Distributed Systems

Tutorial #3
Docker Containers

Yaron Hay
Largely based on the slides of
Amir Dachbash

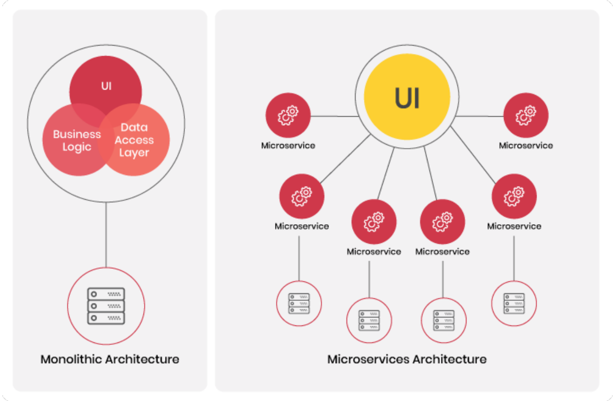
Technion
Israel Institute of Technology
Faculty of Computer Science

Fall 2024/25
November 27, 2024

v1.0 27/11/2024

Motivation
Microservices

- Microservices are self-contained and independent application units
- Each microservice fulfills one specific business function
  - But, still considered as a small application
- What would happen if you create a lots of microservices for your app?
  - Resource scaling would be hard to accomplish
  - Increased deployment complexity
  - But, it worked on my machine!



Docker

Key Concept

Docker is a software platform that allows you to **build**, **test**, and **deploy** applications quickly.

- Flexible
- Lightweight
- Portable
- Loosely Coupled
- Scalable
- Secure

- Docker packages software into standardized units called containers.
- Images that are built using docker files, create containers.
- Images are distributed using registries.
- Containers include everything the software needs to run e.g. libraries, system tools, code, and runtime.

Docker

Images

Key Concept

An image is a **read-only** template with instructions for creating a Docker container.
An image has a designated command to be executed.

You can create a Docker image in one of two ways:

- Interactive Method
  1. Run a container from an existing Docker image.
  2. Manually change that container environment through a series of live steps.
  3. Save the resulting state as a new image.
- Building it from a *Dockerfile*

TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

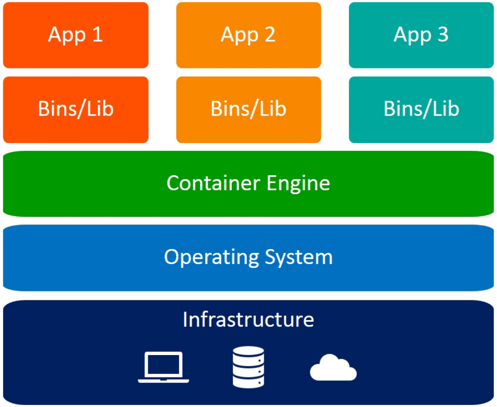
Docker

Containers

Key Concept

A container is a runtime instance of an image, that *runs until the command completes*. Containers are relatively well isolated from other containers and the host machine, but still lightweight.

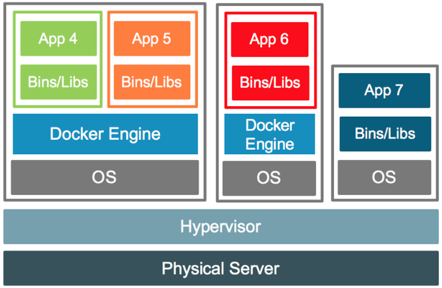
- You can create, start, stop, move, or delete a container using the Docker API or CLI.
- You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state
- The isolation level can be controlled.

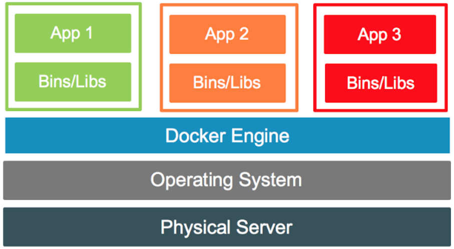


TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

Docker

Containers vs Virtual Machines





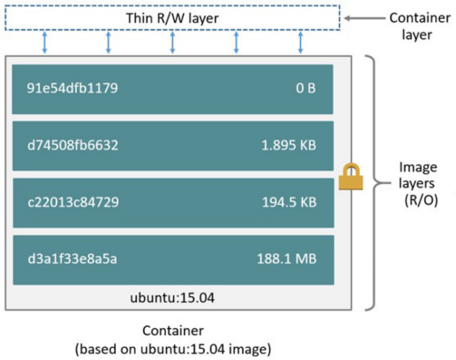
Docker
Containers vs Virtual Machines

	Virtual Machines	Docker Containers
Underlying Framework	Hypervisor	Docker Engine (directly on host's kernel)
Isolation	Strict	Loose
Typical Size	GBs	MBs
Startup Time	Minutes	Seconds
Use Cases	Production of "heavy systems" e.g, datacenter	Development, testing and distribution of an applications

