Project Description:
Security and compliance scanning tool for containers

Bharath Ananthanarayanan
Blake Hina
Vrushali Mahajan
Ben Owens

1. Vision and Goals Of The Project:

OpenScap is an auditing tool to verify whether a system confirms to standards specified in Security Content Automation Protocol (SCAP).

SCAP checks for two features:
Security Compliance: Security compliance is a state where computer systems are in line with a specific security policy.
Vulnerability Assessment: Vulnerability assessment is a continuous process that identifies security flaws and weaknesses in software.

Currently OpenSCAP tools support RedHat Linux distributions.

The goals of the project are:

1) To extend the capabilities of this (platform) for scanning container images of other distributions like Ubuntu.
2) To create an automated process for checking the compliance status & vulnerabilities without installing any tools or content in the system being scanned. Therefore, it is more suitable for scanning container images and running instances in a cloud environment.

Users/Personas Of The Project:

The end-users for the project will be people who have Ubuntu running on their computers. They previously would not have been able to run OpenSCAP on their specific distribution of Linux and this project is built for them.

2. Scope and Features Of The Project:

The project will require the following technology:

- Docker:
    - A "container technology", which wraps software with everything needed to run an instance of that software (code, system tools, libraries, etc.).
    - Ensures that software is able to run on any environment.

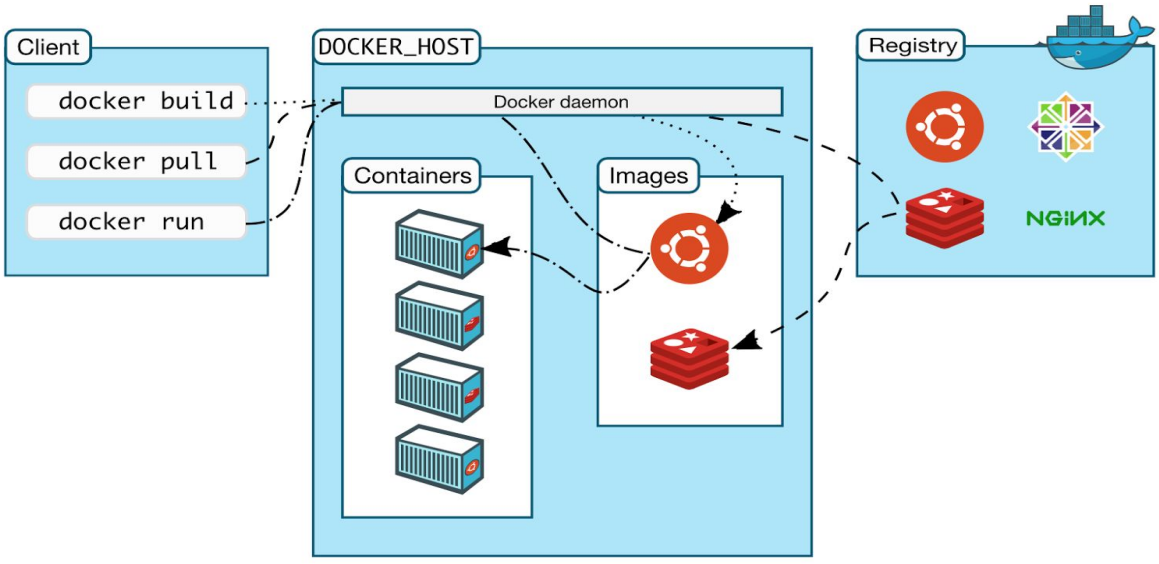
    - Containers are lightweight, and many can be run simultaneously on a host server.
    -
- OpenSCAP
    - An "auditing tool", that "audits" a system or object to see whether it complies to the rules in a compliance policy.
    - SCAP is a protocol for using widely-used rules and standards to enable automated security policy compliance metrics.
    - The National Vulnerability Database (NVD) is a US government content repository for SCAP policies.
- Database
    - SQL or NoSQL (probably MongoDB)
- A publicly accessible UI

