Docker Course

Done By: Gabriel Cassim

Setup & Code:

Get code at https://github.com/Stoksweet/docker-intro-course

What is docker and containerization?

Lesson 1: Introduction to Docker

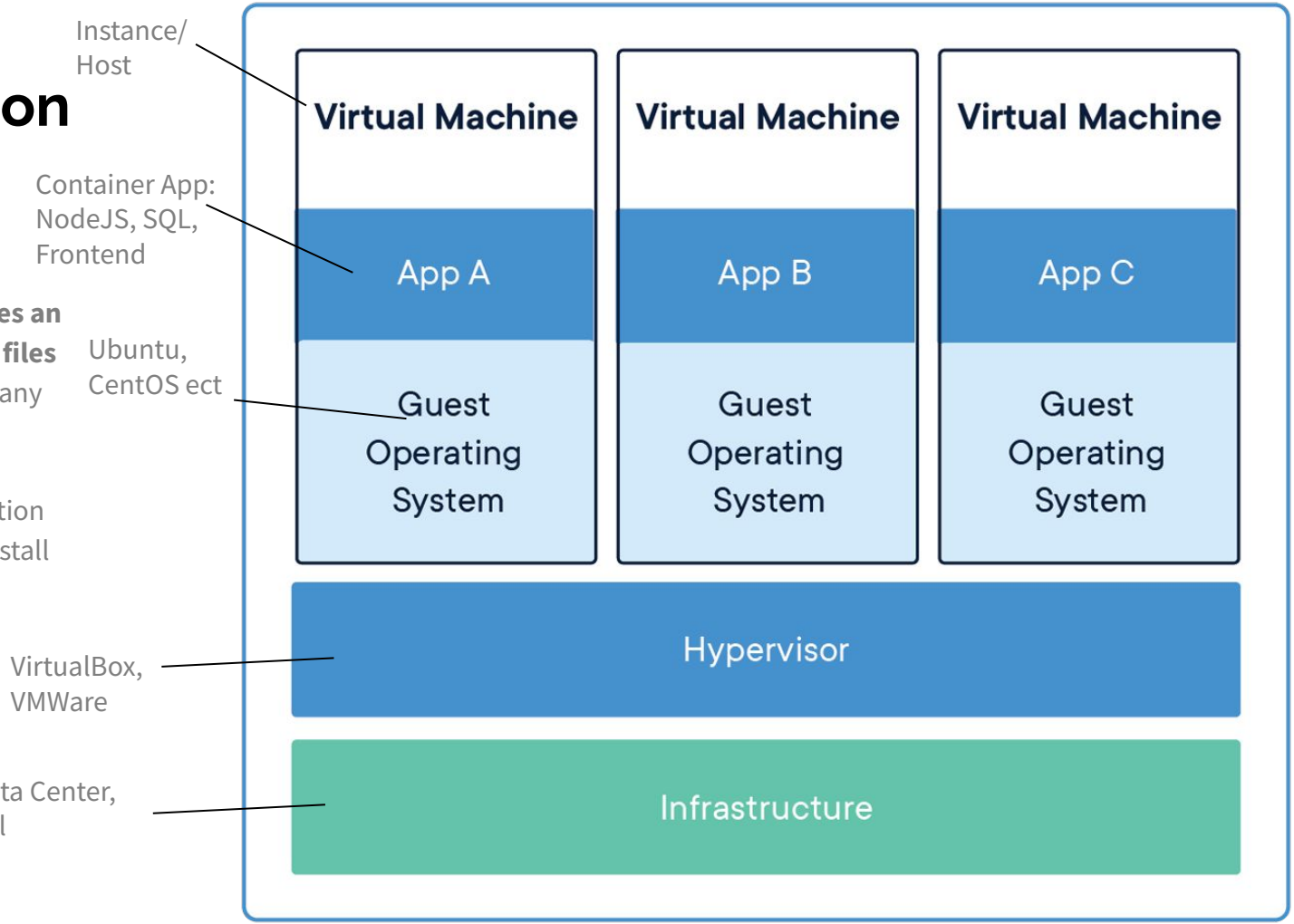
In this lesson, we will delve into the fundamental concepts of Docker and containerization. Docker is a platform that enables the development, deployment, and running of applications in containers. **Containers encapsulate an application and its dependencies**, ensuring consistency across different environments.

