Docker and RC2

SKG1

10.04.2018

—

Mayank Bhushan
15CS30019

Siddharth Jain
15CS30031

Main Aim of the project

The main aim of the project is to use Docker containers for the deployment of geospatial services. By using Docker containers for geospatial services, we can make multiple nodes in an area use web services in an organized manner. These nodes can be used in rescue missions during catastrophe in an area.

What is docker?

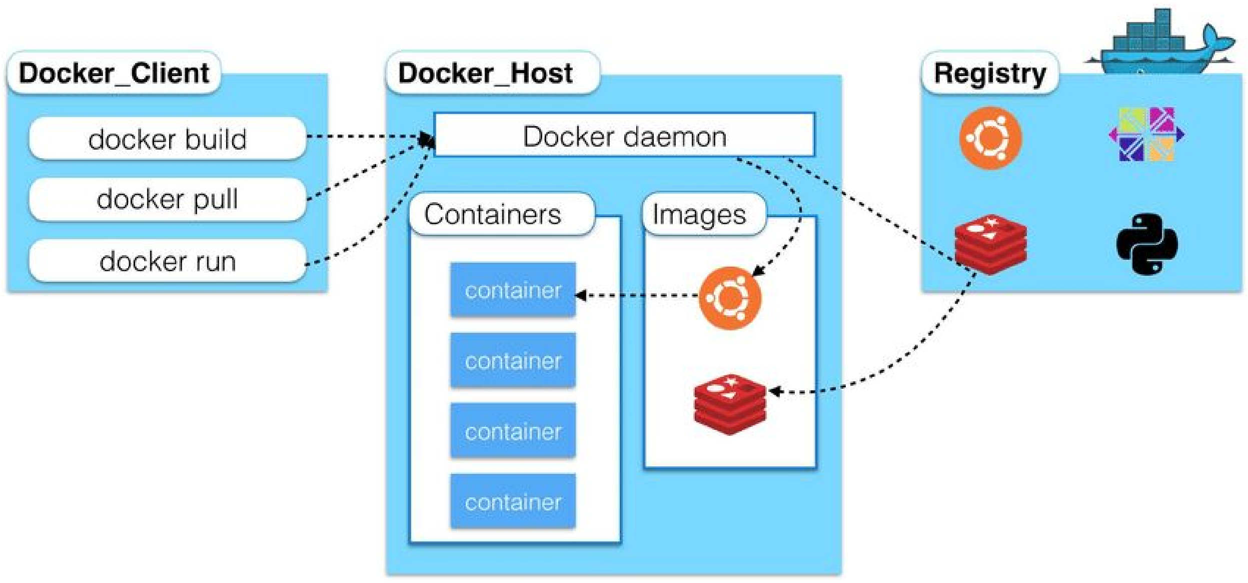
Docker is a container management software. The whole idea is for developers to easily develop applications, ship them into containers which can be deployed anywhere.

Benefits of Docker

- Fast
- Lightweight
- Open Source
- Portable Software
- Version Control

Docker Architecture

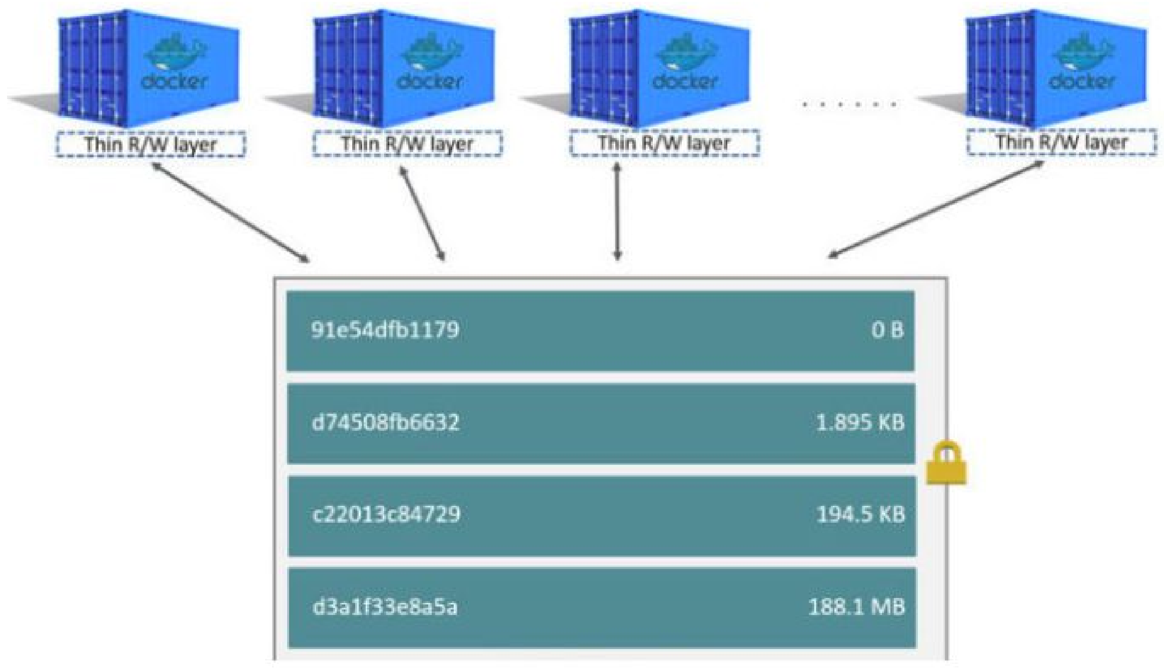
Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing Docker containers. The Docker client and daemon can run on the same system, or we can connect a Docker client to a remote Docker daemon.



