Environments

DEVELOPMENT

Write code

Developers

TESTING

Verify code

QA

STAGING

Test infrastructure

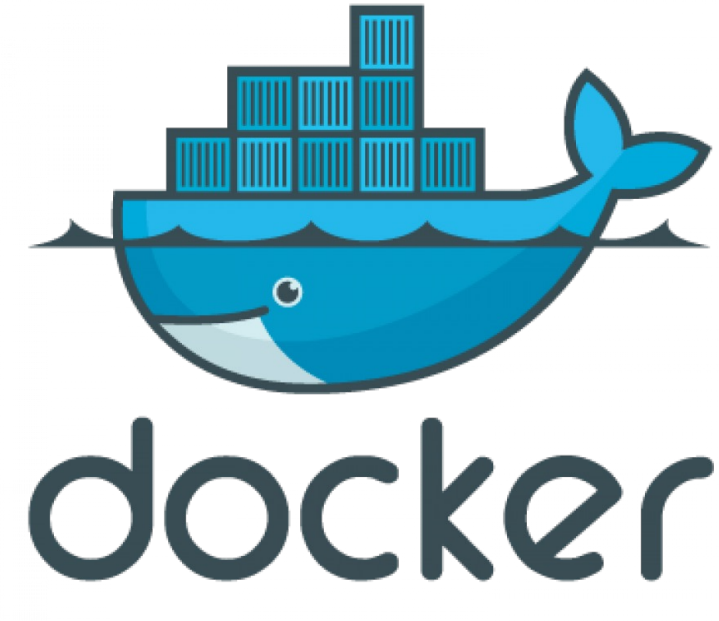
Product & Stakeholders

PRODUCTION

Deploy live

End Users

CHAPTER 8



https://docs.docker.com/get-started/get-docker/

https://docs.docker.com/manuals/

Chapter 8: DOCKER

OUTLINE

1. What is Docker?
2. Docker Architecture
3. What is Docker Images?
4. What is Containers?
5. Containers vs Virtual Machine
7. How to Install Docker?
8. Create Docker Containers
9. Container vs Docker Images
10. Docker Image layers
11. Container Registries
12. Pushing the Docker Image to Docker Hub
13. What is Docker Compose?
14. Install Docker Compose
15. Docker Swarm

What is Docker?

- **Docker** is an open platform for developing, testing, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With docker, you can manage your infrastructure in the same ways you manage your applications.
- By taking advantage of docker's methodologies for build, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

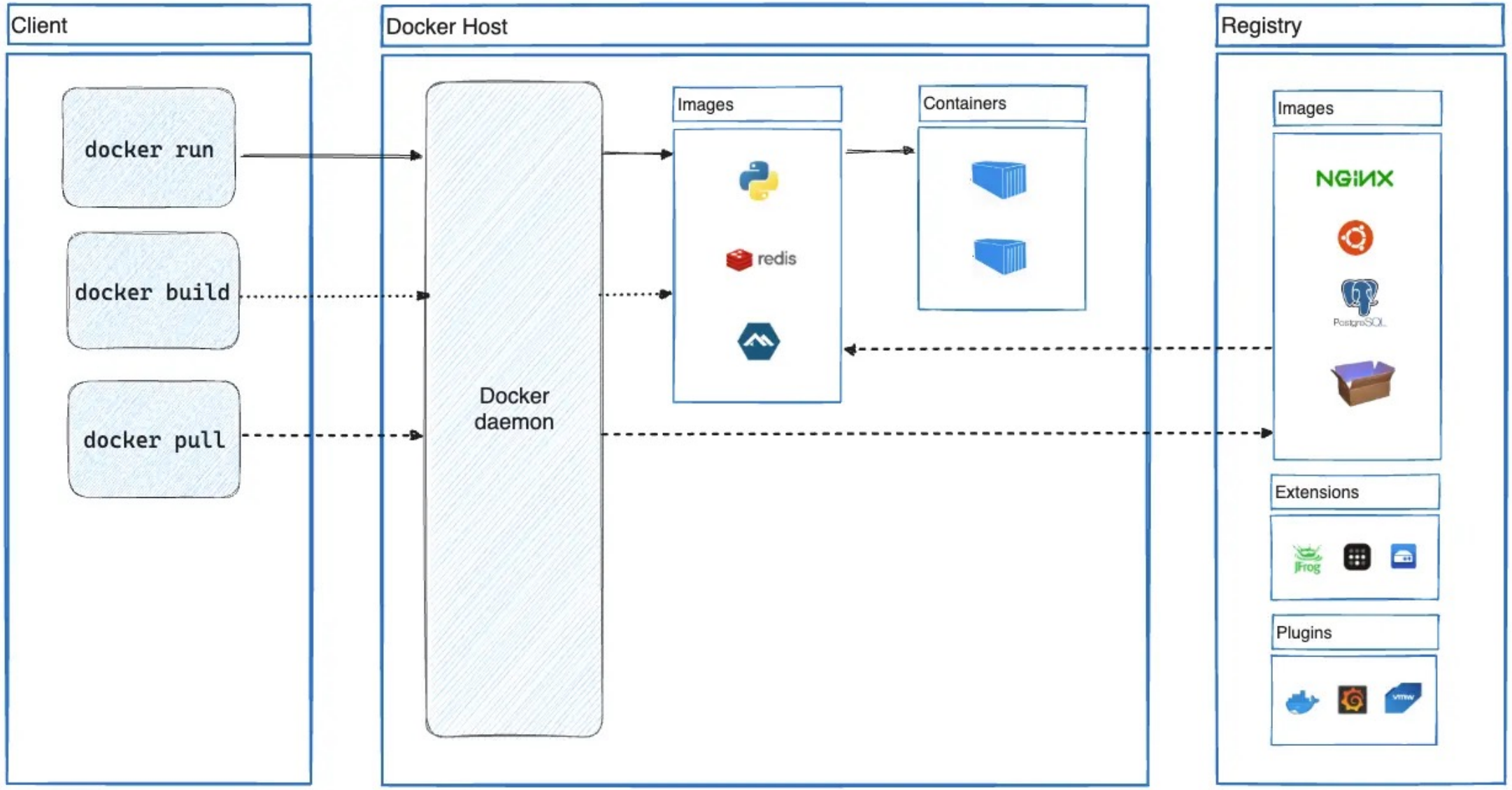
The Docker platform

- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- The isolation and security lets you run many containers simultaneously on a given host.
- Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host.
- You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

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 your Docker containers.
- The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.
- Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

Docker Architecture (cont.)



Components of Docker

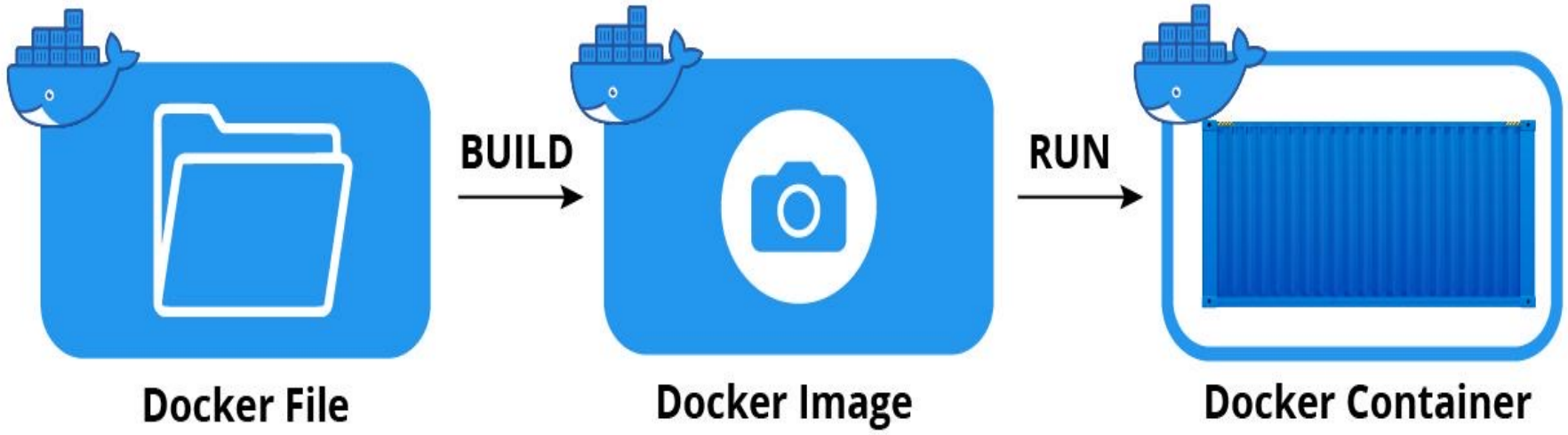
1. ***Docker Engine***: is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:
   - A server with a long-running daemon process dockerd.
   - APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
   - A command line interface (CLI) client docker.
2. ***Docker Client***: the Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Components of Docker

3. ***Docker Desktop***: Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, …
4. ***Docker registries***: A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default.
5. ***Docker objects***: When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects.

