg: GNU General Public License

Questions about 3rd-Party Software or Services

Thanks again for using this portal. Our goal is to keep it current, relevant, and as error-free as possible.

Please refer to the disclaimer as background for this section.

As a reminder, use the methods on the Contact Us page if you encounter issues directly related to this portal, files or product container images provided by Ping, or with any suggestions you may have.

When seeking help with content on this portal, consider the following:

1. **Product-related questions**: For questions about the Ping products that run inside these images, customers with valid support contracts should engage through our Support Portal.
2. **Questions specific to Ping product container images**: Customers with valid support contracts should engage through our Support Portal. Please specify that the problem is with an image provided by the DevOps program
3. **Examples on this portal**: If you are following an example step-by-step as given on this portal, and it does not work, contact us at the link above.
4. **General questions about Ping DevOps**: Consult our FAQ page, or use an internet search for the information.
5. **Any code in these Github repositories**: Contact us using information from the link above or use the bug tracking link on the repository:
    1. Getting Started
    2. Docker Builds
    3. Helm Charts
    4. Server Profiles

Other software mentioned or used on this portal (not comprehensive)

The link will take you to the main page for each product or service. You should also explore the forums and documentation for each if you encounter an issue:

- Helm
- Docker Desktop
- Kubernetes
- kind
- minikube
- Openshift Local
- ingress-nginx
- externalDNS
- MetalLB
- kubectl
- k9s
- Prometheus
- Grafana
- telegraf-operator

Portal Updates

This page provides details on significant changes to this portal.

September 2023

A statement on AWS EFS/EBS

Some of the product documentation does not explicitly explain the recommended storage solution to use when running on AWS. Clarification was added, particularly when running Ping products in containerized environments here.

July 2023

Read-only filesystem example and explanation

Some environments require the use of a read-only filesystem on containers in runtime. As our product images mature to better align with this practice, a way to work around the situation with our images as they stand today can be found here.

November 2022

Video content is arriving

In recent weeks, videos on topics relevant to the DevOps program at Ping have started to be released. Look for links on some pages where the videos are relevant, or you can browse all videos here.

July 2022

Removal of Docker Compose examples

With the focus of using Helm to provision to Kubernetes as the recommended practice, the multi-product Docker Compose examples were no longer maintained or supported and have been removed. The only examples that remain are for single-product deployments. For orchestrating multiple products, see the Helm examples.

Removal of Kubernetes Kustomize examples

Kustomize was the former method for for Kubernetes orchestration. As most of those examples were replaced by Helm, the Kustomize guides and files were also removed from this portal.

Confirmation of Documentation

We received reports in recent weeks that some of examples were not working due to changes in Kubernetes version support, files that had been removed from this repository, and other reasons. All examples and guides on this portal have been reviewed and should work as written.

Container Docker Images Information

Image Information

These documents for Ping Identity Docker images include information on:

- Related Docker images
- Ports exposed for the containers
- Environment variables for the image
- Associated deployment information
- Tagging methodology and support policy

The image-specific documentation is generated from each new build ensuring these documents align with any changes over time.

Storage Considerations on AWS

When deploying Ping products in containers, please consider the information found here on storage options.

Older images based on product versions that are no longer supported under our policy are removed from Docker Hub. See the support policy page for details.

As with many organizations, Ping Identity uses *floating* Docker image tags. This practice means, for example, that the `edge` tag does not refer to the same image over time as product updates occur. The release tags page has information on the `edge` and other tags, how often they are updated, and how to ensure the use of a particular version and release of a product image.

Notification of new image tags

If you want to be notified when new versions of product Docker images are available, see the **Docker Images** section of the FAQ page for instructions on following the docker-builds GitHub repository.

Iron Bank Images

For FedRAMP certification and other United States government compliance, Ping Identity partners to build and host highly-secured images at Iron Bank that are compliant with the increased security requirements. More information is provided in the FAQ section of this documentation. The Iron Bank images are available on the Iron Bank repository and are not provided on Docker Hub.

Product Images

Ping Identity DevOps Docker Image - `pingaccess`

Ping Identity DevOps Docker Image - `pingaccess`

This docker image includes the Ping Identity PingAccess product binaries and associated hook scripts to create and run both PingAccess Admin and Engine nodes.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingcommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
BASE	${BASE:-/opt}	Location of the top level directory where everything is located in image/container
ROOT_USER	administrator	the default administrative user for PingData
JAVA_HOME	/opt/java	
STAGING_DIR	${BASE}/staging	Path to the staging area where the remote and local server profiles can be merged
OUT_DIR	${BASE}/out	Path to the runtime volume
SERVER_ROOT_DIR	${OUT_DIR}/instance	Path from which the runtime executes
IN_DIR	${BASE}/in	Location of a local server-profile volume

ENV Variable	Default	Description
SERVER_BITS_DIR	${BASE}/server	Path to the server bits
BAK_DIR	${BASE}/backup	Path to a volume generically used to export or backup data
LOGS_DIR	${BASE}/logs	Path to a volume generically used for logging
PING_IDENTITY_ACCEPT_EULA	NO	Must be set to 'YES' for the container to start
PING_IDENTITY_DEVOPS_FILE	devops-secret	File name for devops-creds passed as a Docker secret
STAGING_MANIFEST	${BASE}/staging-manifest.txt	Path to a manifest of files expected in the staging dir on first image startup
CLEAN_STAGING_DIR	false	Whether to clean the staging dir when the image starts
SECRETS_DIR	/run/secrets	Default path to the secrets
TOPOLOGY_FILE	${STAGING_DIR}/topology.json	Path to the topology file
HOOKS_DIR	${STAGING_DIR}/hooks	Path where all the hooks scripts are stored
CONTAINER_ENV	${STAGING_DIR}/.env	Environment Property file use to share variables between scripts in container
SERVER_PROFILE_DIR	/tmp/server-profile	Path where the remote server profile is checked out or cloned before being staged prior to being applied on the runtime
SERVER_PROFILE_URL		A valid git HTTPS URL (not ssh)
SERVER_PROFILE_URL_REDACT	true	When set to "true", the server profile git URL will not be printed to container output.
SERVER_PROFILE_BRANCH		A valid git branch (optional)
SERVER_PROFILE_PATH		The subdirectory in the git repo
SERVER_PROFILE_UPDATE	false	Whether to update the server profile upon container restart

ENV Variable	Default	Description
SECURITY_CHECKS_STRICT	false	Requires strict checks on security
SECURITY_CHECKS_FILENAME	.jwk .pin	Perform a check for filenames that may violate security (i.e. secret material)
UNSAFE_CONTINUE_ON_ERROR		If this is set to true, then the container will provide a hard warning and continue.
LICENSE_DIR	${SERVER_ROOT_DIR}	License directory
PD_LICENSE_DIR	${STAGING_DIR}/pd.profile/server-root/pre-setup	PD License directory. Separating from above LICENSE_DIR to differentiate for different products
STARTUP_FOREGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container
PING_IDENTITY_DEVOPS_KEY_REDACT	true	
TAIL_LOG_FILES		A whitespace separated list of log files to tail to the container standard output - DO NOT USE WILDCARDS like /path/to/logs/*.log
COLORIZE_LOGS	true	If 'true', the output logs will be colorized with GREENs and REDs, otherwise, no colorization will be done. This is good for tools that monitor logs and colorization gets in the way.
LOCATION	Docker	Location default value If PingDirectory is deployed in multi cluster mode, that is, K8S_CLUSTER, K8S_CLUSTERS and K8S_SEED_CLUSTER are defined, LOCATION is ignored and K8S_CLUSTER is used as the location

ENV Variable	Default	Description
LOCATION_VALIDATION	true	Any string denoting a logical/physical location
MAX_HEAP_SIZE	384m	Heap size (for java products)
JVM_TUNING	AGGRESSIVE	
JAVA_RAM_PERCENTAGE	75.0	Percentage of the container memory to allocate to PingFederate JVM DO NOT set to 100% or your JVM will exit with OutOfMemory errors and the container will terminate
VERBOSE	false	Triggers verbose messages in scripts using the set -x option.
PING_DEBUG	false	Set the server in debug mode, with increased output
PING_PRODUCT		The name of Ping product, i.e. PingFederate, PingDirectory - must be a valid Ping product type. This variable should be overridden by child images.
PING_PRODUCT_VALIDATION	true	i.e. PingFederate,PingDirectory
ADDITIONAL_SETUP_ARGS		List of setup arguments passed to Ping Data setup-arguments.txt file
LDAP_PORT	1389	Port over which to communicate for LDAP
LDAPS_PORT	1636	Port over which to communicate for LDAPS
HTTPS_PORT	1443	Port over which to communicate for HTTPS
JMX_PORT	1689	Port for monitoring over JMX protocol
ORCHESTRATION_TYPE		The type of orchestration tool used to run the container, normally set in the deployment (.yaml) file. Expected values include: - compose - swarm - kubernetes Defaults to blank (i.e. No type is set)
USER_BASE_DN	dc=example,dc=com	Base DN for user data

ENV Variable	Default	Description
DOLLAR	'$'	Variable with a literal value of '$', to avoid unwanted variable substitution
PD_ENGINE_PUBLIC_HOSTNAME	localhost	PD (PingDirectory) public hostname that may be used in redirects
PD_ENGINE_PRIVATE_HOSTNAME	pingdirectory	PD (PingDirectory) private hostname
PDP_ENGINE_PUBLIC_HOSTNAME	localhost	PDP (PingDirectoryProxy) public hostname that may be used in redirects
PDP_ENGINE_PRIVATE_HOSTNAME	pingdirectoryproxy	PDP (PingDirectoryProxy) private hostname
PDS_ENGINE_PUBLIC_HOSTNAME	localhost	PDS (PingDataSync) public hostname that may be used in redirects
PDS_ENGINE_PRIVATE_HOSTNAME	pingdatasync	PDS (PingDataSync) private hostname
PAZ_ENGINE_PUBLIC_HOSTNAME	localhost	PAZ (PingAuthorize) public hostname that may be used in redirects
PAZ_ENGINE_PRIVATE_HOSTNAME	pingauthorize	PAZ (PingAuthorize) private hostname
PAZP_ENGINE_PUBLIC_HOSTNAME	localhost	PAZP (PingAuthorize-PAP) public hostname that may be used in redirects
PAZP_ENGINE_PRIVATE_HOSTNAME	pingauthorizepap	PAZP (PingAuthorize-PAP) private hostname
PF_ENGINE_PUBLIC_HOSTNAME	localhost	PF (PingFederate) engine public hostname that may be used in redirects
PF_ENGINE_PRIVATE_HOSTNAME	pingfederate	PF (PingFederate) engine private hostname
PF_ADMIN_PUBLIC_BASEURL	https://localhost:9999	PF (PingFederate) admin public baseurl that may be used in redirects
PF_ADMIN_PUBLIC_HOSTNAME	localhost	PF (PingFederate) admin public hostname that may be used in redirects
PF_ADMIN_PRIVATE_HOSTNAME	pingfederate-admin	PF (PingFederate) admin private hostname
PA_ENGINE_PUBLIC_HOSTNAME	localhost	PA (PingAccess) engine public hostname that may be used in redirects

ENV Variable	Default	Description
PA_ENGINE_PRIVATE_HOSTNAME	pingaccess	PA (PingAccess) engine private hostname
PA_ADMIN_PUBLIC_HOSTNAME	localhost	PA (PingAccess) admin public hostname that may be used in redirects
PA_ADMIN_PRIVATE_HOSTNAME	pingaccess-admin	PA (PingAccess) admin private hostname
ROOT_USER_DN	cn=${ROOT_USER}	DN of the server root user
ENV	${BASE}/.profile	
PS1	\${PING_PRODUCT}:\h:\w\n>	Default shell prompt (i.e. productName:hostname:workingDir)
PATH	${JAVA_HOME}/bin:${BASE}:${SERVER_ROOT_DIR}/bin:${PATH}	PATH used by the container
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingAccess	Ping product name
LICENSE_DIR	${SERVER_ROOT_DIR}/conf	License directory
LICENSE_FILE_NAME	pingaccess.lic	Name of license file
LICENSE_SHORT_NAME	PA	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
OPERATIONAL_MODE	STANDALONE	PA_RUN_PA_OPERATIONAL_MODE will override this value for PingAccess 7.3 and later.
PA_ADMIN_PASSWORD_INITIAL	2Access	

ENV Variable	Default	Description
PING_IDENTITY_PASSWORD	2FederateM0re	Specify a password for administrator user for interaction with admin API
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/run.sh	The command that the entrypoint will execute in the foreground to instantiate the container
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/log/ pingaccess.log	Files tailed once container has started
PA_ADMIN_PORT	9000	Default port for PA Admin API and console Ignored when using PingIdentity Helm charts
PA_ADMIN_CLUSTER_PORT	9090	Default port when clustering PA primary administrative node Ignored when using PingIdentity Helm charts
JAVA_RAM_PERCENTAGE	60.0	Percentage of the container memory to allocate to PingAccess JVM DO NOT set to 100% or your JVM will exit with OutOfMemory errors and the container will terminate
FIPS_MODE_ON	false	Turns on FIPS mode (currently with the Bouncy Castle FIPS provider) set to exactly "true" lowercase to turn on set to anything else to turn off PA_FIPS_MODE_PA_FIPS_MODE will override this for PingAccess 7.3 and later.
SHOW_LIBS_VER	true	Defines a variable to allow showing library versions in the output at startup default to true
SHOW_LIBS_VER_PRE_PATCH	false	Defines a variable to allow showing library version prior to patches being applied default to false This is helpful to ensure that the patch process updates all libraries affected
PA_ENGINE_PORT	3000	
ADMIN_WAITFOR_TIMEOUT	300	wait-for timeout for 80-post-start.sh hook script How long to wait for the PA Admin console to be available

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- ${PA_ADMIN_PORT}
- ${PA_ENGINE_PORT}
- ${HTTPS_PORT}

Running a PingAccess container

To run a PingAccess container:

```
docker run \
         --name pingaccess \
         --publish 9000:9000 \
         --publish 443:1443 \
         --detach \
         --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
         --env SERVER_PROFILE_PATH=getting-started/pingaccess \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingaccess:edge
```

Follow Docker logs with:

```
docker logs -f pingaccess
```

If using the command above with the embedded server profile, log in with:

- https://localhost:9000
- Username: Administrator
- Password: 2FederateM0re

Docker Container Hook Scripts

Please go here for details on all pingaccess hook scripts

This document is auto-generated from *pingaccess/Dockerfile*

Ping Identity DevOps Docker Image - `pingauthorize`

Ping Identity DevOps Docker Image - `pingauthorize`

This docker image includes the Ping Identity PingAuthorize product binaries and associated hook scripts to create and run a PingAuthorize instance or instances.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingdatacommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	--shm-size 256m \
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingAuthorize	Ping product name
LICENSE_DIR	${PD_LICENSE_DIR}	PD License directory. This value is set from the pingbase dockerfile
LICENSE_FILE_NAME	PingAuthorize.lic	Name of license file
LICENSE_SHORT_NAME	PingAuthorize	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
MAX_HEAP_SIZE	1g	Minimal Heap size required for PingAuthorize

ENV Variable	Default	Description
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/start-server	The command that the entrypoint will execute in the foreground to instantiate the container
STARTUP_FOREGROUND_OPTS	--nodetach	The command-line options to provide to the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the startup command when the container starts with the server in the background. This is the debug start flow for the container
ROOT_USER_PASSWORD_FILE		Location of file with the root user password (i.e. cn=directory manager). Defaults to /SECRETS_DIR/root-user-password
ENCRYPTION_PASSWORD_FILE		Location of file with the passphrase for setting up encryption Defaults to /SECRETS_DIR/encryption-password
KEYSTORE_FILE		Location of the keystore file containing the server certificate. If left undefined, the SECRETS_DIR will be checked for a keystore. If that keystore does not exist, the server will generate a self-signed certificate.
KEYSTORE_PIN_FILE		Location of the pin file for the keystore defined in KEYSTORE_FILE. You must specify a KEYSTORE_PIN_FILE when a KEYSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a self-signed certificate.

ENV Variable	Default	Description
KEYSTORE_TYPE		Format of the keystore defined in KEYSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the keystore format will be inferred based on the file extension of the KEYSTORE_FILE, defaulting to "jks".
TRUSTSTORE_FILE		Location of the truststore file for the server. If left undefined, the SECRETS_DIR will be checked for a truststore. If that truststore does not exist, the server will generate a truststore, containing its own certificate.
TRUSTSTORE_PIN_FILE		Location of the pin file for the truststore defined in TRUSTSTORE_FILE. You must specify a TRUSTSTORE_PIN_FILE when a TRUSTSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a truststore.
TRUSTSTORE_TYPE		Format of the truststore defined in TRUSTSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the truststore format will be inferred based on the file extension of the TRUSTSTORE_FILE, defaulting to "jks".
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/trace ${SERVER_ROOT_DIR}/logs/policy-decision ${SERVER_ROOT_DIR}/logs/ldap-access	Files tailed once container has started
PD_PROFILE	${STAGING_DIR}/pd.profile	Directory for the profile used by the PingData manage-profile tool
UNBOUNDID_SKIP_START_PRECHECK_NODETACH	true	Setting this variable to true speeds up server startup time by skipping an unnecessary JVM check.

ENV Variable	Default	Description
CERTIFICATE_NICKNAME		There is an additional certificate-based variable used to identify the certificate alias used within the `KEYSTORE_FILE`. That variable is called `CERTIFICATE_NICKNAME`, which identifies the certificate to use by the server in the `KEYSTORE_FILE`. If a value is not provided, the container will look at the list certs found in the `KEYSTORE_FILE` and if one - and only one - certificate is found of type `PrivateKeyEntry`, that alias will be used.
COLUMNS	120	Sets the number of columns in PingAuthorize command-line tool output

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- ${LDAP_PORT}
- ${LDAPS_PORT}
- ${HTTPS_PORT}
- ${JMX_PORT}

Running a PingAuthorize container

The easiest way to test a simple standalone image of PingAuthorize is to cut/paste the following command into a terminal on a machine with docker.

```
docker run \
         --name pingauthorize \
         --publish 1389:1389 \
         --publish 8443:1443 \
         --detach \
         --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
         --env SERVER_PROFILE_PATH=getting-started/pingauthorize \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
        pingidentity/pingauthorize:edge
```

You can view the Docker logs with the command:

```
docker logs -f pingauthorize
```

You should see the ouptut from a PingAuthorize install and configuration, ending with a message the the PingAuthorize has started. After it starts, you will see some typical access logs. Simply `Ctrl-C` after to stop tailing the logs.

Stopping/Removing the container

To stop the container:

```
docker container stop pingauthorize
```

To remove the container:

```
docker container rm -f pingauthorize
```

Docker Container Hook Scripts

Please go here for details on all pingauthorize hook scripts

This document is auto-generated from *pingauthorize/Dockerfile*

Ping Identity DevOps Docker Image - `pingauthorizepap`

Ping Identity DevOps Docker Image - `pingauthorizepap`

This docker image includes the Ping Identity PingAuthorize Policy Editor product binaries and associated hook scripts to create and run a PingAuthorize Policy Editor instance.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingdatacommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingAuthorize-PAP	Ping product name
LICENSE_DIR	${PD_LICENSE_DIR}	PD License directory. This value is set from the pingbase dockerfile
LICENSE_FILE_NAME	PingAuthorize.lic	Name of license File
LICENSE_SHORT_NAME	PingAuthorize	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
MAX_HEAP_SIZE	384m	Minimal Heap size required for PingAuthorize Policy Editor
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/start-server	The command that the entrypoint executes in the foreground to instantiate the container
STARTUP_FOREGROUND_OPTS	--nodetach	The command-line options to provide to the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the startup command when the container starts with the server in the background. This is the debug start flow for the container
KEYSTORE_FILE		Location of the keystore file containing the server certificate. If left undefined, the SECRETS_DIR will be checked for a keystore. If that keystore does not exist, the server will generate a self-signed certificate.

ENV Variable	Default	Description
KEYSTORE_PIN_FILE		Location of the pin file for the keystore defined in KEYSTORE_FILE. You must specify a KEYSTORE_PIN_FILE when a KEYSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a self-signed certificate.
KEYSTORE_TYPE		Format of the keystore defined in KEYSTORE_FILE. One of "jks" or "pkcs12". If not defined, the keystore format will be inferred based on the file extension of the KEYSTORE_FILE, defaulting to "jks".
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/authorize-pe.log ${SERVER_ROOT_DIR}/logs/management-audit.log ${SERVER_ROOT_DIR}/logs/policy-db.log ${SERVER_ROOT_DIR}/logs/setup.log ${SERVER_ROOT_DIR}/logs/start-server.log ${SERVER_ROOT_DIR}/logs/stop-server.log	Files tailed once container has started
REST_API_HOSTNAME	localhost	Hostname used for the REST API (deprecated, use `PING_EXTERNAL_BASE_URL` instead)
DECISION_POINT_SHARED_SECRET	2FederateM0re	Defines the shared secret between PAZ and the Policy Editor
PING_ENABLE_API_HTTP_CACHE	true	When set to `false`, disables the default HTTP API caching in the Policy Manager, Trust Framework, and Test Suite

ENV Variable	Default	Description
PING_POLICY_DB_SYNC		When set to `true`, the container uses the provided policy database admin credentials to either create a PostgreSQL policy database or upgrade it if it already exists. You must also provide the `PING_DB_CONNECTION_STRING`, `PING_DB_ADMIN_USERNAME`, `PING_DB_ADMIN_PASSWORD` `PING_DB_APP_USERNAME`, and `PING_DB_APP_PASSWORD` variables for this to work.
PING_DB_CONNECTION_STRING	jdbc:h2:file:./Symphonic;ALIAS_COLUMN_NAME=TRUE	The JDBC connection string used to connect to the policy database. Use the format `jdbc:postgresql://<host>:<port>/<database>`. Only H2 embedded files and PostgreSQL are supported.
PING_DB_ADMIN_USERNAME	sa	The database administration username to use when creating or upgrading a policy database.
PING_DB_ADMIN_PASSWORD	Passw0rd	The database administration password to use when creating or upgrading a policy database.
PING_DB_APP_USERNAME	pap_user	The username that the Policy Editor should use when accessing the policy database during server runtime.
PING_DB_APP_PASSWORD	Symphonic2014!	The password that the Policy Editor should use when accessing the policy database during server runtime.

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- ${HTTPS_PORT}

Running a PingAuthorize Policy Editor container

A PingAuthorize Policy Editor may be set up in one of two modes:

- **Demo mode**: Uses insecure username/password authentication.
- **OIDC mode**: Uses an OpenID Connect provider for authentication.

To run a PingAuthorize Policy Editor container in demo mode:

```
docker run \
         --name pingauthorizepap \
         --env PING_EXTERNAL_BASE_URL=my-pap-hostname:8443 \
         --publish 8443:1443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingauthorizepap:edge
```

Log in with:

- https://my-pap-hostname:8443/
- Username: admin
- Password: password123

To run a PingAuthorize Policy Editor container in OpenID Connect mode, specify the `PING_OIDC_CONFIGURATION_ENDPOINT` and `PING_CLIENT_ID` environment variables. To provide scopes other than the default (`oidc email profile`), specify the `PING_SCOPE` environment variable:

```
docker run \
         --name pingauthorizepap \
         --env PING_EXTERNAL_BASE_URL=my-pe-hostname:8443 \
         --env PING_OIDC_CONFIGURATION_ENDPOINT=https://my-oidc-provider/.well-known/openid-configuration \
         --env PING_CLIENT_ID=b1929abc-e108-4b4f-83d467059fa1 \
         --env PING_SCOPE="oidc email profile phone" \
         --publish 8443:1443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingauthorizepap:edge
```

Note

If both `PING_OIDC_CONFIGURATION_ENDPOINT` and `PING_CLIENT_ID` are not specified, then Docker sets up the PingAuthorize Policy Editor container in demo mode.

Log in with:

- https://my-pap-hostname:8443/
- Provide credentials as prompted by the OIDC provider

Follow Docker logs with:

```
docker logs -f pingauthorizepap
```

Specifying the external hostname and port

The Policy Editor consists of a client-side application that runs in the user's web browser and a backend REST API service that runs within the container. So that the client-side application can successfully make API calls to the backend, the Policy Editor must be configured with an externally accessible hostname:port. If the Policy Editor is configured in OIDC mode, then the external hostname:port pair is also needed so that the Policy Editor can correctly generate its OIDC redirect URI.

Use the `PING_EXTERNAL_BASE_URL` environment variable to specify the Policy Editor's external hostname and port using the form `hostname[:port]`, where `hostname` is the hostname of the Docker host and `port` is the Policy Editor container's published port. If the published port is 443, then it should be omitted.

For example:

```
docker run \
         --name pingauthorizepap \
         --env PING_EXTERNAL_BASE_URL=my-pap-hostname:8443 \
         --publish 8443:1443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingauthorizepap:edge
```

Changing the default periodic database backup schedule and location

The PAP performs periodic backups of the policy database. The results are placed in the `policy-backup` directory underneath the instance root.

Use the `PING_BACKUP_SCHEDULE` environment variable to specify the PAP's periodic database backup schedule in the form of a cron expression. The cron expression evaluates against the container timezone, UTC. Use the `PING_H2_BACKUP_DIR` environment variable to change the backup output directory.

For example, to perform backups daily at UTC noon and place backups in `/opt/out/backup`:

```
docker run \
         --name pingauthorizepap \
         --env PING_EXTERNAL_BASE_URL=my-pap-hostname:8443 \
         --env PING_BACKUP_SCHEDULE="0 0 12 * * ?" \
         --env PING_H2_BACKUP_DIR=/opt/out/backup \
         --publish 8443:1443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingauthorizepap:edge
```

Creating and upgrading a PostgreSQL policy database

Although the Policy Editor uses an embedded H2 file for its policy database implementation by default, it also has the capability to use a PostgreSQL database. However, this database must first be initialized with database objects.

Set the `PING_POLICY_DB_SYNC` environment variable to `true`, provide the PostgreSQL JDBC connection string in `PING_DB_CONNECTION_STRING`, the database administration credentials through `PING_DB_ADMIN_USERNAME` and `PING_DB_ADMIN_PASSWORD`, and the server runtime credentials through `PING_DB_APP_USERNAME` and `PING_DB_APP_PASSWORD` to indicate that pingauthorizepap should create the necessary policy database objects.

Similarly, use the same environment variables with the same values when a new version of the application is released. pingauthorizepap will use the database administration user to perform any necessary upgrades to the database objects.

In both scenarios, the database administration user and the server runtime user must exist and be able to sign into the PostgreSQL server. In the create scenario, the database administration user must be able to create databases. In the upgrade scenario, the database administration user must own the database objects. Therefore, it is advisable to continually provide the same administration credentials during creation and upgrades to prevent permissions issues.

For example, assume that a system administrator has created a PostgreSQL username `pap_admin` that can sign into a PostgreSQL server hosted at example.com and listening on port 5432. They have also created the runtime user `pap_user`.

To create the database objects under the database named `my_pap_db` and start the Policy Editor, use the following command:

```
docker run \
         --name pingauthorizepap \
         --env PING_EXTERNAL_BASE_URL=my-pap-hostname:8443 \
         --env PING_BACKUP_SCHEDULE="0 0 12 * * ?" \
         --env PING_H2_BACKUP_DIR=/opt/out/backup \
         --publish 8443:1443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingauthorizepap:edge
```

Use the same command when a new pingauthorizepap release requires an upgrade to the policy database schema.

Note that `PING_DB_ADMIN_PASSWORD` and `PING_DB_APP_PASSWORD` are only provided on the command line for illustrative purposes and can instead be provided through a Vault or through a `/run/secrets` `.env` file.

Docker Container Hook Scripts

Please go here for details on all pingauthorizepap hook scripts

This document is auto-generated from *pingauthorizepap/Dockerfile*

Ping Identity DevOps Docker Image - `pingbase`

Ping Identity Docker Image - `pingbase`

This docker image provides a base image for all Ping Identity DevOps product images.

Environment Variables

The following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
BASE	${BASE:-/opt}	Location of the top level directory where everything is located in image/ container
ROOT_USER	administrator	the default administrative user for PingData
JAVA_HOME	/opt/java	
STAGING_DIR	${BASE}/staging	Path to the staging area where the remote and local server profiles can be merged
OUT_DIR	${BASE}/out	Path to the runtime volume
SERVER_ROOT_DIR	${OUT_DIR}/instance	Path from which the runtime executes
IN_DIR	${BASE}/in	Location of a local server-profile volume
SERVER_BITS_DIR	${BASE}/server	Path to the server bits
BAK_DIR	${BASE}/backup	Path to a volume generically used to export or backup data
LOGS_DIR	${BASE}/logs	Path to a volume generically used for logging
PING_IDENTITY_ACCEPT_EULA	NO	Must be set to 'YES' for the container to start
PING_IDENTITY_DEVOPS_FILE	devops-secret	File name for devops-creds passed as a Docker secret
STAGING_MANIFEST	${BASE}/staging-manifest.txt	Path to a manifest of files expected in the staging dir on first image startup

ENV Variable	Default	Description
CLEAN_STAGING_DIR	false	Whether to clean the staging dir when the image starts
SECRETS_DIR	/run/secrets	Default path to the secrets
TOPOLOGY_FILE	${STAGING_DIR}/topology.json	Path to the topology file
HOOKS_DIR	${STAGING_DIR}/hooks	Path where all the hooks scripts are stored
CONTAINER_ENV	${STAGING_DIR}/.env	Environment Property file use to share variables between scripts in container
SERVER_PROFILE_DIR	/tmp/server-profile	Path where the remote server profile is checked out or cloned before being staged prior to being applied on the runtime
SERVER_PROFILE_URL		A valid git HTTPS URL (not ssh)
SERVER_PROFILE_URL_REDACT	true	When set to "true", the server profile git URL will not be printed to container output.
SERVER_PROFILE_BRANCH		A valid git branch (optional)
SERVER_PROFILE_PATH		The subdirectory in the git repo
SERVER_PROFILE_UPDATE	false	Whether to update the server profile upon container restart
SECURITY_CHECKS_STRICT	false	Requires strict checks on security
SECURITY_CHECKS_FILENAME	.jwk .pin	Perform a check for filenames that may violate security (i.e. secret material)
UNSAFE_CONTINUE_ON_ERROR		If this is set to true, then the container will provide a hard warning and continue.
LICENSE_DIR	${SERVER_ROOT_DIR}	License directory
PD_LICENSE_DIR	${STAGING_DIR}/pd.profile/server-root/pre-setup	PD License directory. Separating from above LICENSE_DIR to differentiate for different products

ENV Variable	Default	Description
STARTUP_COMMAND		The command that the entrypoint will execute in the foreground to instantiate the container
STARTUP_FOREGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container
PING_IDENTITY_DEVOPS_KEY_REDACT	true	
TAIL_LOG_FILES		A whitespace separated list of log files to tail to the container standard output - DO NOT USE WILDCARDS like /path/to/logs/*.log
COLORIZE_LOGS	true	If 'true', the output logs will be colorized with GREENs and REDs, otherwise, no colorization will be done. This is good for tools that monitor logs and colorization gets in the way.
LOCATION	Docker	Location default value If PingDirectory is deployed in multi cluster mode, that is, K8S_CLUSTER, K8S_CLUSTERS and K8S_SEED_CLUSTER are defined, LOCATION is ignored and K8S_CLUSTER is used as the location
LOCATION_VALIDATION	true	Any string denoting a logical/physical location
MAX_HEAP_SIZE	384m	Heap size (for java products)
JVM_TUNING	AGGRESSIVE	

ENV Variable	Default	Description
JAVA_RAM_PERCENTAGE	75.0	Percentage of the container memory to allocate to PingFederate JVM DO NOT set to 100% or your JVM will exit with OutOfMemory errors and the container will terminate
VERBOSE	false	Triggers verbose messages in scripts using the set -x option.
PING_DEBUG	false	Set the server in debug mode, with increased output
PING_PRODUCT		The name of Ping product, i.e. PingFederate, PingDirectory - must be a valid Ping product type. This variable should be overridden by child images.
PING_PRODUCT_VALIDATION	true	i.e. PingFederate,PingDirectory
ADDITIONAL_SETUP_ARGS		List of setup arguments passed to Ping Data setup-arguments.txt file
LDAP_PORT	1389	Port over which to communicate for LDAP
LDAPS_PORT	1636	Port over which to communicate for LDAPS
HTTPS_PORT	1443	Port over which to communicate for HTTPS
JMX_PORT	1689	Port for monitoring over JMX protocol
ORCHESTRATION_TYPE		The type of orchestration tool used to run the container, normally set in the deployment (.yaml) file. Expected values include: - compose - swarm - kubernetes Defaults to blank (i.e. No type is set)
USER_BASE_DN	dc=example,dc=com	Base DN for user data
DOLLAR	'$'	Variable with a literal value of '$', to avoid unwanted variable substitution
PD_ENGINE_PUBLIC_HOSTNAME	localhost	PD (PingDirectory) public hostname that may be used in redirects

ENV Variable	Default	Description
PD_ENGINE_PRIVATE_HOSTNAME	pingdirectory	PD (PingDirectory) private hostname
PDP_ENGINE_PUBLIC_HOSTNAME	localhost	PDP (PingDirectoryProxy) public hostname that may be used in redirects
PDP_ENGINE_PRIVATE_HOSTNAME	pingdirectoryproxy	PDP (PingDirectoryProxy) private hostname
PDS_ENGINE_PUBLIC_HOSTNAME	localhost	PDS (PingDataSync) public hostname that may be used in redirects
PDS_ENGINE_PRIVATE_HOSTNAME	pingdatasync	PDS (PingDataSync) private hostname
PAZ_ENGINE_PUBLIC_HOSTNAME	localhost	PAZ (PingAuthorize) public hostname that may be used in redirects
PAZ_ENGINE_PRIVATE_HOSTNAME	pingauthorize	PAZ (PingAuthorize) private hostname
PAZP_ENGINE_PUBLIC_HOSTNAME	localhost	PAZP (PingAuthorize-PAP) public hostname that may be used in redirects
PAZP_ENGINE_PRIVATE_HOSTNAME	pingauthorizepap	PAZP (PingAuthorize-PAP) private hostname
PF_ENGINE_PUBLIC_HOSTNAME	localhost	PF (PingFederate) engine public hostname that may be used in redirects
PF_ENGINE_PRIVATE_HOSTNAME	pingfederate	PF (PingFederate) engine private hostname
PF_ADMIN_PUBLIC_BASEURL	https://localhost:9999	PF (PingFederate) admin public baseurl that may be used in redirects
PF_ADMIN_PUBLIC_HOSTNAME	localhost	PF (PingFederate) admin public hostname that may be used in redirects
PF_ADMIN_PRIVATE_HOSTNAME	pingfederate-admin	PF (PingFederate) admin private hostname
PA_ENGINE_PUBLIC_HOSTNAME	localhost	PA (PingAccess) engine public hostname that may be used in redirects
PA_ENGINE_PRIVATE_HOSTNAME	pingaccess	PA (PingAccess) engine private hostname
PA_ADMIN_PUBLIC_HOSTNAME	localhost	PA (PingAccess) admin public hostname that may be used in redirects

ENV Variable	Default	Description
PA_ADMIN_PRIVATE_HOSTNAME	pingaccess-admin	PA (PingAccess) admin private hostname
ROOT_USER_DN	cn=${ROOT_USER}	DN of the server root user
ENV	${BASE}/.profile	
PS1	\${PING_PRODUCT}:\h:\w\n>	Default shell prompt (i.e. productName:hostname:workingDir)
PATH	${JAVA_HOME}/bin:${BASE}:${SERVER_ROOT_DIR}/bin:${PATH}	PATH used by the container

Docker Container Hook Scripts

Please go here for details on all pingbase hook scripts

This document is auto-generated from *pingbase/Dockerfile*

Ping Identity DevOps Docker Image - `pingcentral`

Ping Identity DevOps Docker Image - `pingcentral`

This docker image includes the Ping Identity PingCentral product binaries and associated hook scripts to create and run PingCentral in a container.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingcommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	

ENV Variable	Default	Description
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_CENTRAL_SERVER_PORT	9022	
PING_PRODUCT	PingCentral	Ping product name
LICENSE_DIR	${SERVER_ROOT_DIR}/conf	License directory
LICENSE_FILE_NAME	pingcentral.lic	Name of license file
LICENSE_SHORT_NAME	PC	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/run.sh	The command that the entrypoint will execute in the foreground to instantiate the container
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/log/application.log	Files tailed once container has started
PING_CENTRAL_LOG_LEVEL	INFO	
PING_CENTRAL_BLIND_TRUST	false	
PING_CENTRAL_VERIFY_HOSTNAME	true	

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- 9022

Running a PingCentral container

To run a PingCentral container with your devops configuration file:

```
docker run -Pt \ --name pingcentral \ --env-file ~/.pingidentity/config \ --env PING_IDENTITY_ACCEPT_EULA=YES \ --env
PING_IDENTITY_DEVOPS_USER \ --env PING_IDENTITY_DEVOPS_KEY \ --tmpfs /run/secrets \ pingidentity/pingcentral:edge
```

or with long options in the background:

```
docker run \
           --name pingcentral \
           --publish 9022:9022 \
           --detach \
           --env-file ~/.pingidentity/config \
           --env PING_IDENTITY_ACCEPT_EULA=YES \
           --env PING_IDENTITY_DEVOPS_USER \
           --env PING_IDENTITY_DEVOPS_KEY \
           --tmpfs /run/secrets \
           pingidentity/pingcentral:edge
```

or if you want to specify everything yourself:

```
  docker run \
           --name pingcentral \
           --publish 9022:9022 \
           --detach \
           --env PING_IDENTITY_ACCEPT_EULA=YES \
           --env PING_IDENTITY_DEVOPS_USER \
           --env PING_IDENTITY_DEVOPS_KEY \
           --tmpfs /run/secrets \
           pingidentity/pingcentral:edge
```

Follow Docker logs with:

```
docker logs -f pingcentral
```

If using the command above with the embedded server profile, log in with: * https://localhost:9022/ * Username: Administrator * Password: 2Federate

Docker Container Hook Scripts

Please go here for details on all pingcentral hook scripts

This document is auto-generated from *pingcentral/Dockerfile*

Ping Identity DevOps Docker Image - `pingcommon`

Ping Identity Docker Image - `pingcommon`

This docker image provides a busybox image to house the base hook scripts and default entrypoint.sh used throughout the Ping Identity DevOps product images.

Docker Container Hook Scripts

Please go here for details on all pingcommon hook scripts

This document is auto-generated from *pingcommon/Dockerfile*

Ping Identity DevOps Docker Image - `pingdatacommon`

Ping Identity Docker Image - `pingdatacommon`

This docker image provides a busybox image based off of `pingidentity/pingcommon` to house the base hook scripts used throughout the Ping Identity DevOps PingData product images.

Related Docker Images

- `pingidentity/pingcommon` - Parent Image

Environment Variables

The following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
REGENERATE_JAVA_PROPERTIES	false	Flag to force a run of dsjavaproperties --initialize. When this is false, the java.properties file will only be regenerated on a restart when there is a change in JVM or a change in the product-specific java options, such as changing the MAX_HEAP_SIZE value.

Docker Container Hook Scripts

Please go here for details on all pingdatacommon hook scripts

---

This document is auto-generated from *pingdatacommon/Dockerfile*

Ping Identity DevOps Docker Image - `pingdataconsole`

Ping Identity Docker Image - `pingdataconsole`

This docker image provides a tomcat image with the PingDataConsole deployed to be used in configuration of the PingData products.

Related Docker Images

- `tomcat:9-jre8` - Tomcat engine to serve PingDataConsole .war file

Environment Variables

The following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingDataConsole	Ping product name
HTTP_PORT	8080	PingDataConsole HTTP listen port
HTTPS_PORT	8443	PingDataConsole HTTPS listen port
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/catalina.sh	The command that the entrypoint will execute in the foreground to instantiate the container
STARTUP_FOREGROUND_OPTS	run	The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS	start	The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container

ENV Variable	Default	Description
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/console.log	Files tailed once container has started

Run

To run a PingDataConsole container:

```
  docker run \
           --name pingdataconsole \
           --publish ${HTTPS_PORT}:${HTTPS_PORT} \
           --detach \
           --env PING_IDENTITY_ACCEPT_EULA=YES \
           --env PING_IDENTITY_DEVOPS_USER \
           --env PING_IDENTITY_DEVOPS_KEY \
           --tmpfs /run/secrets \
           pingidentity/pingdataconsole:edge
```

Follow Docker logs with:

```
docker logs -f pingdataconsole
```

If using the command above with the embedded server profile, log in with: * http://localhost:$\{HTTPS_PORT}/console/login

```
Server: pingdirectory:1636
Username: administrator
Password: 2FederateM0re
```

> *make sure you have a PingDirectory running*

Docker Container Hook Scripts

Please go here for details on all pingdataconsole hook scripts

---

This document is auto-generated from *pingdataconsole/Dockerfile*

Ping Identity DevOps Docker Image - `pingdatasync`

Ping Identity DevOps Docker Image - `pingdatasync`

This docker image includes the Ping Identity PingDataSync product binaries and associated hook scripts to create and run a PingDataSync instance.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingdatacommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/sync	Files tailed once container has started
LICENSE_DIR	${PD_LICENSE_DIR}	PD License directory. This value is set from the pingbase docker file
LICENSE_FILE_NAME	PingDirectory.lic	Name of license file
LICENSE_SHORT_NAME	PD	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
PING_PRODUCT	PingDataSync	Ping product name
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/start-server	The command that the entrypoint will execute in the foreground to instantiate the container

ENV Variable	Default	Description
STARTUP_FOREGROUND_OPTS	--nodetach	The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
RETRY_TIMEOUT_SECONDS	180	The default retry timeout in seconds for manage-topology and remove-defunct-server
ADMIN_USER_NAME	admin	Failover administrative user
ROOT_USER_PASSWORD_FILE		Location of file with the root user password (i.e. cn=directory manager). Defaults to / SECRETS_DIR/root-user-password
ADMIN_USER_PASSWORD_FILE		Location of file with the admin password, used as the password replication admin Defaults to / SECRETS_DIR/admin-user-password
KEYSTORE_FILE		Location of the keystore file containing the server certificate. If left undefined, the SECRETS_DIR will be checked for a keystore. If that keystore does not exist, the server will generate a self-signed certificate.
KEYSTORE_PIN_FILE		Location of the pin file for the keystore defined in KEYSTORE_FILE. You must specify a KEYSTORE_PIN_FILE when a KEYSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a self-signed certificate.
KEYSTORE_TYPE		Format of the keystore defined in KEYSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the keystore format will be inferred based on the file extension of the KEYSTORE_FILE, defaulting to "jks".

ENV Variable	Default	Description
TRUSTSTORE_FILE		Location of the truststore file for the server. If left undefined, the SECRETS_DIR will be checked for a truststore. If that truststore does not exist, the server will generate a truststore, containing its own certificate.
TRUSTSTORE_PIN_FILE		Location of the pin file for the truststore defined in TRUSTSTORE_FILE. You must specify a TRUSTSTORE_PIN_FILE when a TRUSTSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a truststore.
TRUSTSTORE_TYPE		Format of the truststore defined in TRUSTSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the truststore format will be inferred based on the file extension of the TRUSTSTORE_FILE, defaulting to "jks".
PD_PROFILE	${STAGING_DIR}/pd.profile	Directory for the profile used by the PingData manage-profile tool
UNBOUNDID_SKIP_START_PRECHECK_NODETACH	true	Setting this variable to true speeds up server startup time by skipping an unnecessary JVM check.

ENV Variable	Default	Description
PARALLEL_POD_MANAGEMENT_POLICY	false	Whether this container is running as a Pod in a Kubernetes StatefulSet, and that StatefulSet is using the Parallel podManagementPolicy. This property allows for starting up Pods in parallel to speed up the initial startup of PingDataSync topologies. This variable must be set to true when using the Parallel podManagementPolicy. Note: when using parallel startup, ensure the RETRY_TIMEOUT_SECONDS variable is large enough. The pods will be enabling replication simultaneously, so some pods will have to retry while waiting for others to complete. If the timeout is too low, a Pod may end up restarting unnecessarily.
SKIP_WAIT_FOR_DNS	false	Set to true to skip the waiting for DNS step that is normally done just before attempting to join the topology.
CERTIFICATE_NICKNAME		There is an additional certificate-based variable used to identity the certificate alias used within the `KEYSTORE_FILE`. That variable is called `CERTIFICATE_NICKNAME`, which identifies the certificate to use by the server in the `KEYSTORE_FILE`. If a value is not provided, the container will look at the list certs found in the `KEYSTORE_FILE` and if one - and only one - certificate is found of type `PrivateKeyEntry`, that alias will be used.
COLUMNS	120	Sets the number of columns in PingDataSync command-line tool output

ENV Variable	Default	Description
PD_REBUILD_ON_RESTART	false	Force a rebuild (replace-profile) of PingDataSync on restart. Used to ensure that the server configuration exactly matches the server profile. This variable will slow down startup times and should only be used when necessary.

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- ${LDAP_PORT}
- ${LDAPS_PORT}
- ${HTTPS_PORT}
- ${JMX_PORT}

Running a PingDataSync container

```
docker run \
         --name pingdatasync \
         --publish 1389:1389 \
         --publish 8443:1443 \
         --detach \
         --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
         --env SERVER_PROFILE_PATH=simple-sync/pingdatasync \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingdatasync:edge
```

Docker Container Hook Scripts

Please go here for details on all pingdatasync hook scripts

This document is auto-generated from *pingdatasync/Dockerfile*

Ping Identity DevOps Docker Image - `pingdelegator`

Ping Identity Docker Image - `pingdelegator`

This docker image provides an NGINX instance with PingDelegator that can be used in administering PingDirectory Users/Groups.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingcommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PD_DELEGATOR_PUBLIC_HOSTNAME	localhost	
PD_DELEGATOR_HTTP_PORT	6080	
PD_DELEGATOR_HTTPS_PORT	6443	
PING_PRODUCT	PingDelegator	Ping product name
PF_ENGINE_PUBLIC_HOSTNAME	localhost	The hostname for the public Ping Federate instance used for SSO.
PF_ENGINE_PUBLIC_PORT	9031	The port for the public Ping Federate instance used for SSO. NOTE: If using port 443 along with a base URL with no specified port, set to an empty string.
PF_DELEGATOR_CLIENTID	dadmin	The client id that was set up with Ping Federate for Ping Delegator.
PD_ENGINE_PUBLIC_HOSTNAME	localhost	The hostname for the DS instance the app will be interfacing with.

ENV Variable	Default	Description
PD_ENGINE_PUBLIC_PORT	1443	The HTTPS port for the DS instance the app will be interfacing with.
PD_DELEGATOR_TIMEOUT_LENGTH_MINS	30	The length of time (in minutes) until the session will require a new login attempt
PD_DELEGATOR_HEADER_BAR_LOGO		The filename used as the logo in the header bar, relative to this application's build directory. Note about logos: The size of the image will be scaled down to fit 22px of height and a max-width of 150px. For best results, it is advised to make the image close to this height and width ratio as well as to crop out any blank spacing around the logo to maximize its presentation. e.g. '${SERVER_ROOT_DIR}/html/delegator/images/my_company_logo.png'
PD_DELEGATOR_DADMIN_API_NAMESPACE		The namespace for the Delegated Admin API on the DS instance. In most cases, this does not need to be set here. e.g. 'dadmin/v2'
PD_DELEGATOR_PROFILE_SCOPE_ENABLED	false	Set to true if the "profile" scope is supported for the Delegated Admin OIDC client on PingFederate and you wish to use it to show the current user's name in the navigation.
NGINX_WORKER_PROCESSES	auto	The number of NginX worker processes — Default: auto
NGINX_WORKER_CONNECTIONS	1024	The number of NginX worker connections — Default: 1024
STARTUP_COMMAND	nginx	The command that the entrypoint will execute in the foreground to instantiate the container
STARTUP_FOREGROUND_OPTS	-c ${SERVER_ROOT_DIR}/etc/nginx.conf	The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container

ENV Variable	Default	Description
STARTUP_BACKGROUND_OPTS	${STARTUP_FOREGROUND_OPTS}	The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container

Run

To run a PingDelegator container with HTTPS_PORT=6443 (6443 is simply a convention for PingDelegator so conflicts are reduced with other container HTTPS ports):

```
docker run \
         --name pingdelegator \
         --publish 6443:6443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingdelegator:edge
```

PingDelegator requires running instances of PingFederate/PingDirectory.

Docker Container Hook Scripts

Please go here for details on all pingdelegator hook scripts

This document is auto-generated from *pingdelegator/Dockerfile*

Ping Identity DevOps Docker Image - `pingdirectory`

Ping Identity DevOps Docker Image - `pingdirectory`

This docker image includes the Ping Identity PingDirectory product binaries and associated hook scripts to create and run a PingDirectory instance or instances.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

  > *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingdatacommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingDirectory	Ping product name
LICENSE_DIR	${PD_LICENSE_DIR}	PD License directory. This value is set from the pingbase docker file
LICENSE_FILE_NAME	PingDirectory.lic	Name of license File
LICENSE_SHORT_NAME	PD	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
REPLICATION_PORT	8989	Default PingDirectory Replication Port
ADMIN_USER_NAME	admin	Replication administrative user
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/start-server	The command that the entrypoint will execute in the foreground to instantiate the container
PD_DELEGATOR_PUBLIC_HOSTNAME	localhost	Public hostname of the DA app
STARTUP_FOREGROUND_OPTS	--nodetach	The command-line options to provide to the startup command when the container starts with the server in the foreground. This is the normal start flow for the container

ENV Variable	Default	Description
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container
ROOT_USER_PASSWORD_FILE		Location of file with the root user password (i.e. cn=directory manager). Defaults to /SECRETS_DIR/root-user-password
ADMIN_USER_PASSWORD_FILE		Location of file with the admin password, used as the password replication admin Defaults to /SECRETS_DIR/admin-user-password
ENCRYPTION_PASSWORD_FILE		Location of file with the passphrase for setting up encryption Defaults to /SECRETS_DIR/encryption-password
KEYSTORE_FILE		Location of the keystore file containing the server certificate. If left undefined, the SECRETS_DIR will be checked for a keystore. If that keystore does not exist, the server will generate a self-signed certificate.
KEYSTORE_PIN_FILE		Location of the pin file for the keystore defined in KEYSTORE_FILE. You must specify a KEYSTORE_PIN_FILE when a KEYSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a self-signed certificate.
KEYSTORE_TYPE		Format of the keystore defined in KEYSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the keystore format will be inferred based on the file extension of the KEYSTORE_FILE, defaulting to "jks".

ENV Variable	Default	Description
TRUSTSTORE_FILE		Location of the truststore file for the server. If left undefined, the SECRETS_DIR will be checked for a truststore. If that truststore does not exist, the server will generate a truststore, containing its own certificate.
TRUSTSTORE_PIN_FILE		Location of the pin file for the truststore defined in TRUSTSTORE_FILE. You must specify a TRUSTSTORE_PIN_FILE when a TRUSTSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a truststore.
TRUSTSTORE_TYPE		Format of the truststore defined in TRUSTSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the truststore format will be inferred based on the file extension of the TRUSTSTORE_FILE, defaulting to "jks".
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/access ${SERVER_ROOT_DIR}/logs/errors ${SERVER_ROOT_DIR}/logs/failed-ops ${SERVER_ROOT_DIR}/logs/config-audit.log ${SERVER_ROOT_DIR}/logs/debug-trace ${SERVER_ROOT_DIR}/logs/debug-aci ${SERVER_ROOT_DIR}/logs/tools/.log ${SERVER_BITS_DIR}/logs/tools/.log	Files tailed once container has started
MAKELDIF_USERS	0	Number of users to auto-populate using make-ldif templates
RETRY_TIMEOUT_SECONDS	180	The default retry timeout in seconds for dsreplication and remove-defunct-server
PD_PROFILE	${STAGING_DIR}/pd.profile	Directory for the profile used by the PingData manage-profile tool

ENV Variable	Default	Description
FIPS_MODE_ON	false	Turns on FIPS mode (currently with the Bouncy Castle FIPS provider) set to exactly "true" lowercase to turn on set to anything else to turn off
FIPS_PROVIDER	BCFIPS	BCFIPS is the only provider currently supported — do not edit
PD_REBUILD_ON_RESTART	false	Force a rebuild (replace-profile) of a PingDirectoy on restart. Used to ensure that the server configuration exactly matches the server profile. This variable will slow down startup times and should only be used when necessary.
UNBOUNDID_SKIP_START_PRECHECK_NODETACH	true	Setting this variable to true speeds up server startup time by skipping an unnecessary JVM check.
REPLICATION_BASE_DNS		Base DNs to include when enabling replication, in addition to the always-included USER_BASE_DN. Multiple base DNs can be specified here, separated by a `;` character
RESTRICTED_BASE_DNS		Base DNs to set as --restricted when enabling replication. Multiple base DNs can be specified here, separated by a `;` character. See the product documentation for more information on how to configure entry balancing.

ENV Variable	Default	Description
PARALLEL_POD_MANAGEMENT_POLICY	false	Whether this container is running as a Pod in a Kubernetes StatefulSet, and that StatefulSet is using the Parallel podManagementPolicy. This property allows for starting up Pods in parallel to speed up the initial startup of PingDirectory topologies. This variable must be set to true when using the Parallel podManagementPolicy. Note: when using parallel startup, ensure the RETRY_TIMEOUT_SECONDS variable is large enough. The pods will be enabling replication simultaneously, so some pods will have to retry while waiting for others to complete. If the timeout is too low, a Pod may end up restarting unnecessarily.
SKIP_WAIT_FOR_DNS	false	Set to true to skip the waiting for DNS step that is normally done just before attempting to join the topology.
CERTIFICATE_NICKNAME		There is an additional certificate-based variable used to identity the certificate alias used within the `KEYSTORE_FILE`. That variable is called `CERTIFICATE_NICKNAME`, which identifies the certificate to use by the server in the `KEYSTORE_FILE`. If a value is not provided, the container will look at the list certs found in the `KEYSTORE_FILE` and if one - and only one - certificate is found of type `PrivateKeyEntry`, that alias will be used.
PD_FORCE_DATA_REIMPORT	false	Set to true to force PingDirectory to export and re-import its backend data on restart. Note that this process can take a long time for large backends.