We'll discuss the advantages of using Docker, such as improved portability, resource efficiency, and simplified deployment processes. The exercise for this lesson involves installing Docker on your machine, setting the stage for hands-on exploration.

Containerization Explained

Containerization is a software deployment process that **bundles an application's code with all the files and libraries** it needs to run on any infrastructure.

Traditionally, to run any application on your computer, you had to install the version that matched your machine's operating system.



Exercise:

How to Install Docker & Run hello-world

Lesson 2: Docker Architecture

Lesson 2: Docker Architecture

Understanding the architecture of Docker is crucial for effective usage. We'll dissect the roles of the **Docker daemon, Docker client, Docker images, containers, and the Docker registry**.

- The daemon manages containers, while the client serves as the interface for users.
- Docker images are the blueprints for containers, and the registry stores and distributes these images.

We'll explore basic commands to interact with the Docker daemon and gain insights into the overall architecture.

Docker Components

Docker Daemon

The Docker daemon is a background process that manages Docker containers on a host system, handling container execution, communication, and resource management.

Docker Client

The Docker client is the user interface that allows users to interact with the Docker daemon, issuing commands to build, manage, and deploy containers.

Docker Containers and Images

Docker images are lightweight, portable, and executable packages that include application code, libraries, dependencies, and a runtime, providing a consistent environment for running applications across different environments.Docker containers are instances of Docker images that run as isolated processes, encapsulating applications and their dependencies, ensuring consistency and efficiency in deployment.

Docker Registry

The Docker registry is a centralized repository for storing and sharing Docker images, enabling easy distribution and retrieval of container images across different systems and environments.

Exercise:

Exploring Docker Desktop, and deploying a game application using the Docker CLI

Lesson 3: Docker Images

Lesson 3: Docker Images

This lesson focuses on **Docker images** and the **creation process using Dockerfiles**.

- Docker images are **snapshots of a file system and application code**, providing a reproducible and shareable environment.
- We'll delve into the anatomy of Docker images, including layers, and learn how to build custom images using **Dockerfiles**.

The practical exercise involves creating a simple Docker image with a customized Dockerfile, reinforcing the concept of image creation.

Sample Dockerfile

```
FROM node:18.7.0

WORKDIR /code

COPY package.json package.json
COPY package-lock.json package-lock.json

RUN npm install

COPY . .

CMD [ "node", "server.js" ]
```

- **FROM**: Specifies the base image for the Docker image.
- **WORKDIR**: Sets the working directory for subsequent commands.
- **COPY/ADD**: Copies files from the local machine to the image filesystem.
- **RUN**: Executes commands during the image build process.
- **CMD/ENTRYPOINT**: Defines the default command to run when the container starts.
- **EXPOSE**: Informs Docker that the container listens on specific network ports at runtime.
- **ENV**: Sets environment variables in the image.
- **VOLUME**: Creates a mount point for external volumes.
- **USER**: Sets the user or UID to use when running the image.
- **LABEL**: Adds metadata to the image in key-value pairs.

MELSOFT ACADEMY

Exercise:
Create and Build Docker Image using Dockerfile

Lesson 4: Docker Containers

Lesson 4: Docker Containers

Now that we understand images, we'll shift our focus to Docker containers.

- Containers are the **executable packages that include everything needed to run an application**, making it easy to deploy and scale.
- We'll explore commands for **creating, managing, and interacting** with Docker containers.

The exercise for this lesson involves running a basic container, examining its lifecycle, and gaining proficiency in container management.

Docker Commands

- **docker run <image>:<tag>**: Create and start a container from a specific image.
- **docker ps**: List running containers.
- **docker ps -a**: List all containers, including stopped ones.
- **docker stop <container_id or container_name>**: Stop a running container.
- **docker start <container_id or container_name>**: Start a stopped container.
- **docker restart <container_id or container_name>**: Restart a container.
- **docker pause <container_id or container_name>**: Pause a running container.
- **docker unpause <container_id or container_name>**: Unpause a paused container.
- **docker rm <container_id or container_name>**: Remove a stopped container.
- **docker exec -it <container_id or container_name> <command>**: Execute a command inside a running container.
- **docker logs <container_id or container_name>**: View the logs of a container.
- **docker inspect <container_id or container_name>**: Display detailed information about a container.