Table: A comparison between Docker Containers and Virtual Machines

Docker
Layers

- A Docker image is built up from a series of layers.
- Each layer is read-only (except the last one).
- Each layer is only a set of differences from the layer before it.
    - Files are changed in a Copy-on-write manner.
- When you create a new container, a new writable layer is added on top of the underlying layers (AKA the "container layer").
    - All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer.
    - The files won't persist after the container is deleted

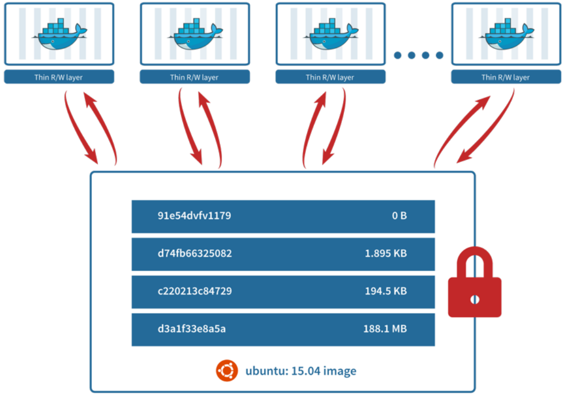


TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

Docker

Layers

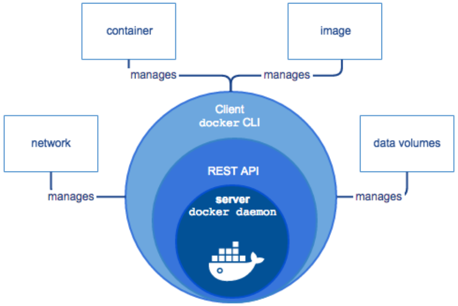
- When you the Dockerfile is changed and rebuilt, only the layers that changed are rebuilt.
- Multiple containers can share access to the same underlying image and yet have their own data state.



Docker

Docker Engine

- Docker Engine is is installed on the host machine.
- The docker engine follows the client-server architecture.
- The docker engine has 3 components:
  1. **The Docker Daemon** called **dockerd** (the server). It can create and manage docker images, containers, networks, etc.
  2. **REST API** used to instruct the docker daemon.
  3. **Command Line Interface** (CLI) for clients to control the docker daemon.



Docker

Registry

Key Concept

A stateless, highly scalable server-side application that **stores** and **distributes** Docker images.

- A Docker registry is organized into Docker repositories.
- A repository holds all the versions of a specific image. A version can be accessed by a corresponding *tag*.
- Allows Docker engines can
  - Pull images from the registry (locally)
  - Push new images to the registry
- If no registry is specified, the Docker engine uses with DockerHub, Docker's public registry instance.

Docker

Using the Registry

Interact with a Registry

```
docker pull ubuntu:16.04
docker tag  ubuntu:16.04 myregistry.domain.com/my-ubuntu
docker push myregistry.domain.com/my-ubuntu
docker pull myregistry.domain.com/my-ubuntu
```

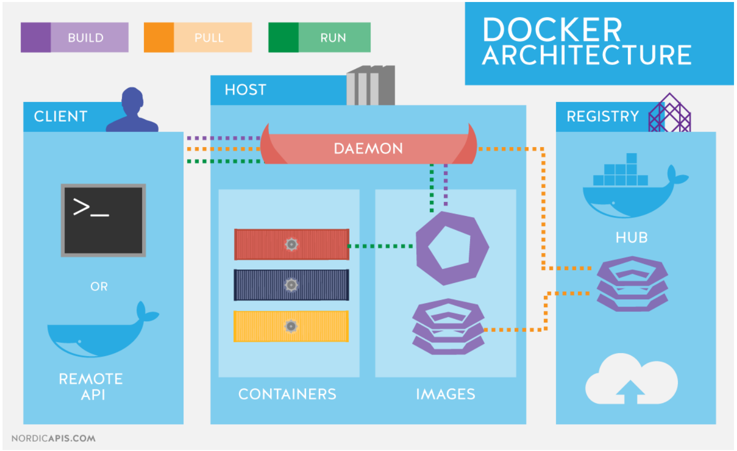
Create your own Registry

```
docker run -d --restart=always --name registry -v "$(pwd)"/certs:/certs \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:443 \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  -p 443:443 registry:2
```