ENV Variable	Default	Description
LOAD_BALANCING_ALGORITHM_NAMES		The load-balancing algorithm names to set for this server instance. This variable is only needed when enabling automatic server discovery with PingDirectoryProxy. Multiple algorithms can be specified here, separated by a `;` character
FAIL_ON_DISABLED_BASE_DN	false	Set to true to fail the container if it is found that replication is not enabled for the USER_BASE_DN during startup. If replication is not enabled for the DN but this variable is not set to true, then a warning will be printed, but the container will not fail.
FAIL_ON_UNSUCCESSFUL_REMOVE_DEFUNCT	false	Set to true to fail the container if it is found that a previous call to remove-defunct-server in the hook scripts failed. If there was a failure but this variable is not set to true, then a warning will be printed, but the container will not fail. Failure of remove-defunct-server is marked by a file at ${SERVER_ROOT_DIR}/logs/remove-defunct-server-marker. Logs for the tool can be found at ${SERVER_ROOT_DIR}/logs/tools/remove-defunct-server.log
COLUMNS	120	Sets the number of columns in PingDirectory command-line tool output

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- ${LDAP_PORT}
- ${LDAPS_PORT}
- ${HTTPS_PORT}
- ${JMX_PORT}

Running a PingDirectory container

The easiest way to test a simple standalone image of PingDirectory is to cut/paste the following command into a terminal on a machine with docker.

```
docker run \
         --name pingdirectory \
         --publish 1389:1389 \
         --publish 8443:1443 \
         --detach \
         --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
         --env SERVER_PROFILE_PATH=getting-started/pingdirectory \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingdirectory:edge
```

You can view the Docker logs with the command:

```
docker logs -f pingdirectory
```

You should see the ouptut from a PingDirectory install and configuration, ending with a message the the PingDirectory has started. After it starts, you will see some typical access logs. Simply `Ctrl-C` after to stop tailing the logs.

Running a sample 100/sec search rate test

With the PingDirectory running from the previous section, you can run a `searchrate` job that will send load to the directory at a rate if 100/sec using the following command.

```
docker exec -it pingdirectory \
      /opt/out/instance/bin/searchrate \
              -b dc=example,dc=com \
              --scope sub \
              --filter "(uid=user.[1-9])" \
              --attribute mail \
              --numThreads 2 \
              --ratePerSecond 100
```

Connecting with an LDAP Client

Connect an LDAP Client (such as Apache Directory Studio) to this container using the default ports and credentials

LDAP Port	1389
LDAP Base DN	dc=example,dc=com
Root Username	cn=administrator

Root Password	2FederateM0re

Stopping/Removing the container

To stop the container:

```
docker container stop pingdirectory
```

To remove the container:

```
docker container rm -f pingdirectory
```

Docker Container Hook Scripts

Please go here for details on all pingdirectory hook scripts

This document is auto-generated from *pingdirectory/Dockerfile*

Ping Identity DevOps Docker Image - `pingdirectoryproxy`

Ping Identity DevOps Docker Image - `pingdirectoryproxy`

This docker image includes the Ping Identity PingDirectoryProxy product binaries and associated hook scripts to create and run a PingDirectoryProxy instance or instances.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingdatacommon` - Common Ping files (i.e. hook scripts)\

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	

ENV Variable	Default	Description
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingDirectoryProxy	Ping product name
LICENSE_FILE_NAME	PingDirectory.lic	Name of license File
LICENSE_DIR	${PD_LICENSE_DIR}	PD License directory. This value is set from the pingbase docker file
LICENSE_SHORT_NAME	PD	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
ADMIN_USER_NAME	admin	Replication administrative user
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/start-server	The command that the entrypoint will execute in the foreground to instantiate the container
PD_DELEGATOR_PUBLIC_HOSTNAME	localhost	Public hostname of the DA app
STARTUP_FOREGROUND_OPTS	--nodetach	The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container
ROOT_USER_PASSWORD_FILE		Location of file with the root user password (i.e. cn=directory manager). Defaults to /SECRETS_DIR/root-user-password

ENV Variable	Default	Description
KEYSTORE_FILE		Location of the keystore file containing the server certificate. If left undefined, the SECRETS_DIR will be checked for a keystore. If that keystore does not exist, the server will generate a self-signed certificate.
KEYSTORE_PIN_FILE		Location of the pin file for the keystore defined in KEYSTORE_FILE. You must specify a KEYSTORE_PIN_FILE when a KEYSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a self-signed certificate.
KEYSTORE_TYPE		Format of the keystore defined in KEYSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the keystore format will be inferred based on the file extension of the KEYSTORE_FILE, defaulting to "jks".
TRUSTSTORE_FILE		Location of the truststore file for the server. If left undefined, the SECRETS_DIR will be checked for a truststore. If that truststore does not exist, the server will generate a truststore, containing its own certificate.
TRUSTSTORE_PIN_FILE		Location of the pin file for the truststore defined in TRUSTSTORE_FILE. You must specify a TRUSTSTORE_PIN_FILE when a TRUSTSTORE_FILE is present. This value does not need to be defined when allowing the server to generate a truststore.

ENV Variable	Default	Description
TRUSTSTORE_TYPE		Format of the truststore defined in TRUSTSTORE_FILE. One of "jks", "pkcs12", "pem", or "bcfks" (in FIPS mode). If not defined, the truststore format will be inferred based on the file extension of the TRUSTSTORE_FILE, defaulting to "jks".
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/access ${SERVER_ROOT_DIR}/logs/errors ${SERVER_ROOT_DIR}/logs/failed-ops ${SERVER_ROOT_DIR}/logs/config-audit.log ${SERVER_ROOT_DIR}/logs/tools/.log ${SERVER_BITS_DIR}/logs/tools/.log	Files tailed once container has started
PD_PROFILE	${STAGING_DIR}/pd.profile	Directory for the profile used by the PingData manage-profile tool
UNBOUNDID_SKIP_START_PRECHECK_NODETACH	true	Setting this variable to true speeds up server startup time by skipping an unnecessary JVM check.
CERTIFICATE_NICKNAME		There is an additional certificate-based variable used to identity the certificate alias used within the `KEYSTORE_FILE`. That variable is called `CERTIFICATE_NICKNAME`, which identifies the certificate to use by the server in the `KEYSTORE_FILE`. If a value is not provided, the container will look at the list certs found in the `KEYSTORE_FILE` and if one - and only one - certificate is found of type `PrivateKeyEntry`, that alias will be used.
RETRY_TIMEOUT_SECONDS	180	The default retry timeout in seconds for manage-topology and remove-defunct-server

ENV Variable	Default	Description
PINGDIRECTORY_HOSTNAME		Set this variable to configure Proxy for automatic server discovery with PingDirectory hostname JOIN_PD_TOPOLOGY must be enabled for PINGDIRECTORY_HOSTNAME to take effect
PINGDIRECTORY_LDAPS_PORT		Set this variable to configure Proxy for automatic server discovery with PingDirectory LDAPS port JOIN_PD_TOPOLOGY must be enabled for PINGDIRECTORY_LDAPS_PORT to take effect
JOIN_PD_TOPOLOGY	false	Setting this variable to true will configure proxy to join the topology of PingDirectory
COLUMNS	120	Sets the number of columns in PingDirectoryProxy command-line tool output

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- ${LDAP_PORT}
- ${LDAPS_PORT}
- ${HTTPS_PORT}
- ${JMX_PORT}

Running a PingDirectoryProxy container

The easiest way to test test a simple standalone image of PingDirectoryProxy is to cut/paste the following command into a terminal on a machine with docker.

```
docker exec -it pingdirectoryproxy \
       /opt/out/instance/bin/searchrate \
               -b dc=example,dc=com \
               --scope sub \
               --filter "(uid=user.[1-9])" \
               --attribute mail \
               --numThreads 2 \
               --ratePerSecond 100
```

You can view the Docker logs with the command:

```
docker logs -f pingdirectoryproxy
```

You should see the output from a PingDirectoryProxy install and configuration, ending with a message the the PingDirectoryProxy has started. After it starts, you will see some typical access logs. Simply `Ctrl-C` after to stop tailing the logs.

Running a sample 100/sec search rate test

With the PingDirectoryProxy running from the previous section, you can run a `searchrate` job that will send load to the directory at a rate if 100/sec using the following command.

```
docker exec -it pingdirectoryproxy \
       /opt/out/instance/bin/searchrate \
               -b dc=example,dc=com \
               --scope sub \
               --filter "(uid=user.[1-9])" \
               --attribute mail \
               --numThreads 2 \
               --ratePerSecond 100
```

Connecting with an LDAP Client

Connect an LDAP Client (such as Apache Directory Studio) to this container using the default ports and credentials

LDAP Port	1389
LDAP Base DN	dc=example,dc=com
Root Username	cn=administrator
Root Password	2FederateM0re

Stopping/Removing the container

To stop the container:

```
docker container stop pingdirectoryproxy
```

To remove the container:

```
docker container rm -f pingdirectoryproxy
```

Docker Container Hook Scripts

Please go here for details on all pingdirectoryproxy hook scripts

---

This document is auto-generated from *pingdirectoryproxy/Dockerfile*

Ping Identity DevOps Docker Image - `pingfederate`

Ping Identity DevOps Docker Image - `pingfederate`

This docker image includes the Ping Identity PingFederate product binaries and associated hook scripts to create and run both PingFederate Admin and Engine nodes.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingcommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
BASE	${BASE:-/opt}	
ROOT_USER	administrator	the default administrative user for PingData
JAVA_HOME	/opt/java	
STAGING_DIR	${BASE}/staging	Path to the staging area where the remote and local server profiles can be merged

ENV Variable	Default	Description
OUT_DIR	${BASE}/out	Path to the runtime volume
SERVER_ROOT_DIR	${OUT_DIR}/instance	Path from which the runtime executes
IN_DIR	${BASE}/in	Location of a local server-profile volume
SERVER_BITS_DIR	${BASE}/server	Path to the server bits
BAK_DIR	${BASE}/backup	Path to a volume generically used to export or backup data
LOGS_DIR	${BASE}/logs	Path to a volume generically used for logging
PING_IDENTITY_ACCEPT_EULA	NO	Must be set to 'YES' for the container to start
PING_IDENTITY_DEVOPS_FILE	devops-secret	File name for devops-creds passed as a Docker secret
STAGING_MANIFEST	${BASE}/staging-manifest.txt	Path to a manifest of files expected in the staging dir on first image startup
CLEAN_STAGING_DIR	false	Whether to clean the staging dir when the image starts
SECRETS_DIR	/run/secrets	Default path to the secrets
TOPOLOGY_FILE	${STAGING_DIR}/topology.json	Path to the topology file
HOOKS_DIR	${STAGING_DIR}/hooks	Path where all the hooks scripts are stored
CONTAINER_ENV	${STAGING_DIR}/.env	Environment Property file use to share variables between scripts in container
SERVER_PROFILE_DIR	/tmp/server-profile	Path where the remote server profile is checked out or cloned before being staged prior to being applied on the runtime
SERVER_PROFILE_URL		A valid git HTTPS URL (not ssh)
SERVER_PROFILE_URL_REDACT	true	When set to "true", the server profile git URL will not be printed to container output.
SERVER_PROFILE_BRANCH		A valid git branch (optional)
SERVER_PROFILE_PATH		The subdirectory in the git repo
SERVER_PROFILE_UPDATE	false	Whether to update the server profile upon container restart

ENV Variable	Default	Description
SECURITY_CHECKS_STRICT	false	Requires strict checks on security
SECURITY_CHECKS_FILENAME	.jwk .pin	Perform a check for filenames that may violate security (i.e. secret material)
UNSAFE_CONTINUE_ON_ERROR		If this is set to true, then the container will provide a hard warning and continue.
LICENSE_DIR	${SERVER_ROOT_DIR}	License directory
PD_LICENSE_DIR	${STAGING_DIR}/pd.profile/server-root/pre-setup	PD License directory. Separating from above LICENSE_DIR to differentiate for different products
STARTUP_FOREGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container
STARTUP_BACKGROUND_OPTS		The command-line options to provide to the the startup command when the container starts with the server in the background. This is the debug start flow for the container
PING_IDENTITY_DEVOPS_KEY_REDACT	true	
TAIL_LOG_FILES		A whitespace separated list of log files to tail to the container standard output - DO NOT USE WILDCARDS like /path/to/logs/*.log
COLORIZE_LOGS	true	If 'true', the output logs will be colorized with GREENs and REDs, otherwise, no colorization will be done. This is good for tools that monitor logs and colorization gets in the way.
LOCATION	Docker	Location default value If PingDirectory is deployed in multi cluster mode, that is, K8S_CLUSTER, K8S_CLUSTERS and K8S_SEED_CLUSTER are defined, LOCATION is ignored and K8S_CLUSTER is used as the location
LOCATION_VALIDATION	true	Any string denoting a logical/physical location
MAX_HEAP_SIZE	384m	Heap size (for java products)
JVM_TUNING	AGGRESSIVE	

ENV Variable	Default	Description
JAVA_RAM_PERCENTAGE	75.0	Percentage of the container memory to allocate to PingFederate JVM DO NOT set to 100% or your JVM will exit with OutOfMemory errors and the container will terminate
VERBOSE	false	Triggers verbose messages in scripts using the set -x option.
PING_DEBUG	false	Set the server in debug mode, with increased output
PING_PRODUCT		The name of Ping product, i.e. PingFederate, PingDirectory - must be a valid Ping product type. This variable should be overridden by child images.
PING_PRODUCT_VALIDATION	true	i.e. PingFederate,PingDirectory
ADDITIONAL_SETUP_ARGS		List of setup arguments passed to Ping Data setup-arguments.txt file
LDAP_PORT	1389	Port over which to communicate for LDAP
LDAPS_PORT	1636	Port over which to communicate for LDAPS
HTTPS_PORT	1443	Port over which to communicate for HTTPS
JMX_PORT	1689	Port for monitoring over JMX protocol
ORCHESTRATION_TYPE		The type of orchestration tool used to run the container, normally set in the deployment (.yaml) file. Expected values include: - compose - swarm - kubernetes Defaults to blank (i.e. No type is set)
USER_BASE_DN	dc=example,dc=com	Base DN for user data
DOLLAR	'$'	Variable with a literal value of '$', to avoid unwanted variable substitution
PD_ENGINE_PUBLIC_HOSTNAME	localhost	PD (PingDirectory) public hostname that may be used in redirects
PD_ENGINE_PRIVATE_HOSTNAME	pingdirectory	PD (PingDirectory) private hostname
PDP_ENGINE_PUBLIC_HOSTNAME	localhost	PDP (PingDirectoryProxy) public hostname that may be used in redirects
PDP_ENGINE_PRIVATE_HOSTNAME	pingdirectoryproxy	PDP (PingDirectoryProxy) private hostname

ENV Variable	Default	Description
PDS_ENGINE_PUBLIC_HOSTNAME	localhost	PDS (PingDataSync) public hostname that may be used in redirects
PDS_ENGINE_PRIVATE_HOSTNAME	pingdatasync	PDS (PingDataSync) private hostname
PAZ_ENGINE_PUBLIC_HOSTNAME	localhost	PAZ (PingAuthorize) public hostname that may be used in redirects
PAZ_ENGINE_PRIVATE_HOSTNAME	pingauthorize	PAZ (PingAuthorize) private hostname
PAZP_ENGINE_PUBLIC_HOSTNAME	localhost	PAZP (PingAuthorize-PAP) public hostname that may be used in redirects
PAZP_ENGINE_PRIVATE_HOSTNAME	pingauthorizepap	PAZP (PingAuthorize-PAP) private hostname
PF_ENGINE_PUBLIC_HOSTNAME	localhost	PF (PingFederate) engine public hostname that may be used in redirects
PF_ENGINE_PRIVATE_HOSTNAME	pingfederate	PF (PingFederate) engine private hostname
PF_ADMIN_PUBLIC_BASEURL	https://localhost:9999	PF (PingFederate) admin public baseurl that may be used in redirects. PF_RUN_PF_ADMIN_BASEURL will override this value for PingFederate 11.3 and later.
PF_ADMIN_PUBLIC_HOSTNAME	localhost	PF (PingFederate) admin public hostname that may be used in redirects. PF_RUN_PF_ADMIN_HOSTNAME will override this value for PingFederate 11.3 and later.
PF_ADMIN_PRIVATE_HOSTNAME	pingfederate-admin	PF (PingFederate) admin private hostname
PA_ENGINE_PUBLIC_HOSTNAME	localhost	PA (PingAccess) engine public hostname that may be used in redirects
PA_ENGINE_PRIVATE_HOSTNAME	pingaccess	PA (PingAccess) engine private hostname
PA_ADMIN_PUBLIC_HOSTNAME	localhost	PA (PingAccess) admin public hostname that may be used in redirects
PA_ADMIN_PRIVATE_HOSTNAME	pingaccess-admin	PA (PingAccess) admin private hostname
ROOT_USER_DN	cn=${ROOT_USER}	DN of the server root user
ENV	${BASE}/.profile	
PS1	\${PING_PRODUCT}:\h:\w\n>	Default shell prompt (i.e. productName:hostname:workingDir)

ENV Variable	Default	Description
PATH	${JAVA_HOME}/bin:${BASE}:${SERVER_ROOT_DIR}/bin:${PATH}	PATH used by the container
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingFederate	Ping product name
LICENSE_DIR	${SERVER_ROOT_DIR}/server/default/conf	License directory
LICENSE_FILE_NAME	pingfederate.lic	Name of license file
LICENSE_SHORT_NAME	PF	Short name used when retrieving license from License Server
LICENSE_VERSION	${LICENSE_VERSION}	Version used when retrieving license from License Server
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/run.sh	The command that the entrypoint will execute in the foreground to instantiate the container
PING_IDENTITY_PASSWORD	2FederateM0re	Specify a password for administrator user for interaction with admin API
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/log/server.log	Files tailed once container has started
PF_LOG_SIZE_MAX	10000 KB	Defines the log file size max for ALL appenders
PF_LOG_NUMBER	2	Defines the maximum of log files to retain upon rotation
PF_ADMIN_PORT	9999	Defines the port on which the PingFederate administrative console and API runs. PF_RUN_PF_ADMIN_HTTPS_PORT will override this for PingFederate 11.3 and later.

ENV Variable	Default	Description
PF_ENGINE_PORT	9031	Defines the port on which PingFederate listens for encrypted HTTPS (SSL/TLS) traffic. PF_RUN_PF_HTTPS_PORT will override this for PingFederate 11.3 and later.
PF_ENGINE_SECONDARY_PORT	-1	Defines a secondary HTTPS port that can be used for mutual SSL/TLS (client X.509 certificate) authentication for both end users and protocol requests. PF_RUN_PF_SECONDARY_HTTPS_PORT (default 9032) will override this value. The default value of -1 disables the port in the product.
PF_ENGINE_DEBUG	false	Flag to turn on PingFederate Engine debugging Used in run.sh
PF_ADMIN_DEBUG	false	Flag to turn on PingFederate Admin debugging Used in run.sh
PF_DEBUG_PORT	9030	Defines the port on which PingFederate opens up a java debugging port. Used in run.sh
SHOW_LIBS_VER	true	Defines a variable to allow showing library versions in the output at startup default to true
SHOW_LIBS_VER_PRE_PATCH	false	Defines a variable to allow showing library version prior to patches being applied default to false This is helpful to ensure that the patch process updates all libraries affected
OPERATIONAL_MODE	STANDALONE	Operational Mode Indicates the operational mode of the runtime server in run.properties Options include STANDALONE, CLUSTERED_CONSOLE, CLUSTERED_ENGINE. PF_RUN_PF_OPERATIONAL_MODE will override this for PingFederate 11.3 and later.
PF_CONSOLE_AUTHENTICATION		Defines mechanism for console authentication in run.properties. Options include none, native, LDAP, cert, RADIUS, OIDC. If not set, default is native. PF_RUN_PF_CONSOLE_AUTHENTICATION will override this for PingFederate 11.3 and later.
PF_ADMIN_API_AUTHENTICATION		Defines mechanism for admin api authentication in run.properties. Options include none, native, LDAP, cert, RADIUS, OIDC. If not set, default is native. PF_RUN_PF_ADMIN_API_AUTHENTICATION will override this for PingFederate 11.3 and later.

ENV Variable	Default	Description
HSM_MODE	OFF	Hardware Security Module Mode in run.properties Options include OFF, AWSCLOUDHSM, NCIPHER, LUNA, BCFIPS. PF_RUN_PF_HSM_MODE will override this for PingFederate 11.3 and later.
PF_BC_FIPS_APPROVED_ONLY	false	Defines a variable that allows instantiating non-FIPS crypto/random
PF_HSM_HYBRID	false	Hardware Security Module Hybrid Mode When PF is in Hybrid mode, certs/keys can be created either on the local trust store or on the HSM. This can used as a migration strategy towards an HSM setup. PF_RUN_PF_HSM_HYBRID will override this for PingFederate 11.3 and later.
PF_LDAP_TYPE	PingDirectory	This is the type of the LDAP directory server. This property is needed by PingFederate to determine how to handle different implementations between the available LDAP directory server types. Valid options include: ActiveDirectory, SunDirectoryServer, OracleUnifiedDirectory, PingDirectory, and Generic.
PF_LDAP_USERNAME		This is the username for an account within the LDAP Directory Server that can be used to perform user lookups for authentication and other user level search operations. Set if PF_CONSOLE_AUTHENTICATION or PF_ADMIN_API_AUTHENTICATION=LDAP PF_LDAP_LDAP_USERNAME will override this for PingFederate 11.3 and later.
PF_LDAP_PASSWORD		This is the password for the Username specified above. This property should be obfuscated using the 'obfuscate.sh' utility. Set if PF_CONSOLE_AUTHENTICATION or PF_ADMIN_API_AUTHENTICATION=LDAP PF_LDAP_LDAP_PASSWORD will override this for PingFederate 11.3 and later.
CLUSTER_BIND_ADDRESS	NON_LOOPBACK	IP address for cluster communication. Set to NON_LOOPBACK to allow the system to choose an available non-loopback IP address. PF_RUN_PF_CLUSTER_BIND_ADDRESS will override this for PingFederate 11.3 and later.

ENV Variable	Default	Description
PF_PROVISIONER_MODE	OFF	Provisioner Mode in run.properties Options include OFF, STANDALONE, FAILOVER. PF_RUN_PF_PROVISIONER_MODE will override this for PingFederate 11.3 and later.
PF_PROVISIONER_NODE_ID	1	Provisioner Node ID in run.properties Initial active provisioning server node ID is 1 PF_RUN_PROVISIONER_NODE_ID will override this for PingFederate 11.3 and later.
PF_PROVISIONER_GRACE_PERIOD	600	Node group ID in cluster-adaptive.conf file. Does not require a .subst file. Provisioner Failover Grace Period in run.properties Grace period, in seconds. Default 600 seconds PF_RUN_PROVISIONER_FAILOVER_GRACE_PERIOD will override this for PingFederate 11.3 and later.
PF_JETTY_THREADS_MIN		Override the default value for the minimum size of the Jetty thread pool Leave unset to let the container automatically tune the value according to available resources PF_RUN_PF_RUNTIME_THREADS_MIN will override this for PingFederate 11.3 and later.
PF_JETTY_THREADS_MAX		Override the default value for the maximum size of the Jetty thread pool Leave unset to let the container automatically tune the value according to available resources PF_RUN_PF_RUNTIME_THREADS_MAX will override this for PingFederate 11.3 and later.
PF_ACCEPT_QUEUE_SIZE	512	The size of the accept queue. There is generally no reason to tune this but please refer to the performance tuning guide for further tuning guidance. PF_RUN_PF_RUNTIME_ACCEPTQUEUESIZE will override this for PingFederate 11.3 and later.
PF_PINGONE_REGION		The region of the PingOne tenant PingFederate should connect with. Valid values are "com", "eu" and "asia" PF_RUN_PF_PINGONE_ADMIN_URL_REGION will override this for PingFederate 11.3 and later.
PF_PINGONE_ENV_ID		The PingOne environment ID to use PF_RUN_PF_PINGONE_ADMIN_URL_ENVIRONMENT_ID will override this for PingFederate 11.3 and later.

ENV Variable	Default	Description
PF_CONSOLE_TITLE	Docker PingFederate	The title featured in the administration console — this is generally used to easily distinguish between environments PF_RUN_PF_CONSOLE_TITLE will override this for PingFederate 11.3 and later.
PF_NODE_TAGS		This property defines the tags associated with this PingFederate node. Configuration is optional. When configured, PingFederate takes this property into consideration when processing requests. For example, tags may be used to determine the data store location that this PingFederate node communicates with. Administrators may also use tags in conjunction with authentication selectors and policies to define authentication requirements. Administrators may define one tag or a list of space-separated tags. Each tag cannot contain any spaces. Other characters are allowed. Example 1: PF_NODE_TAGS=north Example 1 defines one tag: 'north' Example 2: PF_NODE_TAGS=1 123 test Example 2 defines three tags: '1', '123' and 'test' Example 3: PF_NODE_TAGS= Example 3 is also valid because the PF_NODE_TAGS property is optional. PF_RUN_NODE_TAGS will override this for PingFederate 11.3 and later.
PF_CONSOLE_ENV		This property defines the name of the PingFederate environment that will be displayed in the administrative console, used to make separate environments easily identifiable. PF_RUN_PF_CONSOLE_ENVIRONMENT will override this for PingFederate 11.3 and later.
JAVA_RAM_PERCENTAGE	75.0	Percentage of the container memory to allocate to PingFederate JVM DO NOT set to 100% or your JVM will exit with OutOfMemory errors and the container will terminate
BULK_CONFIG_DIR	${OUT_DIR}/instance/bulk-config	
BULK_CONFIG_FILE	data.json	
ADMIN_WAITFOR_TIMEOUT	300	wait-for timeout for 80-post-start.sh hook script How long to wait for the PF Admin console to be available

ENV Variable	Default	Description
CREATE_INITIAL_ADMIN_USER	false	Set to true to create the initial admin user after PingFederate starts up. The initial admin user will only be created on the first startup of the server after the license is accepted.
ENABLE_AUTOMATIC_HEAP_DUMP	true	Set to true to add the following Java flags and enable memory dumps -XX: +HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$PF_HOME_ESC/log"

Ports Exposed

The following ports are exposed from the container. If a variable is used, then it may come from a parent container

- 9031
- 9999

Running a PingFederate container

To run a PingFederate container:

```
docker run \
         --name pingfederate \
         --publish 9999:9999 \
         --detach \
         --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
         --env SERVER_PROFILE_PATH=getting-started/pingfederate \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER \
         --env PING_IDENTITY_DEVOPS_KEY \
         --tmpfs /run/secrets \
         pingidentity/pingfederate:edge
```

Follow Docker logs with:

```
docker logs -f pingfederate
```

If using the command above with the embedded server profile, log in with: * https://localhost:9999/pingfederate/app * Username: Administrator * Password: 2FederateM0re

Docker Container Hook Scripts

Please go here for details on all pingfederate hook scripts

This document is auto-generated from *pingfederate/Dockerfile*

Ping Identity DevOps Docker Image - `pingintelligence`

Ping Identity DevOps Docker Image - `pingintelligence-ase`

DEPRECATION NOTICE: As of July 2024, the PingIntelligence Docker Image is deprecated. No new image versions will be published. Existing versions will be available as indicated in the Docker Image Support Policy.

This docker image includes the Ping Identity PingIntelligence API Security Enforcer product binaries and associated hook scripts to create and run PingIntelligence ASE instances.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingcommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default
SHIM	${SHIM}
IMAGE_VERSION	${IMAGE_VERSION}
IMAGE_GIT_REV	${IMAGE_GIT_REV}
DATE	${DATE}
PING_PRODUCT_VERSION	${VERSION}
PING_PRODUCT	PingIntelligence_ASE
LICENSE_FILE_NAME	PingIntelligence.lic
LICENSE_DIR	${SERVER_ROOT_DIR}/config
LICENSE_SHORT_NAME	pingintelligence

ENV Variable	Default
LICENSE_VERSION	${LICENSE_VERSION}
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/start_ase.sh
STARTUP_FOREGROUND_OPTS	
STARTUP_BACKGROUND_OPTS	
ROOT_USER_PASSWORD_FILE	
ADMIN_USER_PASSWORD_FILE	
ENCRYPTION_PASSWORD_FILE	
PING_INTELLIGENCE_ADMIN_USER	admin
PING_INTELLIGENCE_ADMIN_PASSWORD	2FederateM0re
PING_INTELLIGENCE_ASE_HTTP_PORT	8000
PING_INTELLIGENCE_ASE_HTTPS_PORT	8443
PING_INTELLIGENCE_ASE_MGMT_PORT	8010
PING_INTELLIGENCE_ASE_TIMEZONE	utc
PING_INTELLIGENCE_ASE_ABS_PUBLISH	true

ENV Variable	Default
PING_INTELLIGENCE_ASE_ABS_PUBLISH_REQUEST_MINUTES	10
PING_INTELLIGENCE_ASE_MODE	sideband
PING_INTELLIGENCE_ASE_ENABLE_SIDEBAND_AUTHENTICATION	false
PING_INTELLIGENCE_ASE_HOSTNAME_REWRITE	false
PING_INTELLIGENCE_ASE_KEYSTORE_PASSWORD	OBF:AES:sRNp0W7sSi1zrReXeHodKQ:lXcvbBhKZgDTrjQOfOkzR2mpca
PING_INTELLIGENCE_ASE_ADMIN_LOG_LEVEL	4
PING_INTELLIGENCE_ASE_ENABLE_CLUSTER	false
PING_INTELLIGENCE_ASE_SYSLOG_SERVER	
PING_INTELLIGENCE_ASE_CA_CERT_PATH	
PING_INTELLIGENCE_ASE_ENABLE_HEALTH	false
PING_INTELLIGENCE_ASE_ENABLE_ABS	true
PING_INTELLIGENCE_ASE_ENABLE_ABS_ATTACK_LIST_RETRIEVAL	true
PING_INTELLIGENCE_ASE_BLOCK_AUTODETECTED_ATTACKS	false
PING_INTELLIGENCE_ASE_ATTACK_LIST_REFRESH_MINUTES	10
PING_INTELLIGENCE_ASE_HOSTNAME_REFRESH_SECONDS	60
PING_INTELLIGENCE_ASE_DECOY_ALERT_INTERVAL_MINUTES	180
PING_INTELLIGENCE_ASE_ENABLE_XFORWARDED_FOR	false
PING_INTELLIGENCE_ASE_ENABLE_FIREWALL	true

ENV Variable	Default
PING_INTELLIGENCE_ASE_ENABLE_SIDEBAND_KEEPALIVE	false
PING_INTELLIGENCE_ASE_ENABLE_GOOGLE_PUBSUB	false
PING_INTELLIGENCE_ASE_ENABLE_ACCESS_LOG	true
PING_INTELLIGENCE_ASE_ENABLE_AUDIT	false
PING_INTELLIGENCE_ASE_FLUSH_LOG_IMMEDIATELY	true
PING_INTELLIGENCE_ASE_HTTP_PROCESS	1
PING_INTELLIGENCE_ASE_HTTPS_PROCESS	1
PING_INTELLIGENCE_ASE_ENABLE_SSL_V3	false
PING_INTELLIGENCE_TCP_SEND_BUFFER_BYTES	212992
PING_INTELLIGENCE_TCP_RECEIVE_BUFFER_BYTES	212992
PING_INTELLIGENCE_ASE_ATTACK_LIST_MEMORY	128MB
PING_INTELLIGENCE_CLUSTER_PEER_NODE_CSV_LIST	
PING_INTELLIGENCE_CLUSTER_ID	ase_cluster
PING_INTELLIGENCE_CLUSTER_MGMT_PORT	8020
PING_INTELLIGENCE_CLUSTER_SECRET_KEY	OBF:AES:nPJOh3wXQWK/BOHrtKu3G2SGiAEElOSvOFYEiWfIVSdummo 7LFcqXQlqkW9kldQoFg0nJoLSojnzHDbD3iAy84pT84

ENV Variable	Default
PING_INTELLIGENCE_ABS_ENDPOINT	
PING_INTELLIGENCE_ABS_ACCESS_KEY	
PING_INTELLIGENCE_ABS_SECRET_KEY	
PING_INTELLIGENCE_ABS_ENABLE_SSL	true
PING_INTELLIGENCE_ABS_CA_CERT_PATH	
PING_INTELLIGENCE_ABS_DEPLOYMENT_TYPE	cloud
PING_INTELLIGENCE_ABS_DEPLOYMENT_TYPE_VALIDATION	true
PING_INTELLIGENCE_GATEWAY_CREDENTIALS	
PING_INTELLIGENCE_GATEWAY_CREDENTIALS_REDACT	true
PING_STARTUP_TIMEOUT	8
TAIL_LOG_FILES	${SERVER_ROOT_DIR}/logs/access.log

Running a PingIntelligence container

To run a PingIntelligence container:

```
docker run \
         --name pingintellgence \
         --publish 8443:8443 \
         --detach \
         --env PING_IDENTITY_ACCEPT_EULA=YES \
         --env PING_IDENTITY_DEVOPS_USER=user@pingone.com \
         --env PING_IDENTITY_DEVOPS_KEY=<edvops key here> \
         --env PING_INTELLIGENCE_GATEWAY_CREDENTIALS=<PingIntelligence App JWT here> \
         --ulimit nofile=65536:65536 \
         pingidentity/pingintelligence:edge
```

Follow Docker logs with:

```
docker logs -f pingintelligence `
```

If using the command above, use cli.sh with: * Username: admin * Password: 2FederateM0re

Docker Container Hook Scripts

Please go here for details on all pingintelligence hook scripts

---

This document is auto-generated from *pingintelligence/Dockerfile*

Other Images

Ping Identity DevOps Docker Image - `pingtoolkit`

Ping Identity DevOps Docker Image - `pingtoolkit`

This docker image includes the Ping Identity PingToolkit and associated hook scripts to create a container that can pull in a SERVER_PROFILE run scripts. The typical use case of this image would be an init container or a pod/container to perform tasks aside a running set of pods/containers.

Related Docker Images

- `pingidentity/pingbase` - Parent Image

> *This image inherits inherits, and can use, Environment Variables from pingidentity/pingbase*

- `pingidentity/pingcommon` - Common Ping files (i.e. hook scripts)

Environment Variables

In addition to environment variables inherited from pingidentity/pingbase, the following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	PingToolkit	Ping product name
STARTUP_COMMAND	tail	The command that the entrypoint will execute in the foreground to instantiate the container

ENV Variable	Default	Description
STARTUP_FOREGROUND_OPTS	-f /dev/null	The command-line options to provide to the the startup command when the container starts with the server in the foreground. This is the normal start flow for the container

Docker Container Hook Scripts

Please go here for details on all pingtoolkit hook scripts

This document is auto-generated from *pingtoolkit/Dockerfile*

Ping Identity DevOps Docker Image - `apache-jmeter`

Environment Variables

The following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PING_PRODUCT	Apache-JMeter	Ping product name
JAVA_RAM_PERCENTAGE	90.0	Percentage of the container memory to allocate to PingFederate JVM. **DO NOT** set to 100% or your JVM will exit with OutOfMemory errors and the container will terminate.
STARTUP_COMMAND	${SERVER_ROOT_DIR}/bin/run.sh	The command that the entrypoint will execute in the foreground to instantiate the container.

Docker Container Hook Scripts

Please go here for details on all apache-jmeter hook scripts

This document is auto-generated from *apache-jmeter/Dockerfile*

Ping Identity DevOps Docker Image - `ldap-sdk-tools`

This docker image provides an alpine image with the LDAP Client SDK tools to be used against other PingDirectory instances.

Related Docker Images

- `openjdk:8-jre8-alpine` - Alpine server to run LDAP SDK Tools from

Environment Variables

The following environment `ENV` variables can be used with this image.

ENV Variable	Default	Description
SHIM	${SHIM}	
IMAGE_VERSION	${IMAGE_VERSION}	
IMAGE_GIT_REV	${IMAGE_GIT_REV}	
DATE	${DATE}	
PING_PRODUCT_VERSION	${VERSION}	
PATH	/opt/tools:${PATH}	
PING_PRODUCT	ldap-sdk-tools	Ping product name

List all available tools

```
docker run -it --rm pingidentity/ldap-sdk-tools:edge ls
```

Use LDAPSearch

Get some help

```
docker run -it --rm pingidentity/ldap-sdk-tools:edge ldapsearch --help
```

Simple search

```
docker run -it --rm pingidentity/ldap-sdk-tools:edge \
    ldapsearch \
        -b dc=example,dc=com \
        -p 1389 "(objectClass=*)"
```

Save output to host file

```
docker run -it --rm \
    -v /tmp:/opt/out \
    pingidentity/ldap-sdk-tools:edge \
    ldapsearch \
        --baseDN dc=example,dc=com \
        --port 1389 \
        --outputFormat json "(objectClass=*)" >/tmp/search-result.json
```

Use manage-certificates

trusting certificates

```
PWD=2FederateM0re
mkdir -p /tmp/hibp
docker run -it --rm \
  -v /tmp/hibp:/opt/out \
  pingidentity/ldap-sdk-tools:edge \
  manage-certificates trust-server-certificate \
    --hostname haveibeenpwned.com \
    --port 1443 \
    --keystore /opt/out/hibp-2019.jks \
    --keystore-password ${PWD}
ls -all /tmp/hibp
keytool -list \
  -keystore /tmp/hibp/hibp-2019.jks \
  -storepass ${PWD}
```

Docker Container Hook Scripts

Please go here for details on all ldap-sdk-tools hook scripts

This document is auto-generated from *ldap-sdk-tools/Dockerfile*

Image Details

Using Release Tags

Using Release Tags

Ping Identity uses multiple tags for each released image. On our Docker Hub site, you can view the available tags for each image.

Multi-product deployment

All product containers in a deployment should use the same release tag.

Store images privately

Before discussing tags, it is important to know more about Ping Identity's use of Docker Hub for images. While Docker Hub is very reliable and you can always find the latest images of Ping Identity products hosted there, **do not** rely on Ping to maintain Docker images in Docker Hub over time. See the image support policy for details.

To ensure continued access to any image, pull the image in question and maintain it in your own image registry. Common Docker registry providers include: JFrog, AWS ECR, Google GCR and Azure ACR.

Tagging Format

To specify a release tag for deployments, use the following format:

```
image: pingidentity/<ping-product>:${PING_IDENTITY_DEVOPS_TAG}
```

In the example above, `<ping-product>` is the name of the product and `${PING_IDENTITY_DEVOPS_TAG}` is the assigned release tag value. The file containing the setting for `${PING_IDENTITY_DEVOPS_TAG}` is `~/.pingidentity/config` by default. This file is created by running the `pinctl config` command, documented here. You can also specify the release tag explicitly in your deployments. The release tag must be the same for each container in the deployment. For example:

```
image: pingidentity/<ping-product>:edge
```

Determine Which Tag To Use

The tag to use depends on the purpose of the deployment in question. Along with using a tag, any image on Docker Hub can be referenced using the SHA256 digest to ensure immutability in your environments. The digest for a given image never changes regardless of any tag or tags with which it is associated.

Production Stability

For customers in production environments, stability is often the highest priority. To ensure a deployment with the fewest dependencies and highest product stability:

- Use the digest of a *full sprint tag* that includes the sprint version and product version. For example, consider the image tag `pingidentity/pingfederate:2206-11.1.0`. To pull this image using the corresponding digest:

```
docker pull pingidentity/pingfederate@sha256:8eb88fc3345d8d71dafd83bcdcc38827ddb09768c6571c930b4d217ea177debf
```

Latest Image Features

For demonstrations and testing latest features, use an `edge` based image. In these situations, it is a good practice to use a ***full tag*** variation similar to `pingfederate:11.1.0-edge`, rather than simply `pingfederate:edge`. Doing so avoids dependency conflicts that might occur in server profiles between product versions (for example, 10.x versus 11.x).

Evergreen Bleeding Edge

The `edge` is the absolute latest product version and image features, with zero guarantees for stability. Typically, this tag is only of interest to Ping employees and partners.

Base Release Tags

The base release tags for a product image build are:

- edge
- latest
- sprint

edge

The `edge` release tag refers to "bleeding edge", indicating a build similar to an alpha release. This *sliding* tag includes the latest hooks and scripts, **and is considered highly unstable**. The `edge` release is characterized by:

- Latest product version
- Latest build image enhancements and fixes from our current sprint in progress
- Linux Alpine as the container base OS

Example: `pingidentity/pingfederate:edge`, `pingidentity/pingfederate:11.1.0-edge`

latest

`edge` is tagged as `latest` at the beginning of each month. The release tag indicates the latest stable release. This tag is also a *sliding* tag that marks the stable release for the latest sprint. The `latest` release is characterized by:

- Latest product version
- All completed and qualified enhacements and fixes from the prior monthly sprint

- Linux Alpine as the container base OS

Example: `pingfederate:latest`, `pingfederate:11.1.0-latest`

sprint

In addition to becoming `latest`, `edge` also is tagged as a stable `sprint` each month. The `sprint` release tag is a build number indicating a stable build that will not change over time. The `sprint` number uses the YYMM format. For example, 2208 = August 2022. The `sprint` release is characterized by:

- Latest product version at the time the sprint ended.
- All completed and qualified enhancements and fixes from the specified monthly sprint. The Docker images are generated at the end of the sprint.
- Linux Alpine as the container base OS

Example: `pingfederate:2206`, `pingidentity/pingfederate:2206-11.1.0`

sprint (point release)

Occasionally, a bug might be found on a stable release, whether in the product itself or something from the team building the image. In these situations, to avoid changing a `sprint` tag which is promised to be immutable, a point release would be created to move `latest` forward.

> **Example only**
>
> These example tags do not exist, they are used here only for illustration purposes.

Example: `pingfederate:2206.1`, `pingidentity/pingfederate:2206.1-11.1.0`

Determine Image Version

If you are unsure of the exact version of the image used for a given product container, shell into the container and examine the $IMAGE_VERSION environment variable. For example, if you are running a container locally under Docker, you would run the following commands:

```
docker container exec -it <container id> sh echo $IMAGE_VERSION
```

The IMAGE_VERSION variable returns the version in this format:

```
[product]-[container OS]-[jdk]-[product version]-[build date]-[git revision]
```

For example:

```
IMAGE_VERSION=pingdirectory-alpine_3.16.0-al11-9.1.0.0-220725-c917
```

Where:

Key	Value
Product	pingdirectory
Container OS	alpine_3.16.0
JDK	al11
Product Version	9.1.0.0
Build Date	220725
Git Revision	c917

If the container is running under Kubernetes, use the kubectl exec command to access the container in order to obtain this information.

Date Format

In the $IMAGE_VERSION variable, Date is in YYMMDD format

Product Version, Image Release Matrix

Product Version, Image Release Matrix

It is recommended that you use the most recent Docker release tag available for the product version you want to run.

The tag used to pull the image is in the format `{RELEASE}-{PRODUCT VERSION}`

Examples:

- PingFederate 10.2.5
  - pingidentity/pingfederate:`2108-10.2.5`
- PingDirectory 8.2.0.1
  - pingidentity/pingdirectory:`2101-8.2.0.1`

This file shows the matrix of Ping Identity product software versions and the Ping Docker release tag in which they are available. In accordance with our image support policy, only images from the past 12 months are supported:

../_attachments/productVersionsAndImageTags.pdf

If your browser doesn't support PDFs, please download the PDF to view it: Download PDF.

Evaluation of Docker Base Image Security

Evaluation of Docker Base Image Security

In the Center for Internet Security (CIS) Docker Benchmark v1.2.0, one of the recommendations says, "4.3 Ensure that unnecessary packages are not installed in the container."

It further states, "You should consider using a minimal base image rather than the standard Red Hat/CentOS/Debian images if you can. Some of the options available include BusyBox and Alpine."

The following sections present security aspects of different Linux distributions compared to Alpine Docker image. This doesn't necessarily mean that one is the best for Docker base images. Other factors, such as usability and compatibility, should also be considered when choosing the most suitable Docker image for an organization.

Evaluation overview

To evaluate Alpine's security, we compared it with the following popular Linux distributions: Ubuntu, CentOS, and Red Hat Enterprise Linux 7.

For this comparison, we used the latest version, as of March 12, 2020, of each distribution's Docker image and compared them in four different categories:

- Image size
- Number of packages installed by default
- Number of historical vulnerabilities reported on cvedetails.com
- Number of vulnerabilities reported by the Clair scan

The following table summarizes the numbers for each distribution.

	Alpine	Ubuntu	CentOS	RHEL7
Image Version	alpine:3.11.3	ubuntu:18.04	centos:centos8.1.1911	rhel7:7.7-481
Image Size	5.59MB	64.2MB	237MB	205MB
Number of Packages Installed	14	89	173	162
Number of Historical CVE*s	2	2007	2	662
Number of Vulnerabilities Reported by Clair	0	32	7	0

**CVE - Common Vulnerabilities and Exposures*

Image size

Alpine has an advantage in image size. Although smaller size doesn't directly translate into better security, the smaller size does mean less code packed into the image, which means smaller attack surface.

Number of packages installed

Because of Alpine's smaller size, Alpine has the fewest packages out of box. Fewer packages means lesser chance of having vulnerabilities in the dependencies, which is a plus for security.

Number of historical CVEs

Alpine and CentOS both rank highest in number of historical CVEs even though CentOS has a close relationship with RHEL7, and RHEL7 has 600+ reported vulnerabilities.

Number of vulnerabilities reported by Clair

Some vulnerabilities reported by Clair might not be real issues, but their presence does mean extra overhead for developers or security teams to triage these findings. This overhead can be avoided if unnecessary dependencies are excluded from the image in the first place.

Final evaluation results

Although none of the four categories is perfect on its own for evaluating the security of a Linux distribution, in combination, **Alpine** presents greater advantages for use, which is why we selected it as the disribution for all of our Docker images.

References

- CIS Docker Benchmarks
- Alpine CVEs
- Ubuntu CVEs
- CentOS CVEs
- Redhat Enterprise Linux CVEs

Ping Identity's Docker Image Hardening Guide

For best practices for securing your product Docker image, see Ping Identity's Hardening Guide.

Ping Identity Docker Image Support Policy

Ping Identity Docker image support policy

Ping Identity DevOps Support Policy ¶

The support policy for other Ping DevOps offerings is found at Ping Identity DevOps Support Policy.

Overview

Unlike software delivered as an archive, Docker images include:

- Product artifacts
- OS shim
- An optimized Java virtual machine (JVM) build
- Miscellaneous tools/libraries (Git, SSH, SSL) to run the software and automation scripts

Because of the number of dependency updates and to ensure all patches are kept up to date, Ping Identity actively maintains product images semi-weekly (edge), releasing a stable build each month (sprint and latest).

The build process retrieves the latest versions of:

- Operating System Shim (Alpine)
- Optimized JVM
- Product files
- Supporting tools/libraries

Actively Maintained Product Versions

The DevOps program actively maintains Docker images for:

- The two most recent feature releases (major and minor) of each product
- The latest patch release for each minor version

Examples:

- If we currently maintain images for PingFederate 10.0 and 10.1, when PingFederate 10.2 is released, Docker images with PingFederate 10.0 will no longer be actively maintained.
- If a patch is released for 10.1, it supersedes the previous patch. In other words, if we currently maintain an image for PingFederate 10.1.2, when PingFederate 10.1.3 is released, it replaces 10.1.2.

Active Build Product Versions

To view products and versions actively being built, see the most recent **Release Notes**.

Docker Hub image removal

Security vulnerabilities that arise over time and the continued evolution of our products create a situation in which older product images should be replaced with newer ones in your environment. Images that have fallen out of Ping's active maintenance window **are removed from Docker Hub 1 year after they were last built**. If you need to keep images longer than this period, you will need to store them in a private repository. The Docker documentation has instructions on this process.

Supported OS shim

The DevOps program uses Alpine as its base OS shim. For the rationale, see Evaluation of Docker Base Image Security.

In rare scenarios where the consumer absolutely cannot run an Alpine based image, you can customize the base image. For more information, see Build a Docker Product Image Locally.

Custom Built Images

Using other Linux distributions should not cause an issue, but it cannot be guaranteed that the products will function as expected because these are not verified for compatibility. Ping Identity Support on custom images *might* be challenging and experience longer delays.

The pingctl Utility

The `pingctl` Utility

`pingctl` is our general DevOps command-line utility.

> **Note**
>
> The deprecated ping-devops utility used a configuration file located at `~/.pingidentity/devops`. pingctl, however, uses `~/.pingidentity/config`. While the files generated are compatible, be aware that some older utilities or documentation might refer to the older configuration name.

Dependent Utilities

To perform all of its operations, `pingctl` has a dependency on the following utilities:

- openssl
- base64
- kubectl
- envsubst
- jq
- jwt

Installation and Upgrades

Using Homebrew to install `pingctl` on MacOS, Windows via Windows Subsystem for Linux, or Linux.

1. To install, enter:

   ```
   brew install pingidentity/tap/pingctl
   ```

2. To upgrade, enter:

   ```
   brew upgrade pingidentity/tap/pingctl
   ```

3. To check for upgrades, run the following command.

   > **Note**
   >
   > Check for upgrades regularly.

   ```
   pingctl version
   ```

The dependent utilities for `pingctl` are also installed or upgraded during this process.

Using sh to install `pingctl` on Linux and WSL.

1. To install or upgrade, enter:

   ```
   curl -sL https://bit.ly/pingctl-install | sh
   ```

2. Ensure you have the dependent utilities for `pingctl` installed.

Usage

```
pingctl <command> [options]

Available Commands:
    info            Print pingctl config
    config          Manage pingctl config
    version         Version Details and Check
    clean           Remove ~/.pingidentity/pingctl

    kubernetes      Kubernetes Tools
    license         Ping Identity Licensing Tools
    pingone         PingOne Tools
```

Use `pingctl` for info on available commands.

Use `pingctl <command>` for info on a specific command.

Options

```
-h
```

Provide usage details.

Available Commands

- kubernetes
- LICENSE
- pingone
- info

  Provides a summary of variables defined with pingctl.
- config

Provides an interactive process in which the user can provide all the `pingctl` standard variables (i.e. PingOne and Ping DevOps) as well as custom variables

- version

  Displays the current version of the tool, and checks to see if an update is available.

- clean

  Cleans the cached pingctl work directory containing the latest PingOne Access Token

The `ldap-sdk-tools` utility

The `ldap-sdk-tools` Docker image gives you easy access to our LDAP Client SDK tools for use with PingDirectory.

For complete documentation, see the `pingidentity/ldapsdk` repository.

Setting up the utility

1. From your local `pingidentity-devops-getting-started` directory, enter:

   ```
   ./ldapsdk
   ```

   When you run the `ldapsdk` script for the first time, you're prompted to configure your settings.

   To edit the settings in the future, enter:

   `sh ldapsdk configure`

2. To start the `ldap-sdk-tools` Docker image, enter:

   ```
   docker run  -it --rm  --network pingnet  pingidentity/ldap-sdk-tools:latest
   ```

3. To list the available tools, enter `ls`.

Ping Identity DevOps Registration

Getting Support

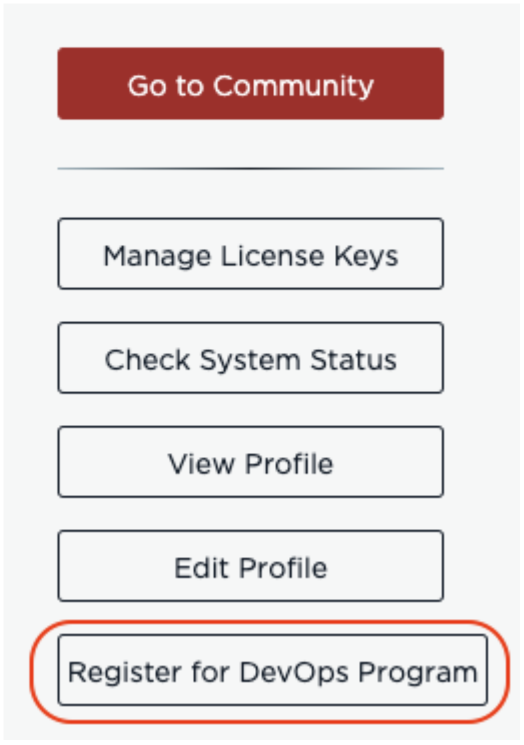
The team responsible for the Ping DevOps program doesn't have access to the user account system on the Ping Identity website. If you have trouble with your account and are unable to follow these instructions to enroll, the issue is likely with your credentials in our system. Please contact your sales representative at Ping Identity Support.

Ping Identity DevOps Registration

Registering for Ping Identity's DevOps Program provides you with credentials that enable you to easily deploy and evaluate Ping Identity products using trial licenses automatically using tools and platforms like Helm or Kubernetes.

To register for the DevOps Program:

- Make sure you have a registered account with Ping Identity. If you're not sure, click the link to Sign On and follow the instructions to access your account.
    - If you don't have an account, create one here.
    - When signing on, select **Support and Community** in the **Select Account** list.
    - After you're signed on, you're directed to your profile page.
    - In the right-side menu, click **Register for DevOps Program**.



A confirmation message will be shown and the DevOps credentials will be forwarded to the email address associated with your Ping Identity account.

Saving Credentials

When you receive your key, follow the instructions below for saving them with the `pingctl` utility.