Docker Images

- An image is the complete standalone package of software that can includes everything needed to run the image.
- The docker run command takes the Docker image as a template and produces a container from it.
- Images are created from a Dockerfile with the docker build command.
- Images are stored in a Docker registry, such as Docker Hub and can be downloaded with the docker pull command.
- There are two types of images based on whether they have a parent image or not:
    - Base Images
    - Child Images
- Images are also classified based on who developed it:
    - Official Images
    - User Images

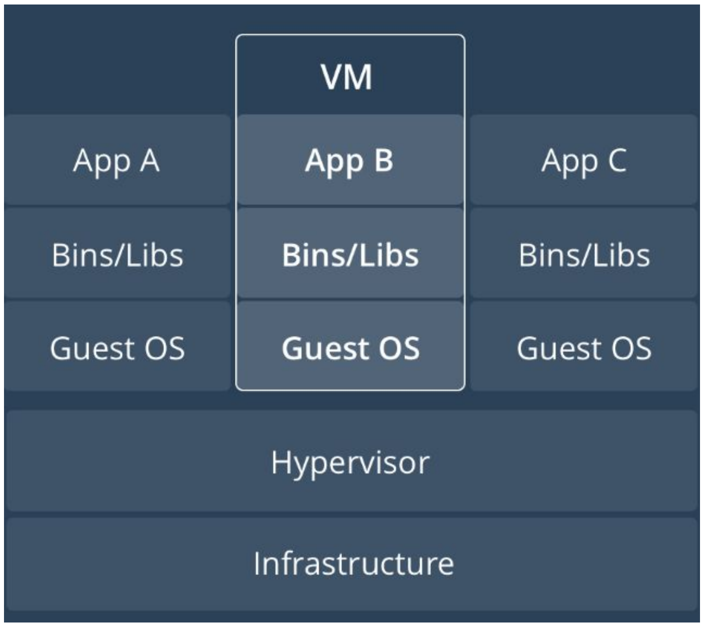
Docker Containers



- A container is a runtime instance of an image—what the image becomes in memory when actually executed.
- Containers are created from images with the docker run command and can be listed with the docker ps command.
- To create a container, Docker engine takes an image, adds the top writable layer and initializes various settings (network ports, container name, ID and resource limits).
- All write operation inside the container are stored in this writable layer, so when the container is deleted, the writable layer is also deleted while the underlying image remains unchanged.
- As each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state.