What is Images?

- **A docker image** is a read-only template that contains a set of instructions for creating a container.
- It provides a convenient way to package up applications and preconfigured server environments, which you can use for your own private use or share publicly with other docker users.

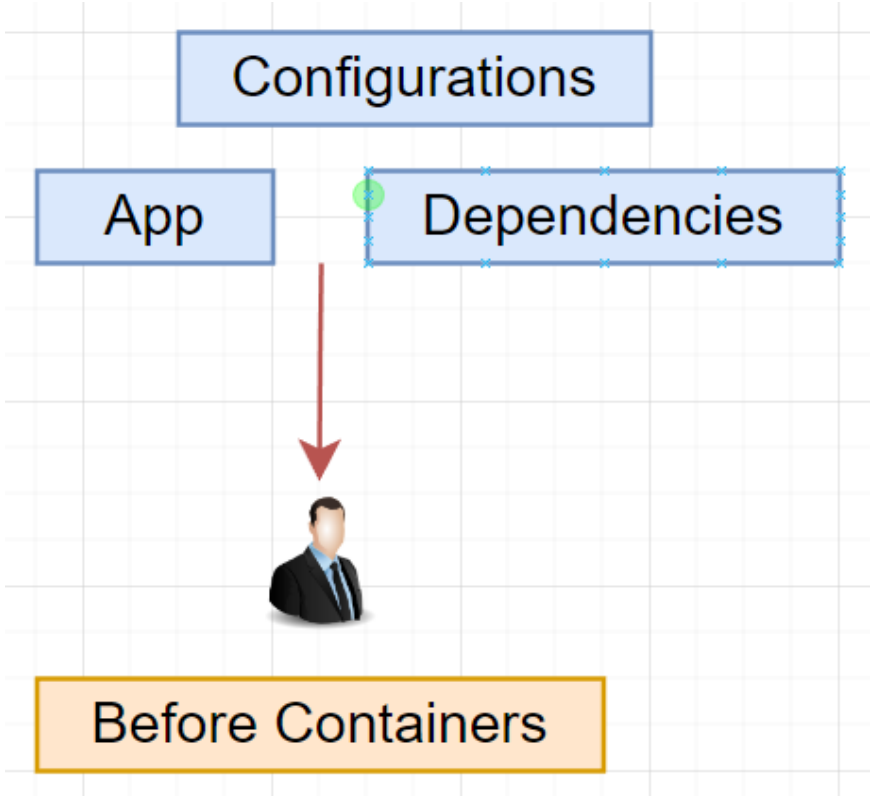


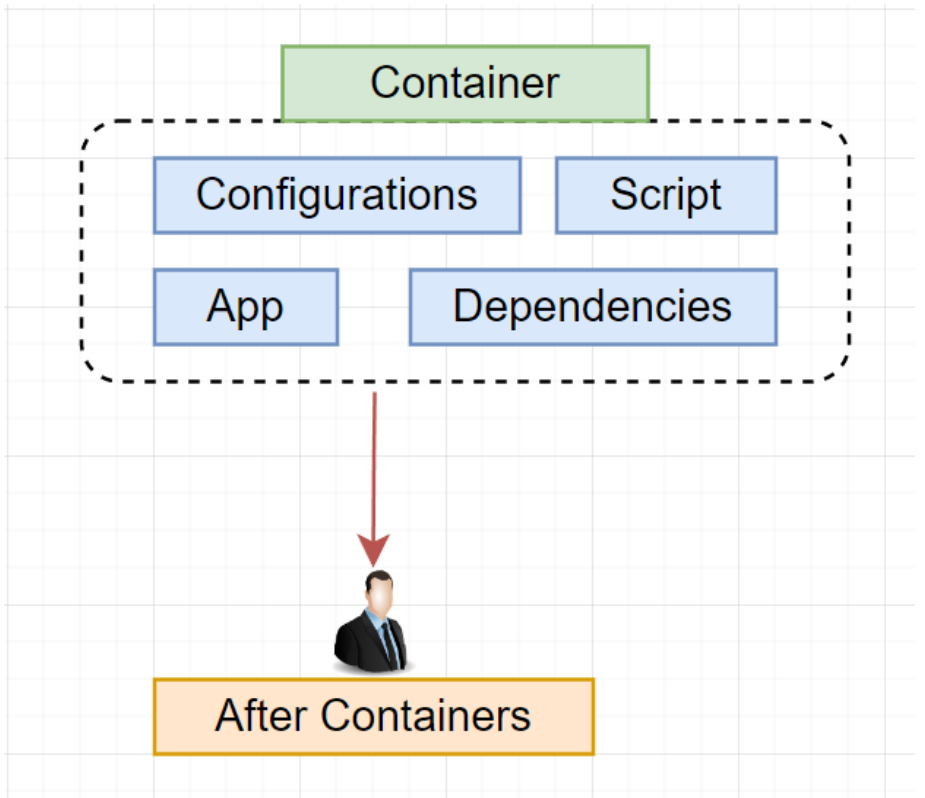
What is Images?

- **A docker image** is made up of a collection of files that bundle together all the essentials – such as installations, application code, and dependencies. You can create a docker image by using one of two methods:
  - **Interactive:** creating image from a container.
  - **Dockerfile:** By constructing a plain-text file, known as a Dockerfile, which provides the specifications for creating a docker image.

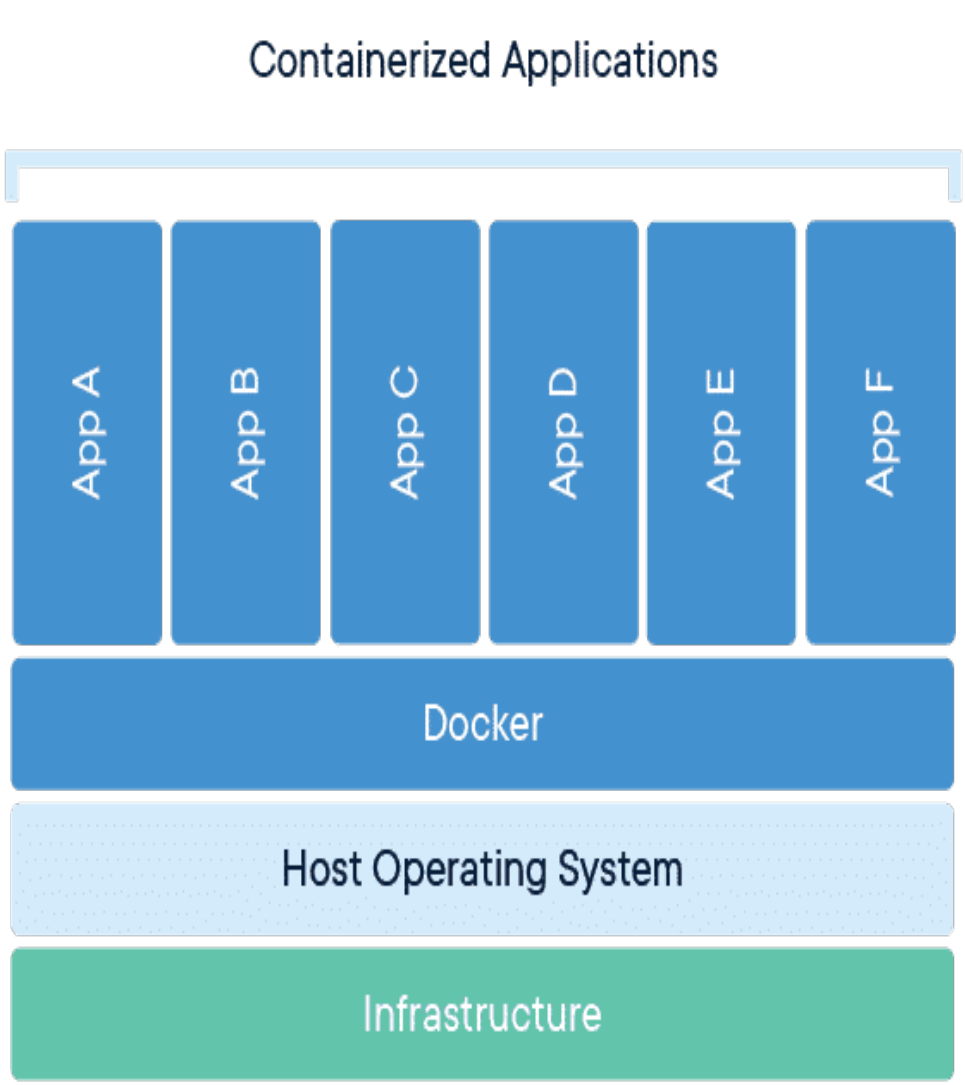
What is Containers?

