

# Docker Containers

## CMIS 545 - Cloud Computing Architecture

Rafael Marino, Eduardo Cassinelli

McGill University

November, 2020

- 1 Just Enough Microservices
- 2 Virtual Machines vs Containers
- 3 Docker & Docker Primitives
- 4 Docker Demo
- 5 Beyond 1 Container

## Section 1

# Just Enough Microservices

# What are Microservices?

- Microservices are a good starting point for containers
- Microservices are self-contained, independent, autonomous, loosely coupled services that work together
- No standard definition, they are usually built around a business domain
- Designed specifically to allow independent deployability of each service

# How do Microservices relate to Containers?

- Containers facilitate the modularized development and deployment of microservices
- For this level of independence and flexibility to exist, a clear definition between service boundaries must also exist
- Having services tied to specific machines would be inefficient and risky
- Using one container per service guarantees independence

## Section 2

# Virtual Machines vs Containers

# Virtual Machines

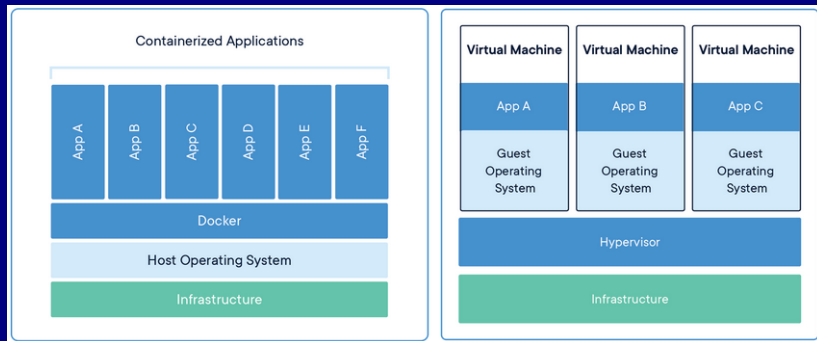
- Virtualization allows the creation of an abstraction layer (Hypervisor) on top of the host OS to divide and virtualize physical resources
- Resources from the host machine are carved out into multiple Virtual Machines
- Each VM runs an independent, guest OS

# Containers

- Containerization is based on the creation of isolated partitions or zones directly from within the host OS, no virtualization software required
- Partitions only have access to their own content, and to specific shared OS Kernel components
- A container is a standard unit of software that can be deployed on any other machine, virtual or otherwise, that runs a compatible OS; independently of any other containers



# Architecture Diagram



**Figure 1:** Container vs VM Architecture

## Section 3

# Docker & Docker Primitives

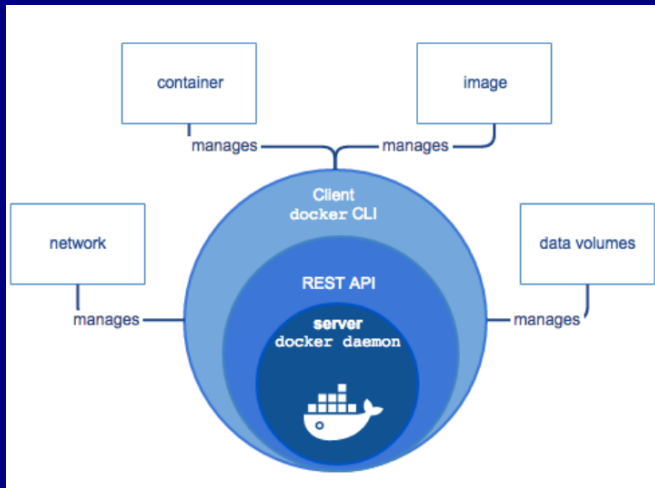
# What is Docker?

