Installing Anchore Enterprise with Kubernetes & Helm

Table of Contents

Anchore Support

Anchore Enterprise customers should have a primary point of contact at Anchore with whom they can raise immediate issues. In addition, our customers can file tickets via Anchore's Freshdesk site located here: https://anchore.freshdesk.com/support/home

All Anchore users are welcome to join our community Slack channel located here: https://anchore.com/slack

Getting started

This document will detail the necessary requirements for installing Anchore Enteprise with Kubernetes and Helm.

Hardware requirements

The following details the minimum hardware requirements needed to run a single instance of all containers:

- 2 CPUs
- 8 GB RAM
- 50 GB disk space

Note: Increased CPUs and RAM is recommended for better performance.

Kubernetes requirements

- A running Kubernetes Cluster is required.
- kubectl configured to access your Kubernetes cluster.

Read more on Kubernetes here: https://kubernetes.io/
Read more on kubectl configuration here: https://kubernetes.io/docs/tasks/tools/install-kubectl/

Helm requirements

- The Helm binary should be installed and available on your path.
- Tiller, the server side component for Helm, should be installed and available inside your Kubernetes cluster.

Read more on Helm here: https://helm.sh/

Operating System requirements

- Ubuntu 16.04x or higher
- CentOS 7.3 or higher
- RHEL 7.3 or higher
- Amazon Linux 2

Software requirements

The Anchore Enterprise UI is a web application with an HTML interface. Accessing the user interface is done via a web browser.

- Chrome
- Firefox
- Safari

Network requirements

Anchore Enterprise Feeds exposes a RESTful API by default on port 8228, however this port can be remapped.

Anchore Enterprise Feeds require access to the upstream data feeds from the following supported distributions and package registries over port 443:

Host	Port	Description
linux.oracle.com	443	Oracle Linux Security Feed
github.com	443	Alpine Linux Security Database
redhat.com	443	Red Hat Enterprise Linux Security Database
security-tracker.debain.org	443	Debian Security Feed
salsa.debian.org	443	Debian Security Feed
replicate.npmjs.com	443	NPM Registry Package Data
s3-us-west-2.amazonaws.com	443	Ruby Gems Data Feed
static.nvd.nist.gov	443	NVD Database
launchpad.net/ubuntu-cve-tracker	443	Ubuntu Data
data.anchore-enterprise.com	443	Snyk data

Note: Air-gapped installs will differ.

Anchore Enterprise UI by default will be accessible over http://localhost:3000.

Database requirements

Anchore Enterprise uses PostgreSQL object-relational database to store data. Before beginning install, determine whether you will be using the managed PostgreSQL service container that is automatically installed with this chart or an external PostgreSQL instance.

Note: It is recommended to use an external PostgreSQL instance for production grade deployments. See configuring external DB instance for more info.

PostgreSQL versions

The PostgreSQL container that is automatically installed with Anchore Enterprise is postgres:9.

Anchore Enterprise supports PostgeSQL version 9.6 or higher

Production Deployment Recommendations

For production deployments of Anchore Enterprise we recommend the following:

- Configuring an external PostgreSQL database instance.
- Configuring an external archive driver.
- Setting non-default password for database and Anchore.

Installation

- Approved Dockerhub username is required to pull Anchore Enterprise images.
- A valid Anchore Enterprise license.yaml file.

About this Helm Chart

This chart will can deploy both the Anchore Engine (OSS) and Anchore Enterprise systems. The chart is split into global and service specific configurations for the OSS Anchore Engine, as well as the global and service specific configurations for the Enterprise components. The recommended way to install Anchore Enterprise via this chart is to create a new file names `anchore_values.yaml` and add the desired fields.

- The `anchoreGlobal` section is for the configuration values required by all Anchore Engine components.
- The `anchoreEnterpriseGlobal` section is for configuration values required by all Anchore Enginer Enterprise components.
- Service specific configuration values allow customization for each individual service.