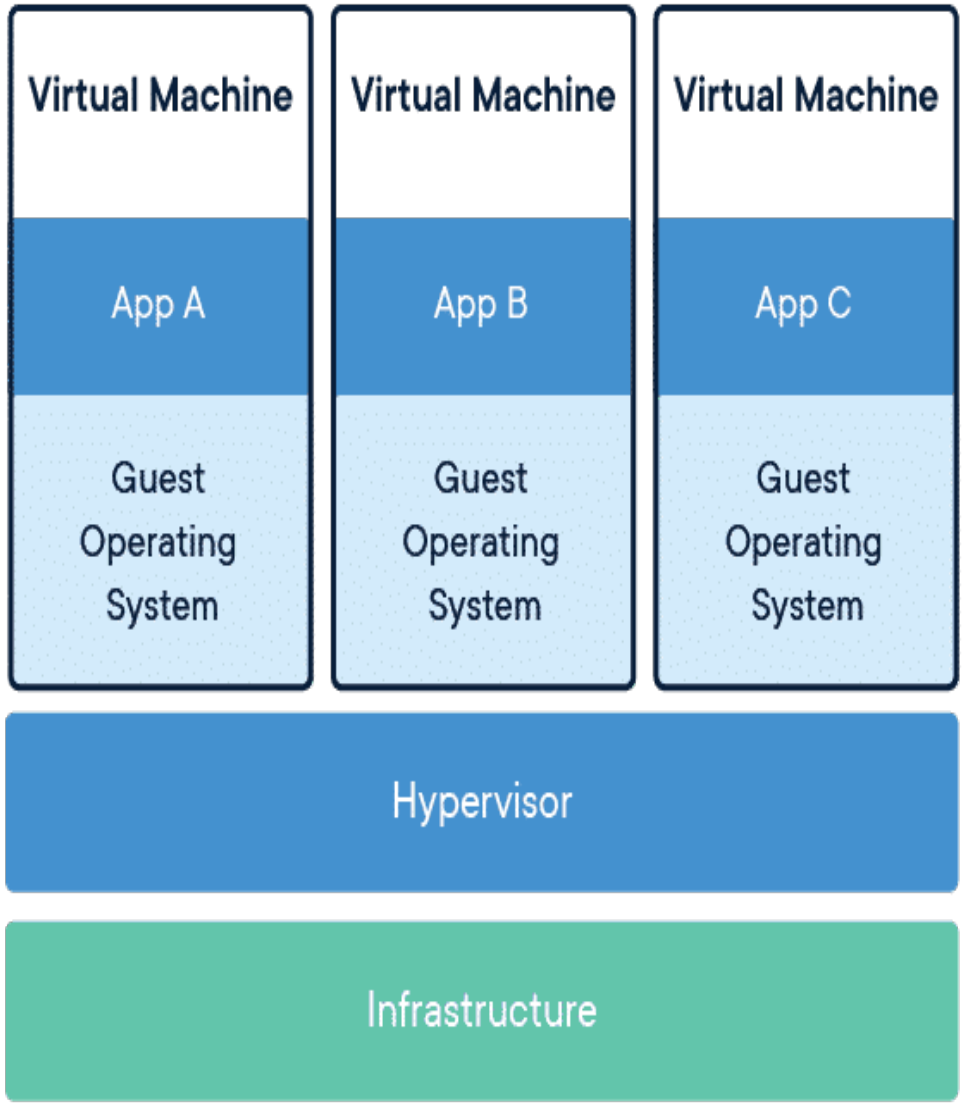
- **Docker Container** is a virtual environment that bundles application code with all the dependencies required to run the application. The application runs quickly and reliably from one computing environment to another.





Containers vs VirtualMachine

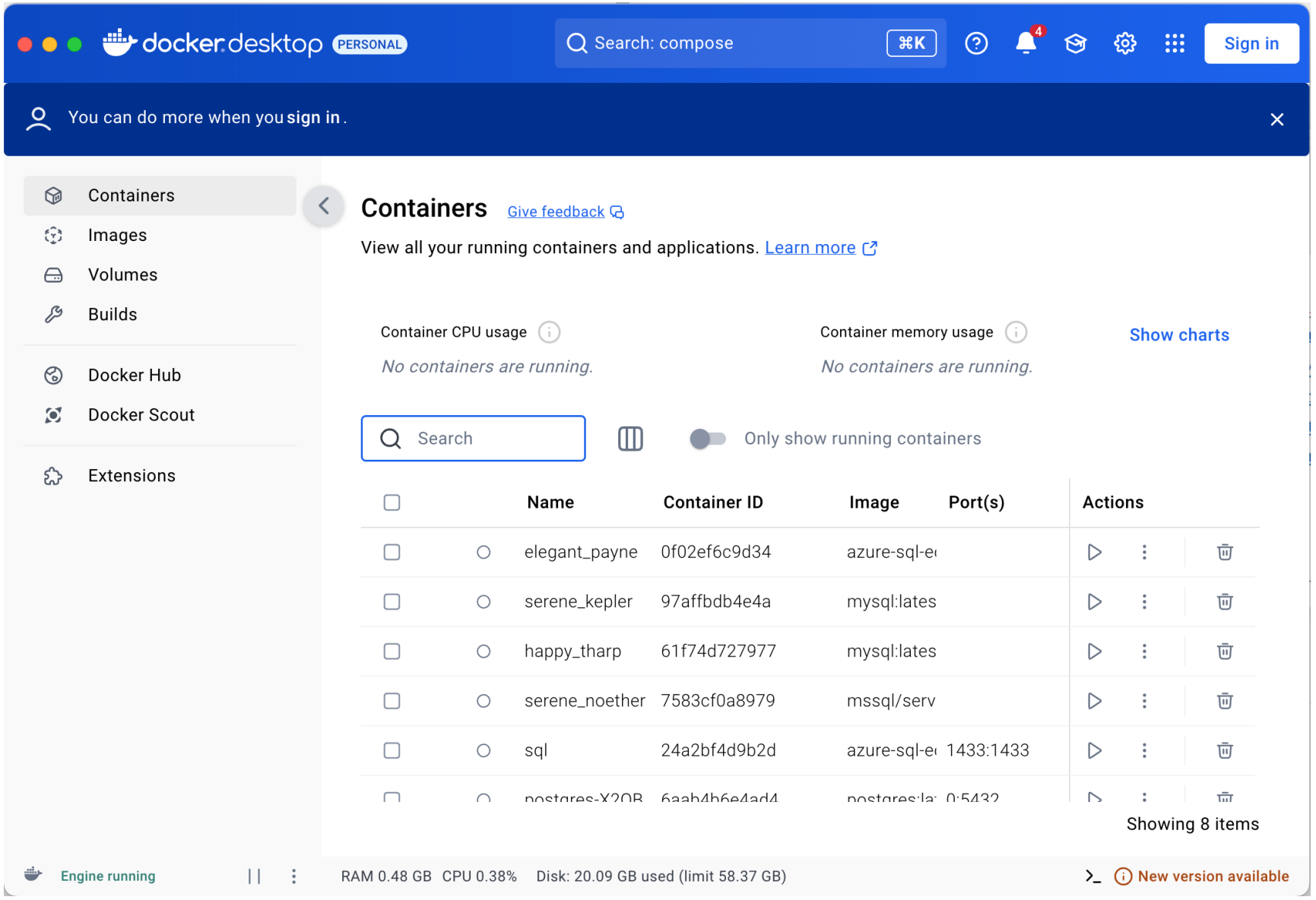




How to install Docker?

- **Install Docker Engine: https://docs.docker.com/engine/install/**
- **Docker Desktop**: https://docs.docker.com/desktop/
  - Windows:https://docs.docker.com/desktop/setup/install/windows-install/
  - Mac: https://docs.docker.com/desktop/setup/install/mac-install/
  - Linux: https://docs.docker.com/desktop/setup/install/linux/

Docker Desktop on Mac



Container Registries

- A container registry is a catalog of storage locations where you can push and pull container images.
- The actual physical locations where images are stored are known as **repositories**.
- Each repository stores a collection of related images with the same name.

Container Registries

- Docker Hub
- Self-Hosted Registries
- Third-Party Registry Services
  - Amazon ECR , Azure Container Registry and Google Container Registry, which are geared towards the public cloud
  - Hybrid-cloud solutions such as GitLab Container Registry and JFrog's own container registry service.

Docker basic commands – Docker build

- ***docker build -t <image_name>***: Build a Docker image from a Dockerfile in the current directory and tag it with a name.
- ***docker build --no-cache -t <image_name>***: Build a Docker image without using the cache.
- ***docker build -f <dockerfile_name> -t <image_name>***: Build a Docker image using a specified Dockerfile.

Docker basic commands – Docker clean up

- ***docker system prune***: Remove all unused Docker resources, including containers, images, networks, and volumes.
- ***docker container prune***: Remove all stopped containers.
- ***docker image prune***: Remove unused images.
- ***docker volume prune***: Remove unused volumes.
- ***docker network prune***: Remove unused networks.