Example:

- `PING_IDENTITY_DEVOPS_USER=jsmith@example.com`
- `PING_IDENTITY_DEVOPS_KEY=e9bd26ac-17e9-4133-a981-d7a7509314b2`

Saving Your DevOps User and Key

The recommended way to save your DevOps User/Key is to use the Ping Identity DevOps utility `pingctl`.

pingctl Setup

You can find installation instructions for `pingctl` in the pingctl Tool document.

To save your DevOps credentials, run `pingctl config` and supply your credentials when prompted.

When `pingctl` is installed and configured, it places your DEVOPS USER/KEY into a Ping Identity property file found at `~/.pingidentity/config` with the following variable names set (see the following example).

```
PING_IDENTITY_DEVOPS_USER=jsmith@example.com
PING_IDENTITY_DEVOPS_KEY=e9bd26ac-17e9-4133-a981-d7a7509314b2
```

After you've configured these settings, you can view them with the `pingctl info` command (credential values are masked by default, use `pingctl info -v` to show unmasked).

Resending your DevOps User and Key

If you have misplaced or lost your DevOps User/Key, there are two convenient ways to recover it.

- If you have configured `pingctl`, the `PING_IDENTITY_DEVOPS_USER` and `PING_IDENTITY_DEVOPS_KEY` can be printed by entering the following command:

```
pingctl info -v
```

- If you did not save the credentials in the `pingctl` tool, you can recover your credentials by logging in to your Ping Identity account.
- Navigate to Sign On and follow the instructions to access your account.
- When signing on, select **Support and Community** in the **Select Account** list.
- After you're signed on, you're directed to your profile page.
- In the right-side menu, click **Register for DevOps Program** again. A confirmation message will be shown and the same DevOps credentials will be resent to the email address associated with your Ping Identity account.

Mount Existing Product License

You can pass the license file to a container via mounting to the container's `/opt/in` directory.

Note

You do not need to do this if you are using your DevOps User/Key. If you have provided license files via the volume mount and a DevOps User/Key, it will ignore the DevOps User/Key.

The `/opt/in` directory overlays files onto the products runtime filesystem, the license needs to be named correctly and mounted in the exact location the product checks for valid licenses.

Example Mounts

Product	File Name	Mount Path
PingFederate	pingfederate.lic	/opt/in/instance/server/default/conf/pingfederate.lic
PingAccess	pingaccess.lic	/opt/in/instance/conf/pingaccess.lic
PingDirectory	PingDirectory.lic	/opt/in/pd.profile/server-root/pre-setup/PingDirectory.lic
PingDirectoryProxy	PingDirectory.lic	/opt/in/pd.profile/server-root/pre-setup/PingDirectory.lic
PingDataSync	PingDirectory.lic	/opt/in/pd.profile/server-root/pre-setup/PingDirectory.lic
PingAuthorize	PingAuthorize.lic	/opt/in/pd.profile/server-root/pre-setup/PingAuthorize.lic
PingAuthorize PAP	PingAuthorize.lic	/opt/in/pd.profile/server-root/pre-setup/PingAuthorize.lic
PingCentral	pingcentral.lic	/opt/in/instance/conf/pingcentral.lic

Volume Mount Syntax

Docker

Sample docker run command with mounted license:

```
docker run \
  --name pingfederate \
  --volume <local/path/to/pingfederate.lic>:/opt/in/instance/server/default/conf/pingfederate.lic \
  pingidentity/pingfederate:edge
```

Sample docker-compose.yaml with mounted license:

```
version: "2.4"
services:
  pingfederate:
    image: pingidentity/pingfederate:edge
    volumes:
      - path/to/pingfederate.lic:/opt/in/instance/server/default/conf/pingfederate.lic
```

Kubernetes

Create a Kubernetes secret from the license file

```
kubectl create secret generic pingfederate-license \
  --from-file=./pingfederate.lic
```

Then mount it to the pod

```
spec:
  containers:
  - name: pingfederate
    image: pingidentity/pingfederate
    volumeMounts:
      - name: pingfederate-license-volume
        mountPath: "/opt/in/instance/server/default/conf/pingfederate.lic"
        subPath: pingfederate.lic
  volumes:
  - name: pingfederate-license-volume
    secret:
      secretName: pingfederate-license
```

Helm

Create a Kubernetes secret from the license file

```
kubectl create secret generic pingfederate-license \
  --from-file=./pingfederate.lic
```

Add the secretVolumes within your values.yaml deployment file

```
pingfederate-admin:
  secretVolumes:
    pingfederate-license:
      items:
        pingfederate.lic: /opt/in/instance/server/default/conf/pingfederate.lic
```

Note on updating the product license when mounting it as a file

If you are updating the license file for a product, simply replacing the file on the filesystem may not update the license of the running server.

For PingData products (PingDirectory, PingDataSync, PingAuthorize, and PingDirectoryProxy) the license can be updated by copying the new license to the expected location in the server profile - `pd.profile/server-root/pre-setup`. After doing so, dsconfig can be used to update the license on the running server. Ensure that the updated license file is still present in the server profile on subsequent restarts of the container.

For example, for PingDirectory:

```
dsconfig set-license-prop \
  --set "directory-platform-license-key<input-file.lic"
```

The exact name of the license property in the above example will depend on which PingData product is being used.

For non-PingData products, the license can be updated on the product with the typical method. This process will depend on the product, but will generally be done either through the administrative console or using an API call. See the product documentation for details.

Server Profile

Server Profile Deployment

Deployment

Any configuration that is deployed with one of our product containers can be considered a "server profile". A profile typically looks like a set of files.

You can use profiles in these ways:

- Pull at startup.
- Build into the image.
- Mount as a container volume.

Pull at startup

Pass a Github-based URL and path as environment variables that point to a server profile.

Pros:

- Easily sharable, inherently source-controlled

Cons:

- Adds download time at container startup

For profiles pulled at startup, the image uses the following variables to clone the repo at startup and pull the profile into the container:

- `SERVER_PROFILE_URL` - The git URL with the server profile.
- `SERVER_PROFILE_PATH` - The location from the base of the URL with the specific server profile. This allows for several products server profile to be housed in the same git repo.
- `SERVER_PROFILE_BRANCH` (optional) - If other than the default branch (usually master or main), allows for specifying a different branch. Example might be a user's development branch before merging into master.

Although there is additional customizable functionality, this is the most common way that profiles are provided to containers because it is easy to provide a known starting state as well as track changes over time. For more information, see Private Github Repos.

Build into the image

Build your own image from one of our Docker images and copy the profile files in.

Pros:

- No download at startup, and no egress required

Cons:

- Tedious to build images when making iterative changes

Building a profile into the image is useful when you have no access to the Github repository or if you're often spinning containers up and down.

For example, if you made a Dockerfile at this location: https://github.com/pingidentity/pingidentity-server-profiles/tree/master/baseline, the relevant entries might look similar to this:

```
FROM: pingidentity/pingfederate:edge COPY pingfederate/. /opt/in/.
```

Mount as a Docker volume

Using `docker-compose` you can bind-mount a host file system location to a location in the container.

Pros:

- Most iterative. There's no download time, and you can see the file system while you are working in the container.

Cons:

- There's no great way to do this in Kubernetes or other platform orchestration tools.

Mount the profile as a Docker volume when you're developing a server profile and you want to be able to quickly make changes to the profile and spin up a container against it.

For example, if you have a profile in same directory as your `docker-compose.yaml` file, you can add a bind-mount volume to /opt/in like this:

```
volumes:    - ./pingfederate:/opt/in
```

Customizing Server Profiles

When you deployed the full stack of product containers in Getting Started, you used the server profiles associated with each of our products. In the YAML files, you'll see entries like the following for each product instance:

```
environment:
  - SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
  - SERVER_PROFILE_PATH=baseline/pingaccess
```

Our pingidentity-server-profiles repository, indicated by the `SERVER_PROFILE_URL` environment variable, contains the server profiles we use for our DevOps deployment examples. The `SERVER_PROFILE_PATH` environment variable indicates the location of the product profile data to use. In the previous example, the PingAccess profile data is located in the `baseline/pingaccess` directory.

We use environment variables for certain startup and runtime configuration settings of both standalone and orchestrated deployments. You can find environment variables that are common to all product images in the PingBase Image Directory. There are also product-specific environment variables. You can find these in the Docker Image Information for each available product.

Before you begin

You must:

- Complete Get Started to set up your DevOps environment and run a test deployment of the products.
- Understand the Anatomy of the Product Containers.

About this task

You will:

- Add or change the environment variables used for any of our server profiles to better fit your purposes.

  You can find these variables in the Server Profiles Repository for each product.

  For example, the location for the `env_vars` file for PingAccess is located in the baseline/pingaccess server profile.
- Modify one of our server profiles to reflect an existing Ping Identity product installation in your organization.

  You can do this by either:

  - Forking our server profiles repository (`https://github.com/pingidentity/pingidentity-server-profiles`) to your Github repository
  - Using local directories

Adding or Changing Environment Variables

1. Select any environment variables to add from either:
   - The product-specific environment variables in the Docker Images Information
   - The environment variables common to all of our products in the PingBase Image Directory
2. From the `baseline`, `getting-started`, or `simple-sync` directories in the Server Profiles Repository, select the product whose profile you want to modify.
3. Open the `env_vars` file associated with the product and either:
   - Add any of the environment variables you've selected.
   - Change the existing environment variables to fit your purpose.

Modifying a Server Profile

You can modify one of our server profiles based on data from your existing Ping Identity product installation.

Modify a server profile by either:

- Using your Github repository

- Using local directories

Using Your Github Repository

In this example PingFederate installation, using the Github Repository uses a server profile provided through a Github URL and assigned to the `SERVER_PROFILE_PATH` environment variable, such as `--env SERVER_PROFILE_PATH=getting-started/pingfederate`).

1. Export a configuration archive as a *.zip file from a PingFederate installation to a local directory.

   > *Make sure this is exported as a .zip rather than compressing it yourself.*

2. Sign on to Github and fork https://github.com/pingidentity/pingidentity-server-profiles into your own GitHub repository.

3. Open a terminal, create a new directory, and clone your Github repository to a local directory. For example:

   ```
   mkdir /tmp/pf_to_docker
   cd /tmp/pf_to_docker
   git clone https://github.com/<github-username>/pingidentity-server-profiles.git
   ```

   Where `<github-username>` is the name you used to sign on to the Github account.

4. Go to the location where you cloned your fork of our `pingidentity-server-profiles` repository, and replace the `/data` directory in `getting-started/pingfederate/instance/server/default` with the `data` directory you exported from your existing PingFederate installation. For example:

   ```
   cd pingidentity-server-profiles/getting-started/pingfederate/instance/server/default
   rm -rf data
   unzip -qd data <path_to_your_configuration_archive>/data.zip
   ```

   Where `<path_to_your_configuration_archive>` is the location for your exported PingFederate configuration archive.

   You now have a local server profile based on your existing PingFederate installation.

   > **Pushing to GitHub**
   >
   > You should push to GitHub only what's necessary for your customizations. Our Docker images create the `/opt/out` directory using a product's base install and layering a profile (set of files) on top.

5. Push your changes (your local server profile) to the Github repository where you forked our server profile repository.

   You now have a server profile available through a Github URL.

6. Deploy the PingFederate container.

Saving Changes

To save any changes you make after the container is running, add the entry `--volume <local-path>:/opt/out` to the `docker run` command, where `<local-path>` is a directory you haven't already created. For more information, see Saving Your Changes.

As in this example, the environment variables `SERVER_PROFILE_URL` and `SERVER_PROFILE_PATH` direct Docker to use the server profile you've modified and pushed to Github:

```
docker run \
  --name pingfederate \
  --publish 9999:9999 \
  --publish 9031:9031 \
  --detach \
  --env SERVER_PROFILE_URL=https://github.com/<your_username>/pingidentity-server-profiles.git \
  --env SERVER_PROFILE_PATH=getting-started/pingfederate \
  --env-file ~/.pingidentity/config \
  pingidentity/pingfederate:edge
```

Private Repo

If your GitHub server-profile repo is private, use the `username:token` format so the container can access the repository. For example, `https://github.com/<your_username>:<your_access_token>/pingidentity-server-profiles.git`. For more information, see Using Private GitHub Repositories.

7. To display the logs as the container starts up, enter:

```
docker container logs -f pingfederate
```

8. In a browser, go to https://localhost:9999/pingfederate/app to display the PingFederate console.

Using Local Directories

This method is particularly helpful when developing locally and the configuration isn't ready to be distributed (using Github, for example).

We'll use PingFederate as an example. The local directories used by our containers to persist state and data, `/opt/in` and `/opt/out`, will be bound to another local directory and mounted as Docker volumes. This is our infrastructure for modifying the server profile.

Bind Mounts in Production

Docker recommends that you never use bind mounts in a production environment. This method is solely for developing server profiles. For more information, see the Docker Documentation.

- The `/opt/out` directory

All configurations and changes during our container runtimes (persisted data) are captured here. For example, the PingFederate image `/opt/out/instance` contains much of the typical PingFederate root directory:

```
.
├── README.md
├── SNMP
├── bin
├── connection_export_examples
├── etc
├── legal
├── lib
├── log
├── modules
├── sbin
├── sdk
├── server
├── tools
└── work
```

- The `/opt/in` directory

  If a mounted `opt/in` directory exists, our containers reference this directory at startup for any server profile structures or other relevant files. This method is in contrast to a server profile provided using a Github URL assigned to the `SERVER_PROFILE_PATH` environment variable, such as, `--env SERVER_PROFILE_PATH=getting-started/pingfederate`.

  > *For the data each product writes to a mounted `/opt/in` directory, see Server profile structures.*

  These directories are useful for building and working with local server-profiles. The `/opt/in` directory is particularly valuable if you don't want your containers to access Github for data (the default for our server profiles).

The following example deployment uses PingFederate.

1. Deploy PingFederate using our sample getting-started Server Profile, and mount `/opt/out` to a local directory:

   ```
   docker run \
     --name pingfederate \
     --publish 9999:9999 \
     --detach \
     --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
     --env SERVER_PROFILE_PATH=getting-started/pingfederate \
     --env-file ~/.pingidentity/config \
     --volume /tmp/docker/pf:/opt/out \
     pingidentity/pingfederate:edge
   ```

   > *Make sure the local directory (in this case, `/tmp/docker/pf`) isn't already created. Docker needs to create this directory for the mount to `/opt/out`.*

2. Go to the mounted local directory (in this case, `/tmp/docker/pf`), then make and save some configuration changes to PingFederate using the management console.

   As you save the changes, you'll be able to see the files in the mounted directory change. For PingFederate, an `instance` directory is created. This is a PingFederate server profile.

3. Stop and remove the container and start a new container, adding another `/tmp/docker/pf` bind mounted volume, this time to `/opt/in`:

   ```
   docker container rm pingfederate

   docker run \
     --name pingfederate-local \
     --publish 9999:9999 \
     --detach \
     --volume /tmp/docker/pf:/opt/out \
     --volume /tmp/docker/pf:/opt/in \
     pingidentity/pingfederate:edge
   ```

Saving Your Configuration Changes

To save any configuration changes you make when using the products in the stack, you must set up a local Docker volume to persist state and data for the stack. If you don't do this, whenever you bring the stack down, your configuration changes will be lost.

Mount a Docker volume location to the Docker `/opt/out` directory for the container. The location must be to a directory you haven't already created. Our Docker containers use the `/opt/out` directory to store application data.

Mounting to /opt/out

Make sure the local directory isn't already created. Docker needs to create this directory for the mount to `/opt/out`.

You can mount a Docker volume for containers in a stack or for standalone containers.

Bind mounting for a stack

1. Add a `volumes` section under the container entry for each product in the `docker-compose.yaml` file you're using for the stack.
2. Under the `volumes` section, add a location to persist your data. For example:

   ```
   pingfederate:...volumes:- /tmp/compose/pingfederate_1:/opt/out
   ```

3. In the `environment` section, comment out the `SERVER_PROFILE_PATH` setting.

   The container then uses your `volumes` entry to supply the product state and data, including your configuration changes.

   When the container starts, this mounts `/tmp/compose/pingfederate_1` to the `/opt/out` directory in the container. You can also view the product logs and data in the `/tmp/compose/pingfederate_1` directory.

4. Repeat this process for the remaining container entries in the stack.

Bind mounting for a standalone container

Add a `volume` entry to the `docker run` command:

```
docker run \
  --name pingfederate \
  --volume <local-path>:/opt/out \
  pingidentity/pingfederate:edge
```

Getting started with Docker Compose mounts

Within many of the docker-compose.yaml files in the Getting-Started repository, volume mounts to `opt/out` have been included to persist your configuration across container restarts.

- To view the list of persisted volumes, enter:

  ```
  docker volume list
  ```

- To view the contents of the /opt/out/ volume when the container is running, enter:

  ```
  docker container exec -it <container id> sh
  cd out
  ```

- To view the contents of the /opt/out/ volume when the container is stopped, enter:

  ```
  docker run --rm -i -v=<volume name>:/opt/out alpine ls
  ```

- To remove a volume, enter:

  ```
  docker volume rm <volume name>
  ```

- To copy files from the container to your local filesystem, enter:

  ```
  docker cp \
    <container id>:<source path> \
    <destination path>
  eg.
  docker cp \
    b867054293a1:/opt/out \
    ~/pingfederate/
  ```

- To copy files from your local filesystem to the container, enter:

```
docker cp \
  <source path> \
  <container id>:<destination path>
eg.
docker cp \
  myconnector.jar \
  bb867054293a186:/opt/out/instance/server/default/deploy/
```

Build Your Own

Build a PingFederate Profile From Your Current deployment

The term "profile" can vary in many instances. Here we will focus on two types of profiles for PingFederate: configuration archive, and bulk export. We will discuss the similarities and differences between two as well as how to build either from a running PingFederate environment.

Before you begin

You must:

- Complete Get Started to set up your DevOps environment and run a test deployment of the products
- Understand our Product Container Anatomy

You should:

- Review Customizing Server Profiles

Overview of profile methods

There are two file-based profile methods that we cover:

- Bulk API Export
  - The resulting `.json` from the admin API at /bulk/export
  - Typically saved as `data.json`
- Configuration Archive
  - Pulled either from the admin UI - Server > Configuration Archive or from the admin API at `/configArchive`
  - We call the result of this output `data.zip` or the `/data` folder

A file-based profile means a "complete profile" looks like a **subset** of files that you would typically find in a running PingFederate filesystem.

This subset of files represents the minimal number of files needed to achieve your PingFederate configuration. All additional files that are not specific to your configuration should be left out because the PingFederate Docker image fills them in. For more information, see Container Anatomy.

Familiarity with the PingFederate filesystem will help you achieve the optimal profile. For more information, see profile structures.

Save files

You should save every file outside of `pingfederate/server/default/data` that you've edited.

Additionally, all files that are included in the profile should also be environment agnostic. This typically means turning hostnames and secrets into variables that can be delivered from the Orchestration Layer.

The Bulk API Export Profile Method

About this method

You will:

1. Export a `data.json` from /bulk/export
2. Configure and run bulkconfig tool
3. Export Key Pairs
4. base64 encode exported key pairs
5. Add `data.json.subst` to your profile at `instance/bulk-config/data.json.subst`

> *In this guide, we will look at the above steps in detail to understand the purpose and flow. Use the steps for reference as needed.*

A PingFederate Admin Console imports a `data.json` on startup if it finds it in `instance/bulk-config/data.json`.

The PF admin API `/bulk/export` endpoint outputs a large .json blob that is representative of the entire `pingfederate/server/default/data` folder, PingFederate 'core config', or a representation of anything you would configure from the PingFedera0te UI. This file can be considered as "the configuration archive in .json format".

Steps

1. On a running PingFederate instance or pod, run:

```
curl \
  --location \
  --request GET 'https://pingfederate-admin.ping-devops.com/pf-admin-api/v1/bulk/export' \
  --header 'X-XSRF-Header: PingFederate' \
  --user "administrator:${passsword}" > data.json
```

2. Save data.json into a profile at `instance/bulk-config/data.json`.
3. Delete everything except `pf.jwk` in `instance/server/default/data`.

Result

You have a bulk API export "profile". This file is useful because the entire config is in a single file and if you store it in source control, then you only have to compare differences in one file. However, there is more value than being in one file.

Making the bulk API export "profile-worthy"

By default, the resulting `data.json` from the export contains encrypted values, and to import this file, your PingFederate needs to have the corresponding master key (`pf.jwk`) in `pingfederate/server/default/data` .

Encrypted values in single file

In the DevOps world, we call this folder `instance/server/default/data` . However, each of the encrypted values also have the option to be replaced with an unencrypted form and, when required, a corresponding password.

Example

The SSL Server Certificate from the PingFederate Baseline Profile when exported to data.json has the following syntax:

```
{
    "resourceType": "/keyPairs/sslServer",
    "operationType": "SAVE",
    "items": [
        {
            "id": "sslservercert",
            "fileData":
"MIIRBwIBAzCCEMAGCSqGSIb3DQEHAaCCELEEghCtMIIQqTCCCeUGCSqGSIb3DQEHAaCCCdYEggnSMIIJzjCCCcoGCyqGSIb3DQEMCgECoIIJezCCCXcw
KQYKKoZIhvcNAQwBAzAbBBQu6vDERQZX3uujWa7v_q3sYN4Q0gIDAMNQBIIJSFtdWbvLhzYrTqeKKiJqiqROgE0E4mkVvmEC6NwhhPbcH37IDNvVLu0um
m--CDZnEmlyPpUucO345-U-6z-cskw4TbsjYIzM10MwS6JdsyYFTC3GwqioqndVgBUzDh8xGnfzx52zEehX8d-
ig1F6xYsbEc01gTbh4lF5MA7E7VfoTa4hWqtceV8PQeqzJNarlZyDSaS5BLn1J6G9BYUze-M1xGhATz7F2l-aAt6foi0mwIBlc2fwsdEPuAALZgdG-
q_V4gOJW2K0ONnmWhMgMLpCL42cmSb            ... more encrypted text ...
Yxpzp_srpy4LHNdgHqhVBhqtDrjeKJDRfc1yk21P5PpfEBxn5MD4wITAJBgUrDgMCGgUABBQLBpq8y79Pq1TzG1Xf6OAjZzBZaQQUC4kD4CkcrH-
WTQhJHud850ddn08CAwGGoA==",
            "encryptedPassword":
"eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjoiRW1JY1UxOVdueSIsInZlcnNpb24iOiIxMC4xLjEuMCJ9..l6PJ55nSSvKHl0
vSWTpkOA.i7hpnnu2yIByhyq_aGBCdaqS3u050yG8eMRGnLRx2Yk.Mo4WSkbbJyLISHq6i4nlVA"
        }
    ]
}
```

You can convert this master key dependent form to:

```
{
    "operationType": "SAVE",
    "items": [{
        "password": "2FederateM0re",
        "fileData":
"MIIRCQIBAzCCEM8GCSqGSIb3DQEHAaCCEMAEghC8MIIQuDCCC28GCSqGSIb3DQEHBqCCC2AwggtcAgEAMIILVQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEM
AQYwDgQIjXWLRGuGNIQCAggAgIILKOgCQ9onDqBPQsshsaS50OjWtj/7s47BUYal1YhO70fBup1a82WGHGhAvb/SY1yOhqQR+TloEBOPI5cExoGN/
Gvw2Mw5/
wkQZZMSHqxjz68KhN4B0hrsOf4rqShB7jsz9ebSml3r2w0sUZWR73GBtBt1Y3wIlXLS2WtqdtHra9VnUqp1eOk+xenjuWM+u2ndDD43GgKB3n8mNBSSVB
qx6ne7aSRJRuAUd+HAzLvSeXjTPMObI1Jod2F+7        ... more base64 encoded exported .p12 ...
5QJ15OJp2iEoVBWxogKf64s2F0iIYPoo6yjNvlidZCevP564FwknWrHoD7R8cIBrhlCJQbEOpOhPg66r4MK1CeJ2poaKRlMS8HGcMRaTpaqD+pIlgmUS6
xFw49vr9Kwfb7KteRsTkNR+I8A7HjUpuCMSUwIwYJKoZIhvcNAQkVMRYEFOb7g1xwDka5fJ4sqngEvzTyuWnpMDEwITAJBgUrDgMCGgUABBRlJ+D+FR/
vQbaTGbKDFiBK/xDbqQQIAjLc+GgRg44CAggA",
        "id": "sslservercert"
    }],
    "resourceType": "/keyPairs/sslServer"
}
```

The process:

1. You exported the private key+cert of the server cert with alias `sslservercert`. When exported, a password is requested and `2FederateM0re` was used. This action results in the download of a password protected `.p12` file.
2. The data.json key name `encryptedPassword` converted to simply `password`.
3. The value for `fileData` is replaced with a base64 encoded version of the exported `.p12` file.

This process can be used for all encrypted items and environment specific items:

- Key Pairs (.p12)
- Trusted Certs (x509)
- Admin Password
- Data Store Passwords
- Integration Kit Properties
- Hostnames

Leaving these confidential items as unencrypted text in source control is unacceptable. The next logical step is to abstract the unencrypted values and replace them with variables. By doing this, the values can be stored in a secrets management tool (such as Hashicorp Vault) and the variablized file can be in source control.

Converting each of the encrypted keys for their unencrypted counterparts and hostnames with variables is cumbersome and can be automated. As we know in DevOps, if it *can* be automated, it *must* be automated. For more information, see Using Bulk Config Tool.

A variablized `data.json.subst` is a good candidate for committing to source control after removing any unencrypted text.

Using Bulk Config Tool

The ping-bulkconfig-tool reads your data.json and can optionally:

- Search and replace (e.g. hostnames)
- Clean, add, and remove json members as required.
- Tokenize the configuration and maintain environment variables.

The bulk export tool can process a bulk `data.json` export according to a configuration file with functions above. After running the tool, you are left with a `data.json.subst` and a list of environment variables waiting to be filled.

The `data.json.subst` form of our previous example will look like:

```
{
    "operationType": "SAVE",
    "items": [{
        "password": "${keyPairs_sslServer_items_sslservercert_sslservercert_password}",
        "fileData": "${keyPairs_sslServer_items_sslservercert_sslservercert_fileData}",
        "id": "sslservercert"
    }],
    "resourceType": "/keyPairs/sslServer"
}
```

Bulk Config Tool Limitations

The bulk config tool can manipulate data.json but it cannot populate the resulting password or fileData variables because there is no API available on PingFederate to extract these. These variables can be filled using with externally generated certs and keys using tools like openssl, but that is out of scope for this document.

The resulting `env_vars` file can be used as a guideline for secrets that should be managed externally and only delivered to the container/image as needed for its specific environment.

Prerequisites

1. The bulk export utility comes in pre-compiled source code. Build a Docker image by running:

   ```
   docker build -t ping-bulkexport-tools:latest .
   ```

2. Copy the data.json to: `pingidentity-devops-getting-started/99-helper-scripts/ping-bulkconfigtool/shared/data.json`

Example

A sample command of the ping-bulkconfig-tool

```
docker run --rm -v $PWD/shared:/shared ping-bulkexport-tools:latest /shared/pf-config.json /shared/data.json /shared/
env_vars /shared/data.json.subst > ./shared/convert.log
```

Where:

- `-v $PWD/shared:/shared` - bind mounts `ping-bulkconfigtool/shared` folder to /shared in the container
- `/shared/pf-config.json` - input path to config file which defines how to process the bulk export `data.json` file from PingFederate.
- `/shared/data.json` - input path to data.json result of /pf-admin-api/v1/bulk/export PingFederate API endpoint.
- `/shared/env_vars` - output path to store environment variables generated from processing
- `/shared/data.json.subst` - output path to processed data.json

After running the above command, you will see `env_vars` and `data.json.subst` in the `ping-bulkconfigtool/shared` folder.

Configure Bulk Tool

Instructions to the bulk config tool are sent by the `pf-config.json` file.

When using the `pf-config.json` file, any unused functions will require an empty array in the file. For example, notice the **add-config** block at the top of this sample:

```
{
  "add-config":[],
  "config-aliases":[
  ],
  "expose-parameters":[
  ]
  ,
  "remove-config":[
    {
    "key": "id",
    "value": "ProvisionerDS"
    }
  ]
}
```

In this file, available commands include:

search-replace - A utility to search and replace string values in a bulk config json file. - Can expose environment variables.

Example: replacing an expected base hostname with a substitution:

```
"search-replace":[
  {
    "search": "data-holder.local",
    "replace": "${BASE_HOSTNAME}",
    "apply-env-file": false
  }
]
```

change-value

- Searches for elements with a matching identifier, and updates a parameter with a new value.

Example: update keyPairId against an element with name=ENGINE:

```
"change-value":[
  {
        "matching-identifier":
        {
          "id-name": "name",
          "id-value": "ENGINE"
        },
    "parameter-name": "keyPairId",
    "new-value": 8
  }
]
```

remove-config

- Remove configuration objects from the bulk export

Example:

```
"remove-config":[
  {
    "key": "id",
    "value": "ProvisionerDS"
  }
]
```

Example:

```
"remove-config":[
  {
    "key": "resourceType",
    "value": "/idp/spConnections"
  }
]
```

This tool works with both PingFederate and PingAccess. This example adds the CONFIG QUERY http listener in PingAccess:

```
"add-config":[
    {
      "resourceType": "httpsListeners",
      "item":
          {
              "id": 4,
              "name": "CONFIG QUERY",
              "keyPairId": 5,
              "useServerCipherSuiteOrder": true,
              "restartRequired": false
          }
    }
]
```

Example: Add an SP connection:

```
"add-config":[
    {
      "resourceType": "/idp/spConnections",
      "item":
      {
                  "name": "httpbin3.org",
                  "active": false,
          ...
      }
    }
]
```

expose-parameters

- Navigates through the JSON and exchanges values for substitions.
- Exposed substition names will be automatically created based on the json path.
    - E.g. ${oauth_clients_items_clientAuth_testclient_secret}
- Can convert encrypted/obfuscated values into clear text inputs (e.g. "encryptedValue" to "value") prior to substituting it. Doing so enables the injection of values in their raw form.

Example: replace the "encryptedPassword" member with a substitution-enabled "password" member for any elements with "id" or "username" members. The following will remove "encryptedPassword" and create "password":

```
  {
    "parameter-name": "encryptedPassword",
    "replace-name": "password",
    "unique-identifiers": [
        "id",
        "username"
    ]
  }
```

config-aliases

- The bulk config tool generates substitution names. However, there might be times you wish to simplify them or reuse existing environment variables.

Example: Rename the Administrator's substitution name using the PING_IDENTITY_PASSWORD environment variable:

```
    "config-aliases":[
      {
        "config-names":[
          "administrativeAccounts_items_Administrator_password"
        ],
        "replace-name": "PING_IDENTITY_PASSWORD",
        "is-apply-envfile": false
      }
    ]
```

sort-arrays

- Configure the array members that need to be sorted. This function ensures the array is created consistently, simplifying git diff analysis.

Example: Sort the roles and scopes arrays:

```
   "sort-arrays":[
          "roles","scopes"
    ]
```

Additional Notes

- The bulk API export is intended to be used as a *bulk* import. The `/bulk/import` endpoint is destructive and overwrites the entire current admin config.
- If you are in a clustered environment, the PingFederate image imports the `data.json` and replicates the configuration to engines in the cluster.
- Your data.json.subst `"metadata": {"pfVersion": "10.1.2.0"}` should match the PingFederate profile version.

The Configuration Archive Profiles Method

About configuration archive-based profiles

You should weigh the pros and cons of configuration archive-based profiles compared to bulk API export profiles. While not fully aligning with pur DevOps principles, many users prefer using bulk API export profiles in most scenarios.

Pros: * The `/data` folder, as opposed to a `data.json` file, is better for profile layering. * Configuration is available on engines at startup, which: * lowers dependency on the admin at initial cluster startup

Cons:

- The `/data` folder contains key pairs in a `.jks` file , which makes externally managing keys very difficult.
- Encrypted data is scattered throughout the folder, creating a dependency on the master encryption key.

About this method

You will:

1. Export a `data.zip` archive
2. Optionally, variablize
3. Replace the data folder

Installing PingFederate Integration Kits

By default, PingFederate is shipped with a handful of integration kits and adapters. If you need other integration kits or adapters in the deployment, manually download them and place them inside the `server/default/deploy` directory of the server profile. You can find these resources in the product download page here.

Building a profile from Your Current Deployment

PingDirectory is built for GitOps through native tools for building profiles. To find the latest tools and profiles, search for "DevOps" in PingDirectory Docs. You can find details of the server profile structure there.

A well-formed PingDirectory profile includes all the configuration details needed for starting up a server in a **new** or **existing** replication topology as a representation of what is actually running.

Use this guide to build a PingDirectory profile from a running instance.

Before you begin

You must:

- Complete Get Started
- Have a running PingDirectory instance of 8.0.0.0 or later with shell access on the machine or in the container
- Understand Product Container Anatomy

You should:

- Review Customizing Server Profiles

Start Building

Generating a profile

To generate a profile, run `manage-profile generate-profile`.

This can be called on a running container in Kubernetes like so:

```
kubectl exec -it <pod-name> \
  -- manage-profile generate-profile \
  --profileRoot /tmp/pd.profile

kubectl exec -it pingdirectory-0 \
  -- manage-profile generate-profile \
  --profileRoot /tmp/pd.profile
```

The Name Matters

Although you don’t have to name your profile `pd.profile`, the default location (the variable `PD_PROFILE`) that PingDirectory looks at is `PD_PROFILE="/opt/staging/pd.profile"`.

Sample Output:

```
Defaulted container "pingdirectory" out of: pingdirectory, telegraf, vault-agent-init (init)
Generating server profile

...

Variables such as PING_INSTANCE_NAME in setup-arguments.txt and in any other
files in the profile will need to be provided through environment variables or
through a profile variables file when using the generated profile with the
manage-profile tool. The PING_SERVER_ROOT and PING_PROFILE_ROOT variables are
provided by manage-profile

Some changes may need to be made to the generated profile. Any desired LDIF
files will need to be added to the profile. Any additional server root files,
server SDK extensions, and dsconfig commands can be manually added, and
variables-ignore.txt can be updated to ignore certain files during variable
substitution. See the README file at /tmp/pd.profile/misc-files/README for
more information on the manual steps that must be taken for the generated
profile to be used with the manage-profile tool

The following files and directories in the server root were excluded from the
generated profile, and can be manually added if necessary. These files can
also be included by generate-profile with the --includePath argument:
config/truststore
config/ads-truststore
config/encryption-settings.pin
config/tools.properties.orig
config/encryption-settings/encryption-settings-db
config/keystore.p12
config/tools.properties
config/encryption-settings/encryption-settings-db.old
config/keystore.pin
config/keystore
config/ads-truststore.pin
config/truststore.p12
config/truststore.pin

The generated profile can be found at /tmp/pd.profile
```

Note other paths that are not included

The `manage-profile generate-profile` command outputs valuable information about what is and isn't included in the generated profile.

Don't put secrets in your profile!

Secrets should *not* be included in your profile, so they are not included in the profile generation by default. However, if you have not already added encryption secrets or keystores to your environment, you can use the `--includePath` argument to collect items from the running server. These items should then be provided to the server on its next restart through some secrets management tool.

Extracting the generated profile

Following the Kubernetes example, you can copy out the generated profile with:

```
kubectl cp pingdirectory-0:/tmp/pd.profile pd.profile
```

Sample output:

```
% tree
.
└── pd.profile
  ├── dsconfig
  │ └── 00-config.dsconfig
  ├── ldif
  │ └── userRoot
  ├── misc-files
  │ └── README
  ├── server-root
  │ ├── post-setup
  │ └── pre-setup
  │   ├── PingDirectory.lic
  │   ├── README.md
  │   └── config
  │     ├── encryption-settings.pin ## Added via --includePath
  │     ├── keystore.pin ## Added via --includePath
  │     └── schema
  │       ├── 80-format-counter-metrics.ldif
  │       ├── 87-local-identities.ldif
  │       ├── 88-grants.ldif
  │       ├── 89-sessions.ldif
  │       └── 90-oauth-clients.ldif
  ├── server-sdk-extensions
  ├── setup-arguments.txt ## REMOVE this
  └── variables-ignore.txt
11 directories, 13 files
```

userRoot data is not included

You might notice that userRoot data (i.e. users) isn't included. Profiles should contain *configuration* only, not data.

Storing a profile

To store the profile, at the root of your profile:

Choose from:

• For an unmounted profile, add to `pd.profile`.

- For a mounted profile, add to `/opt/in/pd.profile`.

Including other files

In addition to what's generated with `manage-profile generate-profile`, you might want to include other files. These files should be siblings to `pd.profile` at the root of the profile.

For an example structure, see baseline.

Profile structure

"A good PingDirectory profile includes all the **configuration** needed for starting up a server in a **new** or **existing** replication topology."

Review the following elements to see what to include in your profile.

dsconfig commands

Because this is how the PingDirectory server is configured, dsconfig commands belong in your profile.

- `manage-profile generate-profile` outputs all of the dsconfig commands of a running server into one file: ` 00-config.dsconfig`.

Keeping dsconfig commands in one file makes sense because they are ingested together but run in order by the server's inherent dependency knowledge of itself. You can work on PingDirectory in a dev environment and make many changes while working toward your desired configuration.

- `generate-profile` exports a representation of your work.

Multiple files

You might see multiple files containing dsconfig commands in our profiles, which serves to show logical separation in our demos. Additionally, our demos might be built of multiple layers coming form different repositories so this prevents overwriting.

users

Data is expected to change at runtime, so this information does *not* belong in your profile structure.

There is built-in protection to enforce this. `ldif/userRoot/` **is only imported on** `GENESIS` **- The first start of the *first PingDirectory in a topology.**

The exceptions to this rule are ephemeral dev and demo environments. This is why you see user files in our sample profiles. These files are intended for bootstrapping demo and test instances.

If you are in this category and wanted to include users, you could use:

```
kubectl exec -it pingdirectory-0 \
  -- export-ldif \
  --backendID userRoot \
  --ldifFile /tmp/userRoot.ldif \
  --doNotEncrypt

kubectl cp pingdirectory-0:/tmp/userRoot.ldif \
  pd.profile/ldif/userRoot/00-users.ldif
```

schema

Schema belongs in your profile structure because you might want to manage your schema as code, and `pd.profile/server-root/pre-setup/config/schema` is where to do that.

java.properties

The `config/java.properties` file in the server root is used by PingDirectory to manage arguments passed to Java for the server process and for any command-line utilities. If you need to set any custom values in this file, provide the entire file in your server profile at `instance/config/java.properties`. Note that this is outside of the `pd.profile` folder.

encryption keys, keystores, truststores, and other secrets

Any and all secrets should be provided by some sort of secrets management (Vault, bitnami sealed secrets, or at least kubernetes secrets), and as such, these do *not* belong in your profile structure.

PingDirectory allows you to define file paths to secrets so they don't need to be in the profile.

Layering Server Profiles

One of the benefits of our Docker images is the ability to layer product configuration. By using small, discrete portions of your configuration, you can build and assemble a server profile based on multiple installations of a product.

A typical organization can have multiple installations of our products, each using different configurations. By layering the server profiles, you can reuse the configurations that are common across environments, leading to fewer configurations to manage.

You can have as many layers as needed. Each layer of the configuration is *copied* on top of the container's filesystem (not merged).

Layer Precedence

The profile layers are applied starting at the top layer and ending at the base layer. This ordering might not be apparent at first.

Before you begin

You must:

- Complete Get Started to set up your DevOps environment and run a test deployment of the products.

About this task

You will:

- Create a layered server profile.
- Assign the environment variables for the deployment.
- Deploy the layered server profile.

Creating a layered server profile

For this guide, PingFederate is used along with the server profile located in the pingidentity-server-profiles repository. You should fork this repository to your Github repository, then pull your Github repository to a local directory. After you have finished creating the layered profile, you can push your updates to your Github repository and reference it as an environment variable to run the deployment.

You will create separate layers for:

- Product license
- Extensions (such as, Integration Kits and Connectors)

For this example, these layers will be applied on top of the PingFederate server profile. However, you can span configurations across multiple repositories if you want.

You can find the complete working, layered server profile of the PingFederate example from this guide in the pingidentity-server-profiles/layered-profiles directory.

Because PingFederate's configuration is file-based, the layering works by copying configurations on top of the PingFederate container's file system.

Files Copied

Files are copied, not merged. It's best practice to only layer items that won't be impacted by other configuration files.

Creating the base directories

Create a working directory named `layered_profiles` and within that directory create `license` and `extensions` directories. When completed, your directory structure should be:

```
└── layered_profiles
  ├── extensions
  └── license
```

Constructing the license layer

1. Go to the `license` directory and create a `pingfederate` subdirectory.
2. Create the PingFederate license file directory path under the `pingfederate` directory.

   The PingFederate license file resides in the `/instance/server/default/conf/` path.

   ```
   mkdir -p instance/server/default/conf/
   ```

   Your license profile path should look like this:

   ```
   └── license
     └── pingfederate
       └── instance
         └── server
             └── default
               └── conf
                 └── pingfederate.lic
   ```

3. Copy your `pingfederate.lic` file to `license/pingfederate/instance/server/default/conf`.

   Using the DevOps evaluation license, when the PingFederate container is running, you can find the license in the container file system `/opt/out/instance/server/default/conf` directory.

   You can copy the `pingfederate.lic` file from the Docker file system using the syntax: `docker cp <container> <source-location> <target-location>`

   For example:

   ```
   docker cp \
     pingfederate \
     /opt/in/instance/server/default/conf/pingfederate.lic \
     ${HOME}/projects/devops/layered_profiles/license/pingfederate/instance/server/default/conf
   ```

   Using the `pingctl` tool (update product and version accordingly):

   ```
   sh pingctl license pingfederate 11.1 > \
     ${HOME}/projects/devops/layered_profiles/license/pingfederate/instance/server/default/conf
   ```

Building the extensions layer

1. Go to the `layered-profiles/extensions` directory and create a `pingfederate` subdirectory.
2. Create the PingFederate extensions directory path under the `pingfederate` directory.

   The PingFederate extensions reside in the `/instance/server/default/deploy` directory path.

```
mkdir -p instance/server/default/deploy
```

3. Copy the extensions you want to be available to PingFederate to the `layered-profiles/extensions/pingfederate/instance/server/default/deploy` directory .

   The extensions profile path should look similar to the following (extensions will vary based on your requirements):

```
└── extensions
  └── pingfederate
    └── instance
        └── server
          └── default
            └── deploy
                ├── pf-aws-quickconnection-2.0.jar
                ├── pf-azure-ad-pcv-1.2.jar
                └── pf-slack-quickconnection-3.0.jar
```

Assigning environment variables

Although this deployment assigns the environment variables for use in a Docker Compose YAML file, you can use the following technique with any Docker or Kubernetes deployment.

If you want to use your own Github repository for the deployment in the following examples, replace:

```
SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
```

with:

```
SERVER_PROFILE_URL=https://github.com/<your-username>/pingidentity-server-profiles.git
```

!!! note "Private Github Repo" If your GitHub server-profile repo is private, use the `username:token` format so the container can access the repository. For example, `https://github.com/<your_username>:<your_access_token>/pingidentity-server-profiles.git`. For more information, see Using Private Github Repositories.

1. Create a new `docker-compose.yaml` file.
2. Add your license profile to the YAML file.

   For example:

```
- SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
- SERVER_PROFILE_PATH=layered-profiles/license/pingfederate
```

> `SERVER_PROFILE` *supports* `URL`, `PATH`, `BRANCH` *and* `PARENT` *variables.*

3. Using `SERVER_PROFILE_PARENT`, instruct the container to retrieve its parent configuration by specifying the `extensions` profile as the parent:

   ```
   - SERVER_PROFILE_PARENT=EXTENSIONS
   ```

   `SERVER_PROFILE` can be extended to reference additional profiles. Because we specified the license profile's parent as `EXTENSIONS`, we can extend `SERVER_PROFILE` by referencing the `EXTENSIONS` profile (prior to the `URL` and `PATH` variables):

   ```
   - SERVER_PROFILE_EXTENSIONS_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
   - SERVER_PROFILE_EXTENSIONS_PATH=layered-profiles/extensions/pingfederate
   ```

4. Set `GETTING_STARTED` as the `EXTENSIONS` parent and declare the `URL` and `PATH`:

   ```
   - SERVER_PROFILE_EXTENSIONS_PARENT=GETTING_STARTED
   - SERVER_PROFILE_GETTING_STARTED_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
   - SERVER_PROFILE_GETTING_STARTED_PATH=getting-started/pingfederate
   ```

   > *Because the `GETTING_STARTED` profile is the last profile to add, it will not have a parent.*

   Your `environment` section of the `docker-compose.yaml` file should look similar to this:

   ```
   environment:# **** SERVER PROFILES BEGIN ****
   # Server Profile - Product License
   - SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
   - SERVER_PROFILE_PATH=layered-profiles/license/pingfederate
   - SERVER_PROFILE_PARENT=EXTENSIONS
   # Server Profile - Extensions
   - SERVER_PROFILE_EXTENSIONS_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
   - SERVER_PROFILE_EXTENSIONS_PATH=layered-profiles/extensions/pingfederate
   - SERVER_PROFILE_EXTENSIONS_PARENT=GETTING_STARTED
   # Base Server Profile
   - SERVER_PROFILE_GETTING_STARTED_URL=https://github.com/pingidentity/pingidentity-server-profiles.git
   - SERVER_PROFILE_GETTING_STARTED_PATH=getting-started/pingfederate
   # **** SERVER PROFILE END ****
   ```

Deploying the layered profile

1. Push your profiles and updated `docker-compose.yaml` file to your GitHub repository.

2. Deploy the stack with the layered profiles.

To view this example in its entirety, including the profile layers and `docker-compose.yaml` file, see the pingidentity-server-profiles/layered-profiles directory.

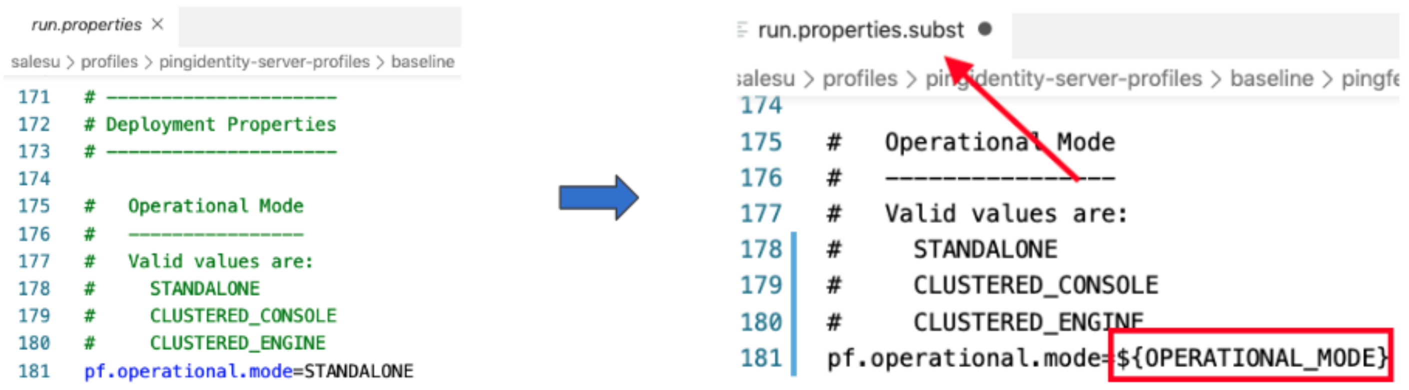
Environment Substitution

In a typical environment, a product configuration is moved from server to server. Hostnames, endpoints, DNS information, and more need a way to be easily modified.

By removing literal values and replacing them with environment variables, configurations can be deployed in multiple environments with minimal change.

> *When templating profiles with variables that reference other products, use the conventions defined in PingBase Image Directory.*

All of our configuration files can be parameterized by adding variables using the syntax: `${filename.ext}.subst`.


run.properties
salesu > profiles > pingidentity-server-profiles > baseline
run.properties.subst
salesu > profiles > pingidentity-server-profiles > baseline > pingfe

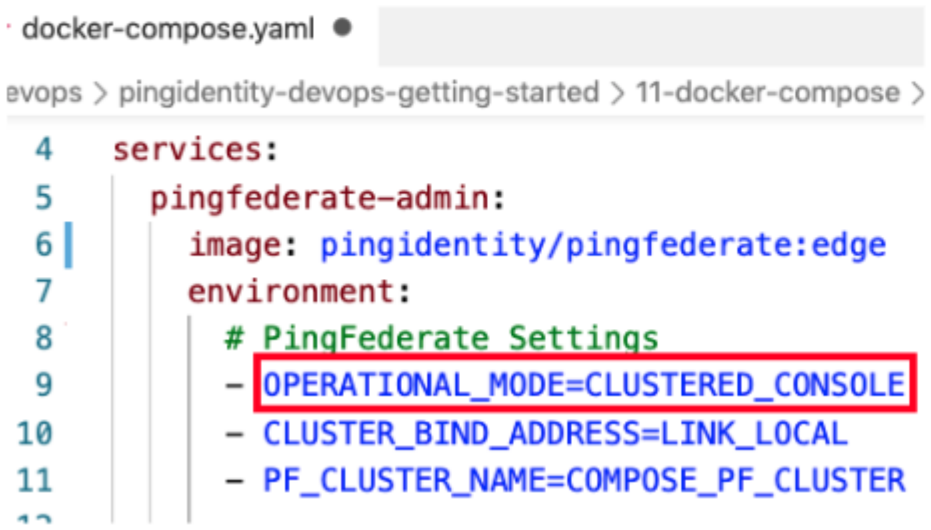
```
# ---------------------
# Deployment Properties
# ---------------------

#   Operational Mode
#   ----------------
#   Valid values are:
#     STANDALONE
#     CLUSTERED_CONSOLE
#     CLUSTERED_ENGINE
pf.operational.mode=STANDALONE
```

```

#   Operational Mode
#   ----------------
#   Valid values are:
#     STANDALONE
#     CLUSTERED_CONSOLE
#     CLUSTERED_ENGINE
pf.operational.mode=${OPERATIONAL_MODE}
```

Passing Values to Containers

Within the environment section of your container definition, declare the variable and the value for the product instance.

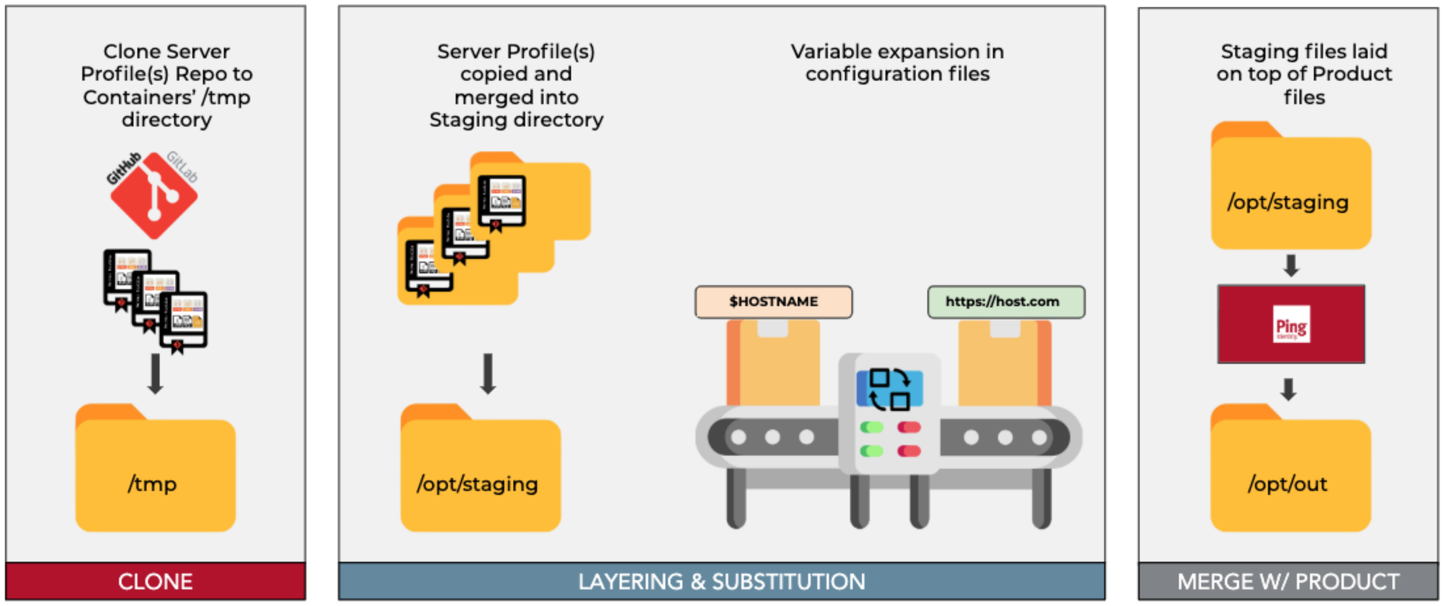
Values can be defined in many sources, such as inline, env_vars files, and Kubernetes ConfigMaps.


docker-compose.yaml
evops > pingidentity-devops-getting-started > 11-docker-compose >

```
services:
  pingfederate-admin:
    image: pingidentity/pingfederate:edge
    environment:
      # PingFederate Settings
      - OPERATIONAL_MODE=CLUSTERED_CONSOLE
      - CLUSTER_BIND_ADDRESS=LINK_LOCAL
      - PF_CLUSTER_NAME=COMPOSE_PF_CLUSTER
```

How it Works

1. A container startup is initiated.
2. The configuration pulls a server profile from Git or from a bind mounted `/opt/in` volume.
3. All files with a `.subst` extension are identified.
4. The environment variables in the identified `.subst` files are replaced with the actual environment values.
5. The `.subst` extension is removed from all the identified files.
6. The product instance for the container is started.