In addition to the database the chart creates two deployments:

- Core services: The core services deployment includes the external api, notification service, kubenetes webhook, catalog and queuing service.
- Worker: The worker service conducts the image analysis and can be scaled out to handle concurrent evaluation of images.

In this example we will deploy the database, core services, a single worker, and the enterprise components.

Enterprise components include:

- Role based access control (RBAC)
- On-prem feeds service
- Snyk vulnerability data
- Graphical User Interface

Step 1: Create kubernetes secret for license file

*Run kubectl command below to generate a secret for the license file: *

```
kubectl create secret generic anchore-enterprise-license --from-
file=license.yaml=<PATH/TO/LICENSE.YAML>
```

Step 2: Create kubernetes secret for dockerhub credentials

Create kubernetes secret containing valid dockerhub credentials with access to private Anchore Enterprise repositories.

Run kubectl command below to generate a secret for dockerhub credentials:

```
kubectl create secret docker-registry anchore-enterprise-pullcreds --docker-
server=docker.io --docker-username=<DOCKERHUB_USER> --docker-password=
<DOCKERHUB_PASSWORD> --docker-email=<EMAIL_ADDRESS>
```

Step 3: Install Helm Chart

Install Helm Chart using custom anchore_values.yaml file

Note: By default, all services (including a bundled DB instance) will be transient, and data will be lost if you shut down/restart.

Note: This chart will install a managed PostgreSQL database and Redis.

Create `anchore_values.yaml` file and enter the following:

```
## anchore_values.yaml

anchoreEnterpriseGlobal:
  enabled: True
```

Set `anchoreEnterpriseGlobal` to true in order to enable the enterprise components.

Install the Helm Chart by running the following command (remember to pass in the custom values file):

```
helm install --name <release_name> -f /path/to/anchore_values.yaml
stable/anchore-engine
```

Step 4: Verify services are up

If the previous command was run successfully, you should see Anchore notes in the terminal describing how to access the Anchore Engine services.

Run the following command to see the pods:

```
kubectl get pods
```

The output should look like the example below:

```
anchore-demo-anchore-engine-analyzer-5c87776cbc-4drvr           1/1
Running   0          47m
anchore-demo-anchore-engine-api-7868d8bc68-xjsqg                3/3
Running   0          47m
anchore-demo-anchore-engine-catalog-546bfbc499-jbwls            1/1
Running   0          47m
anchore-demo-anchore-engine-enterprise-feeds-7fd997b67f-f5pv9   1/1
Running   0          47m
anchore-demo-anchore-engine-enterprise-ui-6688fc7d47-bd4fn      1/1
Running   0          47m
anchore-demo-anchore-engine-policy-74c4956dc7-l66gw             1/1
Running   0          47m
anchore-demo-anchore-engine-simplequeue-784597f666-qssrm        1/1
Running   0          47m
anchore-demo-anchore-feeds-db-64f895cd75-8fshd                  1/1
Running   0          47m
anchore-demo-anchore-ui-redis-master-0                          1/1
Running   0          47m
anchore-demo-postgresql-54f5d746d5-drzrd                        1/1
Running   0          47m
```

In order to check on the status of the Anchore services, run the following command:

```
kubectl run -i --tty anchore-cli --restart=Always --image anchore/engine-cli --
env ANCHORE_CLI_USER=admin --env ANCHORE_CLI_PASS=${ANCHORE_CLI_PASS} --env
ANCHORE_CLI_URL=http://<anchore_service_endpoint>:8228/v1/
```

From within the container, you are able to use 'anchore-cli' commands:

Ex. `anchore-cli system status`

The ouput should look like the example below:

```
Service analyzer (anchore-demo-anchore-engine-analyzer-5c87776cbc-4drvr,
http://anchore-demo-anchore-engine-analyzer:8084): up
Service simplequeue (anchore-demo-anchore-engine-simplequeue-784597f666-
qssrm, http://anchore-demo-anchore-engine-simplequeue:8083): up
Service rbac_authorizer (anchore-demo-anchore-engine-api-7868d8bc68-xjsqg,
http://localhost:8089): up
Service apiext (anchore-demo-anchore-engine-api-7868d8bc68-xjsqg,
http://anchore-demo-anchore-engine-api:8228): up
Service rbac_manager (anchore-demo-anchore-engine-api-7868d8bc68-xjsqg,
http://anchore-demo-anchore-engine-api:8229): up
Service policy_engine (anchore-demo-anchore-engine-policy-74c4956dc7-
l66gw, http://anchore-demo-anchore-engine-policy:8087): up
Service catalog (anchore-demo-anchore-engine-catalog-546bfbc499-jbwls,
http://anchore-demo-anchore-engine-catalog:8082): up

Engine DB Version: 0.0.8
Engine Code Version: 0.3.1
```

Important to note that upon initial install of Anchore Enterprise, it will take some time for vulnerability data to be synced into Anchore. For the most optimal experience, wait until all vulnerability data feeds have synced before performing any image analysis operations.

Run the following command to wait for until Anchore is available and ready

```
anchore-cli system wait
```

You should see output like the example below when Anchore is ready:

```
Starting checks to wait for anchore-engine to be available timeout=-1.0
interval=5.0
API availability: Checking anchore-engine URL (http://0.0.0.0:8228/v1)...
API availability: Success.
Service availability: Checking for service set
(catalog,simplequeue,analyzer,policy_engine,apiext)...
Service availability: Success.
Feed sync: Checking sync completion for feed set (vulnerabilities)...
Feed sync: Success.
```

Checking the status of the Enterprise Data Feeds

You can check on the status of the data feeds by running the following command:

```
anchore-cli system feeds list
```

The ouput should look like the example below:

```
Feed                    Group                   LastSync
RecordCount
snyk                    snyk:java               2019-01-10T18:23:48.169335
1764
snyk                    snyk:js                 2019-01-10T18:23:48.221875
1251
snyk                    snyk:python             2019-01-10T18:23:48.256525
806
snyk                    snyk:ruby               2019-01-10T18:23:48.240023
527
vulnerabilities         alpine:3.3              2019-01-10T18:23:47.646567
457
vulnerabilities         alpine:3.4              2019-01-10T18:23:47.311669
681
vulnerabilities         alpine:3.5              2019-01-10T18:23:44.229436
875
vulnerabilities         alpine:3.6              2019-01-10T18:23:47.285151
918
vulnerabilities         alpine:3.7              2019-01-10T18:23:47.496200
919
vulnerabilities         alpine:3.8              2019-01-10T18:23:47.372342
996
vulnerabilities         amzn:2                  2019-01-10T18:23:45.982926
121
vulnerabilities         centos:5                2019-01-10T18:23:47.442663
1323
vulnerabilities         centos:6                2019-01-10T18:23:44.295297
1312
vulnerabilities         centos:7                2019-01-10T18:23:43.178719
738
vulnerabilities         debian:10               2019-01-10T18:23:47.151327
19156
vulnerabilities         debian:7                2019-01-10T18:23:47.411609
20455
vulnerabilities         debian:8                2019-01-10T18:23:48.033485
20847
vulnerabilities         debian:9                2019-01-10T18:23:45.035600
19550
vulnerabilities         debian:unstable         2019-01-10T18:23:45.814821
19971
vulnerabilities         ol:5                    2019-01-10T18:23:45.255953
1227
vulnerabilities         ol:6                    2019-01-10T18:23:47.395976
1372
vulnerabilities         ol:7                    2019-01-10T18:23:44.197697
826
vulnerabilities         ubuntu:12.04            2019-01-10T18:23:48.067604
```

```
14946
vulnerabilities        ubuntu:12.10          2019-01-10T18:23:48.117023
5652
vulnerabilities        ubuntu:13.04          2019-01-10T18:23:45.213661
4127
vulnerabilities        ubuntu:14.04          2019-01-10T18:23:47.207201
15774
vulnerabilities        ubuntu:14.10          2019-01-10T18:23:47.518278
4456
vulnerabilities        ubuntu:15.04          2019-01-10T18:23:47.338468
5676
vulnerabilities        ubuntu:15.10          2019-01-10T18:23:45.958228
6511
vulnerabilities        ubuntu:16.04          2019-01-10T18:23:47.550738
12751
vulnerabilities        ubuntu:16.10          2019-01-10T18:23:48.094067
8647
vulnerabilities        ubuntu:17.04          2019-01-10T18:23:47.248567
9157
vulnerabilities        ubuntu:17.10          2019-01-10T18:23:45.901606
7632
vulnerabilities        ubuntu:18.04          2019-01-10T18:23:44.117638
6991
```