Docker basic commands – Container Interaction

- ***docker run <image_name>***: Run a Docker image as a container.
- ***docker start <container_id>***: Start a stopped container.
- ***docker stop <container_id>***: Stop a running container.
- ***docker restart <container_id>***: Restart a running container.
- ***docker exec -it <container_id> <command>***: Execute a command inside a running container interactively.

Docker basic commands – Container inspection

- ***docker ps***: List running containers.
- ***docker ps -a***: List all containers, including stopped ones.
- ***docker logs <container_id>***: Fetch the logs of a specific container.
- ***docker inspect <container_id>***: Inspect detailed information about a container

Docker basic commands – Image

- ***docker images***: List available Docker images.
- ***docker pull <image_name>***: Pull a Docker image from a Docker registry.
- ***docker push <image_name>***: Push a Docker image to a Docker registry.
- ***docker rmi <image_id>***: Remove a Docker image.

Docker basic commands – Docker run

- ***docker run -d <image_name>***: Run a Docker image as a container in detached mode.
- ***docker run -p <host_port>:<container_port> <image_name>***: Publish container ports to the host.
- ***docker run -v <host_path>:<container_path> <image_name>***: Mount a host directory or volume to a container.
- ***docker run --name <container_name> <image_name>***: Assign a custom name to the container

Docker basic commands – Docker Registry

- ***docker login***: Log in to a Docker registry.
- ***docker logout***: Log out from a Docker registry.
- ***docker search <term>***: Search for Docker images in a Docker registry.
- ***docker pull <registry>/<image_name>***: Pull a Docker image from a specific registry.

Docker basic commands – Docker Service

- ***docker service create --name <service_name> <image_name>***: Create a Docker service from an image.
- ***docker service ls***: List running Docker services.
- ***docker service scale <service_name>=<replicas>***: Scale the replicas of a Docker service.
- ***docker service logs <service_name>***: View logs of a Docker service.

Docker basic commands – Docker Network

- ***docker network create <network_name>***: Create a Docker network.
- ***docker network ls***: List available Docker networks.
- ***docker network inspect <network_name>***: Inspect detailed information about a Docker network.
- ***docker network connect <network_name> <container_name>***: Connect a container to a Docker network

Docker basic commands – Docker Volume

- ***docker volume create <volume_name>***: Create a Docker volume.
- ***docker volume ls***: List available Docker volumes.
- ***docker volume inspect <volume_name>***: Inspect detailed information about a Docker volume.
- ***docker volume rm <volume_name>***: Remove a Docker volume.

Docker basic commands – Docker Environment Variables

- ***-e or --env***: Set environment variables when running a container.
- ***docker run -e <variable_name>=<value> <image_name>:*** Set an environment variable when running a container

Docker basic commands – Docker Compose

- ***docker compose up***: Create and start containers defined in a Docker Compose file.
- ***docker compose down***: Stop and remove containers defined in a Docker Compose file.
- ***docker compose ps***: List containers defined in a Docker Compose file.
- ***docker compose logs***: View logs of containers defined in a Docker Compose file.

Docker Image - Practices

1. **docker image -- help**
2. **Sign up at: https://hub.docker.com/**
3. **docker login**

```
thoaha@Has-MacBook-Pro ~ % docker login

USING WEB-BASED LOGIN
To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: VXHV-VSVB
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser…

Login Succeeded
```

Docker Image – Practices (cont.)

4. Pull image - mysql 8.0:

https://hub.docker.com/_/mysql/tags?name=8.0

```
thoaha@Has-MacBook-Pro ~ % docker pull mysql:8.0
8.0: Pulling from library/mysql
903087d703a7: Pull complete
9dcae24b624f: Pull complete
d1014e296527: Pull complete
5dd6251984b1: Pull complete
e176816781e5: Pull complete
63ce6cde00d5: Pull complete
46a5e0bee806: Pull complete
6882305ba978: Pull complete
7678a5083faf: Pull complete
f41ddadd1084: Pull complete
dd907af04cee: Pull complete
Digest: sha256:50b5ee0656a2caea76506fe702b1baddabe204cf3ab34c03752d2d7cd8ad83fc
Status: Downloaded newer image for mysql:8.0
docker.io/library/mysql:8.0

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview mysql:8.0
thoaha@Has-MacBook-Pro ~ %
thoaha@Has-MacBook-Pro ~ % docker image ls
REPOSITORY                    TAG                  IMAGE ID       CREATED          SIZE
mysql                         8.0                  5ea077402e99   7 weeks ago      760MB
```

Docker Image – Practices (cont.)

5. Tag and push image - mysql 8.0

```
thoaha@Has-MacBook-Pro ~ % docker image tag mysql:8.0 my-mysql-80:v1
thoaha@Has-MacBook-Pro ~ % docker push my-mysql-80:v1
The push refers to repository [docker.io/library/my-mysql-80]
dd2eea575ebe: Preparing
474d51f5015c: Preparing
213335264498: Preparing
ace2b8657db8: Preparing
744940b63946: Preparing
57c977cd8865: Waiting
23b218434edb: Waiting
55e98753b663: Waiting
7f6728e77522: Waiting
dfaafd4d8599: Waiting
988b327444ca: Waiting
denied: requested access to the resource is denied
thoaha@Has-MacBook-Pro ~ %
```

Docker Image – Practices (cont.)

5. Tag and push image - mysql 8.0

```
[thoaha@Has-MacBook-Pro ~ % docker image tag mysql:8.0 tg61550186/my-mysql-80:v1
[thoaha@Has-MacBook-Pro ~ % docker push tg61550186/my-mysql-80:v1
The push refers to repository [docker.io/tg61550186/my-mysql-80]
dd2eea575ebe: Pushed
474d51f5015c: Pushed
213335264498: Pushed
ace2b8657db8: Pushed
744940b63946: Pushed
57c977cd8865: Pushed
23b218434edb: Pushed
55e98753b663: Pushed
7f6728e77522: Pushed
dfaafd4d8599: Pushed
988b327444ca: Pushed
```

Docker Image – Practices (cont.)

6. **Save image**

   docker image save mysql:8.0 -o mysql-8.0.tar

7. **Load image**

   docker image load mysql-8.0.tar

8. **Remove image**

```
thoaha@Has-MacBook-Pro ~ % docker image rm mysql:8.0
Untagged: mysql:8.0
Untagged: mysql@sha256:50b5ee0656a2caea76506fe702b1baddabe204cf3ab34c03752d2d7cd8ad83fc
```