Docker

Architecture

- Developers build images and push them to the registry.
- Clients contact the docker engine via REST API or a CLI.
- Images are pulled from the image registry.
- The docker daemon creates containers from images.



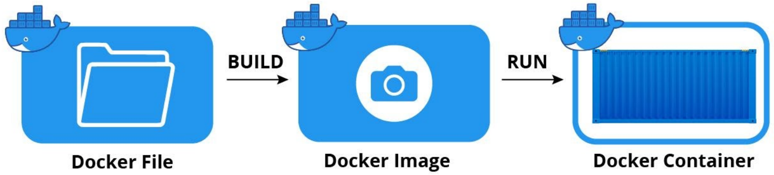
TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

Docker Files

Key Concept

A Dockerfile is a file containing a list of instructions that inform the daemon on how to build a docker image.

- Each layer is represented by a **single** instruction in the image's Dockerfile.
- Often, an image is based on another image, with some additional customization.
    - For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application



Docker Files

Commands

- **FROM** – Defines the base image that is used as the first layer in the build process.
- **RUN** – Executes the command with its arguments in a new layer.
- **ADD** – Copies the files from sourceon the host to the destination inside the container.
- **CMD** – A command, that gets executed only after the container has been instantiated.
  - **ENTRYPOINT** - Also append the specified command line arguments
- **ENV** – Sets environment variables.
- **WORKDIR** – Sets the working directory for any **RUN**, **CMD**, **ENTRYPOINT**, **COPY** and **ADD** instruction that come after in the Dockerfile.
- **EXPOSE** – Informs Docker that the container listens to the specified network ports during runtime. You can specify whether the container listens for TCP or UDP traffic, (the default is TCP if the protocol is not specified).

Full specification can be found at the Docker Builder Documentation.

Docker Files

Example

```
FROM node:10-slim

RUN apt-get update \
    && apt-get install -y
    ↪  --no-install-recommends \
    curl \
    && rm -rf /var/lib/apt/lists/*

ENV TINI_VERSION v0.19.0

ADD https://github.com/...
↪  .../${TINI_VERSION}/tini /tini
RUN chmod +x /tini

WORKDIR /app
RUN npm install -g nodemon

COPY package*.json ./

RUN npm ci \
 && npm cache clean --force \
 && mv /app/node_modules /node_modules

COPY . .

ENV PORT 80
EXPOSE 80

CMD ["/tini", "--", "node", "server.js"]
```

Docker Networking

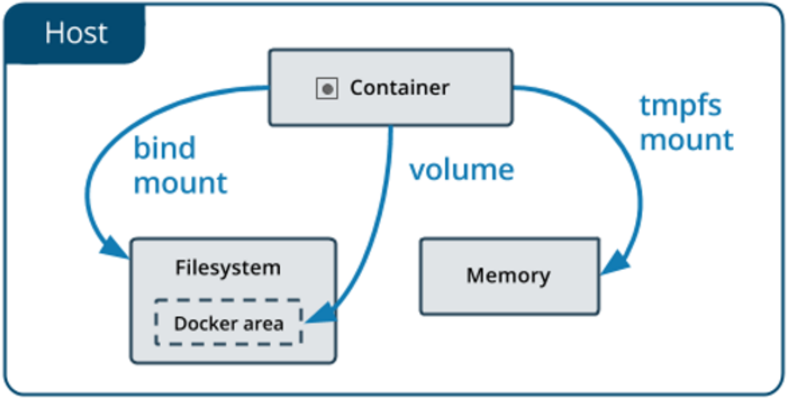
Network Types

- **Bridge** [The default network driver]
  - Usually used when your applications run in standalone containers that need to communicate with each other
- **Host**
  - Removes the network isolation between the container and the host
  - Allows for directly using the host's networking
- **Overlay**
  - Connects multiple Docker daemons together
  - Enables swarm services to communicate with each other
  - Facilitates communication between two standalone containers on different Docker daemons
- **MACvLAN Networks**
  - Allows you to assign a MAC address to a container, making it appear as a physical device on your network.
  - Traffic is routed to containers by their MAC addresses.
  - Best choice when for legacy applications expecting to be directly connected to the physical network.
- **None** Disable all networking

Storage

Storage Drivers

How does the container access the persistent storage on the host?



