**Docker-Bench and Anchore Demonstration**

**Docker-Bench**

Source Code Management:

The docker-bench is an open source tools for docker-bench security. Built by the members of docker themselves, it is a script that checks for dozens of common best-practices around deploying Docker containers in production. The tests are all automated, and are inspired by the [CIS Docker Benchmark v1.2.0](https://www.cisecurity.org/benchmark/docker/).

Here is the official location of the docker-bench repository

<https://github.com/docker/docker-bench-security>

Docker-bench can be run various ways. We are running it by building our own container.

Let’s see how we are managing docker-bench in our repository.

It can be found at :

<https://git.mmi.mig.corp/release-management/systemsadministration/tree/admlama-docker-bench/linux/docker/dockerfiles/migdockerbench>

A screenshot of a cell phone

Description automatically generated

Here,

**Tests:** The tests directory contains the script files that of the checks. Inside of that directory looks like :

**A screenshot of a cell phone

Description automatically generated**

When we run the scan using docker-bench, these are the different methods which docker bench uses to make a scan. From this picture, we can find what kind of vulnerabilities does docker-bench looks for. After running docker-bench, by default we will receive vulnerabilities of our host server, on our docker-daemon configurations, container and images, security wise configurations , swarm configuration and also enterprise configuration.

**Dockerfile** : It is the file to create an image for the docker-bench which we will we using to make a scan on other projects. It is based on the alpine image.

A screenshot of a social media post

Description automatically generated

**Build.properties:** This file contains the value of alpine image tag and the vendor tag of the docker-bench to keep track of the version we are using.

A screenshot of a cell phone

Description automatically generated

**Docker-bench-security.sh**: It is a script file which contains all available CIS tests. The script is run at entry-point. So, once we build a container with the image, we can receive the scan result.

**Functions\_lib.sh**: A file containing a complete list of the checks.

**Using docker-bench**

docker run -it --net host --pid host --userns host --cap-add audit\_control \

-e DOCKER\_CONTENT\_TRUST=$DOCKER\_CONTENT\_TRUST \

-v /etc:/etc:ro \

-v /lib/systemd/system:/lib/systemd/system:ro \

-v /usr/bin/containerd:/usr/bin/containerd:ro \

-v /usr/bin/runc:/usr/bin/runc:ro \

-v /usr/lib/systemd:/usr/lib/systemd:ro \

-v /var/lib:/var/lib:ro \

-v /var/run/docker.sock:/var/run/docker.sock:ro \

--label docker\_bench\_security \

docker/docker-bench-security

The container is being run with a *lot* of privilege -- sharing the host's filesystem, pid and network namespaces, due to portions of the benchmark applying to the running host.

**Docker-bench for security options**

The list of flags we can use to run the docker-bench to our needs.

Example:

-c flag can be used to run specifics check:

-c host\_configuration will run the checks related to host configuration checks.

-b optional Do not print colors

-h optional Print this help message

-l FILE optional Log output in FILE

-c CHECK optional Comma delimited list of specific check(s)

-e CHECK optional Comma delimited list of specific check(s) to exclude

-i INCLUDE optional Comma delimited list of patterns within a container or image name to check

-x EXCLUDE optional Comma delimited list of patterns within a container or image name to exclude from check

Benchmark – Result:

A screenshot of a social media post

Description automatically generated

**Docker-bench base-Image build:**

Like how we were building our base image, Docker-bench base image is also built and pushed to nexus using the same pattern. The job configuration can be found here:

<https://jenkins.mmi.mig.corp/job/docker_base_images/job/Other_Pipeline/job/build_migdockerbench/configure>

**Anchore**

Anchore is an open source tool for deep image inspection and vulnerability scanning. Anchore Engine is fully featured and flexible and can work within a wide variety of environments and development pipelines. The open source package provides most of the features and they have enterprise version as well

**Source Code-Management:**

Anchore is currently running in v0.7.2. The location to gitlab project for anchore is :

<https://git.mmi.mig.corp/release-management/systemsadministration/tree/admlama-docker-bench/linux/docker/dockerfiles/miganchore>

A screenshot of a cell phone

Description automatically generated

**Deployment:**

Anchore is deployed using compose files. I have placed the compose files on svn under aevolume for now.

Link to the compose file:

<https://mgpsbvrw01.mmi.mig.corp/!/#SystemsAdministration/view/head/trunk/linux/docker/composefiles/test.mig.corp/aevolume>

We are currently running 6 anchore services. They are described below:

1. **Engine API:**

The Anchore Engine API is the primary API for the entire system. This service runs the API used to analyze images, get policy evaluations etc.

It consumes:

* Catalog
* Policy Engine
* SimpleQueue

Requires

* Anchore Engine DB

1. **Catalog**

The catalog is the primary state manager of the system and owns both the state machines for images as well as the document archive interface used to store large, unstructured documents like JSON outputs from analysis.

**Consumes**

* Policy Engine
* SimpleQueue

##### **Requires**

* Anchore Engine DB

#### Policy Engine

The policy engine is responsible for loading the result of an image analysis and normalizing and structuring the data in a way that makes it quickly searchable, scans for vulnerabilities in the found artifacts of the image and provides fast policy evaluation over that data.

##### **Consumes**

* Catalog (for archive fetch)
* Enterprise Feed Service (in an enterprise install, for open-source it uses the hosted Anchore feed service at ancho.re)

##### **Requires**

* Anchore Engine DB

#### SimpleQueue

The simplequeue is a postgresql-backed queue service that the other components use for task execution, notifications, and other asynchronous operations.

##### **Requires**

* Anchore Engine DB

#### Analyzer

The anchore engine analyzer is the component that does all of the image download and analysis heavy-lifting. It receives work from the simplequeue service by polling specific queues and executes image analysis, uploading the results to the catalog and the policy engine when complete.

##### **Consumes**

* Policy Engine
* SimpleQueue
* Catalog

##### **Requires**

* Anchore Engine DB

#### Anchore Engine Database

Anchore is built around a single Postgresql database, using the default public schema namespace. This is the standard open source installed db and contains tables for all necessary services. The services do not communicate through the db., only through explicit API calls, but the database tables are consolidated for easier management operations.

Docker-compose-file

**Build Job:**

Like how we were building our base image, Anchore base image is also built and pushed to nexus using the same pattern. The job configuration can be found here:

<https://jenkins.mmi.mig.corp/job/docker_base_images/job/Other_Pipeline/job/build_miganchore/>

**Anchore Plugin Configuration:**

A screenshot of a cell phone

Description automatically generated

* Image list file: a file containing an image for anchore to scan. In our case a plugin searches for a file named anchore\_images and read the file to get an image name to scan.
* Anchore Engine URL: Override plugin's global settings for Anchore Engine URL. The override is applicable only to builds in this job and does not change the plugin wide global settings. In this case we have anchore engine installed on ohulsib0001.test.mig.corp server.
* Engine Credentials: A username and password to access the anchore engine.

**Anchore result:**

After the build is complete in Jenkins, the result executed by the Anchore can be seen as follow

A screenshot of a social media post

Description automatically generated

After clicking ,

A screenshot of a cell phone

Description automatically generated

Anchore provides a very clear results on the vulnerabilities.

So, this is how anchore and Docker-bench work.

**Next: Implementing Anchore and Docker-bench on our base image**

<https://jenkins.mmi.mig.corp/job/docker_base_images/job/migubuntu_1804_Pipeline/job/build_migubuntu/>

Here is the demonstration on how we are implementing both docker scanning tools to our base migubuntu image.