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

Exercise:
Explore Container Lifecycle Commands with CLI

Lesson 5: Docker Volumes

Lesson 5: Docker Volumes

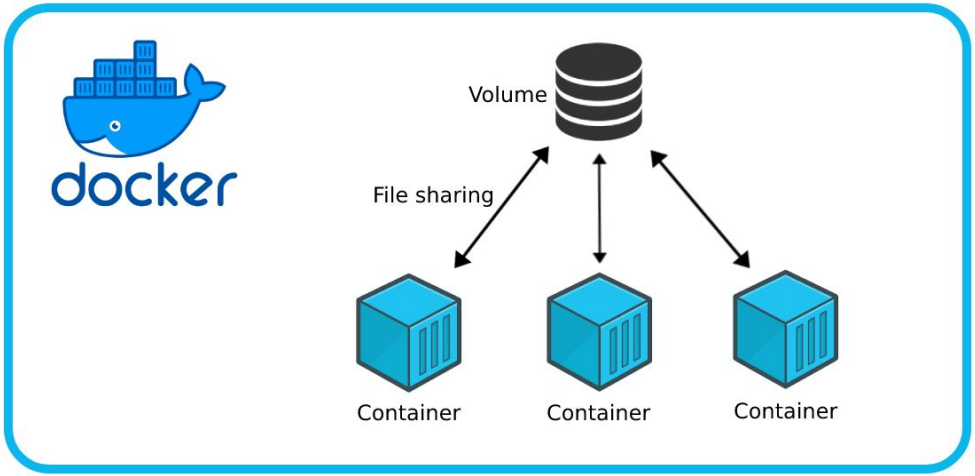
Data persistence is a critical aspect of containerized applications, and Docker volumes play a key role in achieving this.

- We'll discuss the importance of **data volumes for persistent storage** and explore how to **attach volumes to containers**.

The hands-on exercise involves mounting a volume to a container, emphasizing the practical aspects of data persistence in Docker.

Docker Volume Commands

- **docker volume inspect <volume_name>:** Display detailed information about a specific Docker volume.
- **docker volume rm <volume_name>:** Remove a Docker volume
- **docker run -v <host_path>:<container_path> <image>:<tag>:** Mount a host directory or create a named volume inside a container.
- **docker inspect --format='{{.Mounts}}' <container_id or container_name>:** View the volumes associated with a running container.
- **docker cp <local_file> <container_id or container_name>:<container_path>:** Copy files or directories between the host and a running container.
- **docker-compose -f <docker-compose-file.yml> up -d:** Use Docker Compose to start containers defined in a compose file, including any specified volumes.



Exercise:

Mounting volumes
to docker containers.

Lesson 6: Docker Networking

Lesson 6: Docker Networking

This lesson introduces Docker networking, covering the **basics of how containers communicate with each other and the external world**.

- We'll explore the **creation of Docker networks, connecting containers**, and understanding the nuances of container networking.

The exercise includes **setting up a network, connecting containers, and witnessing the communication** between them.

Why use Docker Networks?

Pros of Using Docker Networks:

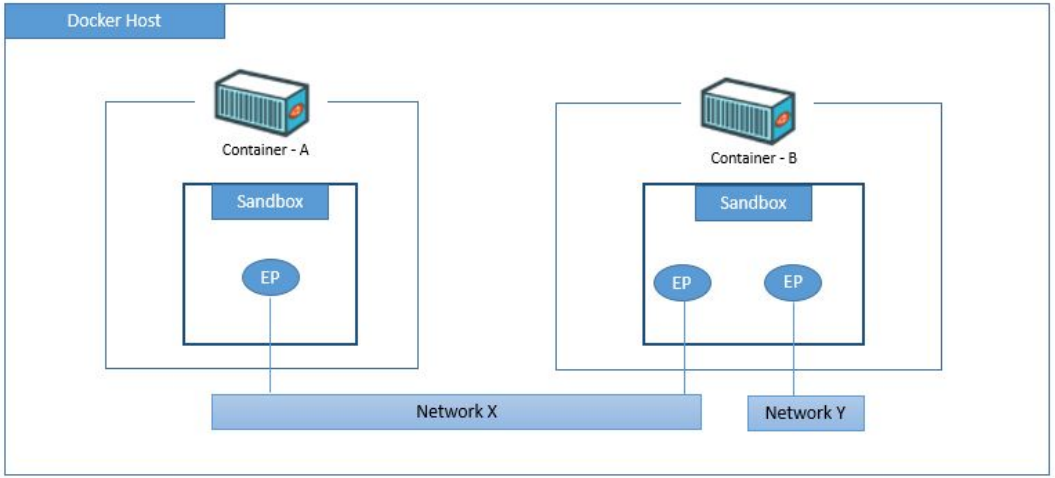
- Isolation and Segmentation
- Service Discovery
- Scalability
- Built-in DNS Resolution
- Load Balancing
- Custom Bridge Networks

Cons of Using Docker Networks:

- Complexity for Simple Applications
- Learning Curve
- Potential for Overhead
- Resource Utilization
- Security Concerns if Misconfigured
- Network Namespacing Overhead

Docker Networks Commands

- **docker network create <network_name>:** Create a new Docker network.
- **docker network ls:** List all Docker networks on the system.
- **docker network inspect <network_name>:** Display detailed information about a specific Docker network.
- **docker network rm <network_name>:** Remove a Docker network.
- **docker run --network <network_name> <image>:<tag>:** Connect a container to a specific Docker network during creation.
- **docker network connect <network_name> <container_id or container_name>:** Connect an existing container to a Docker network.
- **docker network disconnect <network_name> <container_id or container_name>:** Disconnect a container from a Docker network.



Exercise:

Create network, connect containers and test connection.

Lesson 7: Docker Compose

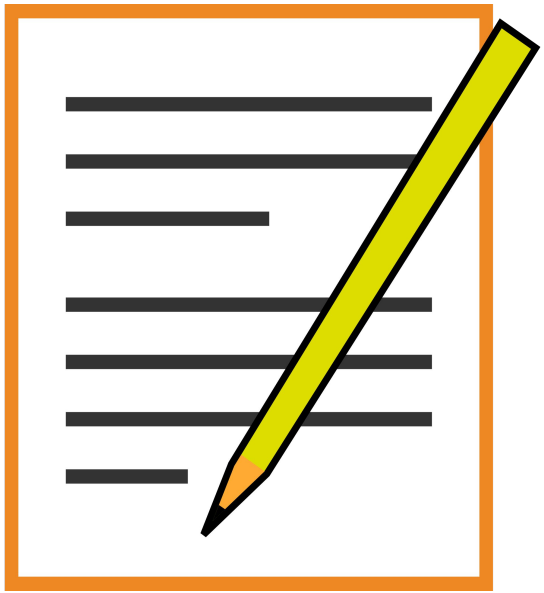
Lesson 7: Docker Compose

Docker Compose simplifies the **orchestration of multi-container applications**.

- We'll explore the **Compose syntax, creating a docker-compose.yml file** to define and manage multi-container applications.