Clone Server Profile(s) Repo to Containers' /tmp directory
GitHub
GitLab
/tmp
CLONE
Server Profile(s) copied and merged into Staging directory
/opt/staging
Variable expansion in configuration files
$HOSTNAME
https://host.com
LAYERING & SUBSTITUTION
Staging files laid on top of Product files
/opt/staging
Ping
/opt/out
MERGE W/ PRODUCT

Including Extensions in PingData Server Profiles

Server SDK extension zip files can be included in your server profile for PingData products (PingAuthorize, PingDataSync, PingDirectory, and PingDirectoryProxy). The zip files can be included directly, or can be pulled from a remote URL when the container starts up.

The pd.profile/server-sdk-extensions/ directory

Any desired extension zip files should be included in the pd.profile/server-sdk-extensions/ directory of your server profile. Extension zip files in this directory will be installed during the setup process.

Pulling extension zip files from an external URL

The hook scripts support pulling down extension zip files from a defined URL, to avoid having to commit zip archives to a Git repository. To do this, a `remote.list` file should be included in the extensions/ directory of your server profile. Any file with a name ending in `remote.list` in the extensions/ directory will be treated as a list of extensions.

Note

Ensure extensions are in the right folder

When listing extensions to pull down via curl, the list must be placed in the `extensions/` directory of the server profile. When directly including extension zip files, the zip files must be placed in the `pd.profile/server-sdk-extensions/` directory of the server profile.

For an example, see the extension list included in our baseline PingDirectory server profile.

Separate multiple extensions with line breaks.

A URL can also be specified in the downloaded zip file. To do this, add a space between the extension zip URL and the URL that will provide the SHA1 hash. For example:

```
https://example.com/extension.zip https://example.com/extension/sha1
```

Set the `ENABLE_INSECURE_REMOTE_EXTENSIONS` environment variable to `true` to allow installing extensions without the SHA1 hash check. By default the SHA1 check will be required. If a SHA1 URL is provided and the SHA does not match or the URL cannot be reached, the extension will not be installed.

Using Private Git Repositories

In general, you do not want your server profiles to be public. Instead, you should persist your server profiles in private git repositories.

To use server profiles with private repositories, you must either:

- Pull using HTTPS
- Pull using SSH

For HTTPS with GitHub:

1. Generate an access token in GitHub.
2. Specify the access token in the URL you assign to the `SERVER_PROFILE_URL` environment variable in your YAML files.

For SSH:

- Include your keys and known hosts in the image under `/home/ping/.ssh`.

Cloning from GitHub using HTTPS

Creating a GitHub Access Token

1. In GitHub, go to `Settings` --> `Developer Settings` --> `Personal access tokens`.
2. Click `Generate new token` and assign the token a name.
3. Grant the token privilege to the `repo` group.

> *Copy the token to a secure location. You won't be able to view the token again.*

4. At the bottom of the page, click `Generate Token`.

Using The Token In YAML

To use the token in your YAML file, include it in the `SERVER_PROFILE_URL` environment variable using this format:

```
html https://<github-username>:<github-token>@github.com/<your-repository>.git
```

For example:

```
SERVER_PROFILE_URL=https://github_user:zqb4famrbadjv39jdi6shvl1xvozut7tamd5v6eva@github.com/pingidentity/
server_profile.git
```

Using Git Credentials in Profile URL

Typically, variables in a `SERVER_PROFILE_URL` string are not replaced. However, certain Git user and password variables can be replaced.

- To substitute for the user and password variables using values defined in your YAML files, include either or both `${SERVER_PROFILE_GIT_USER}` and `${SERVER_PROFILE_GIT_PASSWORD}` in your server profile URL. For example:

  ```
  SERVER_PROFILE_URL=https://${SERVER_PROFILE_GIT_USER}:${SERVER_PROFILE_GIT_PASSWORD}@github.com/pingidentity/
  server_profile.git
  ```

- When using layered server profiles, each layer can use the base user and password variables, or you can define values specific to that layer.

  For example, for a `license` server profile layer, you can use the `SERVER_PROFILE_LICENSE_GIT_USER` and `SERVER_PROFILE_LICENSE_GIT_PASSWORD` variables, and substitute for those variables using values defined in your YAML files.

Cloning using SSH

- To clone using SSH, you can mount the necessary keys and known hosts files using a volume at `/home/ping/.ssh`, the home directory of the default user in our product images.
- To clone from GitHub, you must add the necessary SSH keys to your account through the account settings page.

PingDirectory Metrics with Prometheus

In the past, enabling metrics for PingDirectory required a manual process to setup the statsd configuration to enable the data to be made available to Prometheus. However, PingDirectory now includes an HTTP servlet extension that can be enabled to expose metrics in Prometheus format.

You can refer to the documentation for the `dsconfig` commands to enable the Prometheus metrics. The link above is for PingDirectory 9.2, but the process is the same for newer versions.

These `dsconfig` commands can be included in a server profile to ensure that the configuration is applied when the server is started. See here for an example.

Using Hashicorp Vault

This documentation provides details for using Hashicorp Vault and secrets with Ping Identity DevOps images.

> *Note: the PingIdentity DevOps images and Helm chart only support version 2 of the KV secrets engine API for Vault secrets. PingDirectory itself currently only supports KV version 1 for password storage schemes. Learn more in the Vault KV secrets engine documentation.*

Before you begin

You must:

- Complete Get Started to set up your DevOps environment and run a test deployment of the products.
- Have a running Hashicorp Vault instance.

About this task

The following examples explain and show:

- How to use HashiCorp Vault Secrets in native PingIdentity DevOps images
- How to use HashiCorp Vault Injector in Kubernetes deployments

Kubernetes - HashiCorp Vault Injector

If you are using Kubernetes to deploy your containers, it's highly recommended to use the HashiCorp Vault Injector. The following provides details on how to use secrets in a non-Kubernetes deployment, such as Docker-compose.

If the HashiCorp Vault Injector Agent is installed, annotations can be added to the `.yaml` file of a Pod, Deployment, StatefulSet resource to pull in the secrets. The snippet below provides an example set of annotations (placed in to the metadata of the container) to pull in a `pf.jwk` secret into a container.

Helm Chart Stateful Set

This is an StatefulSet example created using the PingIdentity DevOps Helm Chart.

```
apiVersion: apps/v1
kind: StatefulSet
spec:
  template:
    metadata:
      annotations:
        vault.hashicorp.com/agent-init-first: "true"
        vault.hashicorp.com/agent-inject: "true"
        vault.hashicorp.com/agent-inject-secret-devops-secret.env.json: secret/.../devops-secret.env
        vault.hashicorp.com/agent-inject-template-devops-secret.env.json: |
          {{ with secret "secret/.../devops-secret.env" -}}
          {{ .Data.data | toJSONPretty }}
          {{- end }}
        vault.hashicorp.com/agent-inject-secret-devops-secret.env.json: secret/.../passwords
        vault.hashicorp.com/agent-inject-template-passwords.json: |
          {{ with secret "secret/.../passwords" -}}
          {{ .Data.data | toJSONPretty }}
          {{- end }}
        vault.hashicorp.com/agent-pre-populate-only: "true"
        vault.hashicorp.com/log-level: info
        vault.hashicorp.com/preserve-secret-case: "true"
        vault.hashicorp.com/role: k8s-default
        vault.hashicorp.com/secret-volume-path: /run/secrets
```

Secrets - Variables

Using the previous example, the value for secret `secret/.../devops-secret.env` JSON will be pulled into the container as `/run/secrets/devops-secret.env.json`.

Because this secret ends in the value of `.env`, it will further be turned into a property file with NAME=VALUE pairs and is available to the container environment on start up.

Example of devops-secret.env transformed into files

```
{
  "PING_IDENTITY_DEVOPS_USER": "jsmith@example.com",
  "PING_IDENTITY_DEVOPS_KEY": "xxxxx-xxxx-xxxxx-xxxxx-xxxx"
}
```

creates the files

```
File: /run/secrets/devops-secret.env
Contents: PING_IDENTITY_DEVOPS_USER="jsmith@example.com"
          PING_IDENTITY_DEVOPS_KEY="xxxxx-xxxx-xxxxx-xxxxx-xxxx"
```

Secret - Files

Using the previous example, the value for secret `secret/.../passwords` JSON will be pulled into the container as `/run/secrets/passwords.json` and for every key/value in that secret, a file will be created with the name of the `key` and contents of `value`.

Example of /run/secrets/passwords.json transformed into files

```
{
  "root-user-password": "secret-root-password",
  "admin-password": "secret-admin-password"
}
```

creates the files

```
File: /run/secrets/secret-root-password
Contents: secret-root-password

File: /run/secrets/secret-admin-password
Contents: secret-admin-password
```

Native DevOps HashiCorp Support

Vault secrets can also be used in native PingIdentity DevOps images regardless of the environment they are deployed in, for example, Kubernetes, Docker, and Docker-compose. In these cases, there is no injector agent required.

Warning

This does require some type of authentication to your Vault, such as USERNAME/PASSWORD or TOKEN. The HashiCorp Injector method is recommended.

The following image depicts the components and steps for pulling secrets into a container at start-up.

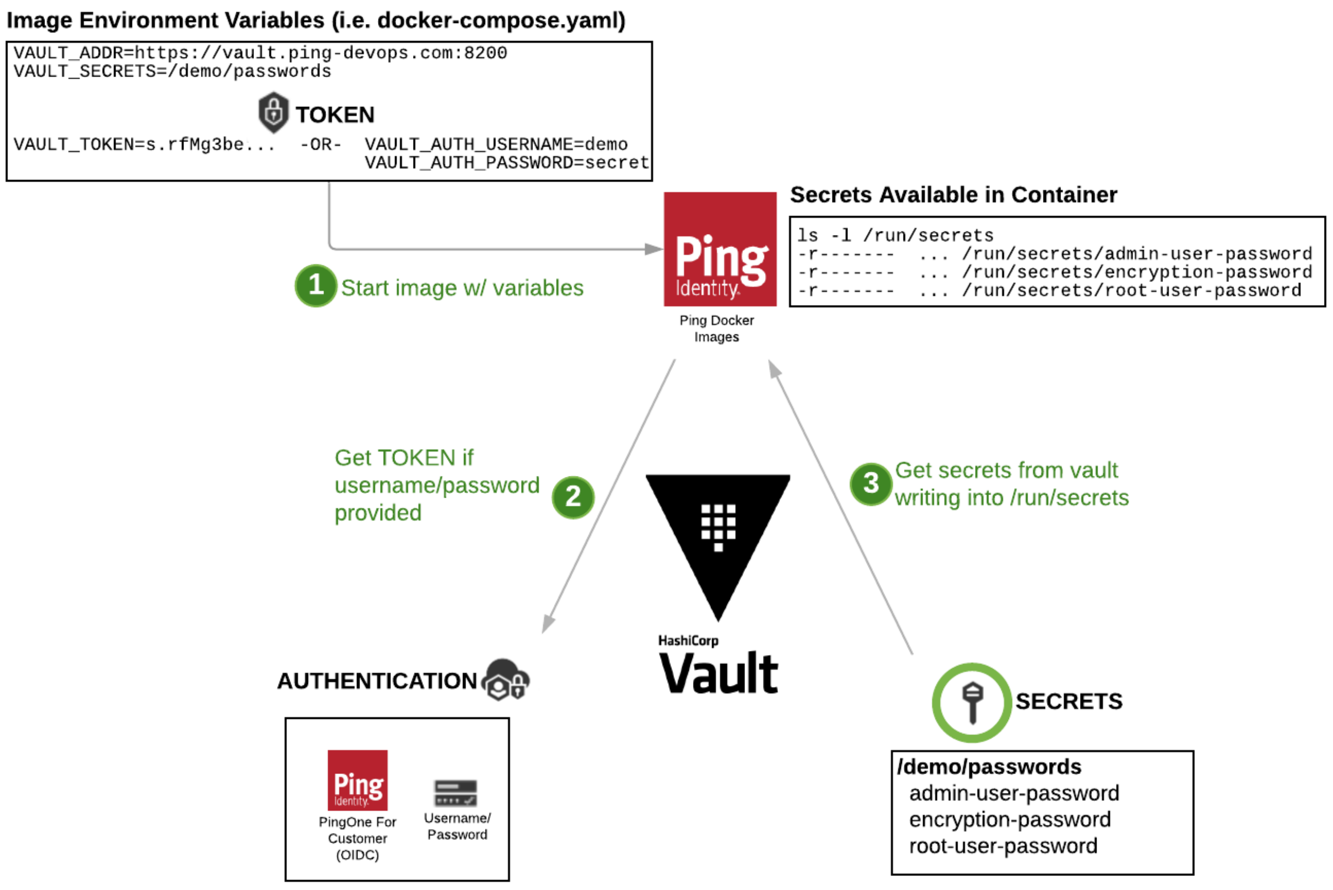


Image Environment Variables (i.e. docker-compose.yaml)
VAULT_ADDR=https://vault.ping-devops.com:8200
VAULT_SECRETS=/demo/passwords
TOKEN
VAULT_TOKEN=s.rfMg3be... -OR- VAULT_AUTH_USERNAME=demo
VAULT_AUTH_PASSWORD=secret
1 Start image w/ variables
Ping Identity
Ping Docker Images
Secrets Available in Container
ls -l /run/secrets
-r------- ... /run/secrets/admin-user-password
-r------- ... /run/secrets/encryption-password
-r------- ... /run/secrets/root-user-password
Get TOKEN if username/password provided 2
3 Get secrets from vault writing into /run/secrets
HashiCorp Vault
AUTHENTICATION
PingOne For Customer (OIDC)
Username/ Password
SECRETS
/demo/passwords
admin-user-password
encryption-password
root-user-password

You can use the following variables to deploy images that will pull secrets from the Vault.

Variable	Example	Description
SECRETS_DIR	/run/secrets	Location for storing secrets. See section below on using a `tmpfs` mounted filesystem to store secrets in a memory location.
VAULT_TYPE	hashicorp	Type of vault used. Currently supporting hashicorp.
VAULT_ADDR	https://vault.example.com:8200	URL for the vault with secrets
VAULT_TOKEN	s.gvC3vd5aFz......JovV0b0A	Active token used to authenticate/ authorize container to vault. Optional if VAULT_AUTH_USERNAME/ VAULT_AUTH_PASSWORD are provided.
VAULT_AUTH_USERNAME	demo	Username of internal vault identity. Optional if VAULT_TOKEN is provided.

Variable	Example	Description
VAULT_AUTH_PASSWORD	2FederateM0re	Password of internal vault identity. Optional if VAULT_TOKEN is provided.
VAULT_SECRETS	/pingfederate/encryption-keys	A list of secrets to pull into the container. Must be the full secret path used in vault.

The following example shows how these would be used in a docker-compose.yaml file. This example provides two secrets, as denoted by the VAULT_SECRETS setting.

```
services:
  pingfederate:
    image: pingidentity/pingfederate:edge
    environment:
      # ...
      ################################################
      # Vault Info
      ################################################
      - VAULT_TYPE=hashicorp
      - VAULT_ADDR=https://vault.ping-devops.com:8200
      - VAULT_AUTH_USERNAME=demo
      - VAULT_AUTH_PASSWORD=2FederateM0re
      - VAULT_SECRETS=/demo/passwords \
                      /demo/getting-started/pingfederated/pf-keys
```

The secret types (Variables/Files) are processed the same way as with the HashiCorp Injector Method above.

Secrets - Base64

Often, there are secrets that might be of a binary format, such as certificates.

Special key name suffixes can be used to perform certain processing on the keys when the file is created. The following table provides examples of how keys with special suffixes.

Key Suffix	Description
.b64 or .base64	Specifies that the value is base64 encoded and the resulting file should be decoded when written, without the suffix.

There is a message that is base64 encoded and stored in the vault as secret /demo/b64-demo hello.b64.

```
Secret: /demo/b64-demo

  KEY         VALUE
  ---         -----
  hello.b64   SGVsbG8gV29ybGQhCg==
```

would result in the following file:

```
/run/secrets/hello
  CONTENTS
  --------
  Hello World!
```

Using tmpfs for Secrets

It's best practice to place secrets in a volume that won't be persisted to storage with the possibility that it might be improperly accessed at any point in the future, such as backups and environment variables.

Kubernetes automatically provides the default `SECRETS_DIR` of `/run/secrets` for this.

If using Docker, you should create a `tmpfs` type volume and size it to `32m` and mount it to a path of /run/secrets.

Docker-compose version > 2.4

Requires Docker-compose version 2.4 or later, due to the options provided to the tmpfs volumes definition.

Creates a `/run/secrets` volume under tmpfs.

```
version: "2.4"
services:
  pingfederate:
    image: pingidentity/pingfederate:edge
    environment:
    #...
    tmpfs: /run/secrets

      # ---- or -----

    volumes:
      - type: tmpfs
        target: /run/secrets
        tmpfs:
          size: 32m
```

See this mount by exec'ing into the container and running:

```
> df -k /run/secrets
Filesystem           1K-blocks      Used Available Use% Mounted on
tmpfs                    16384         0     16384   0% /run/secrets
```

Securing the Containers

Docker Best Practices

Please visit the Docker website for more information on best practices to secure a container.

Kubernetes Best Practices

Please visit the Kubernetes website for more information on best practices to secure a deployment.

Ping Identity's Docker Image Hardening Guide

For best practices on securing your product Docker image, see Ping Identity's Hardening Guide.

Upgrading Deployments

Managing Deployments

In addition to Customizing Deployments, you must maintain your deployments over time as new versions of our products are released and as you tune your deployments to better reflect your changing needs.

Upgrading PingAccess

In a DevOps environment, upgrades can be simplified through automation, orchestration, and separation of concerns.

Notice

- Upgrading from PingAccess versions prior to 6.3.6 will not work using this method.

Caveats

1. This Document Assumes Kubernetes and Helm

   The terms in this document will focus on deployments in a Kubernetes Environment using the ping-devops Helm chart. However, the concepts should apply to any containerized PingAccess Deployment.

2. This Document will Become Outdated

   The examples referenced in this document point to a specific tag. This tag may not exist anymore at the time of reading. To correct the issue, update the tag on your file to `N-1` from the current PF version.

3. Irrelevant Ingress

   The values.yaml files mentioned in this document expects an nginx ingress controller with class `nginx-public`. It is not an issue if your environment does not have this class. In such cases, the created ingresses will not be used.

Configuration Forward

Steps:

1. Deploy your old version of PingAccess with server profile
2. Export the configuration as a data.json file
3. Copy the pa.jwk file to your server profile
4. Deploy new PingAccess version with server profile

Here we will walk through an example upgrade.

Deploy your old version of PingAccess with server profile

Make sure you have a devops-secret

If you are using this example as-is, you will need a devops-secret.

Change Ingress Domain

Be sure to change the ingress domain name value to your domain in 01-original.yaml.

Change Image Tag

Be sure to change the image tag value in 01-original.yaml.

PingFederate and PingAccess Deployment

In order to use the baseline server profile as outlined in this guide, you have to deploy PingFederate along with PingAccess.

Navigate to the getting started repository and deploy your old version of PingAccess.

```
helm upgrade --install pa-upgrade pingidentity/ping-devops -f 30-helm/pingaccess-upgrade/01-original.yaml
```

Export the configuration as a data.json file

After your cluster is healthy, export the configuration as a json file and add it to your server profile so the start-up-deployer can use it to configure your upgraded PingAccess.

```
$ curl -k -u Administrator:2FederateM0re -H "X-Xsrf-Header: PingAccess" https://pa-upgrade-pingaccess-admin.ping-
devops.com/pa-admin-api/v3/config/export >~/<insert path to server profile here>/pingaccess/instance/data/data.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 22002  100 22002    0     0  42664      0 --:--:-- --:--:-- --:--:-- 43056
```

Copy the pa.jwk file to your server profile.

Copy the `/conf/pa.jwk` file.

```
$ kubectl cp pa-upgrade-pingaccess-admin-0:/opt/out/instance/conf/pa.jwk ~/<insert path to server profile here>/
pingaccess/instance/conf/pa.jwk
Defaulted container "pingaccess-admin" out of: pingaccess-admin, wait-for-pingfederate-engine (init), generate-
private-cert-init (init)
tar: removing leading '/' from member names
```

Check to see that the data.json and pa.jwk files have been updated in your server-profile and push these changes to your repository

Deploy new PingAccess version with server profile

Make sure to uninstall your old Ping Access cluster and remove any pvc's created.

```
$ helm uninstall pa-upgrade
release "pa-upgrade" uninstalled
$ kubectl get pvc
NAME                                       STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
out-dir-pa-upgrade-pingaccess-admin-0   Bound    pvc-c1e5cd9b-35f5-4260-8704-3075fcf9b36e   4Gi        RWO
gp2            7m5s
$ kubectl delete pvc out-dir-pa-upgrade-pingaccess-admin-0
persistentvolumeclaim "out-dir-pa-upgrade-pingaccess-admin-0" deleted
```

Finally, update the PingAccess image version to the new target version and run.

Change Ingress Domain

Be sure to change the ingress domain name value to your domain in 02-upgraded.yaml.

Change Image Tag

Be sure to change the image tag value in 02-upgraded.yaml.

Change Server Profile

Be sure to change the server profile url and path in 02-upgraded.yaml.

```
helm upgrade --install pa-upgrade pingidentity/ping-devops -f 30-helm/pingaccess-upgrade/02-upgraded.yaml
```

At this time, you should have an upgraded PingAccess instance

Upgrading PingCentral

Example Only

This example is for demonstration purposes only. It's not intended to reflect the complexities of a full production environment and will need to be adapted accordingly.

Video walkthrough

A video demonstration of this process can be found here.

Caveats

Kubernetes and Helm

This document will focus on deployments in a Kubernetes environment using the ping-devops Helm chart. However, the concepts should apply to any containerized PingCentral deployment.

This document will become outdated

The examples referenced in this document point to a specific tag. This tag may not exist at the time of reading. To correct the issue, update the tag on your files appropriately. This example uses versions 1.10 and 1.14, but the process should be similar for other versions in the future.

MySQL is used as the PingCentral Datastore

A separate MySql container is deployed manually to provide a backing store for PingCentral. In a production environment, you would likely use a managed database service.

> ⚠ **H2 Internal Database**
>
> If you are not using an external database, and are on a version <= 1.10, you will need to be aware of possible issues with the move to the H2v2 database internally. See the PingCentral documentation for more details. It should be noted that the internal H2 database is not supported for production environments.

The Ping-provided baseline Profile is used as a starting point for PingCentral

The default baseline profile is used for this guide. In a production environment, you would likely use a custom repository and profile.

Overall Process

Steps:

1. Deploy the old (1.10) version of PingCentral with the baseline server profile
2. Create a test user in PingCentral and perform other validation steps
3. Copy the `pingcentral.jwk` file to your server profile
4. Deploy the new (1.14) PingCentral version with a custom server profile
5. Validate the upgrade

Prerequisites

Assumptions and requirements:

- You have set up your DevOps environment and can run a test deployment of the products. For more information, see Get Started.
- This example was written using Docker Desktop with Kubernetes enabled on the Mac platform. The version used was `4.27.1 (136059)`, which includes Docker Engine `v25.0.2` and Kubernetes `v1.29.1`. The ingress-nginx controller version was `1.9.6`, deployed from Helm chart version `4.9.1`.

Environment Preparation

Clone the `getting-started` repository

1. Clone the `pingidentity-devops-getting-started` repository to your local `${PING_IDENTITY_DEVOPS_HOME}` directory.

   pingctl utility

   The `${PING_IDENTITY_DEVOPS_HOME}` environment variable was set by running `pingctl config`.

```
cd "${PING_IDENTITY_DEVOPS_HOME}"
git clone \
  https://github.com/pingidentity/pingidentity-devops-getting-started.git
```

Prepare the environment with a namespace and ingress controller

1. Create a namespace for running the stack in your Kubernetes cluster.

```
# Create the namespace
kubectl create ns pingcentral-upgrade
# Set the kubectl context to the namespace
kubectl config set-context --current --namespace=pingcentral-upgrade
# Confirm
kubectl config view --minify | grep namespace:
```

2. Deploy the ingress controller to Docker Desktop:

```
helm upgrade --install ingress-nginx ingress-nginx \
--repo https://kubernetes.github.io/ingress-nginx \
--namespace ingress-nginx --create-namespace
```

3. Wait for the Nginx ingress to reach a healthy state by running the following command. You can also observe the pod status using k9s or by running `kubectl get pods --namespace ingress-nginx`. You should see one controller pod running when the ingress controller is ready. This command should exit after no more than 60 seconds or so, depending on the speed of your computer:

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s
```

4. Create a secret in the namespace you will be using to run the example (*upgrade*) using the `pingctl` utility. This secret will obtain an evaluation license based on your Ping DevOps username and key:

```
pingctl k8s generate devops-secret | kubectl apply -f -
```

5. This example will use the Helm release name `demo` and DNS domain suffix `*pingdemo.example` for accessing applications. Add the expected hostname to `/etc/hosts`:

```
echo '127.0.0.1 demo-pingcentral.pingdemo.example' | sudo tee -a /etc/hosts > /dev/null
```

6. Navigate to your local directory where you cloned the repository (`"${PING_IDENTITY_DEVOPS_HOME}"/pingidentity-devops-getting-started/30-helm/`) directory and run the command shown here to deploy the MySQL pod using `kubectl`. This deployment will be used as the backing store for PingCentral.

```
kubectl apply -f pingcentral-external-mysql-db/mysql.yaml
```

Fork or clone the Ping server profile repository

1. Fork the pingidentity-server-profiles repository to your GitHub account. If you do not have a GitHub account, you can clone the repository instead. For this guide, the repository will be forked to `test-server-profiles` which will be pulled locally to `${HOME}/projects/test-server-profiles`.

```
cd "${HOME}/projects"
git clone <account>/test-server-profiles.git ${HOME}/projects/
```

Deploy the old version of PingCentral with the Ping baseline server profile

1. Install the initial version by running the command shown here. In this example, the release `demo` forms the prefix for all objects created. The ingress is configured to use the ping-local domain:

```
cd "${PING_IDENTITY_DEVOPS_HOME}"/pingidentity-devops-getting-started/30-helm/
helm upgrade --install demo pingidentity/ping-devops -f pingcentral-upgrade/01-original.yaml
```

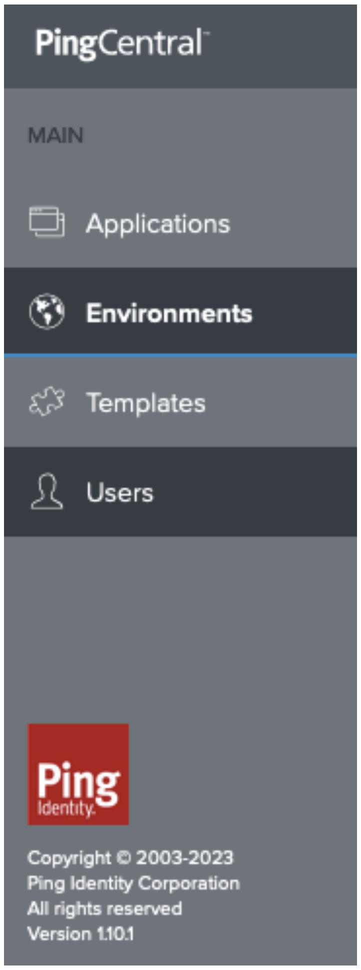
This command will take a few minutes to complete. You can monitor the progress using `kubectl get pods` or `k9s`.

Access the PingCentral Console

1. Login to the PingCentral Administrative Console using the credentials `administrator/2Federate`. The URL is https://demo-pingcentral.pingdemo.example.

2. Select Users from the left navigation menu and click Add User. Create a user to be used to validate PingCentral after the upgrade (the user specifics and role do not matter for this example).

The left navigation panel should indicate version 1.10.0 at the bottom as shown here:



Confirm Database Entries

Shell into the MySQL pod and confirm that the user you created is present in the database. Also, verify the version of PingCentral in the `DATABASECHANGELOG` table.

```
kubectl exec -it mysql-0 -- bash
# Use the password from the mysql.yaml file (2Federate)
mysql -u root -p
use pingcentral;
# The user will be listed here
select * from users;
# The version will be listed here, with the last lines of the table showing the most recent entries
# at version v1.10 and v1.11 (46 lines total at the time of this writing)
select * from DATABASECHANGELOG;
```

Add the `pingcentral.jwk` file to your server profile

1. Copy the `pingcentral.jwk` file from the PingCentral pod. This file will be used in the next step to configure the new PingCentral instance. This example places the file in the repository directory from the fork created earlier.

```
kubectl cp demo-pingcentral-6d4bb97c98-m7vwb:/opt/out/instance/conf/pingcentral.jwk ${HOME}/projects/test-
server-profiles/baseline/pingcentral/external-mysql-db/instance/conf/pingcentral.jwk
```

2. Check to see that the `pingcentral.jwk` file has been placed in your server-profile and push these changes to your repository.

```
cd "${HOME}/projects/test-server-profiles"
git add .
git commit -m "Added pingcentral.jwk file"
git push
```

Pod Name

The pod name from which you copy will vary.

JWK Unique to PingCentral Instance

The `pingcentral.jwk` file encrypts and decrypts the PingCentral configuration. It's unique to each PingCentral instance and you must copy it to the new server profile; otherwise, the new pod won't start.

File copy error

If you get a `tar: removing leading '/' from member names` notification when copying the file, you can ignore it.

Security Warning

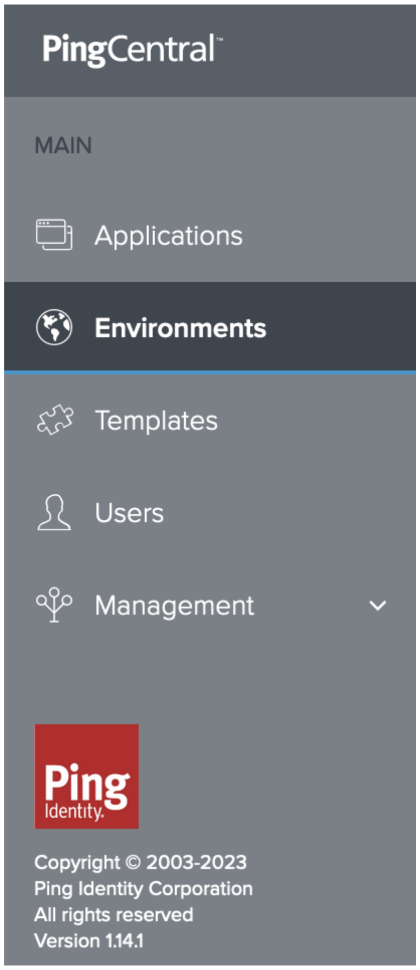
Storing the `pingcentral.jwk` file in the server profile *is not recommended* for production environments. In a production environment, you'd likely use a managed key store service, vault, or other encrypted mechanism.

Deploy the new version of PingCentral with a custom server profile

1. Update the `/pingidentity-devops-getting-started/30-helm/pingcentral-upgrade/02-upgraded.yaml` file to point to the repository and profile directory that contains the JWK file. This example uses the `test-server-profiles` repository, but you should use your own information.

```
helm upgrade --install demo pingidentity/ping-devops -f pingcentral-upgrade/02-upgraded.yaml
```

The new pod will spin up, and when it is healthy, the old pod will be terminated. At this time, you should have an upgraded PingCentral instance. Log in to the administrative console as before. The user you created earlier should still exist, and the version information at the lower left should indicate version 1.14.0. In addition, a new left navigation item (Management) will be present that was not there before:



Copyright © 2003-2023
Ping Identity Corporation
All rights reserved
Version 1.14.1

Finally, a check of the `DATABASECHANGELOG` table in the MySQL pod should show the new version of PingCentral as the last few entries in that table (51 entries as of this writing). These updated entries indicate the database migration was successful.

Not all versions have database migrations

The 1.10 → 1.14 upgrade involved database updates. Not all versions will have database updates, and corresponding `DATABASECHANGELOG` entries might not be present.

Cleanup

After you have finished this demonstration, you can uninstall the Helm release and MySQL deployment, and delete the namespace:

```
helm uninstall demo
kubectl delete -f pingcentral-external-mysql-db/mysql.yaml
kubectl delete ns pingcentral-upgrade
```

Remove the entry from `/etc/hosts` if you do not plan to use the same hostname again:

```
sudo sed -i '' '/demo-pingcentral.pingdemo.example/d' /etc/hosts
```

You can also remove the `test-server-profiles` repository from your local machine and delete the forked repository from GitHub:

```
rm -rf "${HOME}/projects/test-server-profiles"
```

Upgrading PingDirectory

Because PingDirectory is essentially a database, in its container form, each node in a cluster has its own persisted volume. Additionally, because PingDirectory is an application that *focuses* on state, rolling out an upgrade isn't really the same as any other configuration update. However, the product software, and scripts in the image provide a process through which upgrades are drastically simplified.

This use case focuses on a PingDirectory upgrade in a default Kubernetes environment where you upgrade a PingDirectory StatefulSet 9.0.0.1 to 9.1.0.0 using an incremental canary roll-out.

Tips

To ensure a successful upgrade process:

- Avoid combining configuration changes and version upgrades in the same rollout. This adds unnecessary complexity to debugging errors during an upgrade.
- Successfully complete an upgrade in a proper Dev/QA environment before trying anything in production.
- Upgrades will happen on one server at a time. Ensure you have enough resources on the remaining machines to prevent client impact.
- Follow a canary deployment pattern to ensure the changes will be successful before doing a full rollout. There is no *good* way to roll back a completed upgrade, so take any necessary steps to avoid this.

Before you begin

You must:

- Complete Get Started to set up your DevOps environment and run a test deployment of the products using Helm.
- Clone or download the `pingidentity-devops-getting-started/30-helm/pingdirectory-upgrade-partition` repository to your local `${HOME}/projects/devops` directory.
- Understand how to use our DevOps server profiles.
- Have access to a Kubernetes cluster.
- If you're upgrading in your own environment and using mounted licenses, have the license for the existing version in the `/opt/out` persisted volume and a license for the new version needs to be in `/opt/in`.

  The license locations aren't an needed if you're using our DevOps credentials in an evaluation context.

About this task

You will:

- Start with a base stack.
- Set up a partition to make changes on just one node.
- Deploy changes to one node and fix any errors.
- Rollout changes to other nodes.

Upgrade process overview

The key functionality for PingDirectory upgrades is the relationship between the image hooks and the `manage-profile` command in the product.

The upgrade is processed as follows:

1. When a node starts for the first time, it' i's in `SETUP` mode and runs `manage-profile setup`.
2. When a node restarts (for whatever reason), it runs `manage-profile replace-profile`.
   - This command compares the new profile to the old profile, and if there is a change, it tries standing up the server with the new profile.
   - Errors are thrown if there's a configuration in the new profile that prevents it from being applied.
3. If `manage-profile replace-profile` detects a product version difference, it takes the same approach as any other restart. It attempts to migrate PingDirectory to the new version, and if it can't, the command fails.

   Because there is processing that happens automatically in the container during the upgrade, you should roll the change out to a small partition first and test it thoroughly before rolling it out to all. This partition also gives us room to revert back without impacting traffic in case something is not as expected.

   This process in standard Kubernetes is called a Canary Rollout and is derived from Kubernetes documentation on StatefulSet update strategies.

   > *"Canary Rollout" in this scenario focuses only on an incremental rollout of containers, not actually separating traffic. That could be done with additional tools like Istio or standing up another service and pairing separate labels and selectors.*

Setting up the base stack

The YAML configuration files for this use case are in your cloned local copy of the `pingidentity-devops-getting-started` repository

1. To use the `1-initial.yaml` file in your local `pingidentity-devops-getting-started/30-helm/pingdirectory-upgrade-partition` directory to start with a PingDirectory StatefulSet using persistent volumes, enter:

   ```
   helm upgrade --install releasename pingidentity/ping-devops -f 1-initial.yaml
   ```

> *All helm commands for this use case need to be run from the* `pingidentity-devops-getting-started/30-helm/pingdirectory-upgrade-partition` *directory.*

This stands up a two directory topology, each with its own Persistent Volume Claim using the default storage class.

2. Wait for both nodes to be healthy before continuing and then enter:

```
kubectl get pods
```

`pingdirectory-0` and `pingdirectory-1` should show `1/1` in the `READY` column.

Setting up a partition

To use the `2-partition.yaml` file in your local `pingidentity-devops-getting-started/30-helm/pingdirectory-upgrade-partition` directory to add a partition to `StatefulSet` for `updateStrategy`, enter:

```
helm upgrade --install releasename pingidentity/ping-devops -f 2-partition.yaml
```

This partition configuration signifies that any changes to `spec.template` will only be applied to nodes with a cardinal value higher than the partition value.

The only other change is to the image tag. When this change is applied:

1. The `pingdirectory-1` pod is terminated and a new one with the new image is started.
2. The new PingDirectory container is based on a specific version of PingDirectory, found in the `/opt/server` volume.
3. The `manage-profile replace-profile` command is eventually triggered when the container detects PingDirectory is in a `RESTART` state. This command identifies the difference in the database version running, based on the persisted volume attached to `/opt/out`, and then attempts to upgrade. Information similar to the following will be displayed:

```
pingdirectory-1 Validating source and existing servers ..... Done
pingdirectory-1 Updating the server version from 8.0.0.1 to 8.1.0.0. Local database backends
pingdirectory-1 will be exported before the update in case a revert is necessary
pingdirectory-1 Exporting backend with backendID userRoot. This may take a while ..... Done
pingdirectory-1 Running the update tool ..... Done
...
pingdirectory-1 Cleaning up after replace ..... Done
pingdirectory-1   manage-profile replace-profile returned 0
```

> *This process requires having licenses for both server versions available and in the right location.*

If `manage-profile replace-profile` completes without error, you'll see the container continue the migration and eventually start up PingDirectory.

If `manage-profile replace-profile` fails, an error is displayed and the container exits. The errors are because of some conflict in the server profile. The partition you set up previously provides an isolated environment for working out errors.

Work through any errors until you can get `manage-profile replace-profile` to complete successfully before continuing.

Rolling out the changes

When you're confident your upgrade will occur smoothly:

To use the `3-rollout-full-upgrade.yaml` file in your local `pingidentity-devops-getting-started/30-helm/pingdirectory-upgrade-partition` directory to deploy the rollout to the remaining nodes, enter:

```
helm upgrade --install releasename pingidentity/ping-devops -f 3-rollout-full-upgrade.yaml
```

This removes the partition.

Upgrading PingFederate

In a DevOps environment, upgrades can be simplified through automation, orchestration, and separation of concerns.

General Steps:

- Persistent Volume Upgrade of `/opt/out/instance/server/default/data` on pingfederate-admin
- Server Profile Upgrade
- Migrate Cluster Discovery Settings
- Post Upgrade

Persistent Volume Upgrade will include steps helpful to both pieces. Server Profile Upgrade will discuss extracting upgraded files.

Caveats

1. This Document Assumes Kubernetes and Helm

   The terms in this document will focus on deployments in a Kubernetes Environment using the ping-devops Helm chart. However, the concepts should apply to any containerized PingFederate Deployment.

2. This Document will Become Outdated

   The examples referenced in this document point to a specific tag. This tag may not exist anymore at the time of reading. To correct the issue, update the tag on your file to N -1 from the current PF version.

3. Upgrades from Traditional Deployment

It may be desirable to upgrade PingFederate along with migrating from a traditional environment. This is not recommended. Instead you should upgrade your current environment to the desired version of PingFederate and then create a profile that can be used in a containerized deployment.

4. Persistent Volume on `/opt/out`

   The suggested script should not be used if a persistent volume is attached to `/opt/out`. New software bits will not include special files built into the docker image. It is recommended to mount volumes on PingFederate Admin to `/opt/out/instance/server/default/data`.

5. Irrelevant Ingress

   The values.yaml files mentioned in this document expects and nginx ingress controller with class `nginx-public`. It is not an issue if your environment doesn't have this, the created ingresses will not be used.

Persistent Volume Upgrade

Steps needed in both Server-Profile upgrade and Persistent Volume upgrade include:

1. Deploy your PingFederate version and server profile as background process
2. Upgrade profile in container
   1. Backup the files in your profile.
   2. Download the PingFederate software bits for the new version.
   3. Run upgrade utility
   4. diff to view the changes. (optional)
3. Reconfigure any variablized components.
4. Export changes to your profile

Here we will walk through an example upgrade.

This Process Requires Container Exec Access

Your orchestration user will need access to `kubectl exec -it <pod> – sh` for multiple steps here.

Deploy PingFederate as a Background Process

Deploy your PingFederate version and server profile as background process with Helm:

Make sure you have a devops-secret

If you're using this example as is, you'll need a devops-secret.

Change Ingress Domain

Be sure to change the ingress domain name value to your domain in 01-background.yaml.

Change Image Tag

Be sure to change the image tag value in 01-background.yaml.

```
helm upgrade --install pf-upgrade pingidentity/ping-devops \
  --version 0.9.4 -f 30-helm/pingfederate-upgrade/01-background.yaml
```

The `args` section starts PingFederate as a background process and `tail -f /dev/null` as the foreground process.

Upgrade Profile in Container

The steps for upgrading can be automated with a script. Example scripts are included at `30-helm/pingfederate-upgrade/hooks`.

To use the scripts:

Copy the hooks folder to your PingFederate container

```
kubectl cp 30-helm/pingfederate-upgrade/hooks pf-upgrade-pingfederate-admin-0:/opt/staging
```

Copy the target PingFederate license to your PingFederate container See pingctl license documentation to retrieve an evaluation license, or provide an existing product license here.

```
kubectl cp pingfederate.lic pf-upgrade-pingfederate-admin-0:/tmp
```

Copy the target PingFederate software to your PingFederate container. See How to download Product Installation Files.

```
kubectl cp pingfederate-11.1.1.zip pf-upgrade-pingfederate-admin-0:/tmp
```

How to download Product Installation Files

1. Navigate to Ping Identity's Download webpage.
2. Select a Product download page, for example: PingFederate Download Page.
3. Click on the download button for the desired installation method and product version.
    1. If prompted to sign in, please sign in and the download will begin. Alternatively, Sign In Here.
    2. If you do not have a Ping Identity account, you can create one on the Account Creation Page.

Run the Upgrade Utility

The pf-upgrade.sh script will:

- Verify both the PingFederate software bits and new license file is on the container
- Backup the current /opt/out folder to /opt/current_bak
- Run the upgrade utility
- Overwrite /opt/out or /opt/out/instance/server/default/data with upgraded files
- Run diff between /opt/staging (server-profile location) and respective upgraded file. Diffs can be found in `/tmp/stagingDiffs`

Exec into the container and run the script.

```
kubectl exec -it pf-upgrade-pingfederate-admin-0 -- sh
cd /opt/staging/hooks
sh pf-upgrade.sh 10.3.4
```

At the conclusion of the script you will have an upgraded `/opt/out/instance/server/default/data` folder.

Server Profile Upgrade

If your profile is applied on each start of your container, you should keep your profile up to date with the product version you are deploying.

After the previously run script, you can find upgraded profile files in `/opt/new_staging` These files will be hard-coded and you should follow Build a PingFederate Profile as needed for templating.

Additionally, If you use the bulk-config data.json import it will not be found here. It should be imported via the standard process on the next container start.

Migrate Cluster Discovery Settings

To simplify future upgrades, migrate the cluster discovery settings in the `tcp.xml` file to the `jgroups.properties` file.

You can find the default `jgroups.properties` file here.

For more information, see Migrate Cluster Discovery Settings in the PingFederate admin guide.

Post Upgrade

To enable PingFederate admin as a foreground process, scale it down first.

```
kubectl scale sts pf-upgrade-pingfederate-admin --replicas=0
```

Finally, update PingFederate image version to new target PingFederate version and run as normal.

Change Ingress Domain

Be sure to change the ingress domain name value to your domain in 02-upgraded.yaml.

Change Image Tag

Be sure to change the image tag value in 02-upgraded.yaml.

```
helm upgrade --install pf-upgrade pingidentity/ping-devops --version 0.9.4 \
  -f 30-helm/pingfederate-upgrade/02-upgraded.yaml
```

This will restart the admin console, and trigger a rolling update of all the engines.

Old Profile

The final yaml `30-helm/pingfederate-upgrade/02-upgraded.yaml` still points to the same profile. The steps that should have been completed in Server Profile Upgrade were not included.

Connecting to the admin console will now show the upgraded version in cluster management.

Migrating from privileged images to unprivileged-by-default images

In the 2103 release, our product images were updated to run with an unprivileged user by default. Before this release, images ran as root by default. This document describes some important tips when moving from privileged to unprivileged images.

Checklist before migration

- To ensure that any configuration of the pods is maintained, build and commit a server profile from your current workload into a git repository.
  - See the Server Profile Structures page, and/or the product-specific guides for PingFederate and PingDirectory.
- For PingDirectory, export your user data that will be imported into the new server(s). You can include the basic DIT structure in the server profile (in the `pd.profile/ldif/userRoot/` directory), but actual user data should be left out; the server profile should store configuration, not data. You can save the actual user data elsewhere and manually import it after the new pods have started.
  - You can use the `export-ldif` command to export user data, or you can schedule a task via LDAP. The exported ldif file will be written to the pod filesystem.
  - You can use the `import-ldif` command to import user data, or you can schedule a task via LDAP. For the import to run, the file to be imported must exist on the pod filesystem.

Potential issues

Persistent volumes

In Kubernetes, persistent volumes created with our older containers have files owned by the root user. When the default non-privileged user attempts to use these existing volumes, there might be file permission errors.

To avoid this, you can either:

- Create a fresh deployment that doesn't use the old volumes.
- Continue to run the containers as root.

Additionally, the containers using persistent volume claims need to set the securityContext `fsGroup` to a value allowing the container can write to the PVCs. An example of setting this value in the statefulSet workload needs to include the following fsGroup setting.

This example uses the same default groupId set by the image. The Ping Identity Helm Charts already provide this setting by default for the containers.

```
spec:
  template:
    spec:
      securityContext:
        fsGroup:9999
```

Default ports

In our older images, certain default ports (`LDAP_PORT` , `LDAPS_PORT` , `HTTPS_PORT` , and `JMX_PORT`) were set to privileged values (`389` , `636` , `443` , and `689` , respectively). The newer images don't use these values because they run as a non-privileged user. The updated default ports are `1389` , `1636` , `1443` , and `1689` .

If you need, you can maintain the old values by setting the corresponding environment variables and running the container as root.

For our PingDirectory images, port changes aren't allowed on restart. If you're using a volume from an older image you may encounter an error due to changing port values.

You must either:

- Create a fresh deployment for PingDirectory with the new images and import your data from the old deployment.
- Set the environment variables to match the original privileged values and continue to run the container as root.

Running as root with the unprivileged-by-default images

To run as root as mentioned in the two previous examples, you must use your container orchestrator:

- For pure Docker, the `-u` flag allows specifying the user the container should use.
- For Docker Compose, you can define a `user:` .
- In Kubernetes, you can set up a security context for the container to specify the user. To run as root, a user and group ID of `0:0` should be used.

Assigning a Provisioner Node ID for PingFederate Pods

The PingFederate provisioner node id is set in the run.properties file used to configure the server. This value is used to set up failover for provisioning.

When using failover, each provisioning server must be given a unique index. By default with no server profile, the `PF_PROVISIONER_NODE_ID` environment variable is used to set the node id, with a default value of 1.

If it is necessary to set node ids for PingFederate servers, a StatefulSet can be used to provide consistent hostnames for individual Pods. The node id can then be parsed from these hostnames.

Warning

The use of a StatefulSet instead of a Deployment for PingFederate has some consequences. In particular, updates to the StatefulSet will be done as a rolling update, which can increase the time needed for an update.

In the Pod spec for the StatefulSet:

```
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
```

Then, a hook script can be used to parse the pod ordinal, which can then be set as the node id. Note that this will overwrite the default `PF_PROVISIONER_NODE_ID` value.

In your server profile, create a `02-get-remote-server-profiles.sh.post` script to update the environment variable:

```
#!/usr/bin/env sh
. "${HOOKS_DIR}/pingcommon.lib.sh"

# Parse the pod ordinal
PF_PROVISIONER_NODE_ID=${POD_NAME##*-}

# Add one to the ordinal so that node id starts at 1 instead of 0
PF_PROVISIONER_NODE_ID=$((PF_PROVISIONER_NODE_ID+1))

# Save the node id to the environment used by the hook scripts
export_container_env PF_PROVISIONER_NODE_ID
```

Ensure your server profile uses this environment variable if you are providing a custom `instance/bin/run.properties.subst` file:

```
provisioner.node.id=${PF_PROVISIONER_NODE_ID}
```

Adding a Message of the Day (MOTD)

You can create a message of the day (MOTD) JSON file to be provide an MOTD file to our product containers when they start.

Before you begin

You must:

- Complete the Get Started example to set up your DevOps environment and run a test deployment of the products.

Using a MOTD file

To employ a MOTD file:

1. Edit the existing `motd.json` file:
    1. Edit the motd/motd.json file located in your local `pingidentity-devops-getting-started/motd` folder.
2. Create a `motd.json` file in the location of your server profile:
    1. Create a `motd.json` file in the root of the server profile directory being referenced.

   This `motd.json` file will be appended to the `/etc/motd` file used by the provided image.

Testing the MOTD file

Test the new messages in the `motd.json` file using the `test-motd.sh` script. The script supplies the `JQ_EXPR` value used to pass the message data to the container.

1. To test the `motd.json` file locally for our example use cases, from the `pingidentity-devops-getting-started/motd` directory, enter:

   ```
   ./test-motd.sh local
   ```

2. To test the `motd.json` file you created in your server profile directory:
    1. Copy the `test-motd.sh` script located in the `pingidentity-devops-getting-started/motd` directory to your server profile directory.
    2. Enter:

       ```
       ./test-motd.sh local
       ```

3. To test the `motd.json` with a server profile located in a Github repository:
    1. Ensure the `test-motd.sh` script is located in the local, cloned repository.
    2. From the local, cloned repository, enter:

```
./test-motd.sh github
```

Example motd.json

The example below shows the messages that are displayed for all product images.

For this example, the messages are only shown from the `validFrom` to `validTo` dates:

```
{
    "devops" : [
        {
            "validFrom": 20220701,
            "validTo": 20220730,
            "subject": "General Message 1",
            "message": [
                "This is line # 1",
                "",
                "This is line # 3"
            ]
        },
        {
            "validFrom": 20220801,
            "validTo": 20220830,
            "subject": "General Message 2",
            "message": ["Message goes here"]
        }
    ],
    "pingfederate" : [
        {
            "validFrom": 20220701,
            "validTo": 20220830,
            "subject": "PingFederate Message 1",
            "message": ["Message goes here"]
        }
    ]
}
```

Build a Docker Product Image Locally

This page describes the process to build a Docker image of our products with the build tools found in our Docker Builds repository and a local copy of a product .zip archive.

> **Video demonstration**
>
> For a video demonstration of this process, visit this link.

Docker Builds

Cloning a build repository

Open a terminal and clone the `pingidentity-docker-builds` repo:

```
git clone https://github.com/pingidentity/pingidentity-docker-builds.git
```

Download a product .zip archive

1. Go to Product Downloads and download the product to be used to build a Docker image.

   > **Zip file versus installer**
   >
   > Ensure you download the product distribution .zip archive and not the Windows installer.

2. When the download has finished, rename the file to `product.zip`:

   ```
   mv pingfederate-11.0.3.zip product.zip
   ```

3. Move `product.zip` to the Build Directory.

   In the `pingidentity-docker-builds` repository directory for each product, move the `product.zip` file to the `<product>/tmp` directory, where /<product> is the name of one of our available products. For example:

```
mv ~/Downloads/product.zip \
   ~/pingidentity/devops/pingidentity-docker-builds/pingfederate/tmp
```

Build the Docker image

Before building the image, display the `versions.json` file in the product directory. You must specify a valid version for the build script. Since the product .zip archive is being provided, it does not matter which version you select as long as it is valid. For example, you can see that `11.0.3` is a valid product version for PingFederate.