Docker Container

1. Create container from image

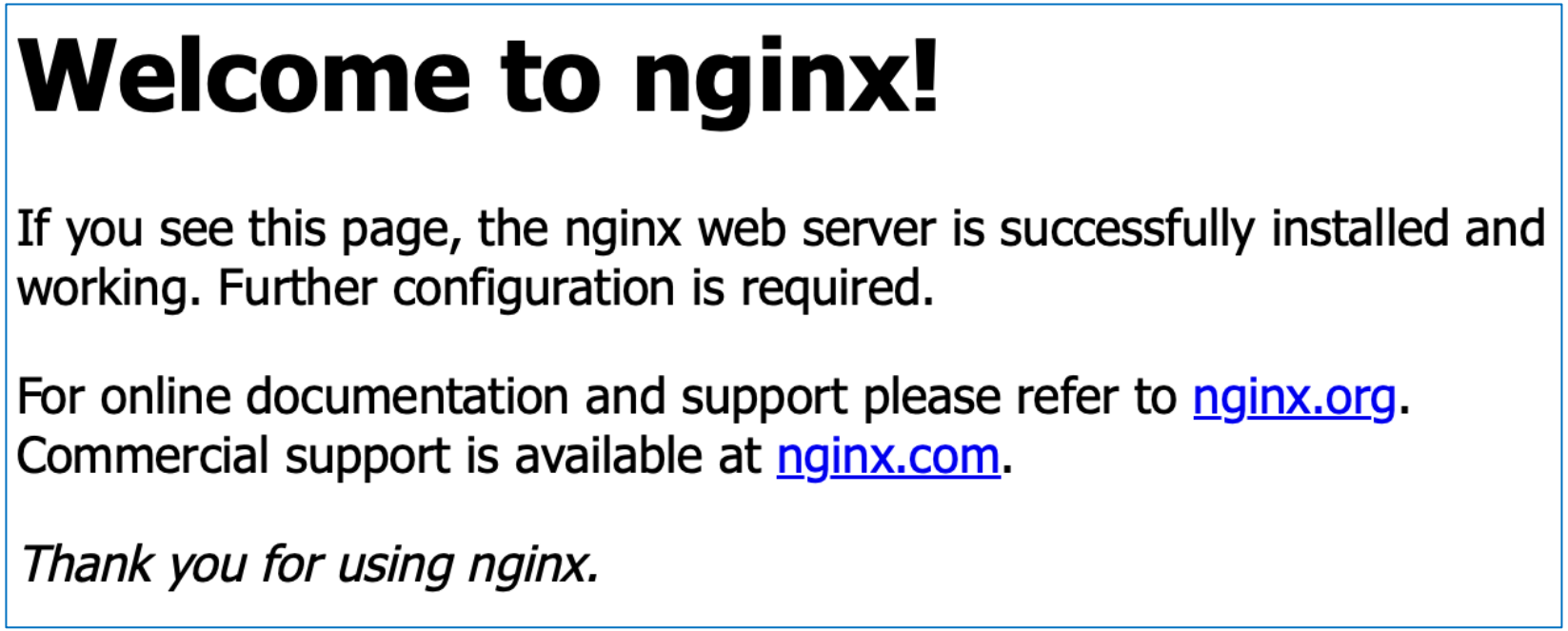
docker run [options] <**image**>

https://docs.docker.com/reference/cli/docker/container/run/

```
thoaha@Has-MacBook-Pro ~ % docker run --name nginx-server -d -p 8080:80 nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
6e771e15690e: Pull complete
0be31969a6d1: Pull complete
9e170776f94c: Pull complete
83f1386059fa: Pull complete
8ea77ffafa6e: Pull complete
76f8ad18306e: Pull complete
3c5c25f3816a: Pull complete
0a329c61f9e1: Pull complete
Digest: sha256:4ff102c5d78d254a6f0da062b3cf39eaf07f01eec0927fd21e219d0af8bc0591
Status: Downloaded newer image for nginx:alpine
3b3426977b139de83d399bcbe69226c42d481fb189cd9193f6628c40c9c5260e
thoaha@Has-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
3b3426977b13   nginx:alpine   "/docker-entrypoint.…"   19 seconds ago   Up 18 seconds   0.0.0.0:8080->80/tcp   nginx-server
```

Docker Container (cont.)

2. **Browser: http://localhost:8080**



Docker Container (cont.)

3. Stop & Start container

```
thoaha@Has-MacBook-Pro ~ % docker stop nginx-server
nginx-server
thoaha@Has-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
thoaha@Has-MacBook-Pro ~ % docker start nginx-server
nginx-server
thoaha@Has-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                   NAMES
3b3426977b13   nginx:alpine   "/docker-entrypoint.…"   4 minutes ago   Up 2 seconds   0.0.0.0:8080->80/tcp   nginx-server
```

Docker Container (cont.)

4. Inspect container

```
thoaha@Has-MacBook-Pro ~ % docker inspect nginx-server
[
    {
        "Id": "3b3426977b139de83d399bcbe69226c42d481fb189cd9193f6628c40c9c5260e",
        "Created": "2025-03-13T11:55:51.716522634Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1360,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-03-13T12:00:30.33597243Z",
            "FinishedAt": "2025-03-13T12:00:12.470536004Z"
        },
        "Image": "sha256:cedb667e1a7b4e6d843a4f74f1f2db0dac1c29b43978aa72dbae2193e
        "ResolvConfPath": "/var/lib/docker/containers/3b3426977b139de83d399bcbe692
        "HostnamePath": "/var/lib/docker/containers/3b3426977b139de83d399bcbe69226
        "HostsPath": "/var/lib/docker/containers/3b3426977b139de83d399bcbe69226c42
        "LogPath": "/var/lib/docker/containers/3b3426977b139de83d399bcbe69226c42d4
1fb189cd9193f6628c40c9c5260e-json.log",
        "Name": "/nginx-server",
```

Docker Container (cont.)

5. Check log

```
thoaha@Has-MacBook-Pro ~ % docker logs nginx-server
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/03/13 11:55:51 [notice] 1#1: using the "epoll" event method
2025/03/13 11:55:51 [notice] 1#1: nginx/1.27.4
2025/03/13 11:55:51 [notice] 1#1: built by gcc 14.2.0 (Alpine 14.2.0)
2025/03/13 11:55:51 [notice] 1#1: OS: Linux 6.10.14-linuxkit
2025/03/13 11:55:51 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/03/13 11:55:51 [notice] 1#1: start worker processes
2025/03/13 11:55:51 [notice] 1#1: start worker process 30
2025/03/13 11:55:51 [notice] 1#1: start worker process 31
2025/03/13 11:55:51 [notice] 1#1: start worker process 32
2025/03/13 11:55:51 [notice] 1#1: start worker process 33
2025/03/13 11:55:51 [notice] 1#1: start worker process 34
2025/03/13 11:55:51 [notice] 1#1: start worker process 35
2025/03/13 11:55:51 [notice] 1#1: start worker process 36
2025/03/13 11:55:51 [notice] 1#1: start worker process 37
172.17.0.1 - - [13/Mar/2025:11:57:32 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (Macintosh
L, like Gecko) Version/17.4.1 Safari/605.1.15" "-"
2025/03/13 11:57:32 [error] 31#31: *2 open() "/usr/share/nginx/html/apple-touch-icon-precomposed
 172.17.0.1, server: localhost, request: "GET /apple-touch-icon-precomposed.png HTTP/1.1", host:
172.17.0.1 - - [13/Mar/2025:11:57:32 +0000] "GET /apple-touch-icon-precomposed.png HTTP/1.1" 404
arwin/21.6.0" "-"
2025/03/13 11:57:32 [error] 31#31: *2 open() "/usr/share/nginx/html/apple-touch-icon.png" failed
 server: localhost, request: "GET /apple-touch-icon.png HTTP/1.1", host: "localhost:8080"
```

Docker Container (cont.)

6. Access inside the container

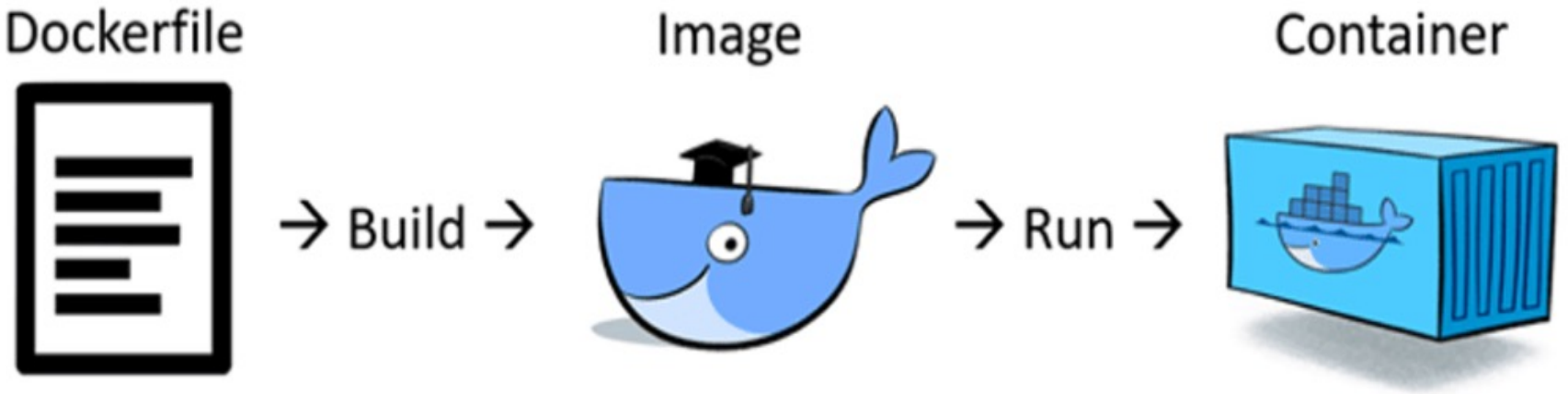
```
thoaha@Has-MacBook-Pro ~ % docker exec -it nginx-server sh
/ # ls
bin                   etc                   mnt                   run                   tmp
dev                   home                  opt                   sbin                  usr
docker-entrypoint.d   lib                   proc                  srv                   var
docker-entrypoint.sh  media                 root                  sys
/ #
```

docker exec –it <container> bash : when the container has ***Bash*** installed - Containers based on full-fledged operating systems like **Ubuntu**, **Debian**, **CentOS**, or others that are more feature-rich will likely have bash installed.

docker exec –it <container> sh : when the container has only ***sh*** installed - Containers based on **Alpine Linux**, **BusyBox**, or similar lightweight operating systems may not include bash and may only provide a simpler shell like sh

What is Dockerfile?

A Dockerfile is a text-based document that's used to create a container image. It provides instructions to the image builder on the commands to run, files to copy, startup command, and more.



Example – Example – Dockerfile Spring Boot

```
FROM maven:3.8-openjdk-17-slim as build      # Build stage - Use the official Maven image to build the application

WORKDIR /app                                 # Set the working directory for the build

COPY mvnw .                                  # Copy the Maven wrapper and pom.xml file
COPY .mvn .mvn
COPY pom.xml .

RUN ./mvnw dependency:go-offline             # Download dependencies (to be cached)

COPY src ./src                               # Copy source code

RUN mvn clean package -DskipTests            # Build the application (this will create the .jar file in target/)

FROM openjdk:17-jdk-slim                     # Runtime stage - Use a smaller base image to run application

WORKDIR /app                                 # Set the working directory for the runtime

COPY --from=build /app/target/*.jar app.jar  # Copy the jar file from the previous stage

EXPOSE 8080                                  # Expose the application port (default is 8080)

ENTRYPOINT ["java", "-jar", "app.jar"]       # Run the application
```

Dockerfile – Common Instructions

- FROM <image> - this specifies the base image that the build will extend.
- LABEL set some info
- WORKDIR <path> - this instruction specifies the "working directory" or the path in the image where files will be copied and commands will be executed.
- COPY <host-path> <image-path> - this instruction tells the builder to copy files from the host and put them into the container image.
- RUN <command> - this instruction tells the builder to run the specified command.
- ADD <src/URL> <dest> It copies the files from source to destination (inside the container).
- ENV <name> <value> - this instruction sets an environment variable that a running container will use.
- EXPOSE <port-number> - this instruction sets configuration on the image that indicates a port the image would like to expose.