At this point, Anchore Enteprise should now be fully installed and you can begin to analyze images.

Installing Anchore-CLI

The Anchore CLI provides a command line interface on top of the Anchore Engine REST API.

Anchore CLI github repo: https://github.com/anchore/anchore-cli Anchore CLI Dockerhub repo: https://hub.docker.com/r/anchore/engine-cli/

Running the Anchore CLI Container

This image provides a simple way to interact with a Anchore Engine service installation.

To use the container run the following command:

```
kubectl run -i --tty anchore-cli --restart=Always --image anchore/engine-cli --
env ANCHORE_CLI_USER=admin --env ANCHORE_CLI_PASS=${ANCHORE_CLI_PASS} --env
ANCHORE_CLI_URL=http://<anchore_service_endpoint>:8228/v1/
```

From this container's shell you can use 'anchore-cli' commands. Ex. `anchore-cli system status`

Note: See below for commmand for analyzing your first image.

Install Anchore CLI from source

The Anchore CLI can be installed from source using the Python pip utility.

```
git clone https://github.com/anchore/anchore-cli
cd anchore-cli
pip install --user --upgrade .
```

Configuring the Anchore CLI

By default the Anchore CLI will try to connect to the Anchore Engine at http://localhost/v1 with no authentication. The username, password and URL for the server can be passed to the Anchore CLI as command line arguments.

```
--u   TEXT   Username     eg. admin
--p   TEXT   Password     eg. foobar
--url TEXT   Service URL  eg. http://localhost:8228/v1
```

Rather than passing these parameters for every call to the cli they can be stores as environment variables.

```
ANCHORE_CLI_URL=http://myserver.example.com:8228/v1
ANCHORE_CLI_USER=admin
ANCHORE_CLI_PASS=foobar
```

Scanning your first image

Now that both Anchore Enterprise and the Anchore CLI have been install and configured, you can begin to scan images.

Run the following command to scan your first image:

```
anchore-cli image add docker.io/library/alpine:latest
```

Configuring an external PostgreSQL instance

As stated in the database requirements above, Anchore requires access to a PostgreSQL database. The database can be run as a container out of the box with a persisted volume or outside of your container environment. If you choose to use an external PostgreSQL Database, the connection string should be specified in the config.yaml file.

Note: The default configuration points to the host anchore-db on port 5432 using username postgres and password mysecretpassword.

If you are configuring an external database service (e.g. Amazon RDS), updated the host, port, username, password, and database name.

Here is the database section of the config.yaml file with environment variables being passed in:

Database section of anchore_values.yaml file

```
  postgresql:
    postgresPassword: <PASSWORD>
    postgresUser: <USER>
    postgresDatabase: <DATABASE>
    enabled: false
    externalEndpoint: <HOSTNAME:5432>

  anchoreGlobal:
    dbConfig:
      ssl: true
```

Anchore should now be able to connect to your external PostgreSQL DB instance.

Accessing logs

There may be cases where you need to inspect the logs for Anchore services.