```
{
    "latest": "11.0.3",
    "versions": [
        {
            "version": "11.0.3",
            "preferredShim": "alpine:3.15.4",
            "shims": [
                {
                    "shim": "alpine:3.15.4",
                    "preferredJVM": "al11",
                    "jvms": [
                        {
                            "jvm": "al11",
                            "build": true,
                            "deploy": true,
                            "registries": [
                                "DockerHub",
                                "Artifactory"
                            ]
                        }
                    ]
                },
```

1. Go to the base of the pingidentity-docker-builds repo. For example:

```
cd ~/pingidentity/devops/pingidentity-docker-builds
```

2. Run the serial_build.sh script with the appropriate options. For example:

```
./ci_scripts/serial_build.sh \
    -p pingfederate \
    -v 11.0.3 \
    -s alpine:3.15.4 \
    -j al11
```

When the build is completed, the product and base images are displayed. For example:

```
REPOSITORY                    TAG                                                        IMAGE ID
CREATED                 SIZE
pingidentity/pingfederate     11.0.3-fsoverride-alpine_3.15.4-al11-master-f1ba-x86_64    404a2b14df0c    7
seconds ago          759MB
pingidentity/pingbase         master-f1ba-x86_64                                         eb7648692b55    About
a minute ago    0B
pingidentity/pingjvm          al11-alpine_3.15.4-master-f1ba-x86_64                      af0e87d8fafd    About
a minute ago    108MB
pingidentity/pingcommon       master-f1ba-x86_64                                         2e82b239e9bb    About
a minute ago    997kB
pingidentity/pingdatacommon   master-f1ba-x86_64                                         13f35b12a918    About
a minute ago    1.11MB
```

Our Docker images are built using common foundational layers required by the product layer such as the Java virtual machine (JVM), pingcommon, and pingdatacommon.

As it is unlikely you will have the foundational layers on your local system, build the first time using the serial_build.sh script. This script will create the foundational images, and if you want to use the same foundational layers for other builds, you need only run the build_product.sh script to build the product layer.

You must specify the appropriate options when you run serial_build.sh. For PingFederate, the options might look like this:

- -p (Product): pingfederate
- -v (Version): 11.0.3
  - NOTE: this is the version retrieved from the versions.json file
- -s (Shim): alpine
- -j (Java): al11

Run from the repository root

It is important to build from the base of the repository as shown in the example.

Re-tagging the local image

To change the tag of the created image and push it to your own Docker registry, use the `docker tag` command:

```
docker tag [image id] \
  [Docker Registry]/[Organization]/[Image Name]:[tag]
```

For example:

```
docker tag 404a2b14df0c \
  gcp.io/pingidentity/pingfederate:localbuild
# Display new tag
docker image ls
# Output snippet
pingidentity/pingfederate            11.0.3-fsoverride-alpine_3.15.4-al11-master-f1ba-x86_64   404a2b14df0c   4
minutes ago   759MB
gcp.io/pingidentity/pingfederate     localbuild
```

Forwarding Logs

Forwarding PingFederate and PingAccess logs to Splunk

This page provides an example of how PingFederate and PingAccess logs can be shipped to Splunk. The principle of using a container for a single purpose is followed, and a sidecar for log collection and forwarding is placed in the appropriate Ping product pods.

Video demonstration

For a video demonstration of this process, visit this link.

Splunk Demonstration Only

This guide is for demonstration purposes only, but the principles will apply to a production implementation. In addition, the process for other logging solutions will be similar.

Components Used

1. Ping DevOps Helm Chart
2. Ping server-profiles repository
3. Splunk Deployment
4. Splunk Universal Forwarder Docker image

Prerequisites

- Access to a Kubernetes cluster. For this guide, a local Kubernetes cluster with the nginx-ingress controller and the MetalLB load balancer was used. You might have to adjust how you access the product interface URLs, depending on your environment.
- Helm pingidentity/ping-devops chart >= 0.9.11

Overall Process

1. Configure the cluster environment
2. Deploy Splunk Enterprise
3. Configure Splunk and generate an HTTP Event Collector (HEC) token
4. Create a configmap with the token for use by the Splunk Universal Forwarder (UF) sidecar
5. Use Helm to deploy PingFederate and PingAccess with the sidecar attached to the engine pods
6. Confirm logs and activity are visible in Splunk

Cluster preparation

```
# Create the namespace
kubectl create ns splunk

# Set the kubectl context to the namespace
kubectl config set-context --current --namespace=splunk

# Confirm
kubectl config view --minify | grep namespace:
```

Splunk Server deployment

Deploy the Splunk application:

```
# Clone the `pingidentity-devops-getting-started` repository to a local directory
git clone \
  https://github.com/pingidentity/pingidentity-devops-getting-started.git

cd pingidentity-devops-getting-started

# Deploy Splunk
# The splunk.yaml file assumes a load balancer is available in the cluster
kubectl apply -f 20-kubernetes/splunk/splunk.yaml

# Determine IP address assigned
# 8000 is HTTP; 8088 is HTTPS
kubectl get svc

NAME     TYPE           CLUSTER-IP     EXTERNAL-IP       PORT(S)
splunk   LoadBalancer   10.105.171.4   192.168.163.172   8000:30416/TCP,8088:30364/TCP,9997:31770/TCP,9990:32292/UDP

# Create corresponding entry in /etc/hosts
# If your cluster has publicly-accessible IPs and DNS support, this step is not necessary
# You would use the DNS entry assigned to the service.
192.168.163.172 splunk.pingdemo.example
```

Configure Splunk

In this section, you will prepare Splunk for the logs from the products.

Data Persistence

In this demo, there is no data persistence for Splunk. If you restart the Splunk pod, you will lose everything that is configured in the following steps.

- Navigate to the UI in a browser at `http://splunk.pingdemo.example:8000/en-US/account`.
- Login with the credentials admin / 2FederateM0re!

Create an index

- Navigate to Settings > Indexes and click the New Index button at the upper-right.
- Provide pinglogs as the Index Name.
- Accept all defaults and click Save.

Create an HTTP Event Collector (HEC)

- Navigate to Settings > Data inputs and click Add New in the HTTP Event Collector row.
- A wizard is launched and you are taken to the Select Source step. Type `pinglogs` as the name and click the Next button in the upper panel.
- In the Input Settings step, add the `pinglogs` index to the Selected item(s) box by clicking on it in the Available item(s) list, then click the Review button in the upper panel.
- Confirm your entries and click the Submit button in the upper panel.
- A token is generated. Save this token to a scratch file or buffer for use in configuring Splunk in a moment.

Add the Ping product applications to Splunk

- Navigate to Apps > Find More Apps. The Apps link is at the upper-left of the UI.
- Filter the list of applications using `Ping`. Add the PingFederate and PingAccess Apps for Splunk.

Splunk Account

You will need valid credentials from Splunk to install the applications. You can use a free trial if necessary.

PingDirectory App

While not shown in this example, Ping also provides a Splunk App for PingDirectory. You would need to attach the Splunk UF sidecar to your PingDirectory pods as done here for PingFederate and PingAccess.

Create a configmap

Use the HEC token generated earlier to update the file `20-kubernetes/splunk/splunk-config-init.yaml` (search for #CHANGEME).

Apply the file:

```
kubectl apply -f 20-kubernetes/splunk/splunk-config-init.yaml
```

Deploy the Ping stack with Splunk UF as a sidecar

```
# Create the DevOps secret for temporary Ping license
pingctl k8s generate devops-secret | kubectl apply -f -

# Install Ping and Ingress
helm upgrade --install myping pingidentity/ping-devops -f 20-kubernetes/splunk/values.yaml -f 30-helm/ingress-
demo.yaml
```

This command deploys PingDirectory, PingFederate, and PingAccess with:

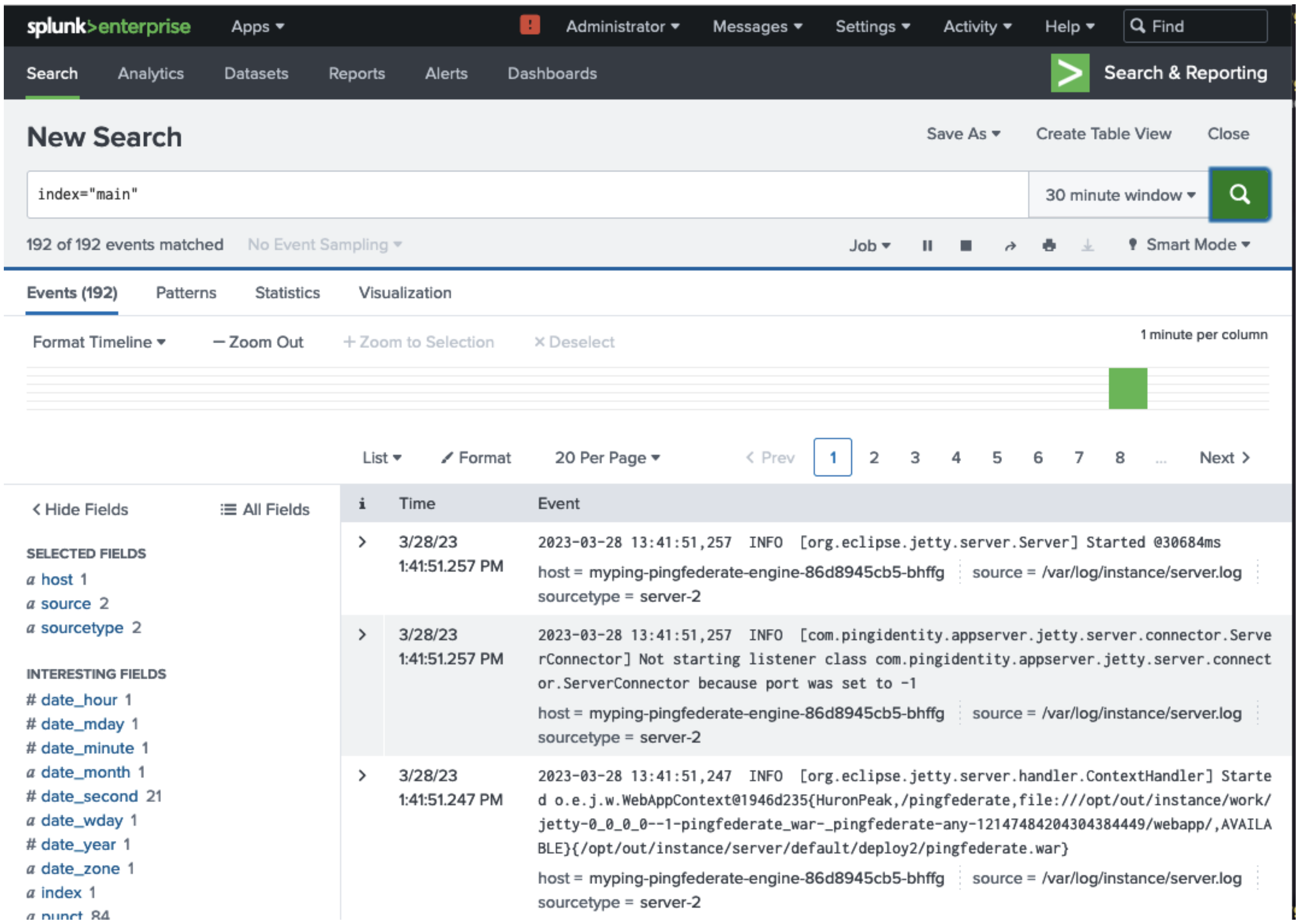
- Baseline Server Profiles
- Splunk Logs Profile layer for the PingAccess and PingFederate engine pods
- Splunk UF sidecar for the PingAccess and PingFederate engine pods

Server Profile Repository

The `values.yaml` file in this guide uses a directory in the Ping server profiles repository. That profile folder has log4j configuration files that format the logs from the PingAccess and PingFederate product containers for use in Splunk. These files are also in the backing repository for this portal under the `20-kubernetes/splunk/pingaccess` and `20-kubernetes/splunk/pingfederate` directories, respectively.

Confirm in Splunk

Eventually you should see product logs in Splunk by searching: `index="main"`. The first logs will appear when the PingFederate engine has launched fully.



```
index="main"
```

192 of 192 events matched

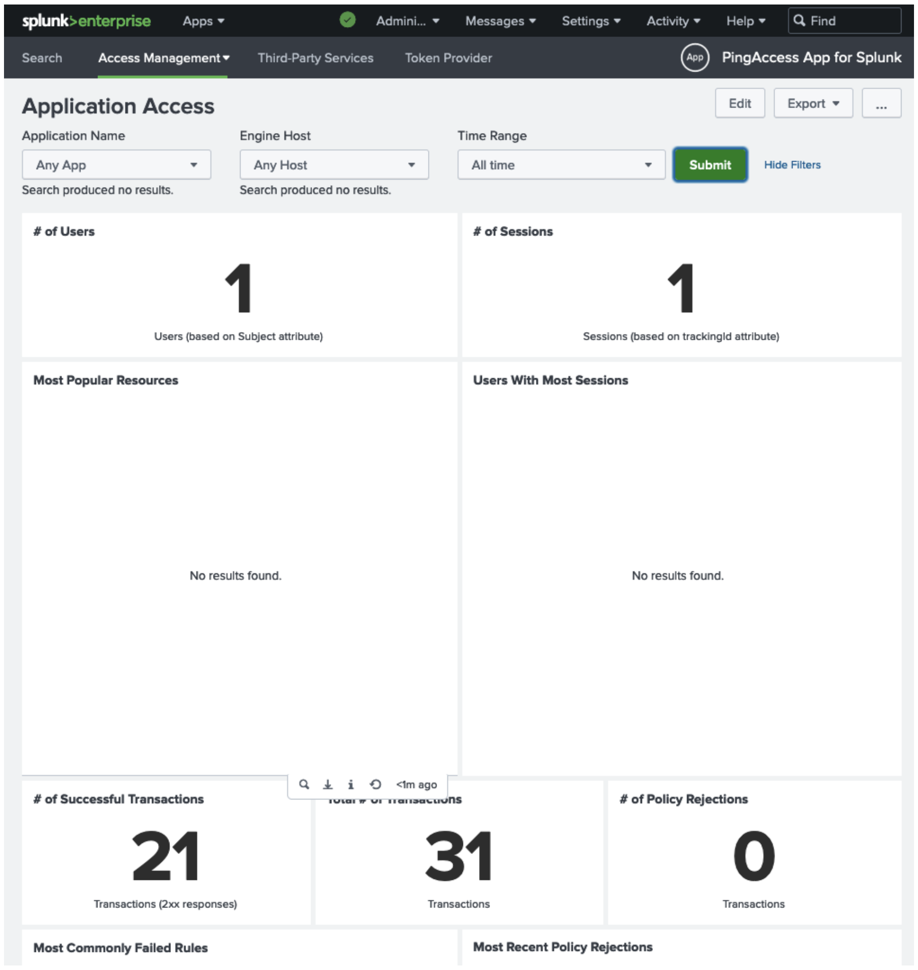
SELECTED FIELDS
a host 1
a source 2
a sourcetype 2

INTERESTING FIELDS
# date_hour 1
# date_mday 1
# date_minute 1
a date_month 1
# date_second 21
a date_wday 1
# date_year 1
a date_zone 1
a index 1
a punct 84

Time	Event
3/28/23 1:41:51.257 PM	2023-03-28 13:41:51,257 INFO [org.eclipse.jetty.server.Server] Started @30684ms host = myping-pingfederate-engine-86d8945cb5-bhffg source = /var/log/instance/server.log sourcetype = server-2
3/28/23 1:41:51.257 PM	2023-03-28 13:41:51,257 INFO [com.pingidentity.appserver.jetty.server.connector.ServerConnector] Not starting listener class com.pingidentity.appserver.jetty.server.connector.ServerConnector because port was set to -1 host = myping-pingfederate-engine-86d8945cb5-bhffg source = /var/log/instance/server.log sourcetype = server-2

3/28/23 1:41:51.247 PM	2023-03-28 13:41:51,247 INFO [org.eclipse.jetty.server.handler.ContextHandler] Started o.e.j.w.WebAppContext@1946d235{HuronPeak,/pingfederate,file:///opt/out/instance/work/jetty-0_0_0_0--1-pingfederate_war-_pingfederate-any-12147484204304384449/webapp/,AVAILABLE}{/opt/out/instance/server/default/deploy2/pingfederate.war}
	host = myping-pingfederate-engine-86d8945cb5-bhffg source = /var/log/instance/server.log sourcetype = server-2

To see the Splunk App dashboards in operation, generate some traffic in the products to populate them. For example, for PingAccess, you can access https://myping-pingaccess-engine.pingdemo.example/anything, which will be rejected, but you will see the activity populated. Also, you can login to the administrative console at http://myping-pingaccess-admin.pingdemo.example with the credentials administrator / 2FederateM0re.



Application Access

Application Name: Any App
Search produced no results.

Engine Host: Any Host
Search produced no results.

Time Range: All time

# of Users
1
Users (based on Subject attribute)

# of Sessions
1
Sessions (based on trackingId attribute)

Most Popular Resources
No results found.

Users With Most Sessions
No results found.

<1m ago

# of Successful Transactions
21
Transactions (2xx responses)

Total # of Transactions
31
Transactions

# of Policy Rejections
0
Transactions

Most Commonly Failed Rules

Most Recent Policy Rejections

References

This list includes some of the references used in the creation of this document:

- PingFederate Logs formatting for Splunk
- PingFederate Dashboard reference
- Splunk Universal Forwarder (SUF) in Kubernetes
- Splunk configuration for inputs via HTTP

Re-encrypting backend data for a set of PingDirectory pods

PingDirectory uses encryption settings definitions to manage how data is encrypted in the database. When setting a new preferred encryption settings definition, the new definition will be used for all subsequent data encryption, but existing data remains encrypted with an older key.

In many cases this is acceptable, and no additional work needs to be done. However in cases such as when an existing key might have been compromised, you will want to completely transition to using the new definition. This page describes the steps necessary to do this.

For PingDirectory documentation on this scenario, see https://docs.pingidentity.com/r/en-us/pingdirectory-93/pd_sec_re-encrypt_database

This page will describe how to follow the steps listed on the above page in a Kubernetes environment with several PingDirectory pods.

Example starting Helm values using the ping-devops Helm chart

For demonstration, we will be using these Helm values with the ping-devops Helm chart:

```
pingdirectory:
  enabled: true
  container:
    replicaCount: 3
  envs:
    SERVER_PROFILE_URL: https://path/to/profile.git
    SERVER_PROFILE_PATH: my-profiles/pingdirectory
    ENCRYPTION_PASSWORD_FILE: /opt/staging/.sec/encryption-passphrase1.txt
```

Updating the encryption settings database with the new preferred definition

The first step is to update the encryption settings database with your new preferred encryptions settings definition. For details on doing this manually, see the PingDirectory documentation.

If you are using the `ENCRYPTION_PASSWORD_FILE` environment variable to control encryption for your pods, you can simply point that variable to a different file with a new passphrase and restart the pods. After the restart, the pods will use the new definition based on the `ENCRYPTION_PASSWORD_FILE` value. For example, with the environment variable updated:

```
pingdirectory:
  enabled: true
  container:
    replicaCount: 3
  envs:
    SERVER_PROFILE_URL: https://path/to/profile.git
    SERVER_PROFILE_PATH: my-profiles/pingdirectory
    ENCRYPTION_PASSWORD_FILE: /opt/staging/.sec/encryption-passphrase2.txt
```

Whatever method you use to update the encryption settings database, ensure that each pod has the new definition in the encryption settings database before continuing. Use the `encryption-settings` command-line tool to view the contents of the encryption settings database.

Disabling replication and deleting the replication database

Now replication must be disabled between the pods before the data is exported and re-imported, and the replication database must be deleted to ensure there are no remaining entries encrypted with the old definition.

Exec into one of the pods and use the `dsreplication disable` command to disable replication between each of the servers. Ensure that each server is in its own single-server topology using the `dsreplication status` command.

Run `rm -r /opt/out/instance/changelogDb/` on each of the pods individually, to remove any lingering entries from the replication database that may have been encrypted with the old definition.

Scaling down to one pod and re-importing the data

The data must now be exported and re-imported with the server offline. To do this, we will scale down to a single pod (however we *do not* need to delete the persistent volumes of the other pods). We will also force the final pod to export and re-import its data so that it is encrypted with the new preferred definition. The `PD_FORCE_DATA_REIMPORT` environment variable can be used to force an export and re-import of the data before the server starts up.

Note that the `PD_FORCE_DATA_REIMPORT` was added in the `2307` docker image release for PingDirectory. Prior to this a custom hook script would be needed to force the data export and re-import.

```
pingdirectory:
  enabled: true
  container:
    replicaCount: 1
  envs:
    SERVER_PROFILE_URL: https://path/to/profile.git
    SERVER_PROFILE_PATH: my-profiles/pingdirectory
    ENCRYPTION_PASSWORD_FILE: /opt/staging/.sec/encryption-passphrase2.txt
    PD_FORCE_DATA_REIMPORT: "true"
```

Scaling back up

Now we can scale back up to the full number of pods, and stop forcing the data export and re-import. When the removed pods restart, they will rejoin the topology and will initialize their data from the seed pod (pod 0), which will be encrypted with the new preferred definition.

```
pingdirectory:
  enabled: true
  container:
    replicaCount: 3
  envs:
    SERVER_PROFILE_URL: https://path/to/profile.git
    SERVER_PROFILE_PATH: my-profiles/pingdirectory
    ENCRYPTION_PASSWORD_FILE: /opt/staging/.sec/encryption-passphrase2.txt
```

The backend data will now be encrypted with the new preferred definition.

Note that in some cases an encryption settings definition may be used for more than encrypting backend data. For example, files can be encrypted using encryption settings definitions. By default some pin files in the server root will be encrypted if encryption was enabled during the first setup of the server, such as `config/keystore.pin`. If you need to manage how files are encrypted with encryption settings definitions, run `encrypt-file --help` for more information.

S3 Archive of a PingDirectory Backup

Demonstration Only

This guide is for demonstration purposes only and is not intended for production use and is just one of many ways of archiving files to S3. Other storage options might be available, depending on your provider.

Before you begin

Prior to attempting these steps, you must:

- Complete the Getting Started steps to set up your DevOps environment and run a test deployment of the products
- Have some means of authenticating the sidecar container to S3. This authentication can use an IAM role or another method and is left for the user to implement.

High-level backup steps

- Configure some means of creating a backup of PingDirectory. For this guide, an extension of the PingDirectory Backup and Sidecar is used.
- After the backup is made, use an archive script to upload the backup to S3.
- (Optional) Clean up the image filesystem of backups.

File exploration

In the `30-helm/s3-sidecar` directory of this repository, you will find the following files:

Dockerfile

This file extends the PingToolkit image, adding the AWS CLI and a few other utilities for demonstration purposes. You can use any platform that meets your requirements.

```
## Dockerfile for AWS CLI
## For demonstration purposes only
## Not intended for production use
FROM pingidentity/pingtoolkit:latest

USER root
# Install AWS CLI
RUN apk add --no-cache \
    aws-cli \
    bash \
    curl \
    less \
    groff \
    shadow \
    sudo \
    unzip

USER 9031:0
```

To build a multi-architecture image, you can use the following command. In order to create an image for a different architecture, you will need to have the `buildx` plugin installed and configured, and the image will have to be pushed at build time:

```
docker buildx build --platform linux/arm64,linux/amd64 -t <registry>/<image>:<tag> --push .
```

Image Availability

Ensure that the registry is accessible to the Kubernetes cluster.

pd-archive-backup-to-s3.yaml

This file will not be repeated in full here. The top section creates ConfigMaps that define four sample scripts:

- archive.sh - This demonstration script is called by the *backup.sh* script to archive the backup to S3. The bucket name and path will need to be updated to match your environment.
- fetch.sh - This demonstration script is called by the *restore.sh* script to fetch backup files from S3. The bucket name and path will need to be updated to match your environment.
- backup.sh - This demonstration script is called by the sidecar container to create a backup of PingDirectory. It then calls the *archive.sh* script (with no error handling or testing).
- restore.sh - This demonstration script would be executed either by a job or in the sidecar container to restore a backup of PingDirectory.

These scripts are placed into the sidecar image under the */opt/in* directory.

Lines 222 and 223 will need modification to point to the registry and tag for the image that has the AWS cli installed as in the `buildx` command above.

Backup Operation

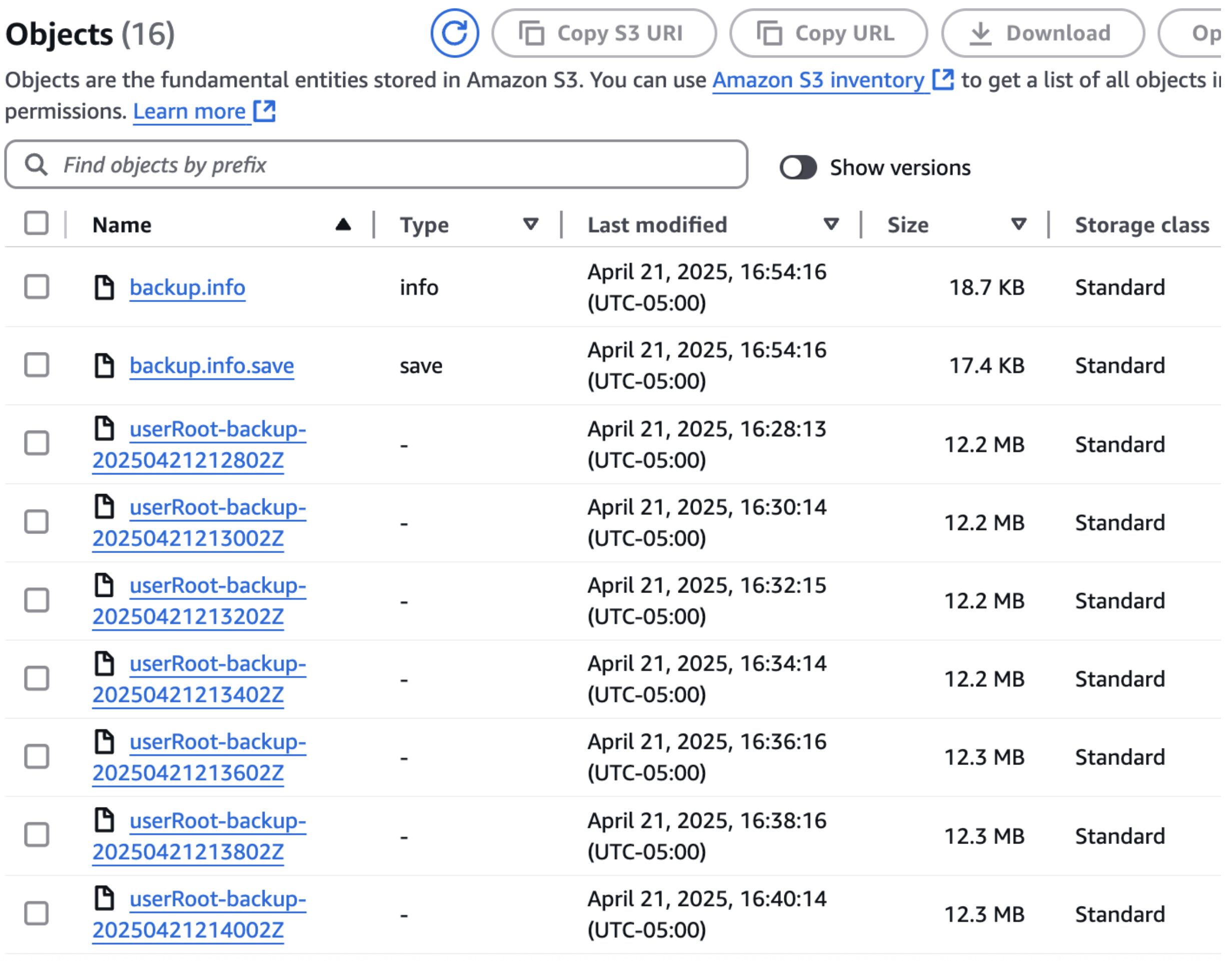
If this demonstration is implemented, the process is straightforward. As per the crontab entry, every 6 hours:

- A backup of the PingDirectory data is created
- The backup is archived to S3

PingDirectory handles the removal of old backups based on the parameters set in the backup script.

If you are observing the cluster at the time the backup takes place, an additional pod launches to execute the cronjob. This pod terminates after the backup is complete.

Over time, the S3 bucket will appear similar to the following screenshot. To create this image, the backup and archive process ran every 2 minutes:



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects i permissions. Learn more

Find objects by prefix

Show versions

	Name	Type	Last modified	Size	Storage class
☐	backup.info	info	April 21, 2025, 16:54:16 (UTC-05:00)	18.7 KB	Standard
☐	backup.info.save	save	April 21, 2025, 16:54:16 (UTC-05:00)	17.4 KB	Standard
☐	userRoot-backup-20250421212802Z	-	April 21, 2025, 16:28:13 (UTC-05:00)	12.2 MB	Standard
☐	userRoot-backup-20250421213002Z	-	April 21, 2025, 16:30:14 (UTC-05:00)	12.2 MB	Standard
☐	userRoot-backup-20250421213202Z	-	April 21, 2025, 16:32:15 (UTC-05:00)	12.2 MB	Standard
☐	userRoot-backup-20250421213402Z	-	April 21, 2025, 16:34:14 (UTC-05:00)	12.2 MB	Standard
☐	userRoot-backup-20250421213602Z	-	April 21, 2025, 16:36:16 (UTC-05:00)	12.3 MB	Standard
☐	userRoot-backup-20250421213802Z	-	April 21, 2025, 16:38:16 (UTC-05:00)	12.3 MB	Standard
☐	userRoot-backup-20250421214002Z	-	April 21, 2025, 16:40:14 (UTC-05:00)	12.3 MB	Standard

Restore Operation

In the event that a restore operation is needed, the restore.sh script can be used. This script will:

- Download the backup from S3
- Restore the backup to the PingDirectory data directory

A sample run of the script is shown below:

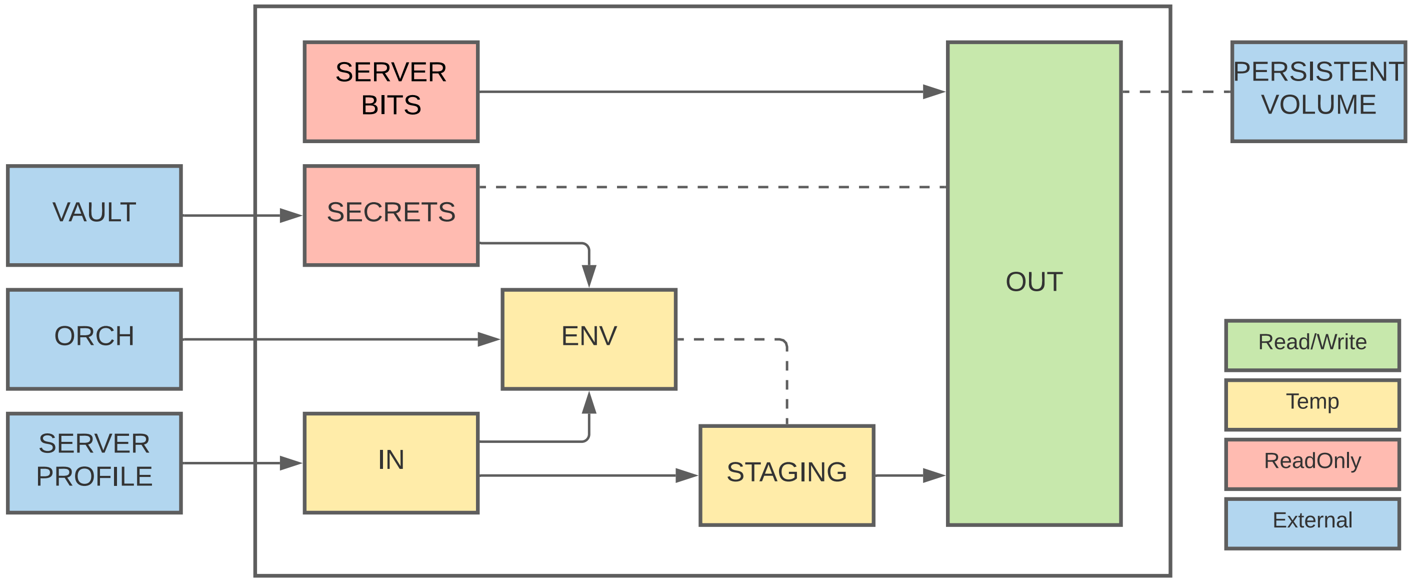
```
PingToolkit:demo-pingdirectory-0:/opt
> /opt/in/restore.sh <admin-password>
download: s3://<bucket-name>/<folder>/userRoot/backup.info to ../tmp/restore/userRoot/backup.info
download: s3://<bucket-name>/<folder>/userRoot/backup.info.save to ../tmp/restore/userRoot/backup.info.save
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421213402Z to ../tmp/restore/userRoot/userRoot-
backup-20250421213402Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421213202Z to ../tmp/restore/userRoot/userRoot-
backup-20250421213202Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421213002Z to ../tmp/restore/userRoot/userRoot-
backup-20250421213002Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421213602Z to ../tmp/restore/userRoot/userRoot-
backup-20250421213602Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421213802Z to ../tmp/restore/userRoot/userRoot-
backup-20250421213802Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421212802Z to ../tmp/restore/userRoot/userRoot-
backup-20250421212802Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421214202Z to ../tmp/restore/userRoot/userRoot-
backup-20250421214202Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421214802Z to ../tmp/restore/userRoot/userRoot-
backup-20250421214802Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421214402Z to ../tmp/restore/userRoot/userRoot-
backup-20250421214402Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421215002Z to ../tmp/restore/userRoot/userRoot-
backup-20250421215002Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421214602Z to ../tmp/restore/userRoot/userRoot-
backup-20250421214602Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421215202Z to ../tmp/restore/userRoot/userRoot-
backup-20250421215202Z
download: s3://<bucket-name>/<folder>/userRoot/userRoot-backup-20250421214002Z to ../tmp/restore/userRoot/userRoot-
backup-20250421214002Z
Replication is not enabled
userRoot
Restoring to the latest backups under /tmp/restore
Restore order of backups: /tmp/restore/userRoot

----- Doing a restore from /tmp/restore/userRoot -----
Restore task 2025042121535304 scheduled to start immediately

NOTE:  This tool is running as a task.  Killing or interrupting this tool will not have an impact on the task
If you wish to cancel the running task, that may be accomplished using the command:  manage-tasks --no-prompt --
hostname localhost --port 1636 --bindDN "cn=administrator" --bindPassword "********" --cancel "2025042121535304"

[21/Apr/2025:21:53:53 +0000] severity="SEVERE_WARNING" msgCount=0 msgID=1880227932 message="Administrative alert
type=backend-disabled id=fc5694f2-7b52-4cf9-8214-89edb41708bb
class=com.unboundid.directory.server.core.BackendConfigManager msg='Backend userRoot is disabled'"
[21/Apr/2025:21:53:53 +0000] severity="NOTICE" msgCount=1 msgID=1880555611 message="Administrative alert type=config-
change id=71fc9c87-4030-427a-ab9c-cf631157d210 class=com.unboundid.directory.server.admin.util.ConfigAuditLog msg='A
configuration change has been made in the Directory Server:  [21/Apr/2025:21:53:53.124 +0000] conn=-4 op=7407
dn='cn=Internal Client,cn=Internal,cn=Root DNs,cn=config' authtype=[Internal] from=internal to=internal
command='dsconfig set-backend-prop --backend-name userRoot --set enabled:false''"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=2 msgID=8847445 message="Restored: .environment-open from
backup with id '20250421215202Z' (size 76)"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=3 msgID=8847445 message="Restored: 00000000.jdb from backup
with id '20250421215202Z' (size 12997816)"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=4 msgID=8847445 message="Restored: esTokenizer.ping from
backup with id '20250421215202Z' (size 39)"
[21/Apr/2025:21:53:54 +0000] severity="SEVERE_WARNING" msgCount=5 msgID=1880227932 message="Administrative alert
type=je-environment-not-closed-cleanly id=8f35bade-1f67-42ca-a506-79ee377a0ace
class=com.unboundid.directory.server.backends.jeb.RootContainer msg='The server has detected that the Berkeley DB JE
environment located in directory '/opt/out/instance/db/userRoot' may not have been closed cleanly the last time it
```

```
was opened (or that the backend has just been restored from a backup taken with the server online).  The database
environment may need to replay changes from the end of the transaction log to guarantee the integrity of the data,
and in some cases this may take a significant amount of time to complete'"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=6 msgID=8847402 message="The database backend userRoot using
Berkeley DB Java Edition 7.5.12 and containing 20008 entries has started"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=7 msgID=1879507338 message="Starting group processing for
backend userRoot"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=8 msgID=1879507339 message="Completed group processing for
backend userRoot"
[21/Apr/2025:21:53:54 +0000] severity="INFORMATION" msgCount=9 msgID=1891631108 message="Starting access control
processing for backend userRoot"
[21/Apr/2025:21:53:54 +0000] severity="INFORMATION" msgCount=10 msgID=12582962 message="Added 2 Access Control
Instruction (ACI) attribute types found in context 'dc=example,dc=com' to the access control evaluation engine"
[21/Apr/2025:21:53:54 +0000] severity="NOTICE" msgCount=11 msgID=1880555611 message="Administrative alert
type=config-change id=550d224f-f8f9-45f5-ace3-4933ca74a76d
class=com.unboundid.directory.server.admin.util.ConfigAuditLog msg='A configuration change has been made in the
Directory Server:  [21/Apr/2025:21:53:54.939 +0000] conn=-4 op=7415 dn='cn=Internal Client,cn=Internal,cn=Root
DNs,cn=config' authtype=[Internal] from=internal to=internal command='dsconfig set-backend-prop --backend-name
userRoot --set enabled:true''"
Restore task 2025042121535304 has been successfully completed
Restore complete
```

Components and Configuration

Container data flows and running state

The diagram below shows the topology of a container with flows of data into the container and how it transitions to the eventual running state.



Data Class	Default Location	Use	Description
VAULT		ext	Secret information from external Vault (i.e. HashiCorp Vault). Items like passwords, certificates, keys, etc.
ORCH		ext	Environment variables from secrets, configmaps and/or env/envfile resources from orchestration (i.e. docker, k8s)
SERVER PROFILE		ext	Product server profile from either an external repository (i.e. git) or external volume (i.e. aws s3).
SERVER BITS	/opt/server	ro	Uncompressed copy of the product software. Provided by image.

Data Class	Default Location	Use	Description
SECRETS	/run/secrets	ro	Read Only secrets residing on non-persistent storage (i.e. /run/secrets).
IN	/opt/in	ro	Volume intended to receive all incoming server-profile information.
ENV	/opt/staging/.env	mem	Environment variable settings used by hooks and product to configure container.
STAGING	/opt/staging	tmp	Temporary space used to prepare configuration and store variable settings before being moved to OUT
OUT	opt/opt	rw	Combo of product bits/ configuration resulting in running container configuration.
PERSISTENT VOLUME		rw	Persistent location of product bits/configuration in external storage (i.e. AWS EBS)

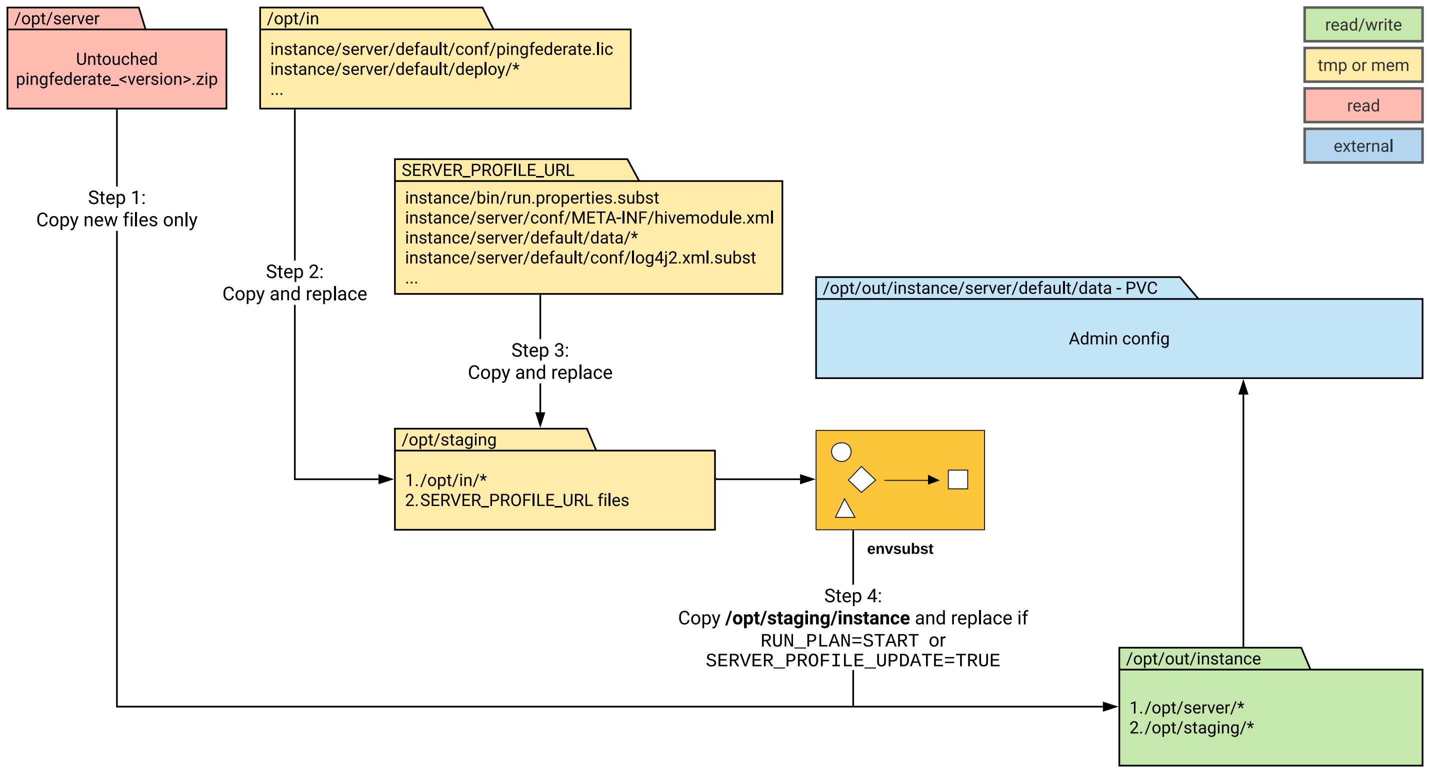
Because of these many factors affecting how an image is deployed, the configuration options for use of the elements in the previous table can vary greatly, depending on factors such as:

- Deployment Environment - Kubernetes, Cloud Vendor, Local Docker
- CI/CD Tools - Kubectl, Helm, Kustomize, Terraform
- Source Maintenance - Git, Cloud Vendor Volumes
- Customer Environment - Development, Test, QA, Stage, Prod
- Security - Test/QA/Production Data, Secrets, Certificates, Secret Management Tools

Examples

File flowchart example

The following diagram shows how files can enter and flow through the container:


/opt/server
Untouched pingfederate_<version>.zip
/opt/in
instance/server/default/conf/pingfederate.lic
instance/server/default/deploy/*
...
SERVER_PROFILE_URL
instance/bin/run.properties.subst
instance/server/conf/META-INF/hivemodule.xml
instance/server/default/data/*
instance/server/default/conf/log4j2.xml.subst
...
read/write
tmp or mem
read
external
/opt/out/instance/server/default/data - PVC
Admin config
Step 1: Copy new files only
Step 2: Copy and replace
Step 3: Copy and replace
/opt/staging
1./opt/in/*
2.SERVER_PROFILE_URL files
envsubst
Step 4: Copy /opt/staging/instance and replace if RUN_PLAN=START or SERVER_PROFILE_UPDATE=TRUE
/opt/out/instance
1./opt/server/*
2./opt/staging/*

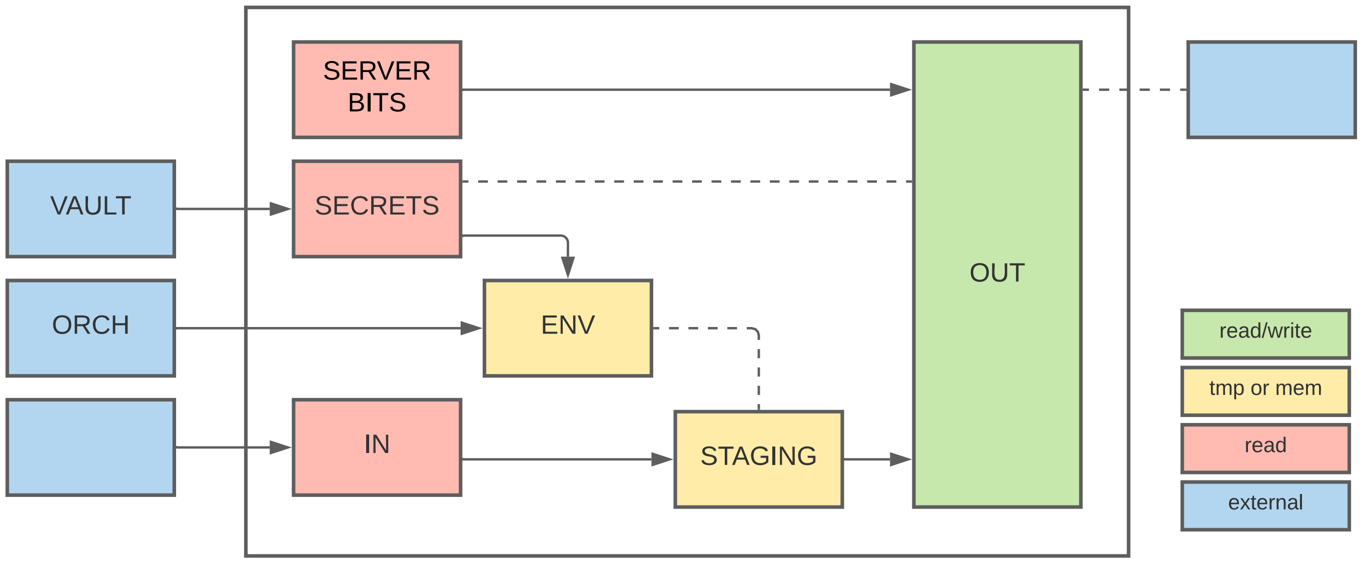
Note

There is a video that goes through the above image in more detail here.

Production Example

The following diagram shows an example in a high-level production scenario in an Amazon Web Services (AWS) EKS environment, where:

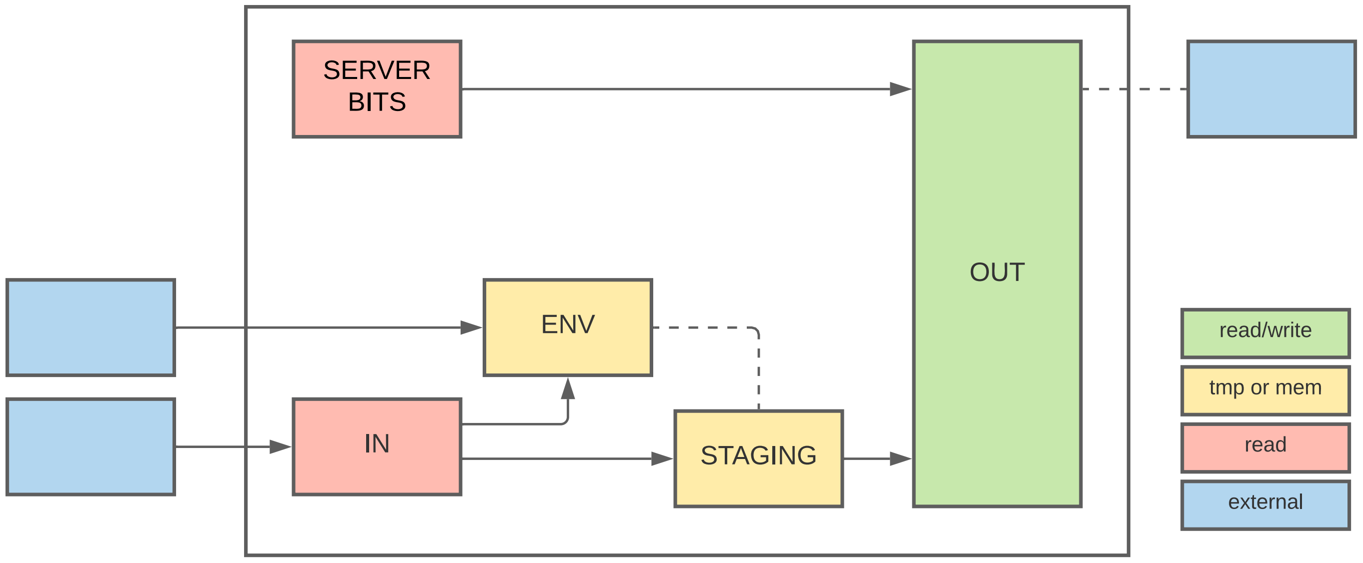
- HashiCorp Vault is used to provide secrets to the container.
- Helm is used to create k8s resources and deploy them.
- AWS EBS volumes is used to persist the state of the container.



Development Example

The following diagram shows an example in a high-level development scenario in an Azure AKS environment, where:

- No secrets management is used.
- Simple kubectl is used to deploy k8s resources.
- AWS EBS volumes is used to persist the state of the container.



Customizing the Containers

You can customize our product containers by:

- Customizing server profiles

  The server profiles supply configuration, data, and environment information to the product containers at startup. You can use our server profiles as-is or as a baseline for creating your own.

  You can find these profiles in Baseline server profiles in our pingidentity-server-profiles repository.

- Environment substitution

  You can deploy configurations in multiple environments with minimal changes by removing literal values and replacing them with environment variables.

- Using DevOps hooks

  Hooks are shell scripts used to automate operations during the lifecycle of a product container. These hook scripts are built into our images: some are common across all products, while others are product-specific. For more information about the repository that houses these scripts, visit the docker-builds repository overview.

- Using release tags

  We use sets of tags for each released build image. These tags identify whether the image is a specific stable release, the latest stable release, or current (potentially unstable) builds. You can find the release tag information in Docker images.

  You can try different tags in either the standalone startup scripts for the deployment examples or the YAML files for the orchestrated deployment examples.

- Adding a message of the day (MOTD)

  You can use a `motd.json` file to add message of the day information for inclusion in the images.

Server Profile Structures

Each of the Docker images use a server profile structure that is specific to each product. The structure (directory paths and data) of the server profile differs between products. Depending on how you Deploy Your Server Profile, it will be pulled or mounted into `/opt/in` on the container and used to stage your deployment.

The following locations are the server profile structures for each of our products with example usage. For help with an example of the basics, see the pingidentity-server-profiles/getting-started examples.

Note

In the getting-started profile examples, you should not use the `.sec` directory when providing passwords to your containers. These examples are only intended for demonstration purposes. Instead, set an environment variable with your secrets or orchestration later:
`PING_IDENTITY_PASSWORD="secret"`

PingFederate

See the example at getting-started/pingfederate.

Table of Paths and Location Descriptions

Path	Location description
instance	Directories and files that you want to be used at product runtime, in accordance with the directory layout of the product.
instance/server/default/data	An extracted configuration archive exported from PingFederate.
instance/bulk-config/data.json	A JSON export from the PingFed admin API `/bulk/export`.
instance/server/default/deploy/OAuthPlayground.war	Automatically deploy the OAuthPlayground web application.
instance/server/default/conf/META-INF/hivemodule.xml	Apply a Hive module config to the container. Used for persisting OAuth clients, grants, and sessions to an external DB.

Note

By default, PingFederate is shipped with a handful of integration kits and adapters. If you need other integration kits or adapters in the deployment, manually download them and place them inside `server/default/deploy` of the server profile. You can find these resources in the product download page here.

PingAccess

Example at getting-started/pingaccess.

Table of Paths and Location Descriptions

Path	Location description
instance	Directories and files that you want to be used at product runtime, in accordance with the directory layout of the product.
instance/conf/pa.jwk	Used to decrypt a `data.json` configuration upon import.
instance/data/data.json	PA 6.1+ A config file that, if found by the container, is uploaded into the container.
instance/data/PingAccess.mv.db	Database binary that would be ingested at container startup if found.

Note

PingAccess profiles are typically minimalist because the majority of PingAccess configurations can be found within a `data.json` or `PingAccess.mv.db` file. You should use `data.json` for configurations and only use `PingAccess.mv.db` if necessary. You can easily view and manipulate configurations directly in a JSON file as opposed to the binary `PingAccess.mv.db` file. This fact makes tracking changes in version control easier as well. PingAccess 6.1.x+ supports using only `data.json`, even when clustering. *However* on 6.1.0.3 make sure `data.json` is only supplied to the admin node.

Note

PingAccess 6.1.0+ now supports native `data.json` ingestion, which is *the recommended method*. Place `data.json` or `data.json.subst` in `instance/conf/data/start-up-deployer`.

> *The JSON configuration file for PingAccess must be named* `data.json`.

A `data.json` file that corresponds to earlier PingAccess versions *might* be accepted. However, after you are on version 6.1.x, the `data.json` file will be forward compatible. This support means you are able to avoid upgrades for your deployments!

Note

For **PingAccess 6.0.x and earlier**, the JSON configuration file for *must* be named `data.json` and located in the `instance/data` directory.

Note

For **all PingAccess versions**, a corresponding file named `pa.jwk` must also exist in the `instance/conf` directory for the `data.json` file to be decrypted on import. To get a `data.json` and `pa.jwk` that work together, pull them both from the same running PingAccess instance.

For example, if PingAccess is running in a local Docker container you can use these commands to export the `data.json` file and copy the `pa.jwk` file to your local Downloads directory:

```
    curl -k -u "Administrator:${ADMIN_PASSWORD}" -H "X-Xsrf-Header: PingAccess" https://
localhost:9000/pa-admin-api/v3/config/export -o ~/Downloads/data.json

    docker cp <container_name>:/opt/out/instance/conf/pa.jwk ~/Downloads/pa.jwk
```

Note

You can find the PingAccess administrator password in `PingAccess.mv.db`, not in `data.json`. For this reason, you can use the following environment variables to manage different scenarios:

- `PING_IDENTITY_PASSWORD`
  Use this variable if:
  - You're starting a PingAccess container without any configurations.
  - You're using only a `data.json` file for configurations.
  - Your `PingAccess.mv.db` file has a password other than the default "2Access".

  The `PING_IDENTITY_PASSWORD` value will be used for all interactions with the PingAccess Admin API (such as importing configurations and creating clustering).
- `PA_ADMIN_PASSWORD_INITIAL`
  Use this variable in addition to `PING_IDENTITY_PASSWORD` to change the runtime admin password and override the password in `PingAccess.mv.db`.

> *If you use only* `data.json` *and do notpass* `PING_IDENTITY_PASSWORD`*, the password will default to "2FederateM0re".* ***Always*** *use* `PING_IDENTITY_PASSWORD`*.*

Ping Data Products

The Ping Data Products (PingDirectory, PingDataSync, PingAuthorize, PingDirectoryProxy) follow the same structure for server-profiles.

Example at getting-started/pingdirectory.

Table of Paths and Location Descriptions

Path	Location description
pd.profile	**Server profile matching the structure as defined by** PingDirectory Server Profiles

Path	Location description
instance	Directories and files that you want to be used at product runtime, in accordance with the layout of the product. In general, this should be non existing or empty.
env-vars	You can set environment variables used during deployment. See Variables and Scope for more info. In general, this should be non existing or empty.
extensions	You can provide URLs to download Server SDK extensions in a remote.list file. See Including Extensions in PingData Server Profiles for more info.

Note

- In most circumstances, the `pd.profile` directory should be the only directory in the server profile.
- All environment variables should be provided through Kubernetes configmaps/secrets and a secret management tool. Be careful providing an `env-vars` and if you do, please review Variables and Scope

Note

Use the `manage-profile` tool (found in product `bin` directory) to generate a `pd.profile` from an existing Ping Data 8.0+ deployment. An example on creating this `pd.profile` looks like:

```
manage-profile generate-profile --profileRoot /tmp/pd.profile
rm /tmp/pd.profile/setup-arguments.txt
```

Follow the instructions provided when you run the `generate-profile` to ensure that you include any additional components, such as `encryption-settings`.

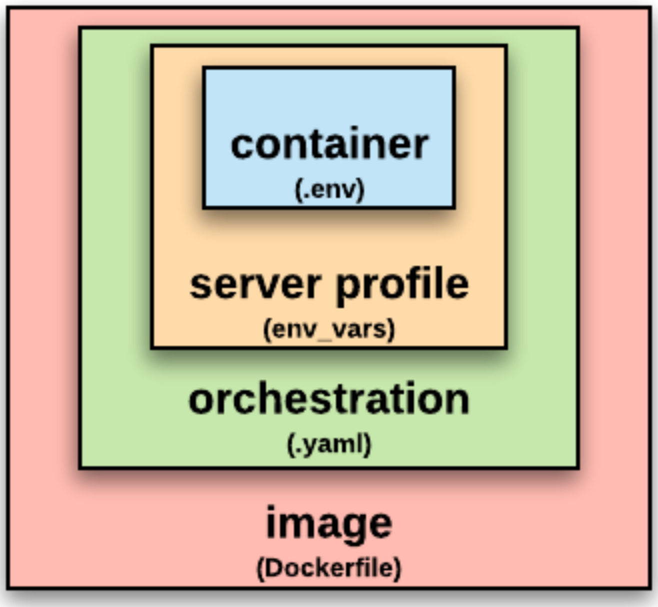
Variables and Scope

Variables provide a way to store and reuse values with our Docker containers which are ultimately used by our Docker image hooks to customize configurations.

It's important to understand:

- The different levels at which you can set variables
- How you should use variables
- Where you should set and use variables

The following diagram shows the different scopes in which variables can be set and applied.



Assume that you are viewing this diagram as a pyramid with the container at the top. The order of precedence for variables is top-down. Generally, you set variables having an orchestration scope.

Image scope

Variables having an image scope are assigned using the values set for the Docker image (for example, from Dockerfiles). These variables are often set as defaults, allowing scopes with a higher level of precedence to override them.

To see the default environment variables available with any Docker image, enter:

```
docker run pingidentity/<product-image>:<tag> env | sort
```

Where <product-image> is the name of one of our products, and <tag> is the release tag (such as `edge`).

For the environment variables available for all products (PingBase) or individual products, see Container Docker Images Information.

Orchestration scope

Variables having orchestration scope are assigned at the orchestration layer. Typically, these environment variables are set using Docker commands, Docker Compose or Helm values. For example:

- Using `docker run` with `--env`:

```
docker run --env SCOPE=env \
  pingidentity/pingdirectory:edge env | sort
```

- Using `docker run` with `--env-file`:

```
echo "SCOPE=env-file"  > /tmp/scope.properties

docker run --env-file /tmp/scope.properties \
  pingidentity/pingdirectory:edge env | sort
```

- Using Docker Compose (docker-compose.yaml):

```
environment:
  - SCOPE=compose
    env_file:
  - /tmp/scope.properties
```

- Using Kubernetes:

```
env:
  - name: SCOPE
    value: kubernetes
```

- Using Helm variables:

```
global:
  envs:
    PING_IDENTITY_ACCEPT_EULA: "YES"
    PING_IDENTITY_PASSWORD: "2Federate"
  ...
```

Server profile scope

Variables having server profile scope are supplied using property files in the server-profile repository. You need to be careful setting variables at this level because the settings can override variables already having an image or orchestration scope value set.

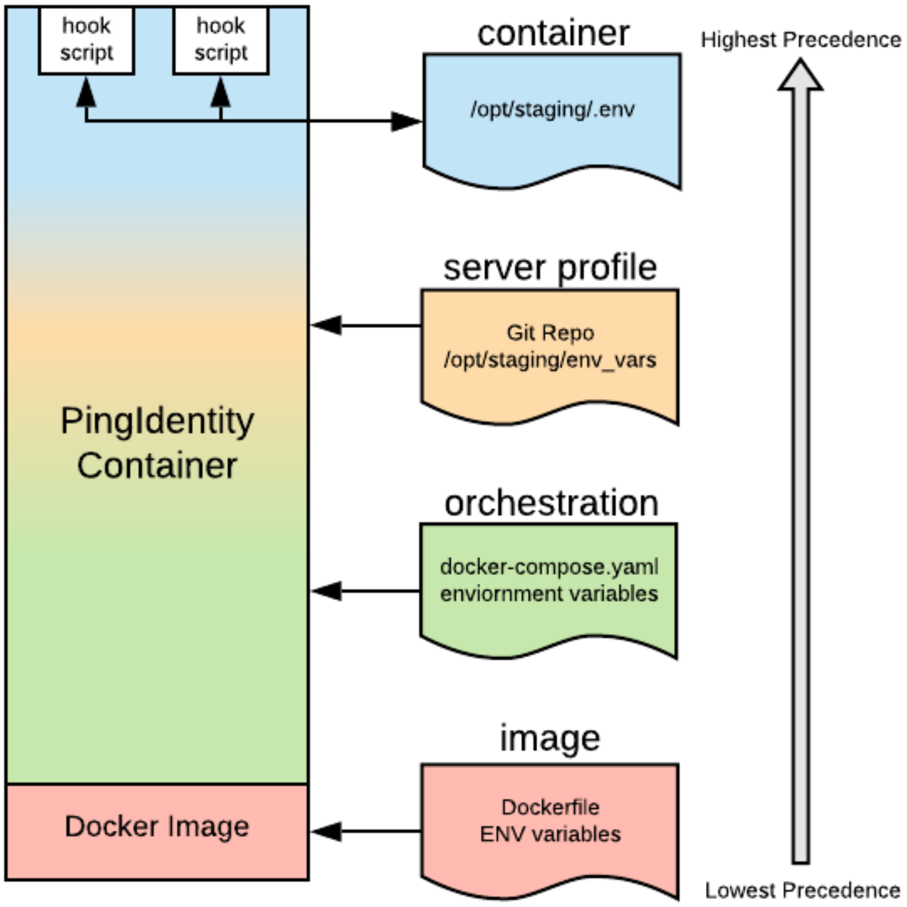
You can use the following masthead in your `env_vars` files to provide examples of setting variables and how they might override variables having a scope with a lower level of precedence. It will also suppress a warning when processing the env_vars file:

```
# .suppress-container-warning
#
# NOTICE: Settings in this file will override values set at the
#         image or orchestration layers of the container.  Examples
#         include variables that are specific to this server profile.
#
# Options include:
#
# ALWAYS OVERRIDE the value in the container
#   NAME=VAL
#
# SET TO DEFAULT VALUE if not already set
#   export NAME=${NAME:=myDefaultValue}  # Sets to string of "myDefaultValue"
#   export NAME=${NAME:-OTHER_VAR}       # Sets ot value of OTHER_VAR variable
#
```

Container scope

Variables having a container scope are assigned in the hook scripts and will overwrite variables that are set elsewhere. Variables that need to be passed to other hook scripts must be appended to the file assigned to `${CONTAINER_ENV}`, (which defaults to `/opt/staging/.env`). This file is sourced by every hook script.

Scoping example



Using Certificates with Images

This page provides details for using certificates with the Ping Identity images. Specifically, it outlines the preferred locations to place the certificate and PIN/key files to provide best security practices and enable use by the underlying Ping Identity product.

Currently, certificates can be provided to the PingData products (PingDirectory, PingDataSync, PingAuthorize, and PingDirectoryProxy) when the containers are started. For non-PingData images, such as PingAccess and PingFederate, the certificates are managed within the product configurations. Those images will not be covered here.

Before you begin

You must:

- Complete Get started to set up your DevOps environment and run a test deployment of the products.
- Strongly recommended: Have a secrets management system, such as Hashicorp Vault, that holds your certificate and places them into your SECRETS_DIR (/run/secrets).

  For information on using a vault, if you have one, see Using Hashicorp Vault.

About this topic

The following examples explain how to deploy a certificate/PIN combination to an image in a secure way.

PingData Image Certificates

The PingData products (PingDirectory, PingDataSync, PingAuthorize, and PingDirectoryProxy) use a file location to determine certificates/PIN files:

- It is best practice to use a non-persistent location, such as /run/secrets, to store these files.
- If no certificate is provided, the container/product will generate a self-signed certificate.

The default location for certificates and associated files are listed below, assuming a default SECRETS_DIR variable of `/run/secrets`.

	Variable Used	Default Location/Value / run/secrets...	Notes
Keystore (JKS)	KEYSTORE_FILE	keystore	Java KeyStore (JKS) Format. Set as default in absence of .p12 suffix.
Keystore (PKCS12)	KEYSTORE_FILE	keystore.p12	PKCS12 Format
Keystore Type	KEYSTORE_TYPE	jks, pkcs12, pem, or bcfks	Based on suffix of KEYSTORE_FILE. Only use BCFKS in FIPS mode.
Keystore PIN	KEYSTORE_PIN_FILE	keystore.pin	

	Variable Used	Default Location/Value / run/secrets...	Notes
Truststore (JKS)	TRUSTSTORE_FILE	truststore	Set as default in absence of .p12 suffix.
Truststore (PKCS12)	TRUSTSTORE_FILE	truststore.p12	PKCS12 Format
Truststore Type	TRUSTSTORE_TYPE	jks, pkcs12, pem, or bcfks	Based on suffix of TRUSTSTORE_FILE. Only use BCFKS in FIPS mode.
Truststore PIN	TRUSTSTORE_PIN_FILE	truststore.pin	
Certificate Nickname	CERTIFICATE_NICKNAME	see below	

Note

There is an additional certificate-based variable used to identity the certificate alias used within the `KEYSTORE_FILE`. That variable is called `CERTIFICATE_NICKNAME`, which identifies the certificate to use by the server in the `KEYSTORE_FILE`. If a value is not provided, the container will look at the list certs found in the `KEYSTORE_FILE` and if one - and only one - certificate is found of type `PrivateKeyEntry`, that alias will be used.

Tip

If you are relying on certificates to be mounted to a different locations than the SECRET_DIR location or a different filename, you can provide your own values for those variables identified above. As an example:

```
KEYSTORE_FILE=/my/path/to/certs/cert-file
KEYSTORE_PIN_FILE=/my/path/to/certs/cert.pin
KEYSTORE_TYPE=jks
CERTIFICATE_NICKNAME=development-cert
```

PingData image certificate rotation

The certificate rotation process for PingData products varies depending on which product is being configured and whether that product is in a topology. For products that are not in a topology, certificates can be rotated by simply updating the environment variables. For products in a topology, certificate rotation must be done via a command-line call with the servers in the topology online.

Rotating the listener certificate by adjusting environment variables

The process described in this section can be used for PingAuthorize, PingDirectoryProxy, and *standalone* (single-server) instances of PingDirectory or PingDataSync.

Warning

If PingDirectory or PingDataSync is deployed with multiple servers, use the process described in the next section.

As mentioned above, for the PingData products there are variables defining the server truststore and keystore. To change certificates, you will need to update the contents of the truststore or keystore in your server profile or secret store. After you update the contents, restart the container. The changes will be picked up automatically when the server restarts. If you have multiple certificates in the keystore, you can use the above-mentioned CERTIFICATE_NICKNAME variable to specify the certificate. The container will pick up that certificate from those stored in the keystore. For updating the product to use the new certificates, perform a rolling update. This action ensures that other servers will remain available as each pod is cycled.

Note

Verify that remaining pods in the cluster have sufficient capacity to handle the increased load during the rolling update.

Rotating the listener certificate with the replace-certificate command-line tool

If multiple PingDataSync or PingDirectory servers are running in a topology, then the servers must be online when updating the listener certificate. Updates to certificates with one or more servers offline (such as rolling updates) can lead to connection issues with the other members of the topology when those servers come back online. Use the PingData `replace-certificate` command-line tool to update certificates with the server online.

Warning

If your keystore and truststore files are on a read-only file system, use the process described in the next section.

Shell into the running instance that needs to be updated, and ensure the keystore containing the needed certificate is mounted on the container. Then, run `replace-certificate`. Replace the `--key-manager-provider` and `--trust-manager-provider` values if necessary when using a non-JKS keystore, as well as the `--source-certificate-alias` value if necessary.

```
replace-certificate replace-listener-certificate \
    --key-manager-provider JKS \
    --trust-manager-provider JKS \
    --source-key-store-file /run/secrets/newkeystore \
    --source-key-store-password-file /run/secrets/newkeystore.pin \
    --source-certificate-alias server-cert \
    --reload-http-connection-handler-certificates
```

For more information on this command, run- `replace-certificate replace-listener-certificate --help`

Running the first command will replace the listener certificate and notify other servers in the topology that this server's certificate has changed.

To update certificates for the other servers in the topology, follow this same process, shelling into each individual instance.

Once this is done, the running pods have been updated. To ensure a restart does not undo these changes, verify that your server profile and orchestration environment variables are updated to point to the new certificates. For example, if you have modified your server configuration to point to `/run/secrets/newkeystore`, then you must update your KEYSTORE_FILE environment variable to point to that new keystore *after* you have completed the `replace-certificate` process on each server.

Rotating the listener certificate when your keystore and truststore are on a read-only filesystem

Typically, the `replace-certificate` tool will edit your keystore and truststore when rotating your listener certificate. Because of this, the above method will not work when the keystore and truststore are read-only. In this case, use one of the two following processes to rotate your listener certificates.

Rotating certificates when they are all signed by the same issuer certificate

If all your listener certs will be signed by the same certificate authority, you can add this CA to your server instance listeners to make rotation easier, as the servers will automatically trust certificates signed by the CA.

Add the issuer certificate to the server instance listeners in the topology

- Copy a PEM file of the your issuer certificate onto your pods. For this example the path will be `/opt/out/instance/config/ca.crt`.
- On each server, use `replace-certificate` to add the issuer certificate to the server instance listener for that server. This will make the servers trust each other provided they are using a cert signed by this issuer. Note that this must be done with all servers online, so that the change gets replicated to the other servers.

```
replace-certificate add-topology-registry-listener-certificate \
    --certificate-file /opt/out/instance/config/ca.crt
```

Point the server's key manager provider at your new certificate

- Ensure your new certificate has been added to your keystore in whatever external storage method you are using (Vault, etc.), and note the alias you have given the new cert. For this example the alias will be `newcert`.
- Set `CERTIFICATE_NICKNAME=newcert` and perform a rolling update. `manage-profile replace-profile` will run and point your connection handlers to `newcert`, while the servers will continue to trust each other since that cert was signed by the trusted CA you added in the previous step.

Removing the old certs

- If desired, you can now remove unused certs from the server instance listeners in the topology registry. You can also remove the old certs from your keystores in your external storage. To remove the old certs from the topology registry, use `replace-certificate`, running the following command on each server in the topology. Note that again this must be done with all servers online, so that the config change gets mirrored across the topology.

  `replace-certificate purge-retired-listener-certificates`

- It is also possible to purge the retired certificates from a single server rather than running a command on each pod, but it requires some configuration changes since it relies on an extended operation and a specific topology admin permission, so it will likely be easier to simply run the previous command on each server. Using dsconfig, the necessary changes would be:

```
dsconfig create-extended-operation-handler \
   --handler-name "Replace Certificate Extended Operation Handler"  \
   --type replace-certificate  \
   --set enabled:true  \
   --set allow-remotely-provided-certificates:true
dsconfig set-topology-admin-user-prop \
   --user-name admin  \
   --add privilege:permit-replace-certificate-request
```

- After these changes are in place on the other servers, the following command can be used to purge retired listener certificates from remote instances:

`replace-certificate purge-remote-retired-listener-certificates`

Rotating certificates without the assumption that they are all signed by the same issuer certificate

Add the new cert to the server instance listeners in the topology

- Add the new desired certificate to your keystore and truststore, in whatever external storage method you are using. Note that you are just adding the new cert, not removing the old one yet. Note the alias that you have given the new cert in the keystore. In these examples the new cert's alias will be `newcert` and the previous one `server-cert`.
- Ensure the pods have the updated keystore and truststore on the filesystem, via a rolling update. At this point the keystores and truststores will have both the old cert and the new cert, but the new one is not yet being used.
- On each server, export the PEM file of the new certificate to a writable location, using the `manage-certificates` tool. This action is necessary because the subsequent command can only use a PEM file, it can't read from a keystore directly.

```
manage-certificates export-certificate \
    --keystore /run/secrets/keystore \
    --keystore-password-file /run/secrets/keystore.pin \
    --alias newcert \
    --output-file /opt/out/instance/config/newcert.crt \
    --output-format PEM
```

- On each server, use `replace-certificate` to add the exported certificate PEM file to the server instance listener in the topology. Note that this step must be done with all servers online, so that the config change is mirrored to the other servers in the topology.

```
replace-certificate add-topology-registry-listener-certificate \
    --certificate-file /opt/out/instance/config/newcert.crt
```

- Now the server will have both the previous cert and the new cert in its server instance listener.

Point the server’s key manager provider at the new cert

- There are two ways to do this - first is to change the CERTIFICATE_NICKNAME environment variable to point to your new certificate’s alias, and then just restart your pods, allowing manage-profile replace-profile to apply the change on startup.
- The second is to edit your keystore and truststore and rename your new cert to the same alias as the previous one (in the case of this document, renaming `newcert` to `server-cert` - the previous cert will have to be either renamed or removed from the keystores). This way, the key manager provider for the server will load in the new cert. Then you can restart the pods and the new cert will be loaded on server startup.

Removing the old cert

- Refer to the removal step in the previous section.

Troubleshooting

Getting started

Examples Not Working

Many common errors using Ping containers arise from using stale images. Our development is highly dynamic and Docker images can rapidly change.

To avoid issues with stale images, remove all local images from your local cache. Doing so will force Docker to pull the latest images:

```
docker rmi $(docker images "pingidentity/*" -q)
```

> **Note**
>
> Even though you might have a local image tagged "latest", this tag does not guarantee it is the newest image in the Docker hub registry. This tag is reapplied for each release image.

Misconfigured `pingctl`

If your containers cannot pull a license based on your DevOps user name and key, there might be some misconfiguration in your `pingctl config` file.

Possible solutions:

1. If you have ran `pingctl config` for the first time, see the Environment Configuration Documentation on how to export configured variables to your environment.
2. Run `pingctl info` and make sure the configured variables in the utility are correct. See the utility's documentation for more information.

Unable To Retrieve Evaluation License

If a product instance or instances cannot retrieve the evaluation license, you might receive an error similar to this:

```
----- Starting hook: /opt/staging/hooks/17-check-license.sh
Pulling evaluation license from Ping Identity for:
              Prod License: PD - v7.3
              DevOps User: some-devops-user@example.com...
Unable to download evaluation product.lic (000), most likely due to invalid PING_IDENTITY_DEVOPS_USER/
PING_IDENTITY_DEVOPS_KEY


        ALERT


#
# No Ping Identity License File (PingDirectory.lic) was found in the server profile.
# No Ping Identity DevOps User or Key was passed.
#
#
# More info on obtaining your DevOps User and Key can be found at:
#     https://devops.pingidentity.com/how-to/devopsRegistration/
#
##
CONTAINER FAILURE: License File absent
CONTAINER FAILURE: Error running 17-check-license.sh
CONTAINER FAILURE: Error running 10-start-sequence.sh
```

This error can be caused by:

1. An invalid DevOps user name or key (as noted in the error). This failure is usually caused by some issue with the variables being passed in. To verify the variables in the `pingctl` configuration are correct for running Docker commands, run the following command:

   `pingctl info`

2. A bad Docker image. Pull the Docker image again to verify.

3. Network connectivity to the license server is blocked. To test this, on the machine that is running the container, run:

   `curl -k https://license.pingidentity.com/devops/license`

   If the license server is reachable, you will receive an error similar to this example:

   `{ "error":"missing devops-user header" }`

Using DevOps Hooks

Note

For more information on hook scripts, see the Ping product image hook script exploration video.

Our DevOps hooks are build-specific scripts that are called, or can be called, by the `entrypoint.sh` script used to start our product containers.

Caution

Use of the hook scripts is intended only for DevOps professionals familiar with the products.

The available hooks are built into the product images and can be found in the `hooks` subdirectory of each product directory in the Docker Builds repository.

In the `entrypoint.sh` startup script, there is an example (stub) provided for the available hooks for all products.

Warning

It is **critical** that the supplied hook names be used if you modify `entrypoint.sh`. For example, they can be used to make subtle changes to a server profile.

Using .pre and .post hooks

When the hook scripts are called during the `entrypoint.sh` initialization, any corresponding `.pre` and `.post` hooks are also called.

The `.pre` and `.post` extensions allow you to define custom scripts to be executed before or after any hook that is run in the container. You can include any custom `.pre` and `.post` hooks in the `hooks` directory of your server profile.

Hooks with a `.pre` extension are run before the corresponding hook, and hooks with a `.post` extension are run after the corresponding hook.

For example, a script named `80-post-start.sh.pre` will execute immediately before the `80-post-start.sh` hook and a script named `80-post-start.sh.post` will be run immediately after that hook completes.

Kubernetes Basics

Although this document cannot cover all aspects of these tools, new Kubernetes users might find other technical documentation too involved for purposes of using Ping Identity images. This document aims to equip new users with helpful terminology in simple terms, with a focus on relevant commands.

> **Note**
>
> This overview uses Ping Identity images and practices as a guide, but generally applies to any interactions in Kubernetes. With these assumptions, this document might feel incomplete or inaccurate to veterans. If you would like to contribute to this document, feel free to open a pull request!

Kubernetes

Terms

Cluster - The ice cube tray

You can consider a Kubernetes cluster as a set of resources into which you can deploy containers. A cluster can be as small as your local computer or as large as hundreds of virtual machines (VMs), called nodes, in a data center. Interaction with the cluster is through an API requiring authentication and role-based access control (RBAC) that allows the actions necessary within the cluster.

In a cloud provider Kubernetes cluster, such as Amazon Web Services (AWS) EKS, Azure AKS, or Google GKE, the cluster can span multiple Availability Zones (AZs), but only *one* region. In AWS terms, a cluster can be in the region us-west-2 but have nodes in the AZs us-west-2a, us-west-2b, and us-west-2c. Kubernetes provides high availability by distributing applications with multiple instances of containers, called replicas, across available AZs.

Nodes - The individual ice cube spaces in the tray

The nodes are the pieces that provide allocatable resources, such as CPU and memory, and make up a cluster. Typically, these are VMs, and for example in AWS, they would be EC2 instances.

Namespace - A loosely defined slice of the cluster

Namespaces are intended to be a virtual delimiter for deploying grouped applications. While it is possible for pods to communicate across namespaces, policies can be put in place with third-party services to prevent this communication.

> **Note**
>
> You can allocate resource limits available to a namespace, but this is not required.

Context - A definition in your ~/.kube/config file that specifies the cluster and namespace where your `kubectl` commands will be executed.

Deployments and Statefulsets - The water that fills ice cube spots.

Applications are deployed as Deployments or Statefulsets. You can consider both of these objects as controllers that define and manage the following:

- Name of an application
- Number of instances (pods) of an application (replicas)

- Persistent storage

Though they are similar, Deployments differ from Statefulsets in a few fundamental ways.

Deployments

- Deployments are typically used for stateless applications - if a pod is lost or removed, any other pod in the same deployment can take on the activity the lost pod was performing.
- Pod names are inconsequential because each pod is identical with no state information required. As a result, the name of the pod does not matter and names are suffixed with a randomly generated string.
- The order in which pods are started is also inconsequential. When starting a deployment all pods are launched at the same time.
- When updating a deployment, you can cycle one, many, or all pods at the same time.

Statefulsets

StatefulSets are more structured in the manner in which the pods are handled.

- StatefulSets - as the name implies - are used for applications in which a known state is required. For example, many clustered products have an instance in the cluster that is considered the leader and all pods in the set need to know which pod is acting in this capacity. A controlled scale-up and scale-down process is needed to maintain known state as application nodes or instances join or leave the cluster or are restarted.
- Pod names are *sticky* in that each pod in the StatefulSet has a known name, with each pod receving an ordinal indicator (unlike the random pod name found in Deployments). For example, a StatefulSet will have pod names similar to: `mypinq-pingdirectory-0`, `myping-pingdirectory-1`, and `myping-pingdirectory-2`
- Controlled startup with health priority: unlike a Deployment, a StatefulSet deploys the first instance (pod name appended with -0) and waits for it to be healthy before adding another to the group.
- Updates occur to instances in a rolling fashion, one-at-a-time, starting with the most recent pod (e.g., `myping-pingdirectory-2`) first.
- With a known Pod name, persistent storage can be maintained for each pod. After persistent storage is created and assigned, the same storage object is provided to the same-named pod every time.

Pod - The molecules that make up the water

A Deployment/StatefulSet specifies the *number* of pods to run for a given application. For example, you can have a `pingfederate-engine` deployment that calls for three replicas with two CPUs and 2 GB of memory, but you cannot make one engine larger or smaller than the others.

Like a molecule, a pod can consist of just one container, or it can have multiple containers, called sidecars. For example, your pod can have a PingFederate container as the main process and a sidecar container, such as Splunk Universal Forwarder, to export logs. All containers in a pod, including these sidecars, share a namespace and IP address.

Pods are are considered disposable and by default do not persist any data. To maintain state or data, external storage or a database of some kind is needed.

PersistentVolume (PV) and PersistentVolumeClaim (PVC) - A virtual external storage device or definition attached to a Pod

The PV is the storage object and PVC is the claim that a given pod makes for that storage.

Service - A slim loadbalancer within the cluster

Pods can come and go, be disposed of or restarted. Every time a Pod is started, it will receive an IP address which often changes. In order to access the application hosted in the Pod, a fixed, known location or address is required.

Services provide a single IP address and cluster-internal DNS resolution that is placed in front of Deployments and Statefulsets to distribute traffic. For service-to-service communication, such as PingFederate using PingDirectory as a user store, the application should be configured to point to a service name and port rather than the individual pods. Services are given fully-qualified domain names (FQDNs) in a cluster. Within the same namespace, services are accessible by their name (`https://myping-pingdirectory:443`), but across namespaces, you must be more explicit (`https://myping-pingdirectory.<namespace>:443`). A FQDN would be `https://myping-pingdirectory.<namespace>.svc.cluster.local` .

Ingress - A network definition used to expose an application external to the cluster. In order for an ingress to work, you need an Ingress Controller.

A common pattern is a deployment of Nginx pods fronted by a physical loadbalancer. The client application traffic hits the loadbalancer first, then is forwarded to Nginx. The header information (hostname and path) of the request is evaluated and forwarded to a corresponding application service in the cluster.

For example, suppose a PingFederate ingress has a host name of myping-pingfederate-engine.pingdemo.example. If a client application makes a request to `https://myping-pingfederate-engine.pingdemo.example/pf/heartbeat.ping` , the traffic flow of the request would be:

- Client -> LoadBalancer -> Nginx k8s Service -> Nginx Pod -> Pingfederate-engine k8s Service -> Pingfederate-engine pod

Commands

To see which cluster and namespace you are using, use the kubectx tool.

Alternatively, you can run the following commands:

```
# Retrieve and set context
kubectl config get-contexts
kubectl config current-context
kubectl config use-context my-cluster-name

# Set Namespace
kubectl config set-context --current --namespace=<namespace>
```

Viewing resources

You can use k9s, which is a UI designed to run in a terminal.

If you cannot use k9s, review the standard commands here.

You can run `kubectl get` for any resource type, such as Pods, Deployments, Statefulsets, and PVCs. Many resources have short names:

- `po` = pods
- `deploy` = Deployments
- `sts` = Statefulsets

- `ing` = ingresses
- `pvc` = persistentvolumeclaims

The most common command is `get pods`:

```
kubectl get pods
```

To show anything that the container prints to `stdout`, use `logs`:

```
kubectl logs -f <pod-name>
```

To show the logs of a pod with multiple containers, specify the container for which you wish to view logs with the `-c` option:

```
kubectl logs -f <pod-name> -c <container-name>
```

To show the logs of a crashed pod (`RESTARTS != 0`):

```
kubectl logs -f <pod-name> --previous
```

To see available host names by ingress:

```
kubectl get ing
```

Debugging

When a pod crashes unexpectedly, you can mine information about the cause with the following commands.

To view logs of the crash:

```
kubectl logs -f <pod-name> --previous
```

To view the reason for exit:

```
kubectl describe pod <pod-name>
```

When looking at `describe`, there are two main sections of the output to note:

- lastState - shows the exit code and the reason for exit.
- Events - this list is most helpful when your pod is not being created. It might be stuck in pending state if:
  - There are not enough resources available for the pod to be created.
  - Something about the pod definition is incorrect, such as a missing volume or secret.

Common exit codes associated with containers are:

Table of Exit Codes and Descriptions

Exit Code	Description
Exit Code 0	Absence of an attached foreground process
Exit Code 1	Indicates failure due to application error

Exit Code	Description
Exit Code 137	**Indicates failure as container received SIGKILL (manual intervention or ‘oom-killer’ [OUT-OF-MEMORY])**
Exit Code 139	**Indicates failure as container received SIGSEGV**
Exit Code 143	**Indicates failure as a container received SIGTERM**

Helm Basics

Although this document cannot cover the depths of this tool, new Helm users might find other technical documentation too involved for the purpose of beginning use of Ping Identity container images. This document aims to equip new users with helpful terminology in simple terms, with a focus on relevant commands. For more in-depth documentation around Helm, check out helm.sh.

> **Note**
>
> This overview uses Ping Identity images and practices as a guide, but generally applies to any interactions using Helm with Kubernetes. With these assumptions, this document might feel incomplete or inaccurate to veterans. If you would like to contribute to this document, feel free to open a pull request!

Helm

> **Note**
>
> All of our instructions and examples are based on the Ping Identity DevOps Helm chart. If you are not using the Ping Identity DevOps Helm chart in production, we still recommend using it to generate your direct Kubernetes manifest files. Using our provided chart to create your files in this manner gives Ping Identity the best opportunity to support your environment.

Everything in Kubernetes is deployed by defining what you want and allowing Kubernetes to achieve the desired state (the declarative model).

Helm simplifies your interaction by building deployment patterns into templates with variables. The Ping Identity Helm chart includes Kubernetes templates and default values maintained by Ping Identity. With these in hand, you only need to provide values for the template variables to match your environment.

For example, a service definition looks like the following file. With this file, Kubernetes is instructed to create a `service` resource with the name `myping-pingdirectory`.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/instance: myping
    app.kubernetes.io/name: pingdirectory
  name: myping-pingdirectory
spec:
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 1443
  - name: ldap
    port: 389
    protocol: TCP
    targetPort: 1389
  - name: ldaps
    port: 636
    protocol: TCP
    targetPort: 1636
  selector:
    app.kubernetes.io/instance: myping
    app.kubernetes.io/name: pingdirectory
  type: ClusterIP
```

Using our Helm chart, you can automatically define this entire resource and all other required resources for a basic deployment by setting `pingdirectory.enabled=true`.

Terminology

Manifests - the final Kubernetes YAML files that are sent to the cluster for resource creation. These files are standard Kubernetes files and will be similar to the service example shown above.

Helm Templates - Go Template versions of Kubernetes YAML files. These templates enable the manifest creation to be parameterized.

Values and values.yaml - A value is the setting you pass to a Helm chart from which the templates produce the manifests you want. Values can be passed individually on the command line, but more commonly they are collected and defined in a file named values.yaml. For example, if this file contained only this entry, the resulting Kubernetes manifest file would be over 200 lines long.

```
pingdirectory:
  enabled: true
```

release - When you deploy resources with Helm, you provide a name for identification. The combination of this name and the resources that are deployed using it make up a `release`. When using Helm, it is a common pattern to prefix all resources managed by a release with the release name. In our examples, `myping` is the release name, so you will see products in Kubernetes running with names similar to `myping-pingfederate-admin`, `myping-pingdirectory`, and `myping-pingauthorize`.

Building the Helm Values File

This documentation focuses on the Ping Identity DevOps Helm chart and the values passed to the Helm chart to achieve your configuration. For your deployment to fit your goals, you must create a values.yaml file.

The most simple values.yaml for our Helm chart would be:

```
global:
  enabled: true
```

By default, this flag is set as `global.enabled=false`. These two lines are sufficient to turn on (deploy) every available Ping Identity software product with a basic configuration.

Providing your own server profile

In the documentation, there is an example for providing your own server profile stored in GitHub to PingDirectory. The documenation provides this snippet in the values.yaml specific to that feature:

```
pingdirectory:
  envs:
    SERVER_PROFILE_URL: https://github.com/<your-github-user>/pingidentity-server-profiles
```

This entry alone will not turn on PingDirectory, because the default value for `pingdirectory.enabled` is false. To complete the deployment, add the snippet to turn deploy and configure PingDirectory in the values.yaml file:

```
global:
  enabled: true
pingdirectory:
  envs:
    SERVER_PROFILE_URL: https://github.com/<your-github-user>/pingidentity-server-profiles
```

This example snippet turns on all products, including PingDirectory, and overwrites the default `pingdirectory.envs.SERVER_PROFILE_URL` with `https://github.com/<your-github-user>/pingidentity-server-profiles`.

This use of substitution and parameters is where the power of Helm to simplify ease of deployment begins to shine. To fully customize your deployment, review all available options by running:

`helm show values pingidentity/ping-devops`

This command prints all of the default values applied to your deployment. To overwrite any default values from the chart, copy the corresponding snippet and include it in your own values.yaml file with any modifications needed. Remember with YAML that tabbing and spacing matters. For most editors, copying all the way to the left margin and pasting at the very beginning of a line in your file should maintain proper indentation.

Helm also provides a wide variety of plugins. A useful one is Helm diff. This plugin shows what changes would be applied between Helm upgrade commands. For example, if this plugin indicates anything in a Deployment or Statefulset would change, you can expect the corresponding pods to be cycled. In this example, Helm diff is useful to note changes that would occur, particularly when you are not prepared for containers to be restarted.

Additional Commands

As you go through the Helm examples, the goal is to build a values.yaml file that works in your environment.

Deploy a release:

```
helm upgrade --install <release_name> pingidentity/ping-devops -f /path/to/values.yaml
```

Delete a release:

This command will remove all resources except PVC and PV objects associated with the release from the cluster:

```
helm uninstall <release name>
```

Delete PVCs associated to a release:

```
kubectl delete pvc --selector=app.kubernetes.io/instance=<release_name>
```

Exit Codes

Exit Code	Description
Exit Code 0	Absence of an attached foreground process
Exit Code 1	Indicates failure due to application error
Exit Code 137	Indicates failure as container received SIGKILL (manual intervention or 'oom-killer' [OUT-OF-MEMORY])
Exit Code 139	Indicates failure as container received SIGSEGV
Exit Code 143	Indicates failure as a container received SIGTERM

Example Configurations

The Helm examples page contains example configurations for running and configuring Ping products using the Ping Devops Helm Chart. Please review the Getting Started Page before trying them.

Container Logging

This document provides an outline of how logging is handled in containerized environments. Please refer to the provided links at the end of this page for details on implementing a logging solution for your deployments.

Note

While providing examples for all logging solutions is impractical, there is an example for using Splunk on this portal here.

Problem statement

In a containerized deployment model, it is expected that containers (or pods under Kubernetes) will be ephemeral. Further, the standard practice for application logging in a container is to use `stdout` and, in some cases, `stderr` as the means of streaming logs. Ping product containers follow this practice. As a result, no logs will persist outside the lifecycle of the container or pod. In particular, if a pod is failing or in crashloop due to a misconfiguration or error, it is impossible to troubleshoot the cause as the logs that might provide information on the crash are lost each time the pod attempts to restart. It is important, then, to insure that logs are stored external to the container.

!!! error "In Case You Missed It" If a container is stopped for any reason, including crashes, all logging information from the container that is not stored elsewhere is lost.

Viewing logs

In a Kubernetes deployment, you can view the streaming logs (stdout/stderr) of a container in a pod by issuing the `kubectl logs` command. This function is useful for quickly examining logs from operational containers.

Persisting logs

Because you cannot rely on the logs from the container itself for long-term use, you must implement some means of storing the logging information apart from the container itself.

Logging sidecar

In the Kubernetes model, a common method of maintaining logs is to use the sidecar model. A logging sidecar is included in the pod and configured to grab the stdout/stderr streams from the application container and persist them to the logging service. Many vendors provide a Docker image for this sidecar that contains the agent for their product. In addition, they usually provide support for configuring the container to connect to their service and format the logs for consumption, such as through environment variables, Kubernetes ConfigMaps, or other means.

Advantages:

- Logs can be sent to different locations at the same time using multiple sidecars
- Access to the cluster node is not required - particularly useful for hosted Kubernetes environments
- No update to the application is required, assuming it dumps logging information to stdout/stderr

Disadvantage:

- Additional resources are required for running the extra container(s), though they tend to be lightweight

The TAIL_LOG_FILES environment variable

Many Ping products were designed and built for a server-deployed implementation. As a result, they write log information to files (the old model for logging), rather than to `stdout`. To ease containerization, an environment variable (`TAIL_LOG_FILES`) is included in the Docker images and this variable is fed to a function that streams these files to `stdout` as they are written.

While Ping includes key log files as defaults, this variable can be modified. You can add additional log files to this variable to include them in the `stdout` stream. See each product Dockerfile for the default value of this variable for the product in question.

References

The list below is not intended to be comprehensive but should provide a good starting point for understanding how logging works and what you can do to retain logs from your deployments.

!!! note "Examples only" Any vendor listed here should not be considered an endorsement or recommendation by Ping Identity for that service. Refer to the documentation for the image in question for further assistance.

- Kubernetes Logging Documentation
- Docker Logging Documentation
- Docker Hub images, listed alphabetically:
    - AWS Cloudwatch
    - Datadog
    - Fluentd
    - Graylog
    - Rsyslog
    - Sematext
    - Splunk Forwarder
    - Sumologic

Running product containers with a read-only root filesystem

Overview

In some environments, there is a requirement that the container filesystem be read-only. Our product images are maturing to support this capability natively in the future. In the meantime, this guide will explain the overall concepts and provide an example with PingDirectory. The other product images can operate in a similar manner.

Warning

This guide is intended to provide an example implementation of solving this problem; your situation might require a different approach.

Note

An excellent starting point for understanding what goes on with our containers as they are instantiated can be found in this video. It is highly recommended that you take the time to view it prior to working through this guide.

High-level process

In Ping product containers, the layered approach of bringing the environment and configuration parameters into the container at launch requires merging of files from server profiles and possibly other locations before the product is launched. This process means that files are modified at runtime, and a read-only root filesystem blocks this action. The overall approach is to use emptyDir volumes to overlay the directories that need modification, allowing the hook scripts to run as normal against the volume rather than the container filesystem. In order to get everything necessary for the scripts in place, an init container (using the same image as the product container) is launched and the files necessary are copied to the shared volume before starting the product container.

Prerequisites

- kustomize CLI utility to serve as a post-renderer for Helm.

File explanation

In the 30-helm/read-only-filesystem folder of the Getting Started repository is a values file and kustomize directory. First, we will explore the the `pd-values.yaml` file; inline comments explain what is going on:

```
initContainers:
pd-init:
name: runtime-init
# CHANGEMETAG TO VERSION NEEDED
# Init container uses the same image as the product container and therefore versions much match
image: pingidentity/pingdirectory:CHANGEMETAG
env:
# Override the startup command so the product is not launched in the init container
- name: STARTUP_COMMAND
value: "ls"
# Use a name different from /opt/staging for holding the copied files from the product image into the
emptyDir volume
- name: STAGING_DIR
value: "/opt/handoff"
# Just in case there is a .env we will need
- name: CONTAINER_ENV
value: "/opt/handoff/.env"
# Another flag for preventing the product from being launched
- name: STARTUP_FOREGROUND_OPTS
value: ""
envFrom:
# CHANGEMERELEASE TO MATCH HELM RELEASE NAME
- configMapRef:
name: CHANGEMERELEASE-global-env-vars
optional: true
- configMapRef:
name: CHANGEMERELEASE-env-vars
optional: true
- configMapRef:
name: CHANGEMERELEASE-pingdirectory-env-vars
- secretRef:
name: devops-secret
optional: true
- secretRef:
name: CHANGEME-pingdirectory-git-secret
optional: true
volumeMounts:
# emptyDir volume: /opt/staging will be copied from the init container to this volume
# This volume will be mounted as /opt/staging in the product container
- mountPath: /opt/handoff
name: staging
readOnly: false
# The location for the license file varies by product
# See https://devops.pingidentity.com/how-to/existingLicense/ for more information
# The license file is required for the init container to operate
- name: pingdirectory-license
mountPath: "/opt/staging/pd.profile/server-root/pre-setup/PingDirectory.lic"
subPath: PingDirectory.lic
# Also an emptyDir
- name: tmp
mountPath: "/tmp"
readOnly: false
# Also an emptyDir
- name: init-runtime
```

```
mountPath: "/opt/out"
readOnly: false
# Mount the slightly modified versions of the bootstrap and start sequence scripts (see below)
- mountPath: /opt/bootstrap.sh
name: bootstrap
readOnly: true
subPath: bootstrap.sh
defaultMode: 0555
- mountPath: /opt/staging/hooks/10-start-sequence.sh
name: init-start
readOnly: true
subPath: 10-start-sequence.sh
defaultMode: 0555

volumes:
# The 3 emptyDir volumes referenced above
init-runtime:
emptyDir: {}
staging:
emptyDir: {}
tmp:
emptyDir: {}
# This secret is created from a license file
pingdirectory-license:
secret:
secretName: pingdirectory-license
# Make the modified bootstrap and start sequence scripts available as configMaps
bootstrap:
configMap:
items:
- key: bootstrap.sh
path: bootstrap.sh
name: bootstrap
init-start:
configMap:
items:
- key: 10-start-sequence.sh
path: 10-start-sequence.sh
name: init-start

configMaps:
init-start:
data:
10-start-sequence.sh: |-
#!/usr/bin/env sh
echo "overwriting 10 hook"
#!/usr/bin/env sh
#
# Ping Identity DevOps - Docker Build Hooks
#
# Called when it has been determined that this is the first time the container has
# been run.
#

        ##############################################################################
        ####### Prevent init container from starting the product normally.  ##########
        ####### These two lines are the only delta from the default script. ##########
        ##############################################################################
        if test ${STARTUP_FOREGROUND_OPTS} != "" ; then
          test "${VERBOSE}" = "true" && set -x

          # shellcheck source=./pingcommon.lib.sh
          . "${HOOKS_DIR}/pingcommon.lib.sh"

          echo "Initializing server for the first time"

          run_hook "17-check-license.sh"

          run_hook "18-setup-sequence.sh"
        fi
  bootstrap:
    data:
      bootstrap.sh: |-
        #!/usr/bin/env sh

##############################################################################################################
        ####### Make a copy of everything under /opt/staging in the product image to /opt/handoff.
##########
        ####### Primarily, this makes the hook scripts available in the emptyDir (writable) volume.
##########
        ####### This line is the only delta from the default script.
##########

##############################################################################################################
        cp -r /opt/staging/* /opt/handoff
        test "${VERBOSE}" = "true" && set -x
        # shellcheck source=./staging/hooks/pingcommon.lib.sh
        . "${HOOKS_DIR}/pingcommon.lib.sh"

        _userID=$(id -u)
        _groupID=$(id -g)

        echo "### Bootstrap"
        if test "${_userID}" -eq 0; then
            echo_yellow "### Warning: running container as root user"
        else
            echo "### Using the default container user and group"

            _effectiveGroupName=$(awk 'BEGIN{FS=":"}$3~/^'"${_groupID}"'$/{print $1}' /etc/group)
            test -z "${_effectiveGroupName}" && _effectiveGroupName="undefined group"

            _effectiveUserName=$(awk 'BEGIN{FS=":"}$3~/^'"${_userID}"'$/{print $1}' /etc/passwd)
            test -z "${_effectiveUserName}" && _effectiveUserName="undefined user"

            echo "### Container user and group"
            echo "###     user : ${_effectiveUserName} (id: ${_userID})"
            echo "###     group: ${_effectiveGroupName} (id: ${_groupID})"
        fi

        # if the current process id is not 1, tini needs to register as sub-reaper
        if test $$ -ne 1; then
            _subReaper="-s"
        fi

        # shellcheck disable=SC2086,SC2048
        exec "${BASE}/tini" ${_subReaper} -- "${BASE}/entrypoint.sh" ${*}

pingdirectory:
enabled: true
envs:
MUTE_LICENSE_VERIFICATION: "yes"
ORCHESTRATION_TYPE: "NONE"
VERBOSE: "true"
# (Optional) Specify a particular tag by uncommenting these two lines and naming the tag to use.
# Otherwise, you will get the latest from Docker Hub.
# If a particular tag is used, be sure the init container tag matches above
# image:
#   tag: "2306"
includeInitContainers:
# Use the init container specification above at pod startup
- pd-init
# Share the volumes between the init container and the product container
includeVolumes:
- staging
- tmp
- pingdirectory-license
- bootstrap
- init-start
- init-runtime
volumeMounts:
# The emptyDir mounted at /opt/handoff in the init container is mounted to /opt/staging here
# Hook scripts and product startup will operate as with a read/write filesystem
- mountPath: /opt/staging
name: staging
readOnly: false
- name: pingdirectory-license
mountPath: "/opt/staging/pd.profile/server-root/pre-setup/PingDirectory.lic"
subPath: PingDirectory.lic
- name: tmp
mountPath: "/tmp"
readOnly: false
```

Note

Kustomize is used as the Ping helm charts do not support setting the readOnlyFileSystem value to the securityContext of a container at this time.

In the 30-helm/read-only-filesystem/kustomize subdirectory is a kustomize script and definition file.

The script simply runs kustomize:

```
#!/bin/bash

cat <&0 > kustomize/all.yaml

kustomize build kustomize && rm kustomize/all.yaml
```

The `kustomization.yaml` file sets the securityContext for the each container's root file system to read-only:

```
resources:
  - all.yaml

patches:
  - target:
      group: apps
      version: v1
      kind: StatefulSet
    patch: |-
      - op: add
        path: /spec/template/spec/containers/0/securityContext
        value:
          readOnlyRootFilesystem: true
      - op: add
        path: /spec/template/spec/initContainers/0/securityContext
        value:
          readOnlyRootFilesystem: true
```

Process

To use the example files to deploy PingDirectory with a read-only root filesystem, follow the steps here:

1. Generate a license file and create a secret. If you have an existing license file, you can use it here:

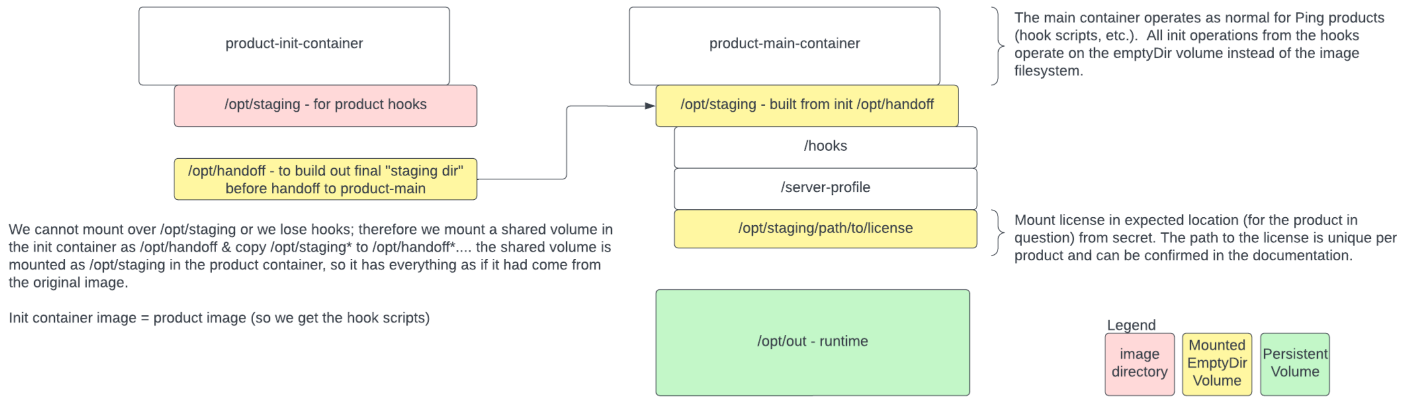
   ```
   # Generate the license file
   pingctl license pingdirectory 9.2 > PingDirectory.lic

   # Create the secret to match the name in the values file:
   kubectl create secret generic pingdirectory-license --from-file=./PingDirectory.lic
   ```

2. Update the `30-helm/read-only-filesystem/pd-values.yaml` file with the appropriate image tag and release name to be used with Helm. Afterward, use Helm to deploy the release:

   ```
   # For this example, the release name of 'rofs' is used
   cd 30-helm/read-only-filesystem
   helm upgrade --install rofs pingidentity/ping-devops -f './pd-values.yaml' \
         --post-renderer kustomize/kustomize
   ```

Diagram

This image provides an overview of what happens. It is best viewed in a separate tab:


product-init-container
/opt/staging - for product hooks
/opt/handoff - to build out final "staging dir" before handoff to product-main
We cannot mount over /opt/staging or we lose hooks; therefore we mount a shared volume in the init container as /opt/handoff & copy /opt/staging* to /opt/handoff*.... the shared volume is mounted as /opt/staging in the product container, so it has everything as if it had come from the original image.
Init container image = product image (so we get the hook scripts)
product-main-container
/opt/staging - built from init /opt/handoff
/hooks
/server-profile
/opt/staging/path/to/license
/opt/out - runtime
The main container operates as normal for Ping products (hook scripts, etc.). All init operations from the hooks operate on the emptyDir volume instead of the image filesystem.
Mount license in expected location (for the product in question) from secret. The path to the license is unique per product and can be confirmed in the documentation.
Legend
image directory
Mounted EmptyDir Volume
Persistent Volume

Note

One issue we encountered is that the **`/etc/motd`** file can be modified at startup by hook scripts, but /etc/ is read-only. We are exploring ways to address this in some way, but at this time, it appears one possible solution is to treat `/etc/` in the same manner as /opt/staging (copying to an emptyDir) if the `motd` file is to be updated. However, `/etc` has many more files and directories and such a solution is not practical. Baking it into a custom image is another possibility.

AWS Storage Considerations

AWS provides many storage options. When considering Ping products deployed in a containerized deployment, the choice typically comes down to two: elastic block storage (EBS) and elastic file system (EFS). Though there are a number of differences between them, on the surface they act similar when attached to an Elastic Kubernetes Service (EKS) node or Elastic Compute Cloud (EC2) instance.

However, Ping products (whether containerized or not) require high I/O performance, and Ping only recommends EBS volumes as the backing store. EFS performance is significantly lower and is not supported.

For additional product-specific requirements, visit the appropriate product page.

PingOne Worker Application and User Configuration

PingOne Worker Application Configuration

To manage PingOne resources using credentials other than your own, you are required to have a PingOne Worker App.