Dockerfile – Common Instructions (cont.)

- **CMD ["<command>", "<arg1>"]** - this instruction sets the default command a container using this image will run.
- **ENTRYPOINT** - similar to CMD but more restrictive. Specify default executable.

Ref: https://docs.docker.com/reference/dockerfile/

- Container images are composed of layers. And each of these layers, once created, are immutable.
- Each layer in an image contains a set of filesystem changes - additions, deletions, or modifications.
- When build a Docker image, each command in the Dockerfile (such as RUN, COPY, ADD) creates a new layer. These layers are stacked on top of each other to form the complete image.

Image layers – Example – Dockerfile Spring Boot

`FROM maven:3.8-openjdk-17-slim as build`	→ ***Layer 1***: *Base Image for Build Stage (Maven + JDK 17)*
`WORKDIR /app`	→ ***Layer 2***: *Set the working directory for the build*

`COPY mvnw .`
`COPY .mvn .mvn`
`COPY pom.xml .`
	→ ***Layer 3***: *Copy the Maven wrapper and pom.xml file*

`RUN ./mvnw dependency:go-offline`	→ ***Layer 4***: *Download dependencies (to be cached)*
`COPY src ./src`	→ ***Layer 5***: *Copy source code*
`RUN mvn clean package -DskipTests`	→ ***Layer 6***: *Build the application (this will create the .jar file in target/)*
`FROM openjdk:17-jdk-slim`	→ ***Layer 7***: *Base image for Runtime – JDK17*
`WORKDIR /app`	→ ***Layer 8***: *Set the working directory for the runtime*
`COPY --from=build /app/target/*.jar app.jar`	→ ***Layer 9***: *Copy the jar file from the previous stage*
`EXPOSE 8080`	→ ***Layer 10***: *Expose the application port (default is 8080)*
`ENTRYPOINT ["java", "-jar", "app.jar"]`	→ ***Layer 11***: *Define the Entry Point for the Container*

Dockerfile - Step by Step – Build and test

1. ***Step 1***: Create a Dockerfile
2. ***Step 2***: Build the Docker Image

   docker build -t your-image-name .

```
thoaha@Has-MacBook-Pro user-service % docker build -t user-service-image .
[+] Building 160.5s (19/19) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 873B
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)
 => [internal] load metadata for docker.io/library/openjdk:17-jdk-slim
 => [internal] load metadata for docker.io/library/maven:3.8-openjdk-17-slim
 => [auth] library/openjdk:pull token for registry-1.docker.io
 => [auth] library/maven:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [build 1/8] FROM docker.io/library/maven:3.8-openjdk-17-slim@sha256:502e781d39f0b40fbd02eb23f5
 => => resolve docker.io/library/maven:3.8-openjdk-17-slim@sha256:502e781d39f0b40fbd02eb23f5b76636
 => => sha256:7337649a0aedd0916ce8bc19a3e1d58779afec34e06ec893fec37fde07661d94 7.71kB / 7.71kB
 => => sha256:4abf2b282e09acdcef9eaddea293475a5694d916e9f9e764ae7ae76ba3bad7aa 1.79kB / 1.79kB
 => => extracting sha256:1d5035d2d5c6c24e610a9317c6907a7c58efd512757d559841e5d0851512ed9c
 => [stage-1 2/3] WORKDIR /app
 => [build 2/8] WORKDIR /app
 => [build 3/8] COPY mvnw .
 => [build 4/8] COPY .mvn .mvn
 => [build 5/8] COPY pom.xml .
 => [build 6/8] RUN ./mvnw dependency:go-offline
 => [build 7/8] COPY src ./src
 => [build 8/8] RUN mvn clean package -DskipTests
 => [stage-1 3/3] COPY --from=build /app/target/*.jar app.jar
 => exporting to image
 => => exporting layers
 => => writing image sha256:a532095eab190630b70248404c50d5d8304d610e14613e7de9555e67a23c2c0f
 => => naming to docker.io/library/user-service-image
```

Dockerfile - Step by Step – Build and test (cont.)

3. ***Step 3***: Verify the Docker Image

   docker images

```
thoaha@Has-MacBook-Pro user-service % docker images
REPOSITORY                      TAG                 IMAGE ID       CREATED          SIZE
user-service-image              latest              a532095eab19   2 minutes ago    464MB
```

4. ***Step 4***: Run the Docker Container

   docker run -p 8080:8080 your-image-name

5. ***Step 5***: Test application

   http://localhost:8080

Dockerfile - Step by Step – Build and test (cont.)

6. ***Step 6***: Export the Docker Image

   docker save -o your-image-name.tar your-image-name

7. ***Step 7***: Load the Docker image on the Target Machine

   docker load -i your-image-name.tar

8. ***Step 8***: Run the Docker container on the Target Machine

   docker run -p 8080:8080 your-image-name

9. ***Step 9***: Test application

   http://localhost:8080

Dockerfile - Reference

1. Write Dockerfile: https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/
2. Build, tag, and publish an image: https://docs.docker.com/get-started/docker-concepts/building-images/build-tag-and-publish-an-image/
3. Using the build cache: https://docs.docker.com/get-started/docker-concepts/building-images/using-the-build-cache/
4. Multi-stage builds: https://docs.docker.com/get-started/docker-concepts/building-images/multi-stage-builds/

Example – Dockerfile Spring Boot – ARG - ENV

```
ARG JDK_VERSION="17"
FROM maven:3.8-openjdk-${JDK_VERSION}-slim as build

WORKDIR /app

……

FROM openjdk:${JDK_VERSION}-jdk-slim

WORKDIR /app

COPY --from=build /app/target/*.jar app.jar
ENV TZ=Asia/Ho_Chi_Minh

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Dockerfile – Build cache

```
thoaha@192 user-service % docker build -t user-service-image .
[+] Building 1.0s (17/17) FINISHED                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 873B                                                0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)      0.0s
 => [internal] load metadata for docker.io/library/openjdk:17-jdk-slim              1.0s
 => [internal] load metadata for docker.io/library/maven:3.8-openjdk-17-slim        1.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [build 1/8] FROM docker.io/library/maven:3.8-openjdk-17-slim@sha256:502e781d39f0b40fbd02eb23  0.0s
 => [stage-1 1/3] FROM docker.io/library/openjdk:17-jdk-slim@sha256:aaa3b3cb27e3e520b8f116863d05  0.0s
 => [internal] load build context                                                   0.0s
 => => transferring context: 5.53kB                                                 0.0s
 => CACHED [stage-1 2/3] WORKDIR /app                                               0.0s
 => CACHED [build 2/8] WORKDIR /app                                                 0.0s
 => CACHED [build 3/8] COPY mvnw .                                                  0.0s
 => CACHED [build 4/8] COPY .mvn .mvn                                               0.0s
 => CACHED [build 5/8] COPY pom.xml .                                               0.0s
 => CACHED [build 6/8] RUN ./mvnw dependency:go-offline                             0.0s
 => CACHED [build 7/8] COPY src ./src                                               0.0s
 => CACHED [build 8/8] RUN mvn clean package -DskipTests                            0.0s
 => CACHED [stage-1 3/3] COPY --from=build /app/target/*.jar app.jar                0.0s
 => exporting to image                                                              0.0s
 => => exporting layers                                                             0.0s
 => => writing image sha256:eafab20f02b678290af56df219c6aa6c6f7978261c289022e27496c69d364140  0.0s
 => => naming to docker.io/library/user-service-image                               0.0s
```

Dockerfile – Multi-stage builds

```
# Stage 1: Build Environment
FROM builder-image AS build-stage
# Install build tools (e.g., Maven, Gradle)
# Copy source code
# Build commands (e.g., compile, package)