- Docker is an open source Platform as a Service for developers and systems engineers to build, ship, and run distributed applications making use of containers
- Docker can package code from an application, including all its dependencies, into a standard container
- Docker includes a comprehensive set of concepts, tools and APIs:
  - Docker Engine
  - Docker Image
  - Docker Container
  - Docker Compose
  - Docker Swarm
  - Data Volumes
  - Networks

# Docker Engine

- The Docker Engine is the underlying software that acts as a client-server for building, executing and orchestrating containers
- It relies on 3 major components:
  - The Docker Daemon
  - The Docker Engine API
  - The Command Line Interface

# Docker Engine Diagram



**Figure 2:** Docker Engine Diagram

# Docker Image

- We can think of a Docker Image as a stopped Docker Container
- Each element within an image represents an image layer. Layers are then stacked on top of each other and ready to run
- Docker Images can be pulled from image registries, which are collections of image repositories
- The most popular repository is the Docker Hub

## Notes

- Docker Inc has recently established rate limits for Docker Hub.
- AWS has announced work on their own public image repository.

# Building a Docker Image

- Define the DockerFile
- The DockerFile declares the starting point of the image: From Scratch or From Parent Image
- It describes and contains all the layers
- The image can be built running the *docker build* command

## Notes

- Images are lightweight. An Ubuntu 20 LTS image weights ~75 MB whereas the full install requires 25 GB
- Images have great boot up speed

# Docker Container

- A Docker Container is a runtime instance of a Docker Image
- It is possible to run many Containers (instances) from one Image
- Containers are the central unit on top of which all Docker is built, and they are better examined practically



## Section 4

### Docker Demo

# Graylog App

- Graylog is an open source log management solution for capturing, storing, and analyzing machine data
- It needs two dependencies:
  - MongoDB: An open-source, “general purpose, document-based, distributed database”
  - Elasticsearch: An open-source, “powerful analytics engine to explore data easily”

# Docker Run

*~\$ docker container run <options> <image>:<tag> <app>*

- <options>: refers to the container command flags
- <image>: refers to the original name or id of the image
- <tag>: refers to the specific version of that image

*~\$ docker container run -d --name ubuntu ubuntu:latest*

# Graylog Setup

```
eduardo@eduardo-L380:~$ docker container run --name mongo -d mongo:3
```

```
eduardo@eduardo-L380:~$ docker run --name elasticsearch \  
-e "http.host=0.0.0.0" \  
-e "ES_JAVA_OPTS=-Xms512m -Xmx512m" \  
-d docker.elastic.co/elasticsearch/elasticsearch-oss:6.8.10
```

```
eduardo@eduardo-L380:~$ docker run --name graylog --link mongo --link  
elasticsearch \  
-p 9000:9000 -p 12201:12201 -p 1514:1514 -p 5555:5555\  
-e GRAYLOG_HTTP_EXTERNAL_URI="http://127.0.0.1:9000/" \  
-d graylog/graylog:3.3
```

**Figure 3:** Graylog Setup Commands

## Note

In Ubuntu 20.04 LTS stock, installing graylog requires adjusting default virtual memory settings using: *sudo sysctl -w vm.max\_map\_count=262144*

# Running Containers

```
eduardo@eduardo-L380:~$ docker container ls
```

CONTAINER ID	IMAGE	CREATED	STATUS	NAMES
1de8801d699f	graylog/graylog:3.3	4 seconds ago	Up 3 seconds (health: starting)	graylog
76b0e75fb7be	./elasticsearch/	22 seconds ago	Up 21 seconds	elasticsearch
229f3aebfe56	mongo:3	27 seconds ago	Up 26 seconds	mongo

**Figure 4:** Running Containers' List

# Testing Graylog

```
eduardo@eduardo-L380: $ echo 'Testing log message for CMIS545 Cloud  
Computing Architecture' | nc localhost 5555
```