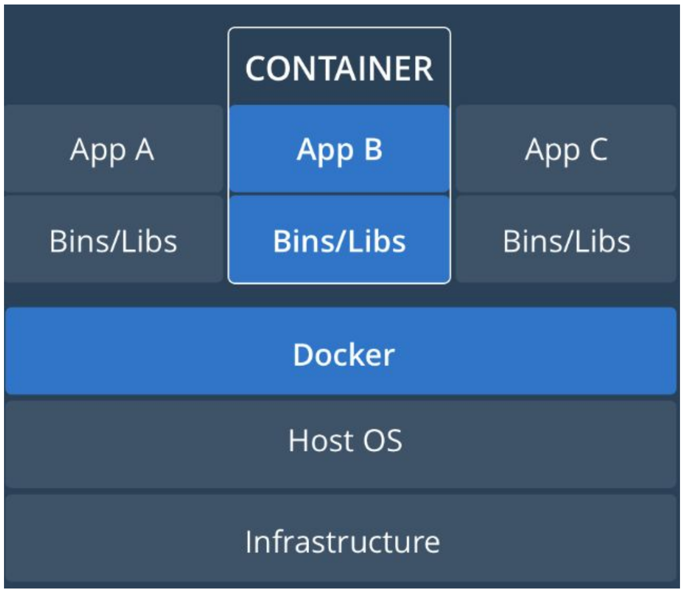
Docker Containers vs Virtual Machines

Virtual Machines



Each Virtual Machine includes a separate operating system image, which adds overhead in memory and storage footprint. This issue adds complexity to all stages of a software development lifecycle—from development and test to production and disaster recovery. This approach also severely limits the portability of applications between public clouds, private clouds, and traditional data centers.

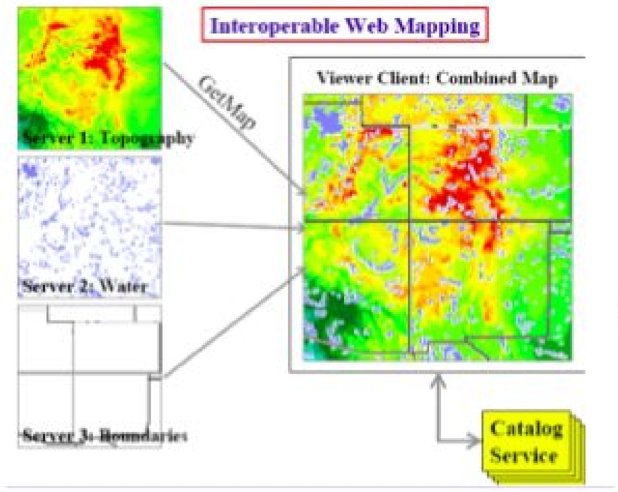
Docker Containers

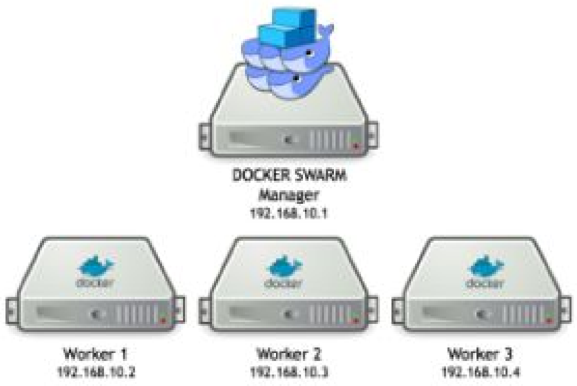


Containers can share a single kernel, and the only information that needs to be in a container image is the executable and its package dependencies, which never need to be installed on the host system.
They contain all their dependencies so there is no configuration entanglement; a containerized app “runs anywhere.”
Containers are thus exceptionally “light”—they are only megabytes in size and take just seconds to start, versus gigabytes and minutes for a VM.

Docker Swarms

Docker Engine now has native clustering via Swarm and offers the ability to instantly scale a container across multiple nodes with one command.





```
# Create Docker image
docker build --tag='your_image' .

# Create Docker service, default to 0 containers, set environment variable appropriately
docker service create --name service_etl --replicas 0 --e "action=etl"  your_image  # Run ETL example
docker service create --name service_analysis --replicas 0 --e "action=analysis"  your_image  # Run analysis example

# Scale as much as your infrastructure will handle
docker service scale service_etl=300
docker service scale service_analysis=300

# To scale down when you are done use
docker service scale service_etl=0
docker service scale service_analysis=0
```

RC2 : A Risk Quantification Framework for Dynamic Access Control in Cloud-based Collaborations

Need

The current models for authorization in the collaborative services of cloud service provider are only concerned with the identity of the requester and the validity of his token. They do not consider the history(previous requests) of the requester, which is important because a valid requester with a valid token can still have malicious intentions.

So, we need a model which takes the history of the requester into account. One such existing model is Risk based Access Control (RAC), where the key step is to quantify the risk associated with each access request, and depending on the value exceeds a threshold or not, the decision is made.