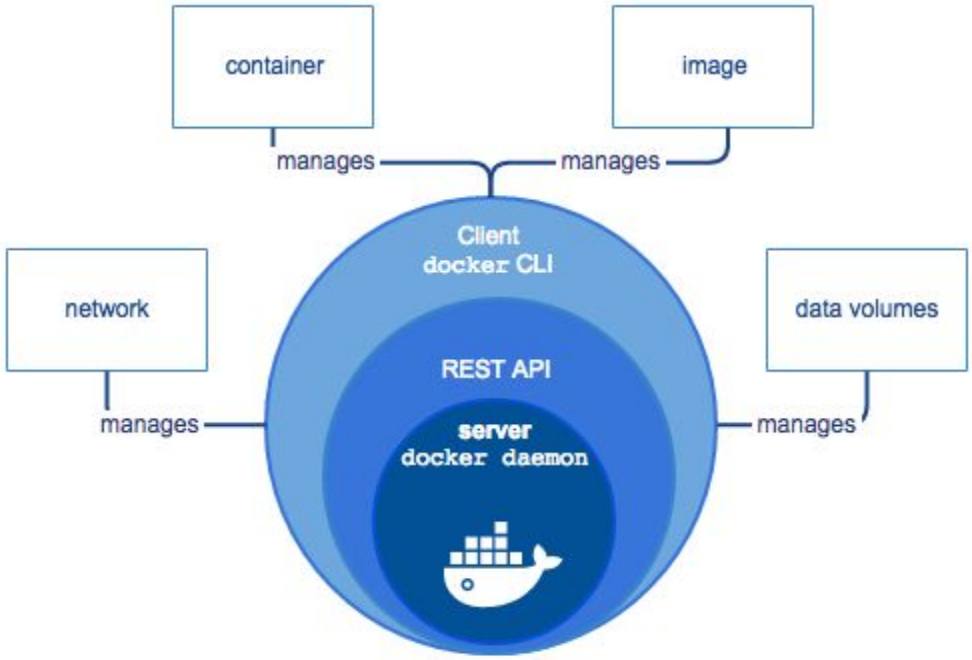
3. Solution Concept

This section provides a high-level outline of the solution.

Global Architectural Structure Of the Project:

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.





We also need to know about images, registries, and containers.

Docker images

A Docker image is a read-only template with instructions for creating a Docker container. For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others. An image may be based on, or may extend, one or more other images. A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.

Docker images are the build component of Docker.

Docker containers

A Docker container is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables. Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.

Docker containers are the run component of Docker.

Docker registries

A docker registry is a library of images. A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server. Docker registries are the distribution component of Docker.

What we plan to achieve:

Once an image is put to registry, we want to know first either through an alert or some feedback mechanism that such an image has been put to the registry and then we need to immediately scan the image

Registry will be known to everyone. Therefore what we intend to do is to have a web-hook on the registry, run the OpenSCAP tool that produces an output either in the form of a text file or an XML file. We will then analyze the scanned output and see if there are problems.

If there are no problems with the output, and there are no security problems verifying the standards such as WSSC or others, and it is security complaint, deploy the image into the public registry and start the container which, others can then pull and access. If the image has vulnerabilities, don't start the container, and we will have to state either the reasons or the conditions for which it failed.

In addition to this, we will have to decide a mechanism of stopping the image from loading into the public registry.

Design Implications and Discussion:

The project will be primarily based around already existing software, with the main challenge of converting OpenSCAP tools to work with Ubuntu.

4. Acceptance criteria

The minimum acceptance criteria is to extend OpenSCAP to be able scan containers for cloud vulnerability in Ubuntu.

Once this is completed, there are three stretch goals:

1. Build a local cache of the security content to make the system more efficient.
2. Continue to make OpenSCAP features available for other operating systems.
3. Build a publically accessible user interface.

5. Release Planning

Demo 1 (due February 23):

- Install docker and OpenSCAP on an existing RedHat-cluster.
- Demonstrate utilization of OpenSCAP tool.

Demo 2 (due March 16):

Demo 3 (due March 31)
Demo 4 (due April 13)
Demo 5 (due April 27): Full