# Stage 2: Runtime environment
FROM runtime-image AS final-stage
#  Copy application artifacts from the build stage (e.g., JAR file)
COPY --from=build-stage /path/in/build/stage /path/to/place/in/final/stage
# Define runtime configuration (e.g., CMD, ENTRYPOINT)
```

Publishing and Expose ports

Running containers

- Publishing a port provides the ability to break through a little bit of networking isolation by setting up a forwarding rule.

  docker run -d -p HOST_PORT:CONTAINER_PORT nginx

- HOST_PORT: The port number on your host machine where you want to receive traffic
- CONTAINER_PORT: The port number within the container that's listening for connections
- Ex: **docker run -d -p 8080:80 nginx**

Ref: https://docs.docker.com/get-started/docker-concepts/running-containers/publishing-ports/

Overriding container defaults

Running containers

- Overriding the network ports

  docker run -d -p HOST_PORT:CONTAINER_PORT postgres

- Setting environment variables

  docker run -e foo=bar postgres env

  docker run --env-file .env postgres env

- Restricting the container to consume the resource

  docker run -e POSTGRES_PASSWORD=secret --memory="512m" --cpus="0.5" postgres

Persisting container data

Running containers

- When a container starts, it uses the files and configuration provided by the image. Each container is able to create, modify, and delete files and does so without affecting any other containers. When the container is deleted, these file changes are also deleted.
- If we want to keep data between runs, Docker volumes and bind mounts can help.

Persisting container data (cont.)

Running containers

- Docker uses the following types of volumes and bind mounts to persist data:
  - Anonymous volumes
  - Named volumes
  - Bind mounts
- Volumes are a storage mechanism that provide the ability to persist data beyond the lifecycle of an individual container. Think of it like providing a shortcut or symlink from inside the container to outside the container.
- Ref: https://docs.docker.com/engine/storage/

Persisting container data (cont.)

Running containers

- **docker volume create [volume_name]**
- **docker run –v [local_dir/volume]:[container_dir]**
- Example:
  - docker volume create **pgdata**
  - docker run -v **pgdata:/var/lib/postgresql/data** --name postgres-container -e POSTGRES_PASSWORD=password -it -p 5433:5432 postgres

Docker Network

- Container networking refers to the ability for containers to connect to and communicate with each other, or to non-Docker workloads.
- Drivers: https://docs.docker.com/engine/network/
  - bridge: The default network driver.
  - host: Remove network isolation between the container and the Docker host.
  - None: Completely isolate a container from the host and other containers.
  - overlay: Overlay networks connect multiple Docker daemons together.
  - ipvlan: IPvlan networks provide full control over both IPv4 and IPv6 addressing.
  - macvlan: Assign a MAC address to a container.

Docker Network (cont.)

- Ref: https://docs.docker.com/reference/cli/docker/network/
- Connects a container to a network

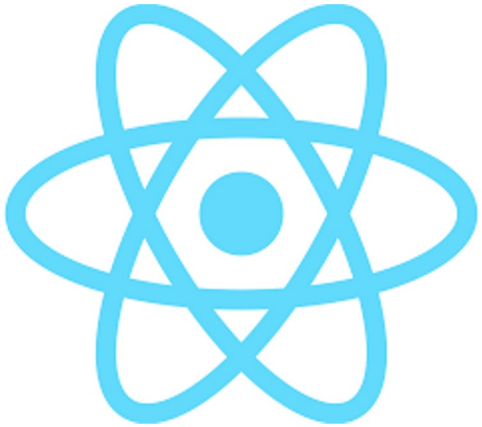
  docker network connect [OPTIONS] NETWORK CONTAINER
- Disconnects a container from a network

  docker network disconnect [OPTIONS] NETWORK CONTAINER

DOCKER COMPOSE

https://docs.docker.com/compose/

Multiple Containers



port: 80
envs:[…]

port: 8080
envs:[…]

port: 3306
envs:[…]

port: 6379
envs:[…]

Docker Compose

- Install Docker Compose: https://docs.docker.com/compose/install/
- Already on Docker Desktop:
  - **docker compose version**

```
thoaha@192 user-service % docker compose version
Docker Compose version v2.31.0-desktop.2
```

Docker Compose (cont.)

- Docker Compose is a tool for defining and running multi-container applications.
- Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

Docker Compose (cont.)

- Compose works in all environments; production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:
  - Start, stop, and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service
- YAML: YAML Ain't Markup Language™

YAML is a human-friendly data serialization language for all programming languages.

Docker Compose (cont.)

- The default path for a Compose file is ***compose.yaml*** (preferred) or ***compose.yml*** that is placed in the working directory.
- Compose also supports ***docker-compose.yaml*** and ***docker-compose.yml*** for backwards compatibility of earlier versions. If both files exist, Compose prefers the canonical ***compose.yaml***.
- Ref: https://docs.docker.com/compose/intro/compose-application-model/

Docker Compose - Command

- **docker compose up: to start all the services**
- **docker compose up <service_name>**
- **docker compose up -d <service_name>**
- **docker compose stop <service_name>**
- **docker compose down <service_name>: to stop and remove the running services**
- **docker compose ps**
- **docker compose logs**

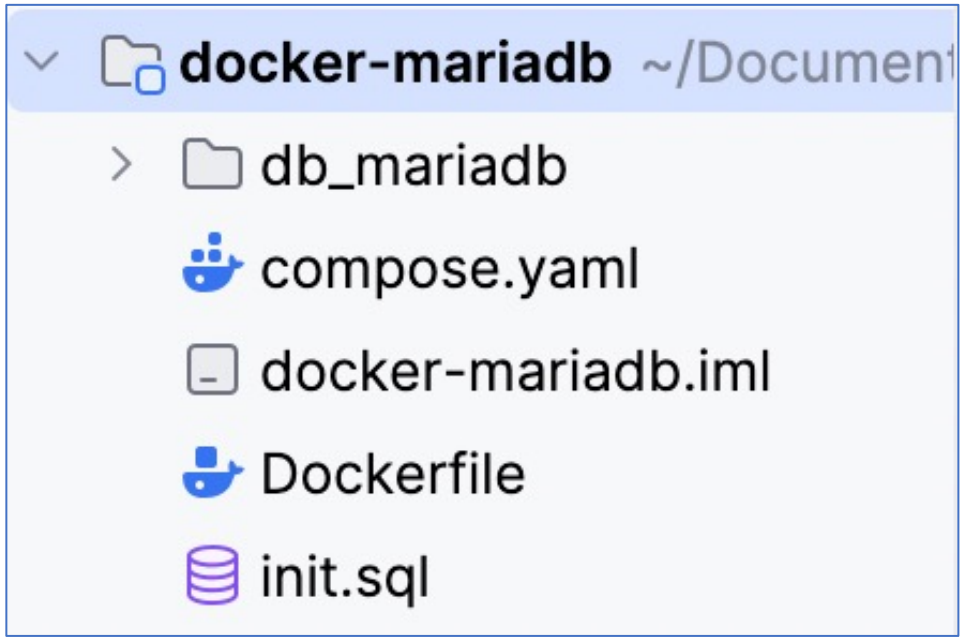
Docker Compose – Template – compose.yaml

```yaml
services: # Defines the containers or services that you want to run.
  service-name-1: # Same as 'docker container run'
    build: . # Build an image from a Dockerfile
    image: # Using build or image: Specifies the Docker image to use
    container_name: # Assigns a custom name to the container
    command: # Override the default command.
    entrypoint: # Specifies the entrypoint of the container (replaces the default one).
    environment: # Add environment variables. You can use either an array or a dictionary
    env_file: # Specifies a file containing environment variables.
    ports: # Expose ports. Either specify both ports (HOST:CONTAINER)
    volumes: # Same as using '-v' with 'docker container run'
    depends_on: # Specifies service dependencies to control the order of service startup.
    networks: # Specifies which networks the service should join.
    restart: # Configures the restart policy for the service.
    logging: # Configures logging options for the service.
    labels: # Adds metadata labels to the container.
  service-name-2:
    ...
volumes:
networks:
  service-name-1:
    driver: bridge
    ipam:
      config:
        subnet: "192.168.1.0/24" # Sub-net
```

Example – mariadb

Docker Compose

1. Project structure



2. Dockerfile

Dockerfile

```
FROM mariadb:11.7.2

LABEL maintainer="demo-mariadb"

# Add spring_security_jwt schema and tables
ADD ./init.sql /docker-entrypoint-initdb.d/

RUN chown -R mysql:mysql /docker-entrypoint-initdb.d/
```

Example – mariadb (cont.)

Docker Compose

3. compose.yaml

compose.yaml

```yaml
services:
  mariadb:
    build: . # Mặc định sẽ run Dockerfile
    container_name: db-mariadb
    restart: unless-stopped
    networks:
      db:
        ipv4_address: 172.28.1.10  # Địa chỉ IP tĩnh
    volumes:
      - ./db_mariadb:/var/lib/mysql/:rw # rw: read + write
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql # Mount thư mục SQL vào thư mục init của container
    ports:
      - "3307:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=123456
      - MYSQL_DATABASE=spring_security_jwt
      - MYSQL_USER=user
      - MYSQL_PASSWORD=123456
networks:
  db:
    driver: bridge
    ipam:
      config:
        - subnet: "172.28.1.0/24" # Sub-net
volumes:
  db_mariadb:
```

Example – mariadb (cont.) - build

Docker Compose