Storage

Volumes

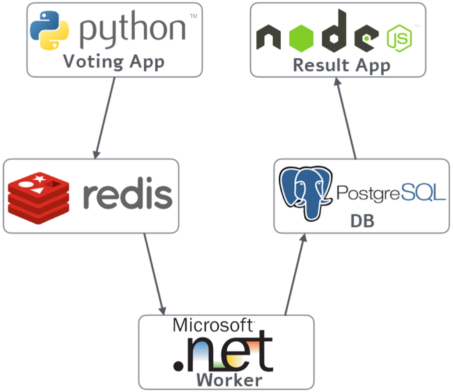
- The recommended way to persist data in Docker.
- Stored in a part of the host file-system which is managed by Docker
- Persists even after a container is deleted.
  - `/var/lib/docker/volumes/` on Linux
- Non-Docker processes should not modify this part of the filesystem (Convention).
- They are completely managed by docker through docker CLI or Docker API.
- To start a container with a volume is done by –v flag in the run command.
  - `$ docker run  v  <host dir>:<container dir>` ···
  - `$ docker run  v  /opt/data:var/lib/mysql mysql`

TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

Storage
Other Options

- Bind Mounts
  - May be stored anywhere on the host system. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
- tmpfs (Linux)
  - Are stored in the host system's memory only
  - Are never written to the host system's filesystem.
  - Why not writing to the 'Container layer'?
    - Performance or security reasons or if the data relates to non-persistent application state.
- Named pipes (Microsoft)

Docker Compose



Source at https://github.com/dockersamples/example-voting-app.

Docker Compose

Example

```yaml
services:
  vote:
    build: ./vote
    command: python app.py
    depends_on:
      redis:
        condition: service_healthy
    volumes:
     - ./vote:/app
    ports:
      - "5000:80"
    networks:
      - front-tier
      - back-tier
  result:
    build: ./result
    # use nodemon rather than node for local dev
    command: nodemon server.js
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
    networks:
      - front-tier
      - back-tier
  worker:
    build:
      context: ./worker
    depends_on:
      redis:
        condition: service_healthy
      db:
        condition: service_healthy
    networks:
      - back-tier
  redis:
    image: redis:5.0-alpine3.10
    volumes:
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
    ports: ["6379"]
    networks:
      - back-tier
  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
    volumes:
      - "db-data:/var/lib/postgresql/dat
      - "./healthchecks:/healthchecks"
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
    networks:
      - back-tier
volumes:
  db-data:
networks:
  front-tier:
  back-tier:
```

Container Orchestration

Why?

Run Containers in a Cluster Mode.

- Redundancy
- High Availability
- Scalability
- Health Monitoring [of containers and hosts]

Further Reading

Docker Swarm `https://docs.docker.com/engine/swarm/key-concepts/`.
Kubernetes `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`.

Container Orchestration

Docker Swarm

- A swarm consists of multiple Docker hosts which run in swarm mode and act as managers and workers.
- A given Docker host can be a manager, a worker, or perform both roles.
- When you create a service, you define its optimal state (number of replicas, network and storage resources available to it, etc..), Docker works to maintain that desired state.
- For instance, if a worker node becomes unavailable, Docker schedules that node's tasks on other nodes.
    - A task is a running container which is part of a swarm service and managed by a swarm manager

Further Reading

- More advanced material https://docs.docker.com/get-started/overview/
    - **Namespaces** - provide the isolated workspace called the container.
    - **cgroups** (control groups) - used to limit and isolate the resource usage (CPU, memory, Disk I/O, network, etc.)
    - **UnionFS** (Union file systems)
- Docker and Microservices
    - https://www.sumologic.com/insight/microservices-architecture-docker-containers/
    - https://timber.io/blog/docker-and-the-rise-of-microservices/
- Running linux containers on windows ?
    - https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers

TECHNION | The Henry and Marilyn Taub Faculty of Computer Science