Viewing logs for pod containers

Use the kubectl logs command to view the logs for an individual pod container:

```
kubectl logs <my-pod-name>
```

Viewing logs for specific Anchore services

You can execute into a specific pod to view logs for an Anchore service.

Use the kubectl exec command to enter a pod container:

```
kubectl exec -it <my-pod-name> /bin/bash
```

The logs are located in the `/var/log/anchore` directory within the container.

Configuring an Archive Driver

Note: It is recommended to use an external archive driver for production deployments.

The archive subsystem of Anchore Engine is what stores large json documents and can consume quite a lot of storage if you analyze a lot of images. A general rule for storage provisioning is 10MB per image analyzed, so with thousands of analyzed images, you may need many gigabytes of storage. The Archive drivers now support other backends than just PostgreSQL, so you can leverage external and scalable storage systems and keep the postgresql storage usage to a much lower level.

Configuring compression

The archive system has compression available to help reduce size of objects and storage consumed in exchange for slightly slower performance and more cpu usage. There are two config values:

To toggle on/off (default is True), and set a minimum size for compression to be used (to avoid compressing things too small to be of much benefit, the default is 100):

```
## anchore_values.yaml
anchoreCatalog:
  archive:
    compression:
      enabled=True
      min_size_kbytes=100
```

The supported archive drivers are:

- S3 - Any AWS s3-api compatible system (e.g. minio, scality, etc)
- OpenStack Swift
- Local FS - A local filesystem on the core pod. Does not handle sharding or replication, so generally only for testing.
- DB - the default postgresql backend

S3

```
## anchore_values.yaml
anchoreCatalog:
  archive:
    storage_driver:
      name: 's3'
      config:
        access_key: 'MY_ACCESS_KEY'
        secret_key: 'MY_SECRET_KEY'
        #iamauto: True
        url: 'https://S3-end-point.example.com'
        region: null
        bucket: 'anchorearchive'
        create_bucket: True
    compression:
    ... # Compression config here
```

Swift

The swift configuration is basically a pass-thru to the underlying pythonswiftclient so it can take quite a few different options depending on your swift deployment and config. The best way to configure the swift driver is by using a custom anchore_values.yaml

The Swift driver supports three authentication methods:

- Keystone V3
- Keystone V2
- Legacy (username / password)

Keystone V3:

```
## anchore_values.yaml
anchoreCatalog:
  archive:
    storage_driver:
      name: swift
      config:
        auth_version: '3'
        os_username: 'myusername'
        os_password: 'mypassword'
        os_project_name: myproject
        os_project_domain_name: example.com
        os_auth_url: 'foo.example.com:8000/auth/etc'
        container: 'anchorearchive'
        ## Optionally
        create_container: True
    compression:
    ... ## Compression config here
```

Keystone V2:

```
## anchore_values.yaml
anchoreCatalog:
  archive:
    storage_driver:
      name: swift
      config:
        auth_version: '2'
        os_username: 'myusername'
        os_password: 'mypassword'
        os_tenant_name: 'mytenant'
        os_auth_url: 'foo.example.com:8000/auth/etc'
        container: 'anchorearchive'
        ## Optionally
        create_container: True
    compression:
    ... ## Compression config here
```

Legacy username/password:

```
anchoreCatalog:
  archive:
    storage_driver:
      name: swift
      config:
        user: 'user:password'
        auth: 'http://swift.example.com:8080/auth/v1.0'
        key:  'anchore'
        container: 'anchorearchive'
```

```
        # Optionally
        create_container: True
    compression:
    ... # Compression config here
```

Scaling individual components

As of Chart version 0.9.0, all services can now be scaled-out by increasing the replica counts. The chart now supports this configuration.

To set a specific number of service containers:

```
anchoreAnalyzer:
  replicaCount: 5

anchorePolicyEngine:
  replicaCount: 3
```

To update the number in a running configuration:

```
helm upgrade --set anchoreAnalyzer.replicaCount=2 <releasename> stable/anchore-engine -f anchore_values.yaml
```