However, these models lack generality and are highly sensitive to the weights assigned to different risk factors by the domain experts. Moreover, they do not model the uncertainty of misusing the shared resources before giving access to any requester from remote domain. In total these models are slow and not up to the mark.

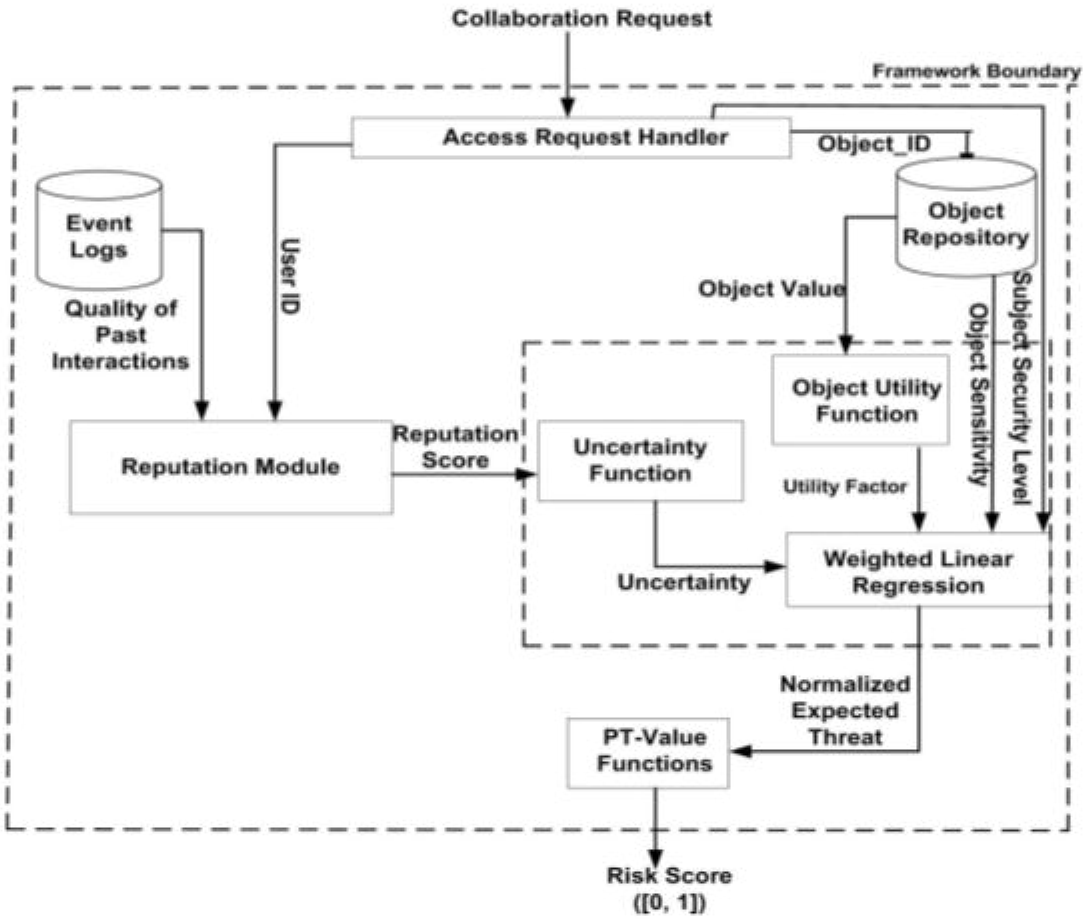
So, there is a need for a risk quantification mechanism which is generic and light-weighted to suit cloud-based applications, and incorporates soft security mechanism (e.g., reputation) to model uncertainty related to resource sharing.

The threats posed on the objects shared with legitimate but malicious users can be categorized into two types-

- *Threats on integrity and availability* : modify or delete the contents
- *Threat on confidentiality* : disclosure of contents to unauthorized users.

Model

RC2 framework is an additional security layer above the token based authorization scheme. To avoid the overhead it can be implemented as a web service which is invoked when collaboration requests are received.



The models approach is such that it maximises the reputation for all the new requesters, and penalize them only if they have been found to misuse the shared resources. Otherwise the requester continues with maximum reputation and all legitimate accesses are permitted.

References

I. Official Docker Documentation

II. forums.docker.com

III. www.stackoverflow.com

IV. RC2: A Risk Quantification Framework for Dynamic Access Control in Cloud-based Collaborations, *by Dr. Nirnoy Ghosh*

Thank You!