Docker

Teaching Faculty: Umur INAN

Prepared by Umur INAN

Docker

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host.

CONTAINER

- Container is a software package that consists of all the dependencies required to run an application.

- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

- Containerization is simply a way of packaging software applications into containers.

VIRTUALIZATION

- Virtualization is the process of creating a simulated computing environment that's abstracted from the physical computing hardware.

- Virtualization allows you to create multiple, virtual computing instances from the hardware and software components of a single machine.

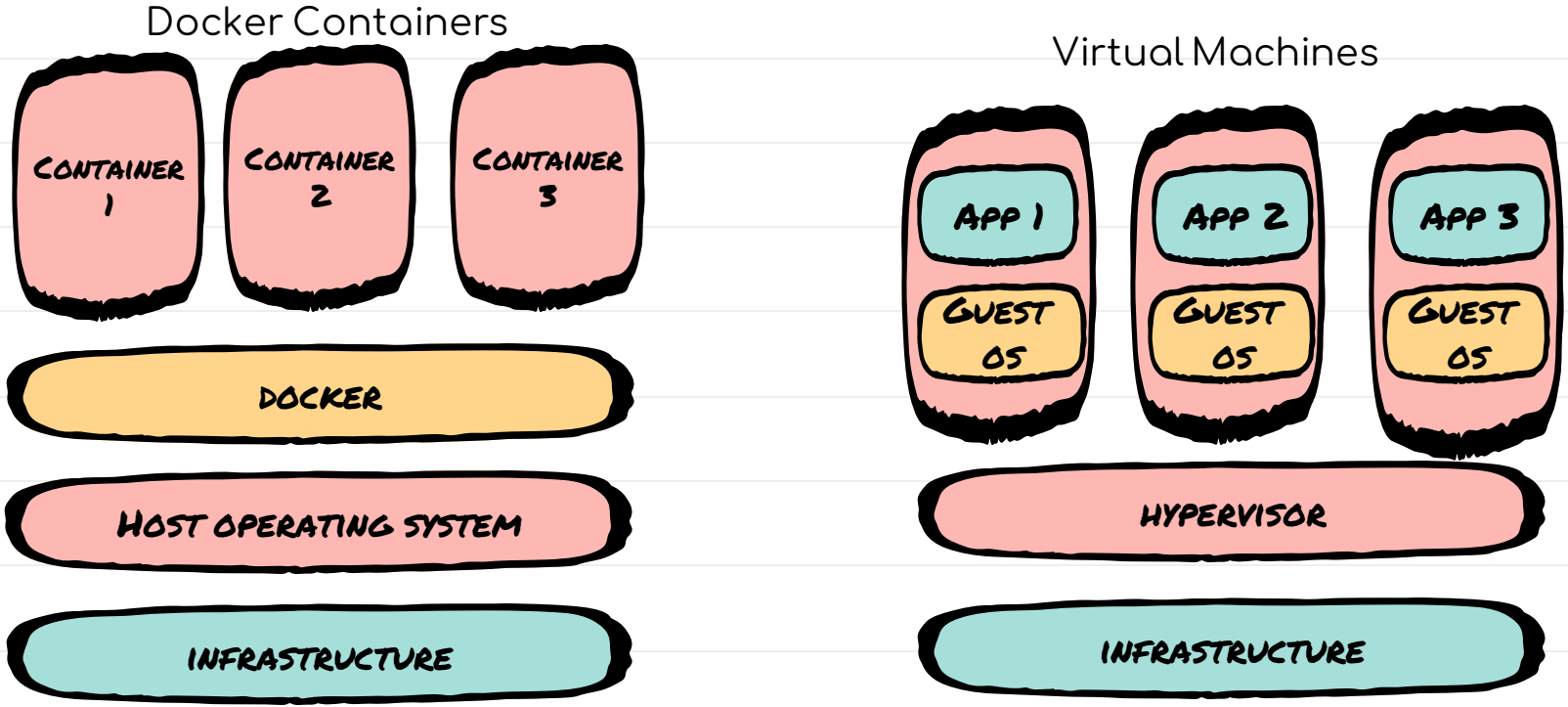
Hypervisor

- The software that enables virtualization is called a hypervisor.
- It's a lightweight software layer that sits between the physical hardware and the virtualized environments and allows multiple operating systems (OS) to run in tandem on the same hardware.
- The hypervisor is the middleman that pulls resources from the raw materials of your infrastructure and directs them to the various computing instances.

Virtual machines

- Separate computers running on hardware that is contained in one physical computer.
- Each VM requires its own OS.
- The OS and any applications running on an individual VM share hardware resources from a single host server, or from a pool of host servers.

Docker vs Virtualization



Building Docker images

- Docker image is built using a Docker file.
- A docker image contains all the project's code, whereas a Dockerfile is a text file which contains commands for building a Docker image.
- With docker images containers can be built.
- From Docker Hub users can pull any Docker Image and build new containers.

Docker Hub

- Docker Hub is a hosted repository service provided by Docker for finding and sharing container images.

- Users get access to free public repositories for storing and sharing images or can choose subscription plan for private repos.

Docker compose

- Docker Compose is used for running multiple containers as a single service.
- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your application's services.