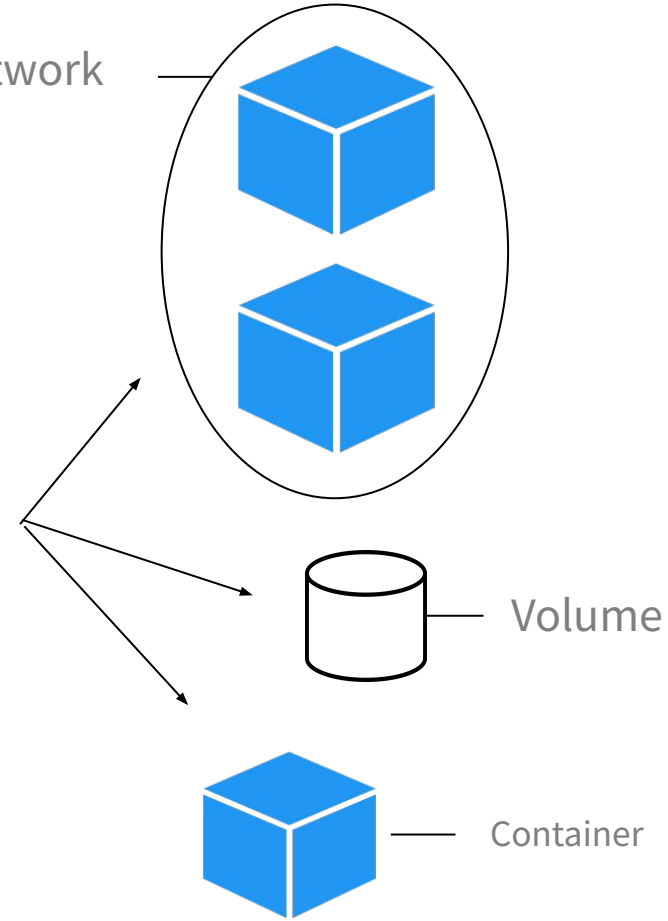
The practical exercise involves composing a simple application with multiple services, providing a hands-on experience in using Docker Compose.

Docker Compose

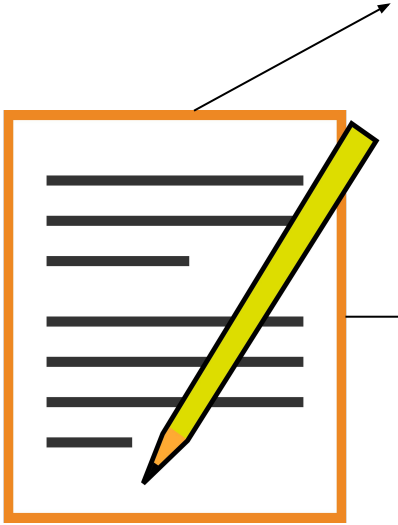


docker-compose.yaml

docker-compose up
Docker compose down



Full Stack App Scenario



Frontend

HTML CSS JS

node JS

Port: 80

Backend

mongoDB.

Port: 27017

SQL

Port: 1433

Docker Compose File

Dockerfile

```
FROM node:14

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "app.js"]
```

docker-compose.yaml

```
version: '3'
services:
  frontend:
    build: .
    ports:
      - "3000:3000"
  mongo:
    image: "mongo:latest"
```

Exercise:

Create and use docker compose for a Node Project

Lesson 8: DockerHub / Registries

Lesson 7: Docker Compose

Docker Hub is a central repository for Docker images, facilitating collaboration and sharing of containerized applications.

- This lesson covers the basics of Docker Hub, including image tagging, pushing, and pulling images.

The exercise involves interacting with Docker Hub, pushing a custom image, and pulling images for various use cases.

Docker Tag & Push

Docker Tag

```
root@iamrj846:/home/jarvis# docker tag ubuntu:latest ubuntu:myubuntu
root@iamrj846:/home/jarvis# docker images
REPOSITORY   TAG         IMAGE ID       CREATED      SIZE
ubuntu       latest      7e0aa2d69a15   9 days ago   72.7MB
ubuntu       myubuntu    7e0aa2d69a15   9 days ago   72.7MB
root@iamrj846:/home/jarvis#
```

Docker Push

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://h
ub.docker.com to create one.
Username: sarab
Password:
Login Succeeded
$ docker push nginx
The push refers to repository [docker.io/library/nginx]
6c7de695ede3: Layer already exists
2f4accd375d9: Layer already exists
ffc9b21953f4: Layer already exists
errors:
denied: requested access to the resource is denied
unauthorized: authentication required
```

Docker Pull

Docker Tag

```
jarvis@iamrj846:~$ docker pull busybox:latest
latest: Pulling from library/busybox
f531cdc67389: Pull complete
Digest: sha256:ae39a6f5c07297d7ab64dbd4f82c77c874cc6a94cea29fdec309d0992574b4f7
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
jarvis@iamrj846:~$
```

Exercise:

Learn to tag, push and pull your docker image with DockerHub

Congrats!

We have made it to the end of our short course on Docker. We hope you enjoyed this!