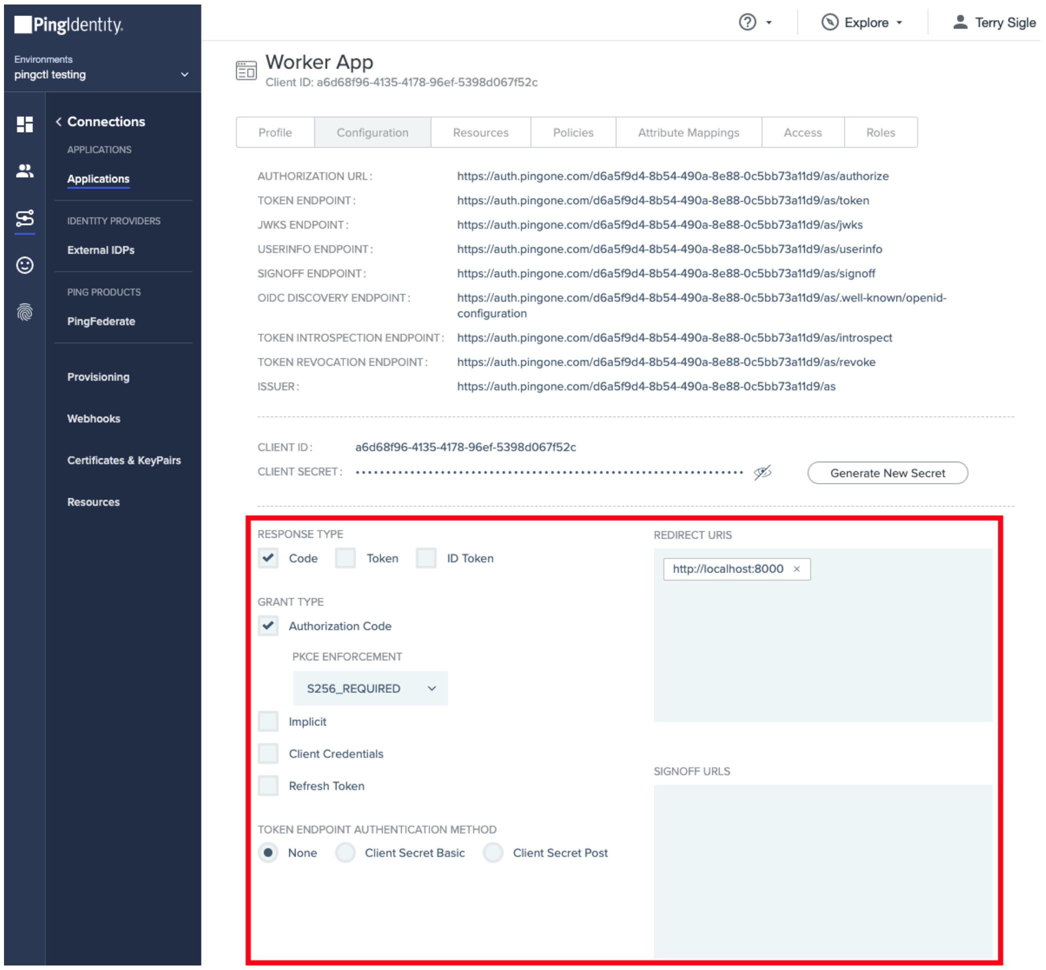
There are three options to authenticate to PingOne from pingctl:

- Authorization Code (w/ PKCE) Flow (Recommended and most secure) - Using a PingOne Admin User
- Implicit Flow - Using a PingOne Admin User
- Client Credentials Flow (Easiest, but most insecure, as a user isn't required)

Additionally, you must set up the proper roles for your Worker App

Authorization Code (w/ PKCE) Flow Settings

The following image shows an example of a Worker App setup for Authorization Code (w/ PKCE) Flow:



Worker App
Client ID: a6d68f96-4135-4178-96ef-5398d067f52c

AUTHORIZATION URL:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/authorize
TOKEN ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/token
JWKS ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/jwks
USERINFO ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/userinfo
SIGNOFF ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/signoff
OIDC DISCOVERY ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/.well-known/openid-configuration
TOKEN INTROSPECTION ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/introspect
TOKEN REVOCATION ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/revoke
ISSUER:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as

CLIENT ID: a6d68f96-4135-4178-96ef-5398d067f52c
CLIENT SECRET: ••

RESPONSE TYPE
✓ Code ☐ Token ☐ ID Token

GRANT TYPE
✓ Authorization Code
PKCE ENFORCEMENT
S256_REQUIRED
☐ Implicit
☐ Client Credentials
☐ Refresh Token

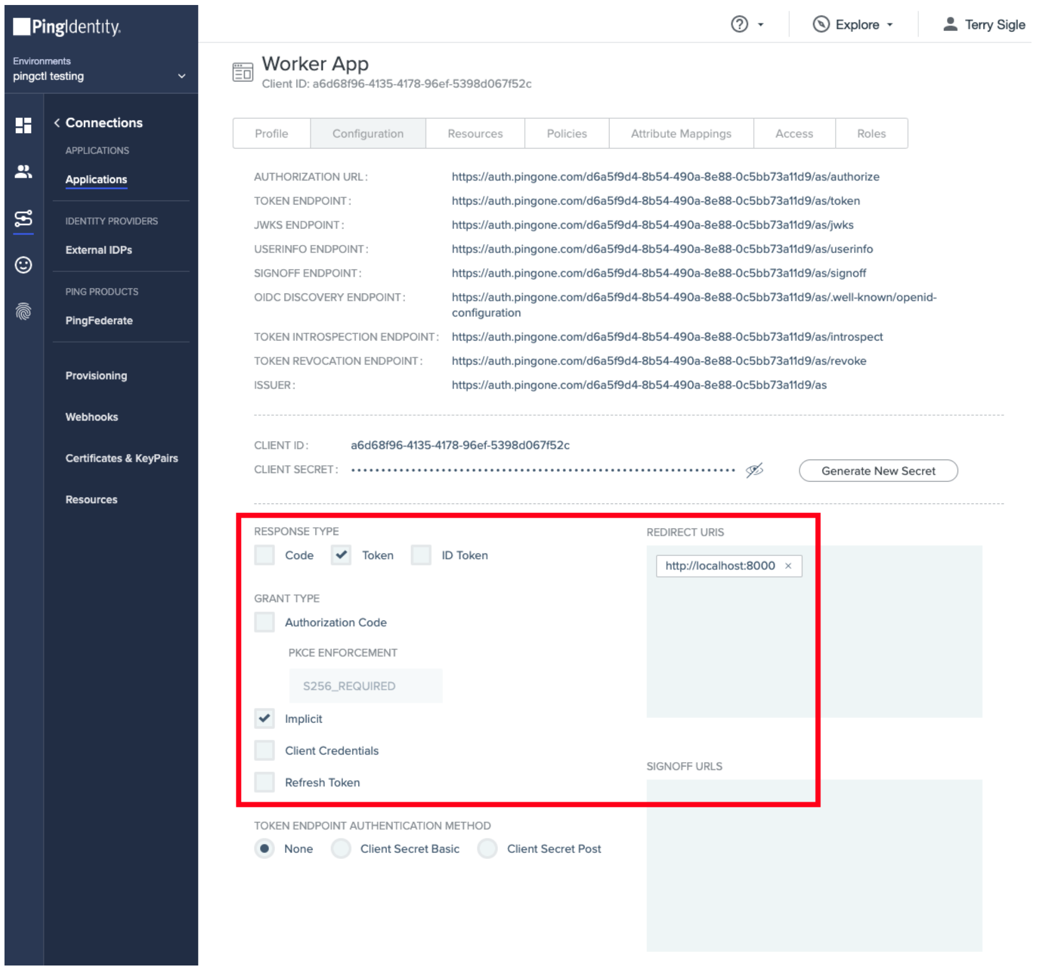
TOKEN ENDPOINT AUTHENTICATION METHOD
● None ○ Client Secret Basic ○ Client Secret Post

REDIRECT URIS
http://localhost:8000

SIGNOFF URLS

Implicit Flow Settings

The following image shows an example of a Worker App setup for Implicit Flow:



Worker App
Client ID: a6d68f96-4135-4178-96ef-5398d067f52c

AUTHORIZATION URL:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/authorize
TOKEN ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/token
JWKS ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/jwks
USERINFO ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/userinfo
SIGNOFF ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/signoff
OIDC DISCOVERY ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/.well-known/openid-configuration
TOKEN INTROSPECTION ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/introspect
TOKEN REVOCATION ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/revoke
ISSUER:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as

CLIENT ID: a6d68f96-4135-4178-96ef-5398d067f52c
CLIENT SECRET: ••

RESPONSE TYPE
Code [✓ Token] ID Token

GRANT TYPE
Authorization Code
PKCE ENFORCEMENT
S256_REQUIRED
[✓ Implicit]
Client Credentials
Refresh Token

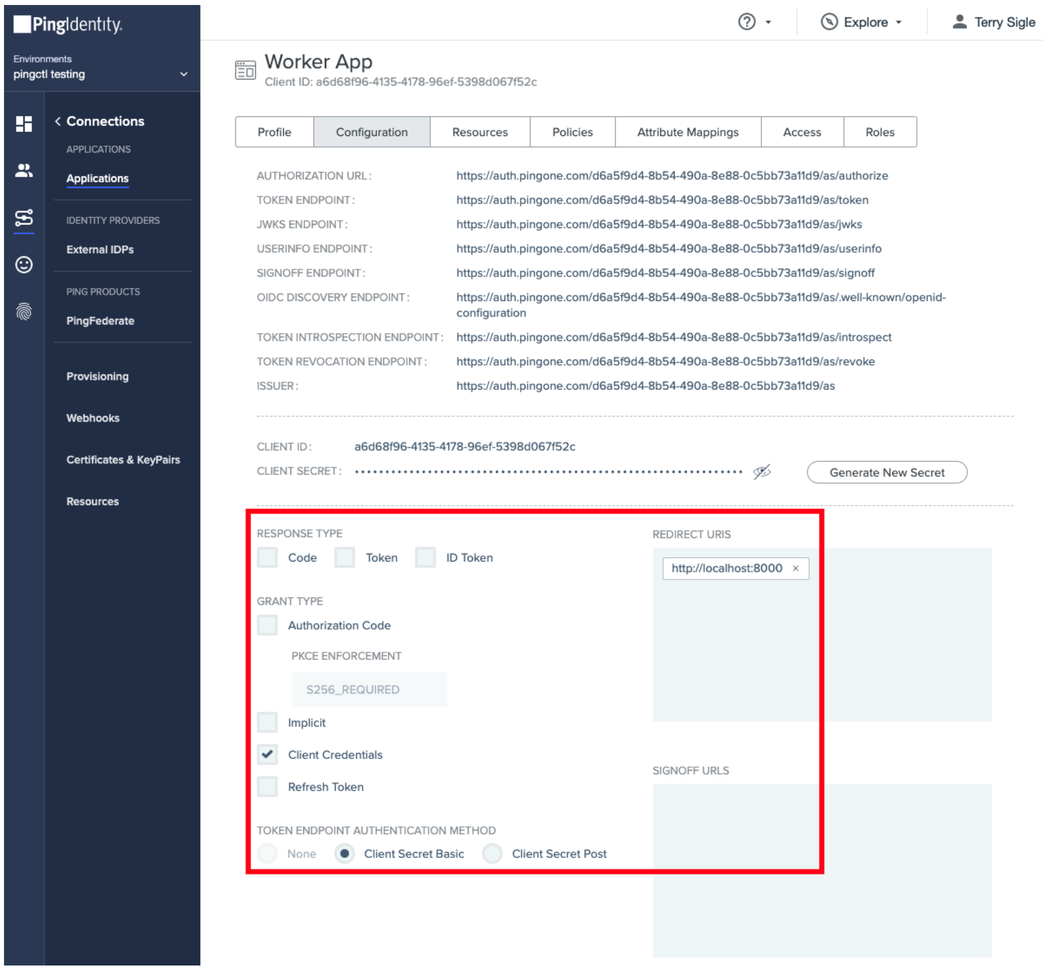
REDIRECT URIS
http://localhost:8000 ×

SIGNOFF URLS

TOKEN ENDPOINT AUTHENTICATION METHOD
[● None] Client Secret Basic Client Secret Post

Client Credentials Flow Settings

The following image shows an example of a Worker App setup for Client Credentials Flow:



Worker App
Client ID: a6d68f96-4135-4178-96ef-5398d067f52c

AUTHORIZATION URL:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/authorize
TOKEN ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/token
JWKS ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/jwks
USERINFO ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/userinfo
SIGNOFF ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/signoff
OIDC DISCOVERY ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/.well-known/openid-configuration
TOKEN INTROSPECTION ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/introspect
TOKEN REVOCATION ENDPOINT:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as/revoke
ISSUER:	https://auth.pingone.com/d6a5f9d4-8b54-490a-8e88-0c5bb73a11d9/as

CLIENT ID: a6d68f96-4135-4178-96ef-5398d067f52c
CLIENT SECRET: ••

RESPONSE TYPE
Code Token ID Token

GRANT TYPE
Authorization Code
PKCE ENFORCEMENT
S256_REQUIRED
Implicit
[✓ Client Credentials]
Refresh Token

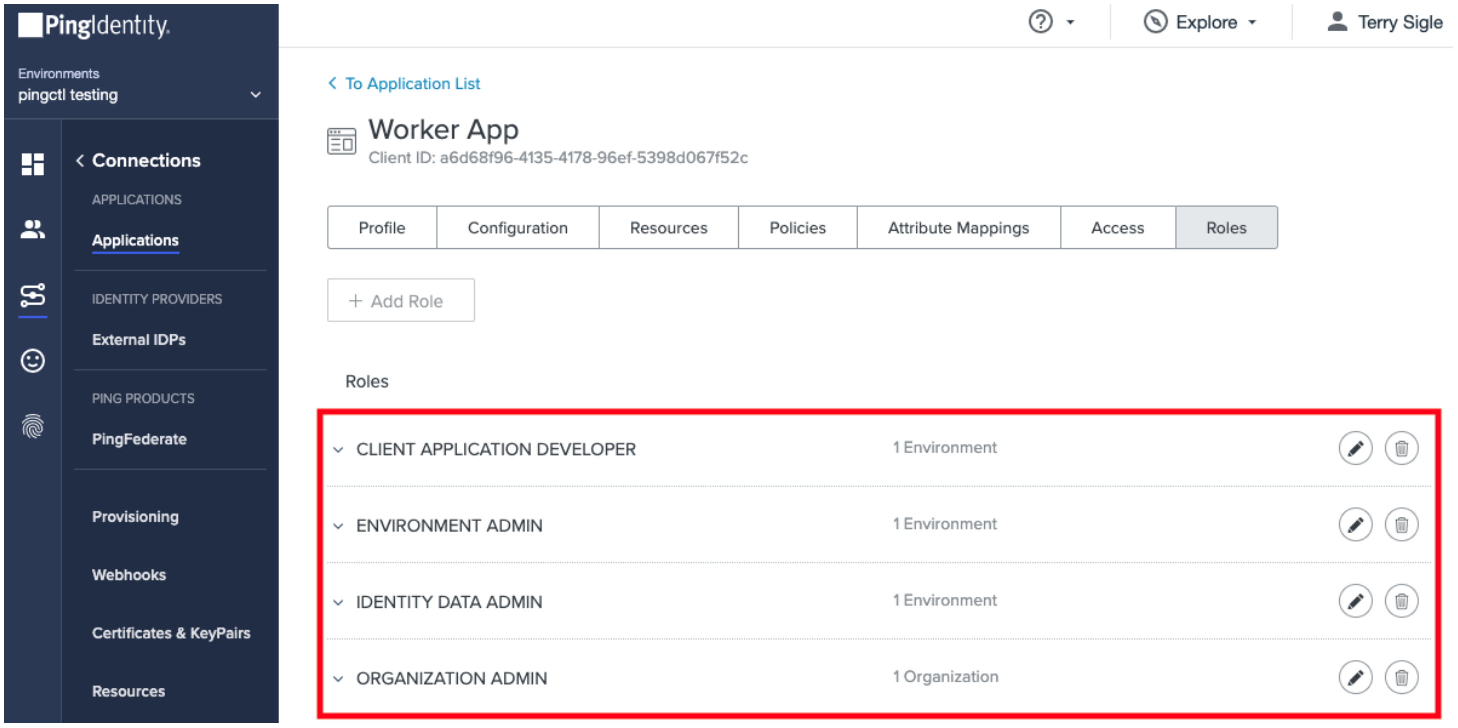
TOKEN ENDPOINT AUTHENTICATION METHOD
None [✓ Client Secret Basic] Client Secret Post

REDIRECT URIS
http://localhost:8000

SIGNOFF URLS

Worker App Roles Settings

The following image shows an example of the minimum roles required. Typically, these are set up by default.



Worker App
Client ID: a6d68f96-4135-4178-96ef-5398d067f52c

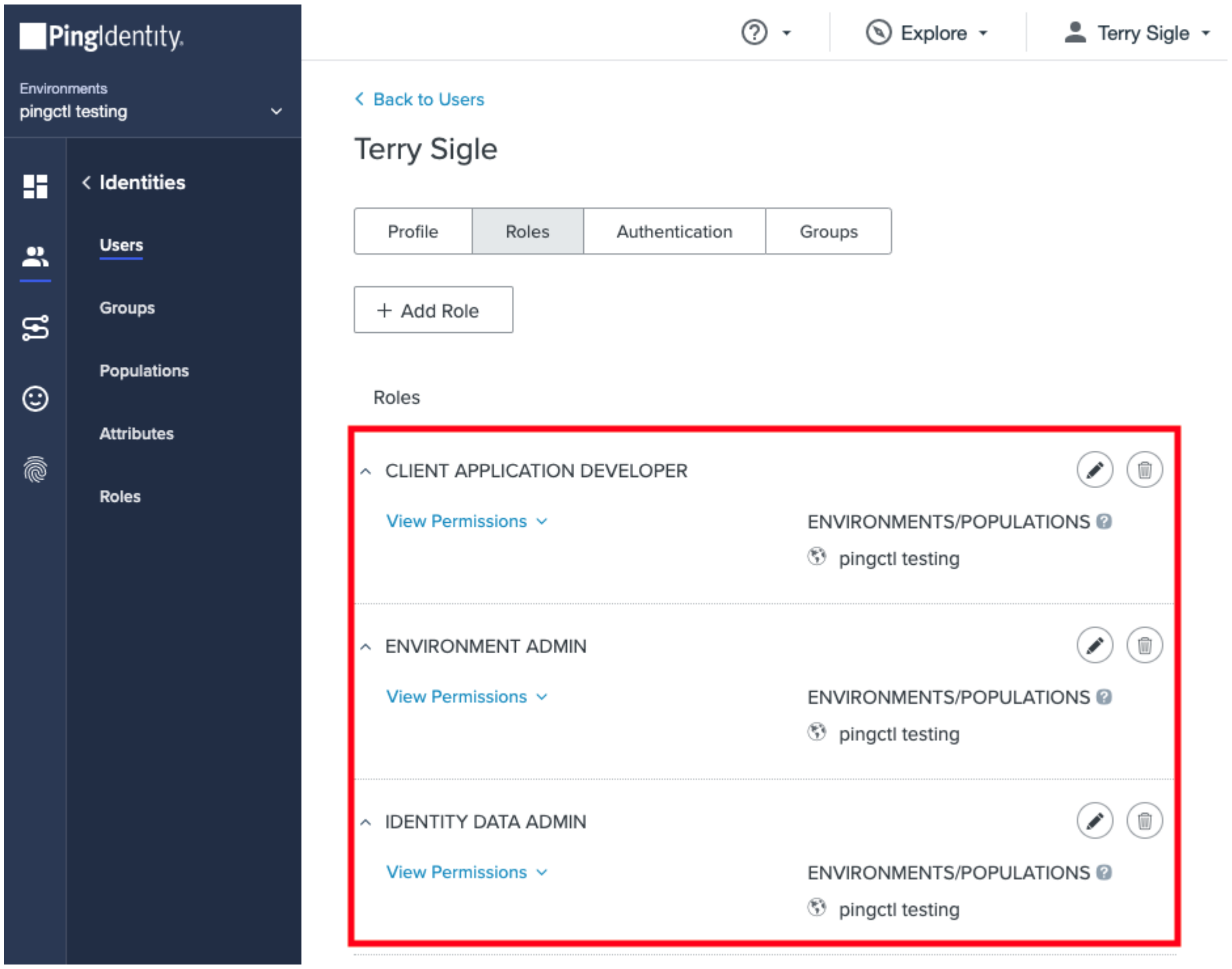
Roles

CLIENT APPLICATION DEVELOPER	1 Environment
ENVIRONMENT ADMIN	1 Environment
IDENTITY DATA ADMIN	1 Environment
ORGANIZATION ADMIN	1 Organization

PingOne User Config

When using Authorization Code or Implicit Flows, you must sign on with an Administrative user to use the Worker App.

It is important to add the proper administrative roles to the user. The following image shows an example of this configuration:



Terry Sigle

Roles

CLIENT APPLICATION DEVELOPER
ENVIRONMENTS/POPULATIONS
pingctl testing

ENVIRONMENT ADMIN
ENVIRONMENTS/POPULATIONS
pingctl testing

IDENTITY DATA ADMIN
ENVIRONMENTS/POPULATIONS
pingctl testing

Videos

You can also find all of these DevOps videos on the Ping Identity website.

Foundational Information

- Getting Started Walkthrough

  Introduction on how to prepare a local Kubernetes environment to deploy Ping products. The video will also show how to use our Helm Charts to deploy our products and how to access product consoles through the ingress controller.

- CICD Demonstration

  This 30-minute video provides a high-level overview of Continuous Integration/Continous Deployment (CICD) principles. Included is a demonstration of a fully operational CICD pipeline using Gitea, Jenkins and Kubernetes. The viewer can obtain the code used from Github to run the demonstration themselves on a local Docker Desktop installation with Kubernetes enabled.

- CICD Reference Pipeline

  This 30-minute video builds on the CICD Demonstration. Watch as a branch of a Github server profile repository uses pipelines to update a stack of Ping products at each code push. Merging the branch to prod then updates the production environment accordingly. The video and repository can serve as a launching point for using your own processes and tools for a similar experience.

Product Docker Images

- Upgrade PingCentral

  This video demonstrates the upgrade of PingCentral in a containerized environment.

- Build a Ping product image locally

  This video demonstrates building a custom Ping product image using the Ping-provided build scripts and a downloaded product .zip file.

- Ping product Docker image exploration

  After viewing this video, you will have a deeper understanding of the structure of a Ping product Docker image filesystem. In addition, you will be shown ways of injecting configuration and customization into the image during the Kubernetes/Helm launch process along with an overview of the hook script functionality used by our images. This video is intended for users with some familiarity with Ping product images.

- Ping product image hook scripts

  This video, intended for users with some familiarity with Ping product images, covers the hook script functionality included in all Ping product Docker images for managing product lifecycle in a containerized environment.

Platform

- Deploy a Local Openshift Cluster

This 12-minute video demonstrates the process of deploying an Openshift Local cluster to your environment, followed by installing a full stack of Ping products using our Helm charts.

- Splunk Logging Example
This 20-minute video demonstrates the use of a Splunk Universal Forwarder (UF) sidecar to forward logs from PingAccess and PingFederate to a local Splunk instance.

- Robust Local Kubernetes Cluster

This 25-minute video demonstrates creating a 3-node local Kubernetes cluster on VMs, including a load balancer, block storage, ingress and service mesh for testing and development.

Version 2507

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2507 (August 1, 2025)

Note

On August 7, 2025, we released a PingFederate 2507.1 patch image for the 12.3.1 version.

Enhancements

- Redhat UBI9 Minimal 9.6-1751286687 → 9.6-1754000177
- Alpine 3.22.0 → 3.22.1
- Liberica JDK17 17.0.15+10 → 17.0.16+12

Features

- (PDI-2252) Support for correct image tag extension when specified as DOCKER_DEFAULT_PLATFORM environment variable.
- (PDI-2253) Updated pingdataconsole 10.3 to run on tomcat 11. This fixes an issue where the console would return a 404 error when running on tomcat 9.
- (PDI-2255) Image hook script 09-build-motd.sh no longer imports external MOTD content, improving security and reliability.

documentation

- (PDI-2246) Migrated content to the new developer experience portal, with redirects, for both https://helm.pingidentity.com and https://devops.pingidentity.com

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

All Releases

Version 2507

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2507 (August 1, 2025)

Enhancements

- Redhat UBI9 Minimal 9.6-1751286687 → 9.6-1754000177
- Alpine 3.22.0 → 3.22.1
- Liberica JDK17 17.0.15+10 → 17.0.16+12

Features

- (PDI-2252) Support for correct image tag extension when specified as DOCKER_DEFAULT_PLATFORM environment variable.
- (PDI-2253) Updated pingdataconsole 10.3 to run on tomcat 11. This fixes an issue where the console would return a 404 error when running on tomcat 9.
- (PDI-2255) Image hook script 09-build-motd.sh no longer imports external MOTD content, improving security and reliability.

documentation

- (PDI-2246) Migrated content to the new developer experience portal, with redirects, for both https://helm.pingidentity.com and https://devops.pingidentity.com

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2506

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2506 (July 2, 2025)

New Product Releases

- UnboundID LDAP SDK for Java 7.0.2 → 7.0.3
- PingFederate 12.3.0 and EOL 12.1.x
- PingFederate 12.2.3 → 12.2.4
- PingAccess 8.3.0 and EOL 8.1.x
- PingData 10.3.0.0 and EOL 10.1.x.x
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.105 → 9.0.106
- Redhat UBI9 Minimal 9.6-1747218906 → 9.6-1751286687

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2505

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2505 (June 4 2025)

New Product Releases

- PingFederate 12.1.7 → 12.1.8

Enhancements

- Apache Tomcat 9.0.104 → 9.0.105
- RedHat UBI9-Minimal 9.5-1745855087 → 9.6-1747218906
- Alpine 3.21.3 → 3.22.0

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2504

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2504 (May 1 2025)

New Product Releases

- PingAccess 8.2.0 → 8.2.1
- PingAccess 8.1.2 → 8.1.3
- PingCentral 2.3.0 and EOL 2.1.x

Enhancements

- Apache Tomcat 9.0.102 → 9.0.104
- RedHat UBI9-Minimal 9.5-1739420147 → 9.5-1745855087
- Liberica JDK17 17.0.14+10 → 17.0.15+10

Features

- (PDI-2196) Updated the PingDirectory docker image to support new FAIL_ON_DISABLED_BASE_DN and FAIL_ON_UNSUCCESSFUL_REMOVE_DEFUNCT environment variables, which can be enabled by being set to "true".

  The first will fail the container if it is found that replication for the USER_BASE_DN is not enabled after all topology setup is complete.

The second will fail the container if it is found that a previous call to remove-defunct-server did not complete successfully, indicating that the topology is in an unknown state. In these cases, manual intervention will be required to correct the topology.

Documentation

- (PDI-2199) Updated the various examples on devops.pingidentity.com to ensure stability and operations.
- (PDI-2204) Added documentation at https://devops.pingidentity.com/reference/usingCertificates/ for certificate rotation in PingData images when using read-only keystores and truststores.
- (PDI-2207) Updated the example on devops.pingidentity.com for archiving to S3 to be platform-agnostic

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2503

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2503 (Apr 1 2025)

New Product Releases

- PingFederate 12.2.1 → 12.2.2
- PingData products 10.2.0.0 → 10.2.0.1
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.100 → 9.0.102

Features

- (PDI-2195) Limit minimum and maximum Jetty threads for PingFederate

Documentation

- (PDI-2197) Updated the DevOps portal with information regarding Iron Bank images

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2502

Note

For information about product changes, refer to the release notes that can be found on each product’s download page.

DevOps Docker Builds, Version 2502 (Feb 28 2025)

New Product Releases

- PingFederate 12.2.0 → 12.2.1
- PingFederate 12.1.5 → 12.1.6

Enhancements

- Apache Tomcat 9.0.98 → 9.0.100
- Redhat UBI9-minimal 9.5-1736404155 → 9.5-1739420147
- Alpine 3.21.2 → 3.21.3

Features

- (PDI-2148) Update PingData images to use product-default java.properties by default in ARM images
- (PDI-2169) Update PingAccess and PingFederate Image Labels for Publication to Redhat Catalog

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2501

Note

For information about product changes, refer to the release notes that can be found on each product’s download page.

DevOps Docker Builds, Version 2501 (Feb 3 2025)

New Product Releases

- PingFederate 12.1.4 → 12.1.5
- PingData products 10.2.0.0 → 10.2.0.1
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Alpine 3.21.0 → 3.21.2
- Redhat UBI9-Minimal 9.5-1734497536 → 9.5-1736404155
- Liberica JDK17 17.0.13+12 → 17.0.14+10

Documentation

- (PDI-2171) Update https://devops.pingidentity.com for clarity on multi-region with all PingData products
- (PDI-2173) Updated https://devops.pingidentity.com examples to latest releases of software and product

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2412

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2412 (Jan 6 2025)

New Product Releases

- PingAccess 8.2.0 and EOL 8.0.x
- PingCentral 2.2.0 and EOL 2.0.x

- PingData products 10.2.0.0 and EOL 10.0.0.X
    - PingDirectory (Dockerhub)
    - PingDirectory Proxy (Dockerhub)
    - PingDataSync (Dockerhub)
    - PingAuthorize (Dockerhub)
    - PingDataConsole (Dockerhub)
- PingFederate 12.2.0 and EOL 12.0.x

Enhancements

- LDAP SDK 7.0.1 → 7.0.2
- Alpine 3.20.3 → 3.21.0
- RedHat UBI9 Minimal 9.5-1731604394 → 9.5-1734497536
- Apache Tomcat 9.0.97 → 9.0.98

Features

- (PDI-2154) Ping product images are now built with Liberica JDK17 installed, replacing Liberica JDK11.
- (PDI-2151) Package managers are no longer removed from Ping product images.

Resolved Defects

- (PDI-2147) Update PingFederate image default log4j2.xml.subst.default files.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2411

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2411 (Dec 2 2024)

New Product Releases

- PingFederate 12.1.3 → 12.1.4
- PingFederate 12.0.5 → 12.0.6

Enhancements

- Apache Tomcat 9.0.96 → 9.0.97
- Redhat Minimal UBI 9.4-1227 → 9.5-1731604394

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2410

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2410 (Oct 31 2024)

New Product Releases

- PingData products 10.0.0.3 → 10.0.0.4
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.95 → 9.0.96
- Liberica JDK11 11.0.24+9 → 11.0.25+11
- Liberica JDK17 17.0.12+10 → 17.0.13+12

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2409

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2409 (Oct 1 2024)

New Product Releases

- PingFederate 12.1.2 → 12.1.3
- PingAccess 8.1.1 → 8.1.2
- PingAccess 8.0.4 → 8.0.5
- PingData products 10.1.0.0 → 10.1.0.2
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.93 → 9.0.95
- Alpine 3.20.2 → 3.20.3

Features

- (PDI-2036) Add future PingFederate 12.2 Administrator Role DATA_COLLECTION_ADMINISTRATOR to 83-configure-admin.sh hook script.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2408

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2408 (Sep 4 2024)

New Product Releases

- PingFederate 12.1.1 → 12.1.2
- PingFederate 12.0.4 → 12.0.5

Enhancements

- Apache Tomcat 9.0.91 → 9.0.93
- Redhat UBI9-minimal 9.4-1194 → 9.4-1227

Documentation

- (PDI-1923) Fix helm examples for Mysql

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2407

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2407 (Aug 5 2024)

New Product Releases

- PingFederate 12.1.0 → 12.1.1
- PingFederate 12.0.3 → 12.0.4
- PingAccess 8.1.0 → 8.1.1
- PingAccess 8.0.3 → 8.0.4
- PingData products 10.0.0.2 → 10.0.0.3
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.90 → 9.0.91
- Redhat UBI9-minimal 9.4-1134 → 9.4-1194
- Alpine 3.20.1 → 3.20.2
- Liberica JDK11 11.0.23+12 → 11.0.24+9
- Liberica JDK17 17.0.11+12 → 17.0.12+10

Documentation

- (PDI-1885) Add notification of deprecation of PingIntelligence Docker images

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2406

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2406 (Jul 2 2024)

New Product Releases

- LDAPSDK 7.0.0 → 7.0.1
- PingFederate 12.1.0
- PingFederate EOL 11.3.X
- PingAccess 8.1.0
- PingAccess EOL 7.3.X
- PingCentral 2.1.0
- PingCentral EOL 1.14.X
- PingData products 10.1.0.0
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)

    - PingAuthorize (Dockerhub)
    - PingDataConsole (Dockerhub)
- PingData EOL 9.3.X.X

Enhancements

- Apache Tomcat 9.0.89 → 9.0.90
- Redhat UBI9 Minimal 9.4-949.1716471857 → 9.4-1134
- Alpine 3.20.0 → 3.20.1

Bug Fixes

- (PDI-1843) Fixed issue where '.pre' and '.post' hooks could not be used for the '81-after-start-process' hook for PingAccess and PingFederate
- (PDI-1867) Updated build-jvm.sh script to use curl rather than wget to avoid an IPV6 issue with some OS shims

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2405

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2405 (Jun 4 2024)

New Product Releases

- PingFederate 12.0.2 → 12.0.3
- PingFederate 11.3.6 → 11.3.7
- PingAccess 8.0.2 → 8.0.3
- PingAccess 7.3.3 → 7.3.4

Enhancements

- Apache Tomcat 9.0.88 → 9.0.89
- Redhat UBI9-minimal 9.3-1552 → 9.4-949.1716471857
- Alpine 3.19.1 → 3.20.0
- Liberica JDK11 11.0.23+10 → 11.0.23+12

- Liberica JDK17 17.0.11+10 → 17.0.11+12

Features

- (PDI-1673) This sprint release adds RHEL UBI9 minimal images for all supported PingAccess and PingFederate versions.

Documentation

- (PDI-1851) Update the Openshift Local documentation in the portal
- (PDI-1854) Updated examples to align with graphics on PF Clustering documentation of the portal.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2404

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2404 (May 1 2024)

New Product Releases

- PingAccess 8.0.1 → 8.0.2
- PingFederate 12.0.1 → 12.0.2
- PingFederate 11.3.5 → 11.3.6
- PingCentral 2.0.1 → 2.0.2

Enhancements

- Apache Tomcat 9.0.87 → 9.0.88
- Liberica JDK17 17.0.10+13 → 17.0.11+10
- Liberica JDK11 11.0.22+12 → 11.0.23+10

Documentation

- (PDI-1634) Update the monitoring implementation on K8s in DevOps portal

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2403

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2403 (Mar 29 2024)

New Product Releases

- PingAccess 7.3.2 → 7.3.3
- PingAccess 8.0.0 → 8.0.1
- PingData products 10.0.0.1 -> 10.0.0.2
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.86 → 9.0.87
- LDAPSDK 6.0.11 -> 7.0.0

Resolved Defects

- (PDI-1505) Fixed an issue where environment variables pulled in from Vault secrets were not available to the server process

Documentation

- (PDI-1475) Remove example for setting up Prometheus in GitHub server profile

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2402

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2402 (Feb 29 2024)

New Product Releases

- PingFederate 12.0.0 → 12.0.1
- PingFederate 11.3.4 → 11.3.5
    - PingData products 10.0.0.0 → 10.0.0.1
        - PingDirectory (Dockerhub)
        - PingDirectory Proxy (Dockerhub)
        - PingDataSync (Dockerhub)
        - PingAuthorize (Dockerhub)
        - PingDataConsole (Dockerhub)
- PingData products 9.3.0.4 → 9.3.0.5
    - PingDirectory (Dockerhub)
    - PingDirectory Proxy (Dockerhub)
    - PingDataSync (Dockerhub)
    - PingAuthorize (Dockerhub)
    - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.85 → 9.0.86

Resolved Defects

- (PDI-1473) Fixed PingDataSync pods not configuring failover when REMOTE_SERVER_REPLICATION_PORT environment variable was not set. This variable now is not necessary for configuring failover.
- (PDI-1474) Updated PingData restart hook scripts to handle updating the java.properties file. If you need to set any custom values in java.properties, provide the entire file in your server profile at `instance/config/java.properties`. Note that this is outside of the `pd.profile` folder.
- (PDI-1476) Fixed an issue with processing env variable json files in the secrets directory where keys with special characters were not handled.

Documentation

- (PDI-1477) Updated ingress definitions and how-to examples and guides to latest versions and formatting
- (PDI-1478) Removed stale helm repo files as well as updated helm documentation
- (PDI-1587) Remove extraneous reference to components now in baseline server profile.
- (PDI-1589) Update old link on the server profiles how-to page.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2401

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2401 (Jan 29 2024)

New Product Releases

- PingCentral 2.0.0 → 2.0.1
- PingData products 9.3.0.3 → 9.3.0.4
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.84 → 9.0.85
- Apache JMeter 5.6.2 → 5.6.3
- Redhat UBI9-minimal 9.1 → 9.3-1552
- Liberica JDK11 11.0.21+10 → 11.0.22+12
- Liberica JDK17 17.0.9+11 → 17.0.10+13
- Alpine 3.19.0 → 3.19.1

Features

- (PDI-1358) Add support for environment variables in utility sidecar in helm charts
- (PDI-1367) Add global annotation support for PVC definitions
- (PDI-1461) Add support for secondary port to PF image

Documentation

- (PDI-1432) Initial documentation for the PingFederate Terraform provider on https://terraform.pingidentity.com/

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2312

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2312 (Dec 29 2023)

New Product Releases

- PingFederate 11.3.3 → 11.3.4
- PingAccess 8.0.0 and EOL 7.2.x
- PingCentral 2.0.0 and EOL 1.12.x
- PingDelegator 5.0.0 and EOL 4.8.x
- PingData products 10.0.0.0 and EOL 9.2.0.x
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.83 → 9.0.84
- Alpine 3.18.4 → 3.19.0

Documentation

- (PDI-1211) Added video demonstration of PingCentral upgrade process in containers
- (PDI-1359) Added FAQ on supported OS shims and JDK version in images
- (PDI-1361) Updated PingCentral upgrade instructions to latest release

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2311

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2311 (Dec 1 2023)

New Product Releases

- PingCentral 1.14.0 → 1.14.1
- PingFederate 11.3.2 → 11.3.3
- UnboundID LDAP SDK 6.0.10 → 6.0.11
- PingData products 9.3.0.2 -> 9.3.0.3
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)
- PingData products 9.2.0.3 → 9.2.0.4
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.82 → 9.0.83

Features

- (BRASS-1291) Added PF_LDAP_TYPE environment variable to support ldap.properties new field 'ldap.type' in PF 11.3.
- (BRASS-1354) Our RHEL UBI9-minimal images now come with tar commandline utility installed by default.

Resolved Defects

- (BRASS-1331) Fixed issue in our Helm charts where the replicas field was being set when autoscaling was enabled.
- (BRASS-1334) Updated the Helm Chart checks for semantic version for apiVersion to use a capability check rather than verifying specific versions. This fixes issues with comparing prerelease versions.
- (BRASS-1345) Update the PingAuthorizePAP readiness check.

Documentation

- (BRASS-1318) Added FAQ concerning trial licenses versus engaging support.
- (BRASS-1329) Reviewed and updated getting-started examples, creating clusters and using kind or minikube documentation.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2310

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2310 (Nov 1 2023)

New Product Releases

- PingAccess 7.3.1 → 7.3.2
- PingIntelligence 5.1.1 → 5.1.3
- PingData products 9.3.0.1 → 9.3.0.2
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)

- PingAuthorize (Dockerhub)
- PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.80 -> 9.0.82
- Liberica JDK11 11.0.20.1+1 -> 11.0.21+10
- Liberica JDK17 17.0.8.1+1 -> 17.0.9+11

Features

- (BRASS-1303) Added support for PingDirectoryProxy joining a multi-region PingDirectory topology.
  - The PingDirectoryProxy servers can also be across multiple regions. The PingDirectoryProxy servers should not be started until the PingDirectory topology is up and ready.
  - The same variables used to run PingDirectory in multiple regions are used by PingDirectoryProxy. In addition, the following variables must be defined:

```
JOIN_PD_TOPOLOGY: Set to true to join a PingDirectory topology
PINGDIRECTORY_HOSTNAME: Hostname of a PingDirectory server in the topology
PINGDIRECTORY_LDAPS_PORT: LDAPS port of the PingDirectory server to join

Added the LOAD_BALANCING_ALGORITHM_NAMES variable to PingDirectory,
which allows defining what load balancing algorithms to set on the server instance,
separated by semicolons. This variable is only needed when using PingDirectoryProxy
automatic server discovery

This change also removes the requirement for PingDirectory to identify
a topology master server when enabling replication. It will instead
always join via the seed server.
```

Resolved Defects

- (BRASS-1287) Updated helm chart to support removed batch API endpoint designation in Kubernetes 1.25
- (BRASS-1239) Corrected broken links on Ping DockerHub repositories

Documentation

- (BRASS-1217) Added demonstration for Archiving and Retrieving Backups from S3

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2309

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2309 (Oct 3 2023)

New Product Releases

- PingFederate 11.3.1 → 11.3.2
- PingAccess 7.3.0 → 7.3.1. EOL 7.1.1 and 7.0.5
- PingCentral 1.14.0 Release and PingCentral 1.11.0 no longer built. EOL 1.10.0 and 1.9.4
- PingData products 9.2.0.2 → 9.2.0.3
    - PingDirectory (Dockerhub)
    - PingDirectory Proxy (Dockerhub)
    - PingDataSync (Dockerhub)
    - PingAuthorize (Dockerhub)
    - PingDataConsole (Dockerhub)

Enhancements

- Alpine 3.18.3 → 3.18.4
- LDAP SDK 6.0.9 → 6.0.10

Resolved Defects

- (BRASS-1087) Update PD_FORCE_DATA_REIMPORT variable name to match documentation
- (BRASS-286) PingAccess Cluster - handle orphaned engines

Features

- (BRASS-1123) Publish images to openshift certified registry

Documentation

- Added Upgrading PingCentral
- Added helm examples for imagePullSecrets to Deploy Ping DevOps Charts using Helm
- Provided clarity on EFS/EBS support with Ping products running on AWS in Recent portal updates

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2308.1

> **Note**
>
> For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2308.1 (Sept 13 2023)

Resolved Defects

- (BRASS-1197) Fix `/opt` file system permissions defect

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2308

> **Note**
>
> For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2308 (Sep 5 2023)

New Product Releases

- PingFederate EOL 11.0.3
- PingAccess EOL 7.0.4 and 7.1.0
- PingData products 9.2.0.1 → 9.2.0.2 and 9.3.0.0 → 9.3.0.1
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.78 → 9.0.80
- Alpine 3.18.2 → 3.18.3
- Liberica JDK11 11.0.20+8 → 11.0.20.1+1
- Liberica JDK17 17.0.8+7 → 17.0.8.1+1

Resolved Defects

- (BRASS-1087) Update PD_FORCE_DATA_REIMPORT variable name to match documentation

Features

- (BRASS-1122) Added support for imagePullSecrets in Helm charts
- (BRASS-1050) Test image integration on openshift cluster
- (BRASS-853) Add "DEBUG" option for intermediate logging, for now just around curl calls in product containers

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2307.1

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2307.1 (Sept 13 2023)

Resolved Defects

- (BRASS-1197) Fix `/opt` file system permissions defect

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2307

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2307 (Aug 2 2023)

New Product Releases

- PingAccess 7.2.1 → 7.2.2 (Dockerhub)
- PingFederate 11.3.0 → 11.3.1 and 11.2.6 → 11.2.7 (Dockerhub)
- PingDirectory Terraform Provider v0.9.0 released

Enhancements

- Apache Tomcat 9.0.76 → 9.0.78
- Apache JMeter 5.6 → 5.6.2
- Liberica JDK11 11.0.19+7 → 11.0.20+8
- Liberica JDK17 17.0.7+7 → 17.0.8+7

Resolved Defects

- (BRASS-1088) Fix defect in PingDirectory Backup Script
- (BRASS-1121) Fix helm ingress demo YAML annotation
- (BRASS-1124) Fix MOTD retrieval and notification script.
- (BRASS-1143) Update 85-import PA and PF script to have better error messaging upon import failure.

Features

- (BRASS-1115 and BRASS-1116) Added new pingaccess-env-config and pingfederate-env-config server profiles that allow managing PingAccess and PingFederate configuration using environment variables in place of certain properties files.

Documentation

- Updated VPC Peering Configuration to Support Multi-Region AWS EKS Deployments to define VPC peering configuration for Multi-region AWS EKS Deployments.
- Added documentation on Running product containers with a read-only root filesystem¶.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2306

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2306 (June 30 2023)

New Product Releases

- PingFederate 11.3.0 released and PingFederate 11.1.x no longer built 11.2.5 → 11.2.6 EOL 10.3.7 and 11.3.0-Beta (Dockerhub)
- PingAccess 7.3.0 released and PingAccess 7.1.x no longer built EOL 6.3.4 and 7.3.0-Beta (Dockerhub)
- PingCentral 1.12.0 released and PingCentral 1.10.x no longer built EOL 1.9.3 and 1.8.2 (Dockerhub)
- PingData products 9.3.0.0 released and PingData products 9.1.0.x are no longer built EOL 8.3.0.6
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)
- PingDelegator EOL 4.6.0

Enhancements

- Apache JMeter 5.5 → 5.6
- LDAP SDK 6.0.8 → 6.0.9
- Apache Tomcat 9.0.75 → 9.0.76
- Alpine 3.18.0 → 3.18.2

Documentation

- Updated Create a simple local Kubernetes Cluster kind and minikube examples to reflect latest releases and supported backends
- Updated Deploy a robust local Kubernetes Cluster 3-VM Kubernetes example to use Ansible

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2305

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2305 (June 01 2023)

New Product Releases

- PingFederate 11.2.4 → 11.2.5 and 11.1.6 → 11.1.7 EOL 11.0.2 and 10.3.6 (Dockerhub)
- PingAccess EOL 7.0.3 (Dockerhub)
- PingData products 9.2.0.0 → 9.2.0.1 EOL 8.3.0.5
    - PingDirectory (Dockerhub)
    - PingDirectory Proxy (Dockerhub)
    - PingDataSync (Dockerhub)
    - PingAuthorize (Dockerhub)
    - PingDataConsole (Dockerhub)

Enhancements

- Apache Tomcat 9.0.74 → 9.0.75
- Alpine 3.17.3 → 3.18.0

Resolved Defects

- (BRASS-1024) Update PingAccess 51-add-engine.sh hook script to correctly handle engine crashes and restarts
- (BRASS-897) Improve check for the current topology in PingDirectory to fail if the server can't reach itself
- (BRASS-1033) Fix incorrect result code capture in hook scripts

Features

- (BRASS-833) Update images to include product license at /licenses.

Documentation

- Added Ping Identity Support Portal as a support link for our Ping Identity customers
- Added PingDevOps Community as a support link for our Non-Ping Identity customers
- Fix missing CERTIFICATE_NICKNAME row from product image documentation on devops site

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2304

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2304 (May 04 2023)

New Product Releases

- PingAccess EOL 7.0.3 (Dockerhub)
- PingFederate EOL 11.0.2 and 10.3.6 (Dockerhub)
- PingIntelligence EOL 5.1.0 (Dockerhub)

Enhancements

- Liberica JDK11 11.0.18+10 → 11.0.19+7
- Liberica JDK17 17.0.6+10 → 17.0.7+7
- Apache Tomcat 9.0.73 → 9.0.74
- Docker in docker (dind) 18.09 → 23.0.2

Resolved Defects

- (BRASS-893) Update Hook Scripts to Work with Curl 8
- (BRASS-979) Prevent PingDirectory container readiness probe from succeeding before replication configuration was complete on restart

Features

- (BRASS-869) Preserve docker cache when using serial_build.sh to build images locally.

Documentation

- Added video to Deploy a robust local Kubernetes Cluster
- Added Forwarding PingFederate and PingAccess logs to Splunk

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2303

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2303 (April 04 2023)

New Product Releases

- PingAccess 7.2.0 -> 7.2.1 and build 7.3.0-Beta (Dockerhub)
- PingFederate 11.2.3 -> 11.2.4 and build 11.3.0-beta EOL 11.0.1 and 10.3.5 (Dockerhub)
- PingData products 9.1.0.1 -> 9.1.0.2 EOL 9.0.0.0
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)
- PingIdentity LDAPSDK 6.0.7 -> 6.0.8 (Dockerhub)
- Apache Tomcat 9.0.72 -> 9.0.73 (Dockerhub)

Enhancements

- Alpine 3.17.2 -> 3.17.3

Resolved Defects

- (BRASS-650) Updated helm template tests to check array values
- (BRASS-735) Added integration-tests for arm pods of pingauthorize and pingauthorizepap
- (BRASS-762) Fixed broken Kubernetes links on the Ping Identity devops site
- (BRASS-808) Set Proxy topology host and port correctly
- (BRASS-835) Fix replication result code not being recorded correctly in VERBOSE mode
- (BRASS-839) Added Splunk log formatting examples for PingFederate and PingDirectory
- (BRASS-842) Update defaultDomain to pingdemo.example across the board

Features

- (BRASS-806) Add default matchLabels values for topologySpreadConstraints

- (BRASS-645) Enable pf.cluster.bind.address to be set with variable

Documentation

- Added Forward PingFederate and PingAccess logs to Splunk
- Added Upgrading PingAccess
- Added Multi-node local K8s cluster guide
- Added Clarification for 3rd party software support
- Added Deploy a Local Openshift Cluster
- Added Splunk Logging Example
- Updated FAQs page to describe how to get notified of a release

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2302

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2302 (March 01 2023)

New Product Releases

- PingFederate 11.2.2 → 11.2.3 and 11.1.5 → 11.1.6 (Dockerhub)

Enhancements

- (BRASS-288) Pin every product to IPv4 for consistent liveness performance in dual-stack settings
- (BRASS-752) Added a check for indeterminate CRLF characters in sourced files in hook scripts

Resolved Defects

- (BRASS-598) Fix setting custom java.properties for PingData images functionality
- (BRASS-743) Added support for PA clustered admin configuration apart from Ping Helm charts
- (BRASS-744) Hook script update to check if the start-up-deployer was used to configure PA admin and respond accordingly

Features

- (BRASS-544) Support new logging level settings of PF 11.2
- (BRASS-591) Allow PingDirectoryProxy to join PingDirectory's topology with automatic backend discovery
- (BRASS-767) Updated the example PingDirectory backup script to run only on the master of the topology

Documentation

- Added FAQ on removal of direct Docker integration support from Kubernetes
- Added Restoring a Multi Region PingDirectory Deployment on Seed Region Failure
- Replaced references to the configuration of the old devops tool(~/.pingidentity/devops), with the configuration for pingctl tool (~/.pingidentity/config)

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2301

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2301 (February 07 2023)

New Product Releases

- PingFederate 11.2.0 → 11.2.2 and 11.1.3 → 11.1.5 (Dockerhub)

Enhancements

- Alpine 3.17.0 → 3.17.1
- Apache Tomcat 9.0.70 → 9.0.71
- Liberica JDK11 11.0.17+7 → 11.0.18+10
- Liberica JDK17 17.0.5+8 → 17.0.6+10

Resolved Defects

- (BRASS-646) Support PingData multi-region multi-loadbalancer use case
- (BRASS-660) Update helm charts to support K8s 1.25+ in
- (BRASS-674) Fixed error output problem from tags 2211 or 2212 with 2211.1 or 2212.1

- (BRASS-683) Updated PingDataConsole baseline server profile to handle spring.mvc.pathmatch.matching-strategy=ant_path_matcher
- (BRASS-688) Fixed issue where PingDirectoryProxy was unable to start with mounted license
- (BRASS-705) Handle Restarts in PingAccess 81 Hook Script

Documentation

- Added Openshift Local Demonstration
- Added Migrating from privileged images to unprivileged-by-default images
- Added Deploy a local Kubernetes Cluster
- Added Deploy a Local Openshift Cluster
- Added Migrating cluster discovery settings

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2212

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2212 (January 03 2023)

New Product Releases

- PingAccess 7.2.0 releases and PingAccess products 7.0.x are no longer built (Dockerhub)
- PingData products 9.2.0.0 released and PingData products 9.0.0.x are no longer built
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Resolved Defects

- (BRASS-522) Remove PingDataGovernance/PingDataGovernancePAP images and documentation
- (BRASS-604) Update our server profiles for PF to not use tcp.xml.subst

Documentation

- Added Reference CICD Pipeline Demonstration

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2211

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2211 (December 09 2022)

New Product Releases

- PingDirectory 9.1.0.1 and EOL 9.1.0.0 (Dockerhub)
- PingFederate 11.2.0 and EOL 11.0.5 (Dockerhub)
- PingFederate 11.1.2 → 11.1.3 (Dockerhub)

Resolved Defects

- (BRASS-615) Remove DevOps User/Key Requirement from PingFederate Upgrade Script.
- (BRASS-570) Updated the 90-shutdown-sequence.sh hook script to not attempt to remove the seed server from its topology.

Documentation

- Added Ping Product Docker Image Exploration
- Added Hook Script Exploration
- Added CICD Demonstration

Enhancements

- Apache Tomcat 9.0.70 and EOL 9.0.69
- LDAPSDK 6.0.7 and EOL 6.0.6
- Alpine 3.16.2 → 3.17.0
- Apache Tomcat 9.0.68 → 9.0.69
- OpenSSL 1.1.1 → 3.0.7

- Added support for creating a PingAccess cluster using Helm without a server-profile

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2210

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2210 (November 02 2022)

New Product Releases

- PingFederate 11.1.2 (Dockerhub)
- PingAccess 7.1.3, EOL 7.1.2 and 7.2.0-Beta (Dockerhub)

Resolved Defects

- (BRASS-392) - Configure baseline server profile pf-connected-identities for DA configuration

Enhancements

- Added support for the necessary dsreplication commands and arguments to deploy an entry-balanced PingDirectory topology.
- Use the RESTRICTED_BASE_DNS environment variable to define the restricted base DNs for the topology. The multi-region environment variables (such as K8S_CLUSTER and K8S_SEED_CLUSTER) must also be defined when using entry balancing
- com.unboundid.directory.server.MaintainConfigArchive=false has been set in the PingData images

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2209

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2209 (October 04 2022)

New Product Releases

- PingAccess 7.1.2 and 7.0.6. EOL 7.1.1 and 7.0.5 (Dockerhub)
- LdapSDK to 6.0.6 (Dockerhub)

Resolved Defects

- (BRASS-545) - Refined the Operating Patterns document for clarity and grammar: (Deployment Patterns)
- (BRASS-556) - Corrected link to the product support matrix in recent release notes

Enhancements

- Apache-Tomcat 9.0.65 → 9.0.67

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2208

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2208 (September 01 2022)

New Product Releases

- PingAccess 7.1.1 and 7.0.5 (Dockerhub)
- PingFederate 11.0.4 (Dockerhub)

Resolved Defects

- (BRASS-346) Added documentation for setting the PingFederate provisioner node ID
- (BRASS-366) Added .default to any .subst files built into the image that did not have the extension. This prevents .subst files from overwriting the equivalent files when defined in a server profile.
- (BRASS-469) PingFederate Upgrade Documentation has been updated
- (BRASS-484) Fixed layered profile documentation page referring to a profile that no longer exists.
- (BRASS-516) Updated documentation with new recommended process for PingData certificate rotation

Enhancements

- Liberica JDK 11.0.16+8 -> 11.0.16.1+1
- Liberica JDK 17.0.4+8 -> 17.0.4.1+1
- Alpine 3.16.1 -> 3.16.2

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2207

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2207 (August 05 2022)

New Product Releases

- PingFederate 11.1.1 released (Dockerhub)
- PingData products 9.0.0.2 released
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)

Documentation

- Added documentation for using extensions with PingData server profiles
- Split the Helm example pingaccess-cluster.yaml to more accurately represent what the file accomplishes, and added a pingaccess-pingfederate-integration.yaml example.
- (BRASS-486) Added clarification for pingtoolkit initContainer security context to the Openshift documentation in the Helm portal
- (BRASS-497) Added documentation about container logging
- (BRASS-501) Added new Helm examples to replace old docker compose and kustomize examples.
  - pingauthorize-pingdirectory.yaml

- pingdataconsole-pingone-sso.yaml
- pingdatasync-failover.yaml
- pingcentral-external-mysql-db
- pingdirectory-upgrade-partition

- (BRASS-490) The documentation portal was extensively updated for grammar, examples, and validation

Enhancements

- (BRASS-508) PingFederate and PingAccess now use the heartbeat from the API for the startup success check (rather than a fixed timeout). Added a new variable $ADMIN_WAITFOR_TIMEOUT (default 300 seconds) as a backstop against a hung startup process in the 80-post-start.sh script.
- Liberica JDK -> 11.0.16+8
- Alpine -> 3.16.1
- Apache Tomcat in PingDataConsole -> 9.0.65

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2206

Note

For information about product changes, refer to the release notes that can be found on each product's download page.