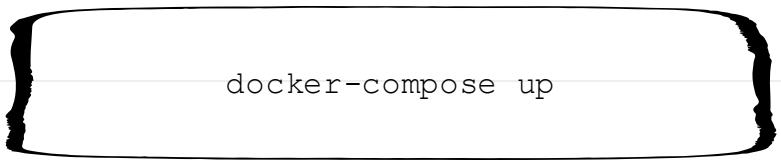
Docker compose

- Then, with a single command, you create and start all the services from your configuration.

- Docker Compose works by applying many rules declared within a single docker-compose.yml configuration file.

DOCKER-COMPOSE.YML

- Contains
  - services
  - volumes (optional)
  - networks



Volumes + networks

- Volume
  - is a shared directory in the host, visible from a container.

- Network
  - define the communication rules between containers, and between a container and the host.
  - Common network zones will make containers' services discoverable by each other, while private zones will segregate them in virtual sandboxes.

SERVICES

- Refers to containers' configuration.
- Let's dockerize an app consisting of frontend-app, backend-app, database and app-umur.

```
services:
  frontend-app:
    image: my-frontend-app
    ports:
      - "8080:2000"
  backend-app:
    image: my-backend-app
  db:
    image: postgres
  umur-app:
    image: awesome-app
```

Port 2000 will be available in 8080 on the host.

SERVICES

```yaml
services:
  frontend-app:
    image: my-frontend-app
    ports:
      - "8080:2000"
    networks:
      - network1
  backend-app:
    image: my-backend-app
    networks:
      - network1
  db:
    image: postgres
  umur-app:
    image: awesome-app
    networks:
      - network2
```

```yaml
networks:
  network1: {}
  network2: {}
```

my-backend-app can reach frontend-app whereas it cannot reach umur-app.

VOLUMES

```
services:
  frontend-app:
    image: my-frontend-app
    ports:
      - "8080:2000"
    networks:
      - network1
    volumes:
      - /tmp:/volume1/from-host
  backend-app:
    image: my-backend-app
    networks:
      - network1
  db:
    image: postgres
  umur-app:
    image: awesome-app
    networks:
      - network2
networks:
  network1: {}
  network2: {}
```

The /tmp folder of the host is mapped to /volume1/from-host folder of the container.

Docker compose commands

- `docker compose up`
  - Creates and start containers.
- `docker compose down`
  - Stops and removes containers.
- `docker compose create`
  - Creates containers for a service.
- `docker compose ls`
  - Lists running compose projects.

Docker commands

- `docker ps`
  - Lists the running containers.
- `docker ps -a`
  - Lists all the running and exited containers.
- `docker stop <container id>`
  - Stops a running container
- `docker images`
  - Lists all the locally stored docker images.

Docker commands

- `docker rm <container id>`
  - Deletes a stopped container.
- `docker build <path to docker file>`
  - Builds an image from a specified docker file.
- `docker run –pHOST:CONTAINER -d –-name custom-name –net network-name name-of-image`

Docker commands

- `docker logs container-id`
- `docker pull name-of-image`
- `docker exec –it container-id /bin/bash`
- `docker start name-of-image`

Docker commands

- `docker network ls`
- `docker network create name-of-network`
- `docker exec –it container-id /bin/bash`
- `docker start name-of-image`

Dockerfile

- Each Dockerfile is a script, composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image in order to create (or form) a new one.

- Dockerfiles begin with defining an image FROM which the build process starts. Followed by various other methods, commands, and arguments (or conditions), in return, provide a new image which is to be used for creating docker containers.

```
docker build
```

Dockerfile commands

- `MAINTAINER`
  - Sets the author.
  - `MAINTAINER umur_inan`
- `ADD`
  - Takes 2 arguments, a source and a destination.
  - It copies files from source on the host machine into the container.
  - The source can also be a URL.
  - `ADD /from/host /to/container`

Dockerfile commands

- `CMD`
  - Executes a specific command.
  - It is not executed during build but when the container is created/started.
  - `CMD “echo” “Hello World”`
- `RUN`
  - It is executed during build unlike CMD command.

Dockerfile commands

- `FROM`
  - Specifies the base image to use to create the container.
  - It can be default images from docker hub or your own custom image.
  - `FROM <image>`
  - `FROM <image>:<tag>`
  - Tag is `latest` by default.

Dockerfile commands

- `WORKDIR`
  - Sets the current path or the path where `CMD`, `RUN`, `COPY`, `ADD` commands going to be executed.
  - `WORKDIR /source`
- `VOLUME`
  - Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

Dockerfile commands

- `EXPOSE`
  - EXPOSE is used as documentation for the port. This is just a communication between the person who builds the image and the person who runs the container.
  - `EXPOSE 8080`

Dockerfile commands

- `docker images`
  - List all images.

- `docker run -it -d image-name`
  - Run the image.

- `docker build -t image-name:tag-name .`
  - Builds a docker image.

Dockerfile

```
# Comment Line
FROM ubuntu
MAINTAINER uinan@miu.edu

WORKDIR /home/umurinan

RUN apt-get update
RUN apt-get install –y nginx

CMD [“echo”, ”Hello World”]

COPY from-my-computer to-container
```

Spring Boot -- Docker

```
FROM openjdk

WORKDIR /path/to/jar/file
COPY app.jar app.jar
EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```