Figure 5: Echo Command

The screenshot displays the Graylog web interface. At the top, there's a header with a hamburger menu icon and the text "All Messages". On the right side of the header, there are icons for zooming in and out. Below the header, a table lists log messages. The first message is highlighted. It has a "timestamp" of "2020-11-16 16:39:19 +00:00" and a "source" of "172.17.0.1". The message content is "Testing log message for CMIS545 Cloud Computing Architecture". Below the message content, there's a row of buttons: an envelope icon, the ID "46318500-282a-11eb-a630-0242ac11000", "Permalink", "Copy ID", "Show surrounding messages" (with a dropdown arrow), and "Test against stream" (with a dropdown arrow). Below this row, there's a section titled "Timestamp" with the value "2020-11-16 16:39:19.560". Another section titled "Received by" shows "Text Input on 7 d49f27d0 / 8759155980c7". A section titled "Stored in Index" shows "graylog\_0". A section titled "Routed into streams" shows a list with "All messages". At the bottom, there's a section titled "message" with the value "Testing log message for CMIS545 Cloud Computing Architecture". Below that, a section titled "source" shows "172.17.0.1". Finally, a section titled "timestamp" shows "2020-11-16 16:39:19 +00:00".

timestamp	source
2020-11-16 16:39:19 +00:00	172.17.0.1

Testing log message for CMIS545 Cloud Computing Architecture

✉ 46318500-282a-11eb-a630-0242ac11000 [Permalink](#) [Copy ID](#) [Show surrounding messages](#) [Test against stream](#)

**Timestamp**  
2020-11-16 16:39:19.560

**Received by**  
Text Input on 7 d49f27d0 / 8759155980c7

**Stored in Index**  
graylog\_0

**Routed into streams**

- All messages

**message**  
Testing log message for CMIS545 Cloud Computing Architecture

**source**  
172.17.0.1

**timestamp**  
2020-11-16 16:39:19 +00:00

Figure 6: Graylog Dashboard

# Stopping and Removing Containers

```
eduardo@eduardo-L380:~$ docker container stop mongo
```

```
eduardo@eduardo-L380:~$ docker container ls -a
```

CONTAINER ID	IMAGE	CREATED	STATUS	NAMES
1de8801d699f	graylog/graylog:3.3	2 minutes ago	Up 2 minutes (healthy)	graylog
76b0e75fb7be	./elasticsearch/	53 seconds ago	Up 52 seconds	elasticsearch
229f3aebfe56	mongo:3	2 minutes ago	Exited(0) 5 seconds ago	mongo

**Figure 7:** Stopping Single Container

```
eduardo@eduardo-L380:~$ docker container stop mongo elasticsearch graylog
```

```
eduardo@eduardo-L380:~$ docker container rm mongo elasticsearch graylog
```

**Figure 8:** Stopping and Removing all Containers

# Further Container Commands[1]

Command	Description
<i>docker container prune</i>	Remove all stopped containers
<i>docker container start</i>	Start one or more stopped containers
<i>docker container diff</i>	Inspect file or directory changes
<i>docker container exec</i>	Run a command in a running container
<i>docker container export</i>	Export a container's filesystem as a tar
<i>docker container inspect</i>	Display detailed information
<i>docker container kill</i>	Kill one or more running containers
<i>docker container logs</i>	Fetch the logs of a container

[1]

Documentation: [https://docs.docker.com/engine/reference/commandline/container\\_run/](https://docs.docker.com/engine/reference/commandline/container_run/)



## Section 5

# Beyond 1 Container

# Multiple Containers and 1 host: Docker Compose

- Compose is an orchestrator tool for multiple-container applications (usually microservices)
- An app designed using microservices will need 1 container per microservice
- Compose is best suited for development and automated-testing environments, given its capacity to create multiple isolated environments on a single host
- Compose is useful but it doesn't work to scale out of a single host into distributed systems

# Multiple Containers and Hosts: Docker Swarm

- Docker Swarm is Docker's distributed computation solution and a cluster management system
- A cluster of Docker Engines (Hosts) is called a swarm
- The key concepts to understand Swarm are: nodes, services & tasks, and load balancing
- A node refers to an instance of a Docker Engine
- Load balancing. Balancing and work distribution can be configured internally or be outsourced to an external balancer

## Notes

- Swarm competes with Kubernetes