DevOps Docker Builds, Version 2206 (July 01 2022)

New Product Releases

- PingData products 9.1.0.0 released and PingData products 8.3.x are no longer built
  - PingDirectory (Dockerhub)
  - PingDirectory Proxy (Dockerhub)
  - PingDataSync (Dockerhub)
  - PingAuthorize (Dockerhub)
  - PingDataConsole (Dockerhub)
- PingFederate 11.1.0 released and PF 10.3.x no longer built (Dockerhub)
- PingAccess 7.1.0 released and PA 6.3.x no longer built (Dockerhub)

- PingCentral 1.10.0 released (Dockerhub)
- PingDelegator 4.10.0 released and PDA 4.9.x no longer built (Dockerhub)

Documentation

- How-to documentation for PingIntelligence
- Added page for Openshift configuration

Resolved Defects

- (BRASS-464) Fixed PingAuthorize baseline server profile overwriting certain environment variables from the orchestration layer

Enhancements

- Liberica JDK → 11.0.15.1+2
- Apache Tomcat in PingDataConsole → 9.0.64
- Apache JMeter → 5.5

Features

- (BRASS-440) Kubernetes deployment script of Ping Intelligence

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2205

DevOps Docker Builds, Version 2205 (June 02 2022)

New Product Releases

- PingAccess
  - PingAccess 7.0.4 is now available on Dockerhub.
- PingData products
  - Updated all PingData products to build 8.3.0.6
    - PingDirectory (Dockerhub)
    - PingDirectory Proxy (Dockerhub)
    - PingDataSync (Dockerhub)
    - PingAuthorize (Dockerhub)

        - PingDataConsole (Dockerhub)

- PingFederate
    - PingFederate 11.0.3 and 10.3.7 are now available on Dockerhub.
- PingIdentity LDAPSDK
    - PingIdentity LDAPSDK upgraded to 6.0.5 in Docker image Dockerhub.

Documentation

- Sidecar Example created
    - Page with details and recommendations for using a sidecar, with example
- Migrating root-based deployment documentation updated
    - Refined this page with recommendations of pre-migration steps

Resolved Defects

- (BRASS-402) - Updates to PingAccess and PingFederate
    - Updated the PingAccess and PingFederate builds to generate a run.properties.subst.default file based on the product default run.properties file pulled from /opt/server. This ensures that any other defaults in the product are included in the default run.properties. This change allows for setting the PingAccess operational mode through the OPERATIONAL_MODE environment variable, without requiring a server profile.
- (BRASS-428) - PingAccess FIPS mode properties issues
    - Updated the default FIPS properties file for PingAccess to use the correct filename and the correct property name to enable FIPS mode.

Enhancements

- Docker Images
    - Apache Tomcat to Version 9.0.63
    - Alpine to version 3.16.0

Features

- (BRASS-434) Support Null SecurityContext in Helm Charts for Openshift
    - Enables the helm charts to generate with workload.securityContext as null, permitting the Openshift environment to generate the security context properly.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2204

DevOps Docker Builds, Version 2204 (May 05 2022)

New Product Releases

- PingAccess
  - PingAccess 6.3.4 is now available on Dockerhub.
- PingIntelligence
  - PingIntelligence 5.1.1 is now available on Dockerhub.
- pingctl
  - pingctl 1.0.5 released Release Notes

Documentation

- Environment Considerations added
  - Workaround for NFS and PingData
- Getting Started Examples updated
  - Getting-started docker-compose examples now use default registry and image tag values
  - docker-compose examples now use pingctl's config file over deprecated ping-devops's config file
- Pingctl configuration updated
  - Instructions to export environment variables if wanted

Resolved Defects

- (BRASS-389) - Fix an issue with the getSemanticImageVersion function that was causing "bad number" errors during hook scripts.
- (BRASS-397) - Updated PingDirectory and PingFederate server profiles to remove hard-coded instances of dc=example,dc=com and replace them with the USER_BASE_DN environment variable.

Enhancements

- Docker Images
  - Apache Tomcat to Version 9.0.62
  - Alpine to version 3.15.4
  - Liberica JDK to 11.0.15+10

Features

- (BRASS-393) PingDirectory supports multiple base DNS for replication
  - Updated the PingDirectory image to support enabling and initializing replication for multiple base DNs with the REPLICATION_BASE_DNS variable, in addition to the USER_BASE_DN variable. Multiple DNs can be delimited with a ';' character.
    - For example: REPLICATION_BASE_DNS=dc=additional,dc=com;dc=another,dc=com
- (BRASS-394) PingDataSync supports persistent volume for restarts
  - Updated the PingDataSync restart logic to include use of the manage-profile replace-profile command to support running PingDataSync with a persistent volume. This allows for updating the PingDataSync configuration on container restart, without requiring deploying a fresh container or volume.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2203

DevOps Docker Builds, Version 2203 (April 01 2022)

New Product Releases

- PingFederate
  - PingFederate 11.0.2 is now available on Dockerhub.
- Documentation
  - Helm and Kustomize Documents added DevOps Getting Started GitHub Repo has been updated
    - 20-kubernetes directory has been renamed to 20-kustomize, as well as kustomize examples reduced
    - 30-helm directory added with examples included

Resolved Defects

- (BRASS-313) - Update docs and pingdataconsole server profile(s) for breaking application.yaml change between 8.3 and 9.0

Helm Chart Releases

- Release 0.9.0
- Release 0.8.9
- Release 0.8.8
- Release 0.8.7

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2202

DevOps Docker Builds, Version 2202 (March 03 2022)

New Product Releases

- PingCentral
    - PingCentral 1.9.3 is now available on Dockerhub.

Enhancements

- Docker Images
    - Apache Tomcat to Version 9.0.59
    - Liberica JDK to 11.0.14.1+1
- Documentation
    - DevOps Getting Started GitHub Repo has been updated
        - Complex Docker Compose examples deprecated and removed
- Helm Charts Release 0.8.6
    - Issues Resolved
        - Update default global.image.tag to 2202

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2201

DevOps Docker Builds, Version 2201 (February 07 2022)

New Product Releases

- PingFederate
    - PingFederate 11.0.1 is now available on Dockerhub.
- PingAccess
    - PingAccess 6.3.3 is now available on Dockerhub.

- PingDirectory
  - PingDirectory 8.3.0.5 is now available on Dockerhub.
- PingDataConsole
  - PingDataConsole 8.3.0.5 is now available on Dockerhub.
- PingDirectoryProxy
  - PingDirectoryProxy 8.3.0.5 is now available on Dockerhub.
- PingDataSync
  - PingDataSync 8.3.0.5 is now available on Dockerhub.
- PingAuthorize
  - PingAuthorize 8.3.0.5 is now available on Dockerhub.
- PingAuthorizePAP
  - PingAuthorizePAP 8.3.0.5 is now available on Dockerhub.

Enhancements

- Docker Images
  - Apache Tomcat to Version 9.0.58
  - Liberica JDK to 11.0.14+9
- Helm Charts Release 0.8.5
  - Features
    - PingCentral now supported. Example values application found here
  - Issues Resolved
    - Issue #119 Workload template not honoring false values from values.yaml. Previously, false did not overwrite true in the Ping Identity Helm Chart template. This fix in _merge-util.tpl will resolve multiple cases within the Ping Identity Helm Chart.

      ```
      {{- $globalValues := deepCopy $top.Values.global -}}
      {{- $prodValues := deepCopy (index $top.Values $prodName) -}}
      {{- $mergedValues := mergeOverwrite $globalValues $prodValues -}}
      ```

    - Issue #264 Update default global.image.tag to 2201

Resolved Defects

- (BRASS-315) - PingFederate server profiles in getting-started and baseline no longer contain an invalid runtime certificate

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2112

DevOps Docker Builds, Version 2112 (January 05 2022)

New Product Releases

- PingFederate
  - PingFederate 11.0 is now available on Dockerhub.
- PingAccess
  - PingAccess 7.0 is now available on Dockerhub.
- PingDirectory
  - PingDirectory 9.0 is now available on Dockerhub.
- PingDelegator
  - PingDelegator 4.8 is now available on Dockerhub.
- PingDirectoryProxy
  - PingDirectoryProxy 9.0 is now available on Dockerhub.
- PingDataSync
  - PingDataSync 9.0 is now available on Dockerhub.
- PingAuthorize
  - PingAuthorize 9.0 is now available on Dockerhub.
- PingIntelligence
  - PingIntelligence 5.1 is now available on Dockerhub.

Enhancements

- Docker Images
  - Apache Tomcat to Version 9.0.56
  - jmeter image to 5.4.3
  - LDAP SDK to 6.0.3
  - PingAccess to log4j 2.12.2, 2.12.3 patch
  - Set UNBOUNDID_SKIP_START_PRECHECK_NODETACH environment variable to true for PingData

- Helm Charts Release 0.8.3
  - Features
    - Document supported values
  - Issues Resolved
    - Issue #233 Ingress - semverCompare now retrieves correct K8 version for applying the correct apiVersion

      ```
      {{- if semverCompare ">=1.19.x" $top.Capabilities.KubeVersion.Version }}
      ```
    - Issue #254 Update default global.image.tag to 2112

Resolved Defects

- (BRASS-60) - Bulk Config Tool Document has been deprecated. Building a PingFederate profile has taken its place.
- (BRASS-181) - Update to PingDirectory liveness and readiness probes to use timeoutSeconds 5 and failureThreshold 3. Update to PingDirectory readiness probes to use readiness.sh.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2111.1

DevOps Docker Builds, Version 2111.1 (December 16 2021)

New Product Releases

- PingAccess
  - PingAccess 7.0.1 is now available on Dockerhub.
- PingCentral
  - PingCentral 1.8.1 is available on Dockerhub.
- PingFederate
  - PingFederate 11.0.0 is available on Dockerhub.

Enhancements

- Docker Images
  - Applied the log4j2 patch updated zip files to PingAccess and PingFederate per recommendation of the Ping Identity CVE knowledge article.
  - The applied patches are available on the Ping Identity CVE knowledge article.
  - All images tagged with the sprint 2111.1 do not contain the Log4j2 vulnerability CVE-2021-44228.

    - Purged all DockerHub images vulnerable to the Log4j2 vulnerability CVE-2021-44228. This is to ensure all PingIdentity images published do not have the Log4j2 vulnerabilities.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2111

DevOps Docker Builds, Version 2111 (December 06 2021)

New Product Releases

- PingFederate
    - PingFederate 10.3.4 is available on Dockerhub.
- PingAccess
    - PingAccess 6.3.2 is now available on Dockerhub.

Enhancements

- Tools
    - The pingctl tool is now POSIX compliant in preparation for the pingdevops tool's deprecation.
- Docker Images
    - PingDirectory, PingDirectoryProxy, and PingDataSync will now start without a server profile.
    - Update JDK to 11.0.13+8.
    - Azul JVM has been deprecated in Favor of Liberica JVM.
    - PingData images are now killed with a TERM signal.
    - Update Apache Tomcat to Version 9.0.55.
    - Update Alpine to 3.15 and UBI8 to 8.5.
    - PingFederate PF_LDAP_USERNAME and PF_LDAP_PASSWORD variables are no longer required by default.
- Helm Charts
    - View the detailed release notes for Ping's Helm Charts here.
        - Release 0.7.9
            - Support for HPA Scaling Behavior
            - Support for shareProcessNamespace in pod spec

- Helm test image pull policy no longer hard-coded in helm-charts/charts/ping-devops/templates/pinglib/_tests/tpl
- Cluster service for pingaccess-admin

Resolved Defects

- (BRASS-80) - 07-apply-server-profile hook now handles PingDirectory restart correctly.
- (BRASS-172) - Default values have been added for PF_LDAP_USERNAME and PF_LDAP_PASSWORD to work around startup errors for PingFederate images.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2110

DevOps Docker Builds, Version 2110 (November 01 2021)

New Features

- PingFederate
  - PingFederate 10.3.3 and 10.2.7 are now available on Dockerhub.
- PingDirectory
  - PingDirectory 8.3.0.3 is now available on Dockerhub.
- PingAuthorize
  - PingAuthorize 8.3.0.3 is now available on Dockerhub.
- PingCentral
  - PingCentral 1.9 is now available on Dockerhub.
- UnboundID LDAP SDK
  - UnboundID LDAP SDK tool set 6.0.2 is now available on Dockerhub.

Enhancements

- Documentation
  - Improved documenation around certificate rotation for PingDirectory.
  - Update DevOps support policy statement.
- Docker Images
  - Images that include Apache Tomcat have been updated to 9.0.54.

- Startup time for PingDirectory has been improved.
- PF_LDAP_USERNAME and PF_LDAP_PASSWORD variables are now required with PingFederate to promote best security practices.

- Helm Charts
    - View the detailed release notes for Ping's Helm Charts here
        - Release 0.7.7 - Update default security context group id to root.
        - Release 0.7.8 - Server profile updates, generate master password for Ping services.

Resolved Defects

- (BRASS-72) - Resolved issue in which numbers were not rendered correctly in some cases in public docs.
- (BRASS-71) - Resolved issue in which PingDirectory seed name is not rendered correctly.

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2109

DevOps Docker Builds, Version 2109 (October 06 2021)

Warning

PingFederate deployments prior to sprint release 2108 (Aug 27th, 2021) may be at risk. Please visit here for details on impacted and patched versions.

New Features

- PingFederate
    - PingFederate 10.3.2 and 11.0 Beta are now available on Dockerhub
- PingAccess
    - PingAccess 6.3.1 and 7.0 Beta are now available on Dockerhub
- PingDirectory
    - PingDirectory 8.3.0.2, 8.2.0.6, and 9.0 EA are now available on Dockerhub
- PingAuthorize
    - PingAuthorize 8.3.0.2, 8.2.0.6, and 9.0 EA are now available on Dockerhub
- PingIntelligence
    - PingIntelligence 5.0.1 is now available on Dockerhub

Enhancements

- Documentation
  - Improved Introduction to Image/Container anatomy
- Docker Images
  - JDK Liberica 11.0.12+7 is now supported
  - Images now include a startup probe script /opt/startup.sh
- Helm Charts
  - View the detailed release notes for Ping's Helm Charts here
    - Release 0.7.6 - Support for scheduler name on pods
- Kubernetes
  - PingDirectory now waits for its pod DNS hostname to match expected K8 pod IP

Resolved Defects

- (GDO-896) - Resolved issue where PingDirectory failed to pick up the product license during deployment
- (GDO-989) - Resolved issue in which PingDirectory seed failure in multi-region topology causes a replication island

Supported Product Releases

- See the Product Version, Image Release Matrix for currently supported image and product versions.

Version 2108

DevOps Docker Builds, Version 2108 (August 27 2021)

New Features

- PingFederate
  - PingFederate 10.3.1 and 10.2.5 are now available on Dockerhub
- PingAccess
  - PingAccess 6.3.1 is now available on Dockerhub

Enhancements

- Documentation
  - Our DevOps documentation now supports both light and dark modes. Toggle between the two by clicking the icon in the top navigation bar.

- Docker Images
  - Upgraded the Image OS from Alpine 3.13 to 3.14
- Helm Charts
  - View the detailed release notes for Ping's Helm Charts here
    - Release 0.7.0 - ServiceAccount/Role/RoleBinding for testFramework
    - Release 0.7.1 - Public hostname/ports
    - Release 0.7.2 - PingFederate PF_ADMIN_PUBLIC_BASEURL variable
    - Release 0.7.3 - Support full definition of initContainers attributes in testSteps and finalStep
    - Release 0.7.4 - Set initContainer settings from values.yaml instead of hard coded templates

Resolved Defects

- (GDO-945) - Resolved issue where PingCentral was unable to communicate with PingAccess in the docker-compose full-stack example.
- (GDO-872) - Resolved issue in tooling when building images locally (`serial_build.sh`).

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.3.1 6.2.2	6.3.0
PingAuthorize	8.3.0.1	
PingAuthorize PAP	8.3.0.1	
PingCentral	1.8.0 1.7.0	
PingDataConsole	8.3.0.1 8.2.0.5	
PingDataGovernance	8.2.0.5	
PingDataGovernance PAP	8.2.0.5	
PingDataSync	8.3.0.1 8.2.0.5	
PingDelegator	4.6.0 4.4.1	

Product	Active Build	Build EOL
PingDirectory	8.3.0.1 8.2.0.5	
PingDirectoryProxy	8.3.0.1 8.2.0.5	
PingFederate	10.3.1 10.2.5	10.3.0 10.2.4
PingIntelligence	5.0 4.4.1	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2107

DevOps Docker Builds, Version 2107 (August 4 2021)

New Features

- PingFederate
  - Added support for pf.admin.baseurl within baseline Server Profile
- PingAccess
  - PingAccess 6.2.2 is now available on Dockerhub
- PingDirectory
  - PingDirectory 8.3.0.1 is now available on Dockerhub
- PingDirectoryProxy
  - PingDirectoryProxy 8.3.0.1 is now available on Dockerhub
- PingDataSync
  - PingDirectory 8.3.0.1 is now available on Dockerhub
- PingAuthorize
  - PingAuthorize 8.3.0.1 is now available on Dockerhub

Enhancements

- PingDelegator - Baseline now works on both local and Kubernetes environments
- Helm Charts - Release 0.6.8 - Probes & Ingress

Resolved Defects

- (GDO-860) - Resolved issue where the PingAuthorize Policy Editor auto-generated documentation uses wrong ports
- (GDO-907) - Restored functionality for prepending the name of the log file to each log line
- (GDO-887) - All Docker images are now signed

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.3.0 6.2.2	6.2.1
PingAuthorize	8.3.0.1	8.3.0.0
PingAuthorize PAP	8.3.0.1	8.3.0.0
PingCentral	1.8.0 1.7.0	
PingDataConsole	8.3.0.1 8.2.0.5	8.3.0.0
PingDataGovernance	8.2.0.5	
PingDataGovernance PAP	8.2.0.5	
PingDataSync	8.3.0.1 8.2.0.5	8.3.0.0
PingDelegator	4.6.0 4.4.1	
PingDirectory	8.3.0.1 8.2.0.5	8.3.0.0
PingDirectoryProxy	8.3.0.1 8.2.0.5	8.3.0.0
PingFederate	10.3.0 10.2.4	

Product	Active Build	Build EOL
PingIntelligence	5.0 4.4.1	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2106

DevOps Docker Builds, Version 2106 (July 6 2021)

New Features

- ARM-Based Images
  - Ping Identity now offers ARM-based Docker images!
  - These images are currently experimental and are not intended for production deployment
  - View the available tags on Dockerhub
- PingFederate
  - PingFederate 10.3.0 and 10.2.4 are now available on Dockerhub
- PingAccess
  - PingAccess 6.3.0 is now available on Dockerhub
- PingDirectory
  - PingDirectory 8.3.0 and 8.2.0.5 are now available on Dockerhub
- PingAuthorize
  - PingAuthorize 8.3.0 is now available on Dockerhub
- PingCentral
  - PingCentral 1.8 is now available on Dockerhub
- PingDelegator
  - PingDelegator 4.6.0 is now available on Dockerhub
- PingIntelligence (ASE)
  - PingIntelligence 5.0 is now available on Dockerhub

- LDAP SDK
    - LDAP SDK 6.0.0 is now available on Dockerhub

Enhancements

- PingFederate - Allow logging level to be set via an environment variable (PF_LOG_LEVEL) - Added property `pf.admin.baseurl` to run.properties configuration file - Added ability to generate the run.properties and jvm-memory.options files based on supplied environment variables
- HEAP Awareness - All Ping Identity Docker images can now calculate the heap based on the memory allocated to the container
- Java Tools - Added jcmd, jstat, jinfo, jmap, jps, jstack tools to images
- Docker-Compose - Added tmpfs secrets directory to all of the docker-compose examples in the Getting-Started repository

Resolved Defects

- (GDO-657) - Resolved PingDelegator self-signed certificate issue
- (GDO-834) - Resolved issue where PingDataConsole doesn't build correctly when providing a local product.zip file
- (GDO-836) - Resolved issue where PingDirectory restart failed due to startup hook syntax error
- (GDO-885) - Resolved HTTPS/LDAPS port variables in PingAuthorize profiles to support Helm charts

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.3.0 6.2.1	6.3.0-Beta 6.1.4
PingAuthorize	8.3.0.0	
PingAuthorize PAP	8.3.0.0	
PingCentral	1.8.0 1.7.0	1.6.0
PingDataConsole	8.3.0.0 8.2.0.5	8.3.0.0-EA 8.2.0.3 8.1.0.3
PingDataGovernance	8.2.0.5	8.2.0.3 8.1.0.3
PingDataGovernance PAP	8.2.0.5	8.2.0.3 8.1.0.3

Product	Active Build	Build EOL
PingDataSync	8.3.0.0 8.2.0.5	8.3.0.0-EA 8.2.0.3 8.1.0.3
PingDelegator	4.6.0 4.4.1	4.5.0 4.4.1 4.2.1
PingDirectory	8.3.0.0 8.2.0.5	8.3.0.0-EA 8.2.0.3 8.1.0.3
PingDirectoryProxy	8.3.0.0 8.2.0.5	8.3.0.0-EA 8.2.0.3 8.1.0.3
PingFederate	10.3.0 10.2.4	10.3.0-Beta 10.2.3 10.1.5
PingIntelligence	5.0 4.4.1	4.4

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2105

DevOps Docker Builds, Version 2105 (June 3 2021)

New Features

- PingFederate
  - PingFederate 10.2.3 is now available on Dockerhub
- PingDelegator
  - PingDelegator 4.5.0 is now available on Dockerhub

Resolved Defects

- (GDO-813) - Resolved issue where OAuth APIS were broken using baseline server profile and pingfederate:edge

- (GDO-818) - Resolved issue where users were unable to build images locally due to a file permission error
- (GDO-829) - Resolved issue where a `dsconfig` command was unable to run due to a quoting error

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.3.0-Beta 6.2.1 6.1.4	
PingCentral	1.7.0 1.6.0	
PingDataConsole	8.3.0.0-EA 8.2.0.3 8.1.0.3	
PingDataGovernance	8.2.0.3 8.1.0.3	
PingDataGovernance PAP	8.2.0.3 8.1.0.3	
PingDataSync	8.3.0.0-EA 8.2.0.3 8.1.0.3	
PingDelegator	4.5.0 4.4.1 4.2.1	
PingDirectory	8.3.0.0-EA 8.2.0.4 8.2.0.3 8.1.0.3	
PingDirectoryProxy	8.3.0.0-EA 8.2.0.3 8.1.0.3	
PingFederate	10.3.0-Beta 10.2.3 10.1.5	10.2.2
PingIntelligence	4.4	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2104

DevOps Docker Builds, Version 2104 (April 2021)

New Features

- Early Access and Beta Release Docker Images
  - PingAccess 6.3.0-Beta
  - PingAuthorize 8.3.0.0-EA
  - PingAuthorize PAP 8.3.0.0-EA
  - PingDataConsole 8.3.0.0-EA
  - PingDataSync 8.3.0.0-EA
  - PingDirectory 8.3.0.0-EA
  - PingDirectoryProxy 8.3.0.0-EA
  - PingFederate 10.3.0-Beta

Enhancements

- `watch-fs-changes`

  We've updated the `watch-fs-changes` utility to accept command-line parameters to watch additional locations.
- Startup Time Performance

  We've updated the start-server.sh script to improve container start up times for all PingData products.
- Helm Charts for PingDirectoryProxy

  PingDirectoryProxy has been integrated into Ping's Helm Charts.

Resolved Defects

- (GDO-649) - Resolved issue where the provided self-signed certificates for PingDataConsole didn't function in Chrome on MacOS
- (GDO-770) - Resolved issue where PingDataConsole didn't log console messages by default
- (GDO-773) - Resolved issue where the collect-support-data tool couldn't find the required JDK

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.2.1 6.1.4	
PingCentral	1.7.0 1.6.0	
PingDataConsole	8.2.0.3 8.1.0.3	
PingDataGovernance	8.2.0.3 8.1.0.3	
PingDataGovernance PAP	8.2.0.3 8.1.0.3	
PingDataSync	8.2.0.3 8.1.0.3	
PingDelegator	4.4.0	
PingDirectory	8.2.0.3 8.1.0.3	
PingDirectoryProxy	8.2.0.3 8.1.0.3	
PingFederate	10.2.2 10.1.5	
PingIntelligence	4.4	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2103

DevOps Docker Builds, Version 2103 (March 2021)

New Features

- Images run as non-privileged user by default

  Critical: We've greatly improved the security of our images by having them run as a non-privileged user by default. See Migrating from privileged images to unprivileged-by-default images for information about migrating existing deployments.

- Layer simplification

  We've consolidated layers in our images where possible.

Enhancements

- PingFederate
  - The baseline image now uses data.json instead of the former use of the /data folder.
  - New variables have been added to run.properties for controlling provisioning failover and grace period.
  - Versions 10.1.5 and 10.3-Beta are now available.
- PingAccess
  - Versions 6.2.1 and 6.3-Beta are now available.
- PingCentral
  - Versions 1.7.0 is now available.
- PingDirectory
  - The number of layers present in the image has been reduced and simplified.
  - Version 8.2.0.3 is now available.
- PingDataGovernance
  - Version 8.2.0.3 is now available.
- PingDataSync
  - Version 8.2.0.3 is now available.

Resolved Defects

- (GDO-742) - Resolved issue which may cause permissions errors creating files under /run/secrets during PingDirectory setup
- (GDO-746) - Resolved issue in which PingDirectory cannot rejoin its replication topology after restart

- (GDO-749) - Addressed documentation issue in which bulleted lists are not printed correctly

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.2.1 6.1.4	6.2.0
PingCentral	1.7.0 1.6.0	1.5.0
PingDataConsole	8.2.0.3 8.1.0.3	8.2.0.2
PingDataGovernance	8.2.0.3 8.1.0.3	8.2.0.2
PingDataGovernance PAP	8.2.0.3 8.1.0.3	8.2.0.2
PingDataSync	8.2.0.3 8.1.0.3	8.2.0.2
PingDelegator	4.4.0	4.2.1
PingDirectory	8.2.0.3 8.1.0.3	8.2.0.2
PingDirectoryProxy	8.2.0.3 8.1.0.3	8.2.0.2
PingFederate	10.2.2 10.1.5	10.1.4
PingIntelligence	4.4	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2102

DevOps Docker Builds, Version 2102 (February 2021)

Enhancements

- PingFederate
  - Support for creation and loading of certificates for admin.
  - Version 10.2.2 is now available.
- PingAccess
  - Baseline now has clustering support.
  - Version 6.1.4 is now available.
- PingDirectory
  - Improve speed of replace-profile process during PingDirectory restart.
  - Indexes are automatically rebuilt upon server restart.
  - Version 8.2.0.2 is now available.
- PingDataGovernance
  - Helm charts have been added for the PingDataGovernance policy editor.
  - Version 8.2.0.2 is now available.
- PingDataSync
  - Version 8.2.0.2 is now available.

Resolved Defects

- (GDO-382) - Resolved issue where PingDirectory is unable to restart when upgrading 7.3 to 8.1 due to a license error.
- (GDO-543) - Updated "Related Docker Images" documentation in PAP Dockerfile.
- (GDO-672) - Resolved issue with 'manage-profile setup' signaling a dsconfig error.
- (GDO-680) - Resolved issue with PingDirectory set_server_available and set_server_unavailable methods being very.
- (GDO-311) - Updated 05-expand-templates.sh to no longer build data.zip if a *data.zip* directory is found in the profile.

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.2.0 6.1.4	6.1.3
PingCentral	1.6.0 1.5.0	
PingDataConsole	8.2.0.2 8.1.0.3	8.2.0.1
PingDataGovernance	8.2.0.2 8.1.0.3	8.2.0.1
PingDataGovernance PAP	8.2.0.2 8.1.0.3	8.2.0.1
PingDataSync	8.2.0.2 8.1.0.3	8.2.0.1
PingDelegator	4.4.0	4.2.1
PingDirectory	8.2.0.2 8.1.0.3	8.2.0.1
PingDirectoryProxy	8.2.0.2 8.1.0.3	8.2.0.1
PingFederate	10.2.2 10.1.4	10.2.1
PingIntelligence	4.4	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2101

Devops Docker Builds, Version 2101 (January 2021)

Enhancements

- PingFederate
  - Versions 10.2.1 and 10.1.4 are now available.
- PingDirectory
  - Versions 8.2.0.1 and 8.1.0.3 are now available.
  - PingDirectory now delays its readiness state until replication has completed (Kubernetes).
  - Improved container restart time by regenerating java.properties only when changes are made to JVM or JVM options.
- PingDataGovernance
  - Versions 8.2.0.1 and 8.1.0.3 are now available.
- PingDataSync
  - Versions 8.2.0.1 and 8.1.0.3 are now available.
- PingDelegator 4.4.1
  - Version 4.4.1 is now available.
- LDAP SDK
  - Version 5.1.3 is now available.
- Container Secrets
  - Sourcing of secret_envs is now recursive.

Resolved Defects

- (GDO-577) - Resolved issue to suppress environment variables in cn=monitor for PingData products.
- (GDO-658) - Enhanced error messages returned by the evaluation license service.
- (GDO-659) - Resolved issue where evaluation license server used incorrect calculation for checking image expiration.
- (GDO-668) - Resolved issue where remnants of previous server profile remained in place when restarting a container.
- (GDO-674) - Resolved issue where hashing contents of the SECRETS_DIR risked leaving passwords stored insecurely on the container filesystem.

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.2.0 6.1.3	
PingCentral	1.6.0 1.5.0	
PingDataConsole	8.2.0.1 8.1.0.3	8.2.0.0 8.1.0.0
PingDataGovernance	8.2.0.1 8.1.0.3	8.2.0.0 8.1.0.0
PingDataGovernance PAP	8.2.0.1 8.1.0.3	8.2.0.0 8.1.0.0
PingDataSync	8.2.0.1 8.1.0.3	8.2.0.0 8.1.0.0
PingDelegator	4.4.0	4.2.1
PingDirectory	8.2.0.1 8.1.0.3	8.2.0.0 8.1.0.0
PingDirectoryProxy	8.2.0.1 8.1.0.3	8.2.0.0 8.1.0.0
PingFederate	10.2.1 10.1.4	10.2.0 10.1.3
PingIntelligence	4.4	

Note

- **Bolded** product version number is version within 'latest' image tag.
- Build EOL denotes product versions that are no longer built as of this release.

Version 2012

Devops Docker Builds, Version 2012 (December 2020)

New Features

- DevOps Documentation

  We've moved from GitBook to MKDocs to provide a richer DevOps documentation experience.

Enhancements

- PingFederate
  - Version 10.2 now available.
- PingAccess
  - Version 6.2 is now available.
- PingDirectory
  - Version 8.2.0 is now available.
- PingDataGovernance
  - Version 8.2.0 is now available.
- PingDataSync
  - Version 8.2.0 is now available.
- PingCentral
  - Version 1.6.0 is now available.
- LDAP SDK
  - Version 5.1.3 is now available.
  - Updated to latest Tomcat version.
- PingData Console SSO Example
  - We've provided an example of running the Admin Console in Docker with SSO configured.

Resolved Defects

- (GDO-362) Resolved issue where PingDirectory instances become active prior to being fully synchronized.
- (GDO-502) Resolved potential vulnerability by updating Ping Data products to Spring Framework v4.3.29.
- (GDO-544) Resolved issue where PingDataGovernance PAP images' MAX_HEAP_SIZE variable had no effect.
- (GDO-618) Resolved issue where base layer was missing JMX agent.
- (GDO-640) Resolved issue where wait-for command didn't honor timeout when waiting for host:port.

Product Build Matrix

The following table includes product versions and their accompanying Image build status for this release.

Product	Active Build	Build EOL
PingAccess	6.2.0 6.1.3	6.0.4

Product	Active Build	Build EOL
PingCentral	1.6.0 1.5.0	
PingDataConsole	8.2.0.0 8.1.0.0	8.0.0.1
PingDataGovernance	8.2.0.0 8.1.0.0	8.0.0.1
PingDataGovernance PAP	8.2.0.0 8.1.0.0	8.0.0.1
PingDataSync	8.2.0.0 8.1.0.0	8.0.0.1
PingDelegator	4.4.0	4.2.1
PingDirectory	8.2.0.0 8.1.0.0	8.0.0.1
PingDirectoryProxy	8.2.0.0 8.1.0.0	8.0.0.1
PingFederate	10.2.0 10.1.3	10.1.2
PingIntelligence	4.4	4.3

Note

- The **bold** product version number is the version within the 'latest' image tag.
- The build EOL denotes product versions that are no longer built as of this release.

Version 2011

Devops Docker Builds, Version 2011 (November 2020)

New Features

- Internal XRay Scanning

  We've automated the process to scan all Sprint Release Docker Images for CVE's

Enhancements

- PingFederate
  - Version 10.1.3 now available.
  - Parameterized run.properties, ldap.properties and tcp.xml now included in Docker Image.
- Helm Charts
  - We added a number of enhancements to our Helm charts. See the Helm Release Notes for details.
- Misc.
  - Updated EULA check to be case insensitive
  - Add Java back into pingtoolkit Image
  - Updated example docker run commands in Dockerfile documentation
  - Info message when Server Profile URLs are not present

Resolved Defects

- (GDO-549) - Resolved issue where SCIM Swagger test pages don't work in PingDataGovernance Docker Image
- (GDO-567) - Resolved issue where changes made to PingDirectory's java.properties were erased on container restart
- (GDO-599) - Change wait-for localhost to use IP address
- (GDO-604) - Modified simple-sync server profile to work in Kubernetes environment with different service names
- (GDO-606) - Resolved issue where copy of server bits throws errors when running under non-root security context

Version 2010

Devops Docker Builds, Version 2010 (October 2020)

New Features

- PingIdentity Helm Charts

Looking to deploy the PingDevOps stack into your Kubernetes cluster? We've published our Helm Charts to help streamline deployment.

- PingIntelligence (ASE) Docker Image

  PingIntelligence (ASE) is now available on DockerHub! Pull the 4.3 ASE image here.

- PingFederate Bulk API Configuration Management

  We've added tooling and documentation for managing PingFederate configuration using the build API export and import. View the latest documentation here.

Enhancements

- PingFederate
  - Version 10.0.6 now available.
  - Image now includes tcp.xml.subst for cluster parameterization.
  - Updated image to support easier enablement/use of Bouncy Castle FIPS provider with PingFederate.
- PingAccess
  - Version 6.1.3 is now available.
- LDAP SDK
  - Updated to version 5.1.1.
- ping-devops CLI
  - Added functionality to generate K8s license and version secret directly from the evaluation license service.
  - Added ACCEPT_EULA value to K8s devops-secret.

Resolved Defects

- (GDO-411) Resolved issue where access token was logged when using private Git repository.
- (GDO-444) Resolved PingDirectory issue with keystore exception on restart.
- (GDO-491) Removed GPG from base Docker image.
- (GDO-495) Removed gosu from base Docker image.
- (GDO-513) Resolved issue with replication topology list on PingDirectory restart.

Version 2009

Devops Docker Builds, Version 2009 (September 2020)

New Features

- PingDataSync Clustering

  Within PingDataSync 8.2.0.0-EA we've introduced clustering, ensuring your deployment is highly available.

- Certificate Management Usage

  We've added documentation for DevOps Certificate Management.

PingAccess Release

PingAccess 6.1.2 is now available using edge, latest and 2009 image tags.

Product Betas and Release Candidates

Looking to see what the next official product release will contain? Start using the beta and early access builds today.

- PingFederate 10.2.0-Beta
- PingAccess 6.2.0-Beta
- PingDirectory 8.2.0.0-EA
- PingDirectoryProxy 8.2.0.0-EA
- PingDataGovernance 8.2.0.0-EA
- PingDataGovernance 8.2.0.0-EA PAP
- PingDataSync 8.2.0.0-EA

Improvements

- Image Hardening

  We've updated our Image hardening Guide to help secure your production deployments.

Version 2008

DevOps Docker Builds, Version 2008 (August 2020)

New Features

- Secret Management

A number of key enhancements have been made to natively support secret management within our Docker Images. See Using Hashicorp Vault for implementation details.

- DevOps Development Mode

  We've added a 'Continue on Failure' option to all Docker Images. This allows the Container to say alive while any potential issues are being investigated.

- DevOps Program Registration

  Signing up for the Ping DevOps program is now self-service! Simply follow the instructions found in Ping Identity DevOps Registration.

Improvements

- Ping-DevOps Utility

  We've added secret management commands to ping-devops, allowing you to quickly integrate secrets into your deployments.

- Image Restart State

  A number of enhancements have been made to improve the overall restart flow in our Docker Images.

Resolved Defects

- (GDO-352) Resolved restart issue in PingDataGovernance PAP.
- (GDO-392) Resolved issue within PingDelegator when DS_PORT variable was undefined.
- (GDO-395) Resolved issue within PingDirectory restart when Java versions changed.
- (GDO-397) Resolved issue where PingFederate failed to start in Kubernetes using the full-stack example.
- (GDO-404) Resolved issue where some users were unable to log into the PingAccess console using the Image edge tag and Baseline server profile.

Version 2007

DevOps Docker Builds, Version 2007 (July 2020)

New Features

- Signed Docker Images

  All DockerHub Images are now signed and conform to the Docker Content Trust specification.

- Variablize PingAccess Ports

  We've updated the PingAccess start up hooks to allow users to customize application ports.

- PingAccess Upgrade Utility

  The PingAccess upgrade utility is now part of Docker Image.

- Certificate Management

  Add consistency and flexibility with the injection of certs/pins.

- Docker Image Startup Flexibility

  We've added the ability for end users to customize the startup sequence for Docker Images using pre and post hooks. See Using DevOps Hooks for implementation details.

Improvements

- Docker Build Pipeline

  We've made several CI/CD enhancements to improve Image qualification (smoke/integration tests).

Resolved Defects

- (GDO-345) Resolved issue where PingDelegator was using PRIVATE rather than PUBLIC hostnames.
- (GDO-346) Resolved issue regarding the default minimum heap for PingDirectory.
- (GDO-380) Resolved issue within PingAccess Clustering (Admin Console) Kubernetes examples.
- (GDO-371) Resolved issue where PingDelegator wouldn't start using non-privileged user.

Version 2006

DevOps Docker Builds, Version 2006 (June 2020)

New Features

- Docker Compose Volumes

  Applications that create and manage configuration now have mounted volumes in Docker-Compose Examples, ensuring that your configuration changes are persisted across restarted.

- PingAccess Image Enhancements

  We've updated the PingAccess Image to support the new features available in version 6.1.

- Customer Support Data Collection

  Included in this release is the Java diagnostic tool to enable embedded customer support data collection. This tool set includes jstat, jmap and jhat.

New Product Versions

The following new product versions are available using edge, latest and 2006 image tags:

- PingFederate 10.1.0
- PingAccess 6.1.0
- PingDirectory 8.1.0.0

- PingDirectoryProxy 8.1.0.0
- PingDataGovernance 8.1.0.0
- PingDataGovernance 8.1.0.0 PAP
- PingDataSync 8.1.0.0
- PingCentral 1.4.0

Improvements

- Liveness Check

  We've made improvements to PingDirectory's liveness check to better inform dependant services on the status of the Directory service.

- Docker Build Pipeline
  - We've published documentation on how to build a Ping Identity Docker Image using a local zip artifact.
  - We have improved our reference pipeline to allow for the build of a single product.
  - We've made several CI/CD enhancements to improve Image qualification (smoke/integration tests).
- Configuration Substitution

  We've made enhancements to explicitly send the variables to be substituted.

Resolved Defects

- (GDO-218) Resolved an issue where PingDirectory threw an error on manage-profile during setup.
- (GDO-289) Resolved an issue where Alpine based image couldn't install pip3.
- (GDO-329) Resolved an issue where PingCentral docs were not syncing to GitHub.

Version 2005

DevOps Docker Builds, Version 2005 (May 2020)

New Features

- PingDelegator Docker Image

  The PingDelegator Docker image is now available. View on Docker Hub for more information.

  Test drive PingDelegator using the supplied docker-compose file in our Simple-Stack example.

- PingAccess Image Version 6.0.2

  We've updated the PingAccess Image to version 6.0.2.

- PingFederate Version 9.3.3

We've updated the PingFederate 9.3.3 Docker image to include patch 4.

- Docker Builds Pipeline

  We've made a number of CI/CD enhancements to improve Image qualification (smoke/integration tests).

- Image Enhancements

  Improved the `wait-for` command to optionally wait for a path or file to become available.

Resolved Defects

- (GDO-187) Resolved issue where MAX_HEAP_SIZE wasn't applied during container restart.
- (GDO-220) Resolved issue where log message didn't contain log file source name.
- (GDO-238) Resolved issue where ping-devops kubernetes start fails if DNS_ZONE variable not set.
- (GDO-245) Resolved issue where PingAccess didn't exit when configuration import failed.
- (GDO-263) Resolved issue within deploy_docs.sh which had resulted in some documentation to not be pushed to GitHub.
- (GDO-278) Resolved issue with PingAccess clustering Server Profile.

Version 2004

DevOps Docker Builds, Version 2004

New Features

- PingCentral

  The PingCentral Docker image is now available. See the Ping Identity Docker hub.

- Docker Compose

  We've standardized our Docker Compose references.

- Performance

  We've built a performance framework.

- PingFederate version 10.0.2

  We've updated the PingFederate 10 Docker image for the 10.0.2 release.

- The ping-devops utility

  We've added major enhancements to our ping-devops utility. See The ping-devops Utility.

- PingDirectory replication

  We've added support for PingDirectory replication using Docker Compose.

- Variables and scope

  We've added documentation to help with understanding the effective scope of variables. See Variables and Scope.

Resolved Defects

- (GDO-1) Resolved issue where users were unable to override root and admin user passwords (PingDirectory).
- (GDO-129) Removed the console from Ping Data products when the server profile isn't specified.
- (GDO-54) Resolved PingDataGovernance issues within the baseline server profile.
- (GDO-138) Resolved issue regarding PingDataGovernance Policy Administration Point (PAP) launch.
- (GDO-189) Resolved issue with PingAccess heartbeat check.
- (GDO-196) Replaced nslookup with getent due to issues running in Alpine.
- (GDO-180) Resolved issue where extension signature verification may return a false positive.
- (GDO-169) Resolved issues with Ping Data Console by upgrading to Tomcat 9.0.34.
- (GDO-166) Resolved issue with make-ldif template processing.

Version 2003

DevOps Docker Builds, Version 2003

New Features

- PingDirectoryProxy

  The PingDirectoryProxy Docker image is now available. See the Ping Identity Docker Hub

- PingCentral

  The PingCentral Docker image is now available. See the Ping Identity Docker Hub

- Docker Compose Port Mappings

  We now support the Docker Compose best practice of quoting all port mappings.

- Docker Images (Tag: edge)

  We've built a pipeline to support nightly public builds of all Ping Identity Docker images using the `edge` tag.

- PingDirectory

  We've upgraded the PingDirectory Docker image to the current product version 8.0.0.1.

- PingFederate Version 10.1.0

  We've built a beta PingFederate 10.1.0 Docker image.

- PingAccess Version 6.1.0

We've built a beta PingAccess 6.1.0 Docker image.

- Ping Tool Kit

  The Ping Tool Kit Docker image is now available. See Ping Identity Docker Hub. Both `kubectl` and `kustomize` are supported in the image.

- PingFederate Version 9.3

  We've updated the PingFederate 9.3 Docker image to include the latest product patches.

- The ping-devops Utility

  We've added Kubernetes license secret generation, and server profile generation for PingDirectory to the ping-devops utility. See The pingctl Utility.

- A New Hook

  We've added a security start-up hook notifying administrators of keys and secrets found in the server profile.

- DevOps Evaluation License

  We've added retry functionality to attempt getting the DevOps evaluation license if the initial request fails.

- Product Artifacts and Extensions

  We've created operations to retrieve product artifacts and extensions using the DevOps credentials.

- Java 11

  We've migrated all Alpine-based Docker images to Java 11 (Azul).

- PingDirectory Replication Timing

  We've added a profile and reference example to test PingDirectory replication timing. See the pingidentity-devops-getting-started Repo.

- Docker Base Image Security

  We've documented an evaluation of Docker base image security. See Evaluation of Docker Base Image Security.

Resolved Defects

- (GDO-85) Resolved an issue where PingAccess 6.0 loaded a 5.2 license.
- (GDO-87) Resolved an issue where Data Console wasn't allowing users to authenticate (edge tag).
- (GDO-124) Resolved an issue in with pipeline where starting containers using Docker-Compose timed out.
- (GDO-89) Resolved an issue where `*.subst` template files were able to overwrite the server profile configuration.
- (GDO-72) Resolved an issue where `motd.json` did not parse correctly when the product was missing.
- (GDO-88) Resolved an issue where PingFederate profile metadata did not expand `hostname`, breaking OAuth flows.

Changed

- (GDO-97) Removed WebConsole HTTP servlet from the baseline server profile. See the pingidentity-server-profiles repo.

Qualified

- (GDO-42) Verified the ability to run our Docker containers as a non-root user. See Securing the Containers.

FAQs

AWS

What storage option should I use for container volumes on EKS?
Ping recommends the use of EBS volumes for container volumes on EKS. EFS is not supported. For more information, please visit AWS Storage Considerations.

Docker Images

I see Ping product container images hosted in Iron Bank. What are the differences between these images and those found on Docker Hub?
Iron Bank is a container image repository intended to host images for those environments requiring additional security, such as for FedRAMP certification and similar situations. Ping does not build these images, but rather they are created by a Ping partner. These images contain the same product code as found on Docker Hub; however, the OS and JDK used in building the container images are chosen by the partner as per their requirements. You can have full confidence in these images. If you encounter a problem related to an image provided through Iron Bank, you can open a ticket through your normal Ping support channels, indicating that it is an Iron Bank image in question.
What OS and Java versions are included in Ping Docker images?
The operating system (OS) shims used for our images are Alpine and Red Hat UBI. The UBI-based images are intended for Openshift deployments, while Alpine should be used in most other situations. For more information on the choice of Alpine, please visit Supported OS Shim. The Java version currently included in our images is OpenJDK 17 and the distribution used is BellSoft Liberica.
When are new Ping product Docker images released?
Typically, Docker images are released on a monthly basis during the first full week of the month. The images are tagged YYMM, with the month indicating the complete month prior. So, tag "2303", representing the work from March 2023, would be released in early April. As we mature our processes, the frequency and timing of these images will more closely align with product releases.
How can I be informed when new images are available?
You can watch the docker-builds GitHub repository for the Ping Identity product line. Select the "custom" option to receive notification when a release occurs. Releases in the docker-builds repository correspond to the publishing of images in Docker Hub.
What are the latest Ping product versions available as Docker images?
The latest Ping product images are tagged with {RELEASE}-{PRODUCT VERSION}. You can find more information about our latest product images by consulting the Product Version matrix.
Do the images come as product only or combined with an OS layer?
The DevOps program uses #Alpine as its base OS shim for all images. For more information please visit Supported OS Shim.
I have created a custom product installation. If we require a specific image, can that be supplied by Ping?
We do not provide custom images, but you are welcome to build the image locally with your customized bits. For more information, see Build Local Images.
It is important to note using a custom image might affect support options and timing.

Container Operations

How do files move around when the container starts up?
To find out how our files are moved at start up, please visit File Flowchart.
How do I turn off the calls to the Message of the Day (MOTD)?
Set the environment variable in PingBase to: MOTD_URL=""
For more information about the PingBase environment variables, please visit PingBase.

How do I get more verbosity in log outputs?
Set the environment variables in PingBase to: `VERBOSE="true"`
For more information about the PingBase environment variables, please visit PingBase.

Orchestration / Helm / Kubernetes

How can I be informed when a new release of the Helm charts are available?
You can watch the Ping helm-charts GitHub repository. Select the "custom" option to receive notification when a release occurs. As with the product Docker images, the Helm charts are usually updated once a month.
Kubernetes has dropped direct integration support for Docker. Does this change impact Ping product containers?
No. The underlying container runtime has not caused problems with our images. Please let us know if you encounter errors. The CRI-O and containerd runtimes have been tested without any known issues. For more background:
The Kubernetes blog post on Docker removal is here.
An excellent write up of how it looks is on this page.
My container environment is not allowed to make any external calls to services such as Github or Docker Hub. Can I still use Ping Identity containers?
Yes. This practice is common in production scenarios. To use Ping Identity containers in this situation:

1. Use an Existing License.
2. Use an empty remote profile SERVER_PROFILE_URL="". Optionally, you can build your profile into the image, visit Customizing Server Profiles for more information.
3. Turn off license verification with MUTE_LICENSE_VERIFICATION="true".
4. Turn off calls to the Message of the Day (MOTD) with MOTD_URL="".

How do we run the console and engines in a container environment?
The helm chart supports instantiating both consoles and engines. Ingress to the consoles would have to be laid out for UI access.
For more information about the Ping's Helm Charts, please visit Ping Helm.
Can I use Podman instead of Docker?
Yes, just like Docker, you will be able to use Podman for container orchestration.
Why does Ping recommand K8s vs docker?
1. Docker or a pure container solution like ECS by itself is generally not as robust or resilient as a K8s environment. While managed Docker services like ECS provide some of the functionality of Kubernetes, you are locked into that provider and you would have a different experience at Google, Azure, or another cloud provider. Kubernetes, even managed services like EKS, provides more flexibility and portability.
2. It is the model we use for our SaaS offerings, so internal teams at Ping are more familiar with this model.
3. Orchestration among multiple applications and services is native to Kubernetes, a bit of an add-on with Container-only services.
4. Workload management using Kubernetes native objects, such as Horizontal Pod Autoscaling, Node scaling and so on.
5. Management through Infrastructure-as-Code principles using Helm Charts and Values files.

Configuration and Server Profile

How do I customize a container?
There are many ways to customize the container for a Ping product. For example, you can create a customized server profile to save a configuration. To find more ways on how to customize a container, see Customizing Containers.
How do I save product configurations?
In order to save configurations, create a server profile and store in a server profile repository. This repository can be used to pass the configuration into the runtime environment. For help with creating a custom server profile, visit Server Profiles.
Examples of how to get the profile data from the different products:

PingFederate Profile

```
curl -k https://localhost:9999/pf-admin-api/v1/bulk/export?includeExternalResources=false \
-u administrator:2FederateM0re \
-H 'X-XSRF-Header: PingFederate' \
-o data.json
```

PingAccess Profile

```
curl -k https://localhost:9000/pa-admin-api/v3/config/export \
-u administrator:2FederateM0re \
-H "X-XSRF-Header: PingAccess" \
-o data.json
```

PingDirectory Profile

```
kubectl exec -it pingdirectory-0 \ + − manage-profile generate-profile \
--profileRoot /tmp/pd.profile
```

What should be in my server profile?
For more information about what information should be in the server profile consist, please visit Container Anatomy and Profile Structures.
Does my server profile have to be hosted on Github?
No, it can be any Public or Private git repository. You are also able to use a Local Directory as your repository, which is convenient for testing and development.

Product related

How do I access various product consoles?
For a Helm-deployed stack, there are two basic ways you can access the consoles.
1. PortForward to the pod to access with localhost.

```
kubectl port-forward <podName> <containerPort>:<localPort>
```

2. Using Helm, add the ingress definition in the yaml file in order to access the container with a URL. See Creating Ingresses. You must have an ingress controller in your cluster for the ingress to work.
How do I use an existing license?
You can mount the license in the container's `opt/in` directory. Please see using existing licenses for more information.
Where do I get a license? How do I obtain a trial license?
The DevOps team at Ping is not responsible for issuing supported product licenses. We provide a temporary license through the DevOps program. After signing up, you can use the provided credentials to get a short-term license to use in evaluating Ping products running in containers. If you want to use Ping products in production environments, you are required to purchase a valid license. Contact our sales department for more information.
How do I turn off the license verification?
Set the environment variable in PingBase to: MUTE_LICENSE_VERIFICATION="true"
For more information about the PingBase environment variables, please visit PingBase.

Troubleshoot

How do I run Collect-Support-Data in the devops environment?
You will need to modify the liveness probe to always exit 0 and the readiness probe to always exit 1. These changes will give you enough time to capture the CSD without it crashing or trying to serve live traffic.
For more information about the Collect-Support-Data, please visit CSD.
How much overhead memory and CPU is needed to run the Collect-Support-Data tool?
By default, this value is set to 1GB. You would need to add additional memory (1GB to 2GB) to the heap for the server. In terms of CPU, the CSD uses whatever is available.
For more information about the Collect-Support-Data, please visit CSD.

Contributing

Thanks for taking the time to help improve our tools!

How Can I Contribute?

Reporting Bugs

How Do I Submit a Bug Report?

Bugs are tracked as GitHub Issues. You can report a bug by submitting an issue in the Ping Identity DevOps Issue Tracker. To help the maintainers understand and reproduce the problem, please provide information such as:

- A clear and descriptive title.
- A description of what happened and a description of what you expected to happen.
- An example with the exact steps needed to reproduce the problem. If relevant, provide sample code.

Please understand that bug reports are reviewed and prioritized internally, and we might not be able to address all bug reports or provide an estimated time for resolution.

Suggesting Enhancements

As with bugs, requests are tracked as GitHub Issues. You can suggest an enhancement by submitting an issue in the Ping Identity DevOps Issue Tracker.

Please understand that enhancement requests are reviewed and prioritized internally, and we might not be able to address all requests or provide an estimated time for resolution.

Alternate Route for Submitting Bugs and Suggesting Enhancements

If you would rather not have your issue discussed on the public repository, you can open an issue from Ping Identity's Support Portal.

Contributing Code Changes

Ping Identity does not accept third-party code submissions.

Community

Ping Identity maintains a community where you can ask questions of Ping employees and other users of our products.

You can submit your questions at the Cloud DevOps Community.