Installing Anchore Enterprise with Docker Compose

Anchore Support

Join community Slack channel: https://anchore.com/slack

Getting started

This document will detail the necessary requirements for installing Anchore Enteprise with Docker Compose.

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.

Docker requirements

Anchore Enterprise is delivered as a Docker container, so a Docker comptabile runtime is a requirement.

Anchore Enterprise supports Docker runtime versions 1.12 or higher and Compose version 2.x.

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

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 PostgreSQL database container that is automatically install or an external PostgreSQL instance.

Note: 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

Installation

- Approved Dockerhub username is required to pull Anchore Enterprise images.
- A valid Anchore Enterprise license.yaml file.
- docker-compose.yaml file (will detail how to obtain in steps below)

Step 1: Create installation location

Create a directory to store the configuration files and license file.

```
mkdir ~/aevolume
```

Step 2: Copy configuration files

Download the latest Anchore Enterprise container image which contains the necessary docker-compose and configuration files needed. In order to download the image, you'll need to login to docker using the dockerhub account that you provided to Anchore when you requested your license.

Run the following commands to do so:

```
docker login
```

Enter username and password.

```
docker pull docker.io/anchore/enterprise:latest
```

Next, copy the included docker-compose.yaml file into the directory you created in step 1.

Via the following commands:

```
docker create --name ae docker.io/anchore/enterprise:latest

docker cp ae:/docker-compose.yaml ~/aevolume/docker-compose.yaml

docker rm ae
```

Next, copy the license.yaml file that provided into the directory you created in step 1.

Via the following command:

```
cp /path/to/your/license.yaml ~/aevolume/license.yaml
```

Once these steps are completed, your Anchore directory workspace should look like the following.

Check by running the following commands:

```
cd ~/aevolume
```

```
find .
.
./docker-compose.yaml
./license.yaml
```

Step 3: Download and run the containers

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

Run the following commands within the directory created in step 1 to pull and run the containers:

```
docker-compose pull

docker-compose up -d
```

Step 4: Verify services are up

After a bit of time, run the following command to verify the containers are running:

```
docker-compose ps
```

The output should look like the example below:

```
aevolume_anchore-db_1_732e4d561243                  docker-entrypoint.sh
postgres    Up             5432/tcp
aevolume_engine-analyzer_1_d10cdb8b34f1             /docker-entrypoint.sh
anch ...   Up (healthy)   8228/tcp
aevolume_engine-api_1_89fd746624f3                  /docker-entrypoint.sh
anch ...   Up (healthy)   0.0.0.0:8228->8228/tcp
aevolume_engine-catalog_1_680e4226efad              /docker-entrypoint.sh
anch ...   Up (healthy)   8228/tcp
aevolume_engine-policy-engine_1_79ef08176b38        /docker-entrypoint.sh
anch ...   Up (healthy)   8228/tcp
aevolume_engine-simpleq_1_42c62abcaf9d              /docker-entrypoint.sh
anch ...   Up (healthy)   8228/tcp
aevolume_enterprise-feeds-db_1_244f869bdc97         docker-entrypoint.sh
postgres    Up             5432/tcp
aevolume_enterprise-feeds_1_1810c017b6d7            /docker-entrypoint.sh
anch ...   Up (healthy)   0.0.0.0:8448->8228/tcp
aevolume_enterprise-rbac-authorizer_1_8b1d8c63ad8c  /docker-entrypoint.sh
anch ...   Up (healthy)   8089/tcp, 8228/tcp
aevolume_enterprise-rbac-manager_1_3f7aa316211c     /docker-entrypoint.sh
anch ...   Up (healthy)   0.0.0.0:8229->8228/tcp
aevolume_enterprise-ui-redis_1_50e706cb20aa         docker-entrypoint.sh
redis ...   Up             6379/tcp
aevolume_enterprise-ui_1_dafff06270b2               /bin/sh -c node
/home/node ...   Up             0.0.0.0:3000->3000/tcp
```

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

```
docker-compose exec engine-api anchore-cli system status
```

The ouput should look like the example below:

```
Service policy_engine (anchore-quickstart, http://engine-policy-
engine:8228): up
Service catalog (anchore-quickstart, http://engine-catalog:8228): up
Service analyzer (anchore-quickstart, http://engine-analyzer:8228): up
Service rbac_authorizer (anchore-quickstart, http://enterprise-rbac-
authorizer:8228): up
Service simplequeue (anchore-quickstart, http://engine-simpleq:8228): up
Service apiext (anchore-quickstart, http://engine-api:8228): up
Service rbac_manager (anchore-quickstart, http://enterprise-rbac-
manager:8228): 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:

```
docker-compose exec engine-api 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

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 config.yaml file

```
credentials:
  database:
    db_connect:
'postgresql+pg8000://${ANCHORE_DB_USER}:${ANCHORE_DB_PASSWORD}@${ANCHORE_D
B_HOST}:${ANCHORE_DB_PORT}/${ANCHORE_DB_NAME}'
    db_connect_args:
      timeout: 120
      ssl: false
    db_pool_size: 30
    db_pool_max_overflow: 100
```

Within the docker-compose.yaml file you can specify the database environment variables to be passed into the config.yaml file like so:

API service section of docker-compose.yaml file

```
services:
  # The primary API endpoint service
  engine-api:
    image: anchore/anchore-engine:v0.3.1
    depends_on:
    - engine-catalog
    volumes:
    - ./config-engine.yaml:/config/config.yaml:z
    ports:
    - "8228:8228"
    logging:
      driver: "json-file"
      options:
        max-size: 100m
    environment:
    - ANCHORE_ENDPOINT_HOSTNAME=engine-api
    - ANCHORE_DB_HOST=anchore-db-instance.<123456>.us-east-
2.rds.amazonaws.com
    - ANCHORE_DB_NAME=anchore_db
    - ANCHORE_DB_USER=dbusername
    - ANCHORE_DB_PASSWORD=dbpassword
```

```
    - ANCHORE_DB_PORT=dbport
    - ANCHORE_AUTHZ_HANDLER=external
    - ANCHORE_EXTERNAL_AUTHZ_ENDPOINT=http://enterprise-rbac-
authorizer:8228
    - ANCHORE_ENABLE_METRICS=false
    command: ["anchore-manager", "service", "start",  "apiext"]
```

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