```
thoaha@Has-MacBook-Pro docker-mariadb % docker compose up -d
[+] Building 0.0s (0/1)                                          docker:desktop-linux
[+] Running 0/1
[+] Building 1.0s (9/9) FINISHED                                 docker:desktop-linux
 => [mariadb internal] load build definition from Dockerfile                    0.0s
 => => transferring dockerfile: 237B                                            0.0s
 => [mariadb internal] load metadata for docker.io/library/mariadb:11.7.2       1.0s
 => [mariadb internal] load .dockerignore                                       0.0s
 => => transferring context: 2B                                                 0.0s
 => [mariadb internal] load build context                                       0.0s
 => => transferring context: 30B                                                0.0s
 => [mariadb 1/3] FROM docker.io/library/mariadb:11.7.2@sha256:310d29fbb58169dcddb384b0ff1  0.0s
 => CACHED [mariadb 2/3] ADD ./init.sql /docker-entrypoint-initdb.d/            0.0s
 => CACHED [mariadb 3/3] RUN chown -R mysql:mysql /docker-entrypoint-initdb.d/  0.0s
 => [mariadb] exporting to image                                                0.0s
 => => exporting layers                                                         0.0s
 => => writing image sha256:8f3173c20eac8ffcafcbdbb617d89ae41b8625911d309b53e0964f1ac328c6  0.0s
[+] Running 3/3o docker.io/library/docker-mariadb-mariadb                      0.0s
 ✔ Service mariadb            Built                                           1.1s
 ✔ Network docker-mariadb_db  Created                                         0.0s
 ✔ Container db-mariadb       Started                                         0.1s
```

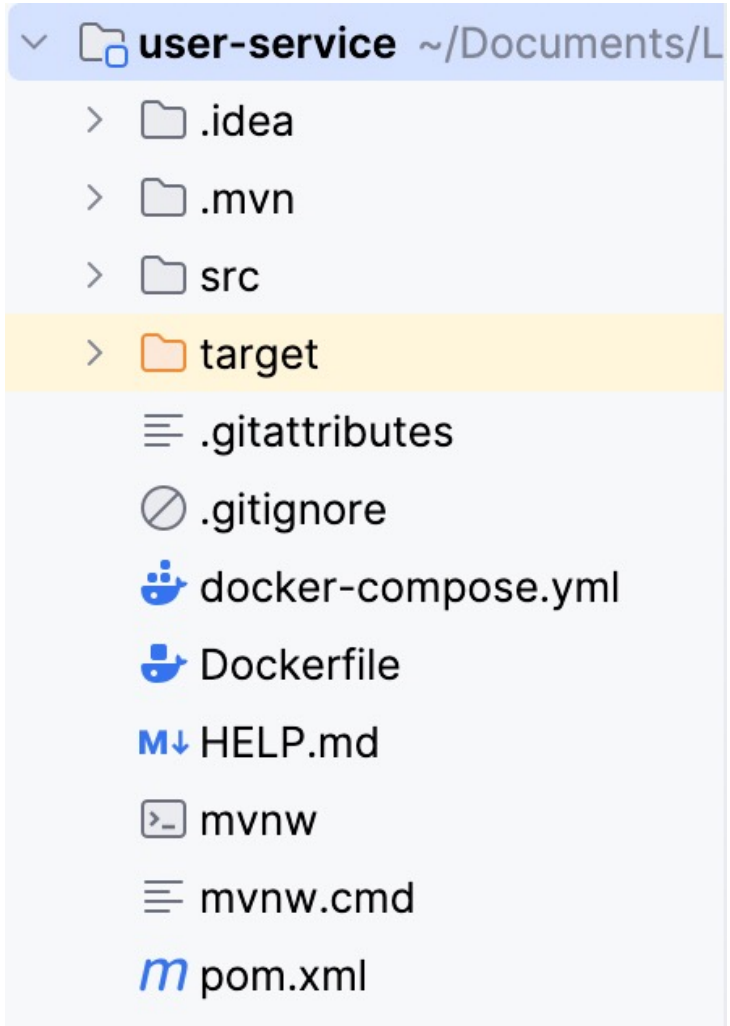
Example – mariadb (cont.)

Docker Compose

```
thoaha@Has-MacBook-Pro docker-mariadb % docker compose down
[+] Running 2/2
 ✔ Container db-mariadb        Removed                                0.3s
 ✔ Network docker-mariadb_db   Removed                                0.1s
```

Example – Springboot
Docker Compose

1. Project structure



2. Dockerfile

Dockerfile

```
FROM maven:3.8-openjdk-17-slim as build
WORKDIR /app
COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
RUN ./mvnw dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests

FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Example – Springboot (cont.)

Docker Compose

3. docker-compose.yml

docker-compose.yml

```yaml
services:
  user-service:
    build: .
    container_name: user-service
    restart: unless-stopped
    networks:
      user-service:
        ipv4_address: 192.168.1.10  # Địa chỉ IP tĩnh
    ports:
      - "8080:8080"
networks:
  user-service:
    driver: bridge
    ipam:
      config:
        - subnet: "192.168.1.0/24" # Sub-net
```

Example – Springboot (cont.) - build

Docker Compose

```
thoaha@Has-MacBook-Pro docker-mariadb % docker compose up -d
[+] Building 0.0s (0/1)                                          docker:desktop-linux
[+] Running 0/1
[+] Building 1.0s (9/9) FINISHED                                 docker:desktop-linux
 => [mariadb internal] load build definition from Dockerfile                    0.0s
 => => transferring dockerfile: 237B                                            0.0s
 => [mariadb internal] load metadata for docker.io/library/mariadb:11.7.2       1.0s
 => [mariadb internal] load .dockerignore                                       0.0s
 => => transferring context: 2B                                                 0.0s
 => [mariadb internal] load build context                                       0.0s
 => => transferring context: 30B                                                0.0s
 => [mariadb 1/3] FROM docker.io/library/mariadb:11.7.2@sha256:310d29fbb58169dcddb384b0ff1  0.0s
 => CACHED [mariadb 2/3] ADD ./init.sql /docker-entrypoint-initdb.d/            0.0s
 => CACHED [mariadb 3/3] RUN chown -R mysql:mysql /docker-entrypoint-initdb.d/  0.0s
 => [mariadb] exporting to image                                                0.0s
 => => exporting layers                                                         0.0s
 => => writing image sha256:8f3173c20eac8ffcafcbdbb617d89ae41b8625911d309b53e0964f1ac328c6  0.0s
[+] Running 3/3o docker.io/library/docker-mariadb-mariadb                      0.0s
 ✔ Service mariadb            Built                                           1.1s
 ✔ Network docker-mariadb_db  Created                                         0.0s
 ✔ Container db-mariadb       Started                                         0.1s
```

Example – Using Network
Docker Compose

1. **Step 1**: mariaDb → private docker
2. **Step 2**: Springboot → private docker
3. **Step 3**: Springboot sử dụng Database ở Step 1

```
# user-service and DB use Docker
spring.datasource.url=jdbc:mariadb://172.28.1.10/spring_security_jwt?createDatabaseIfNotExist=true
```

Example – Using Network (cont.)

Docker Compose

4. **Step 4**: docker network ls

```
thoaha@Has-MacBook-Pro docker-mariadb % docker network ls
NETWORK ID     NAME                      DRIVER    SCOPE
540c9d1cdc55   bridge                    bridge    local
a487e033ed00   docker-mariadb_db         bridge    local
```

5. **Step 5**: docker container ls

```
thoaha@Has-MacBook-Pro user-service % docker container ls
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS          PORTS                    NAMES
9f195c04b9b6   docker-mariadb-mariadb      "docker-entrypoint.s…"   13 minutes ago   Up 13 minutes   0.0.0.0:3307->3306/tcp   db-mariadb
082ba75f2a7c   user-service-user-service   "java -jar app.jar"      43 hours ago     Up 16 seconds   0.0.0.0:8080->8080/tcp   user-service
thoaha@Has-MacBook-Pro user-service %
```

Example – Using Network (cont.)

Docker Compose

6. Step 6: user-service lỗi không kết nối được với database

```
user-service  | 2025-03-20T07:20:38.718Z  INFO 1 --- [user-service] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Starting...
user-service  | 2025-03-20T07:21:09.795Z  WARN 1 --- [user-service] [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : SQL Error: 0, S
QLState: 08000
user-service  | 2025-03-20T07:21:09.795Z ERROR 1 --- [user-service] [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : Socket fail to
connect to address=(host=172.28.1.10)(port=3306)(type=primary). Connect timed out
user-service  | 2025-03-20T07:21:09.797Z  WARN 1 --- [user-service] [           main] o.h.e.j.e.i.JdbcEnvironmentInitiator     : HHH000342: Coul
d not obtain connection to query metadata
user-service  |
user-service  | org.hibernate.exception.JDBCConnectionException: unable to obtain isolated JDBC connection [Socket fail to connect to address=(h
ost=172.28.1.10)(port=3306)(type=primary). Connect timed out] [n/a]
```

Example – Using Network (cont.)

Docker Compose

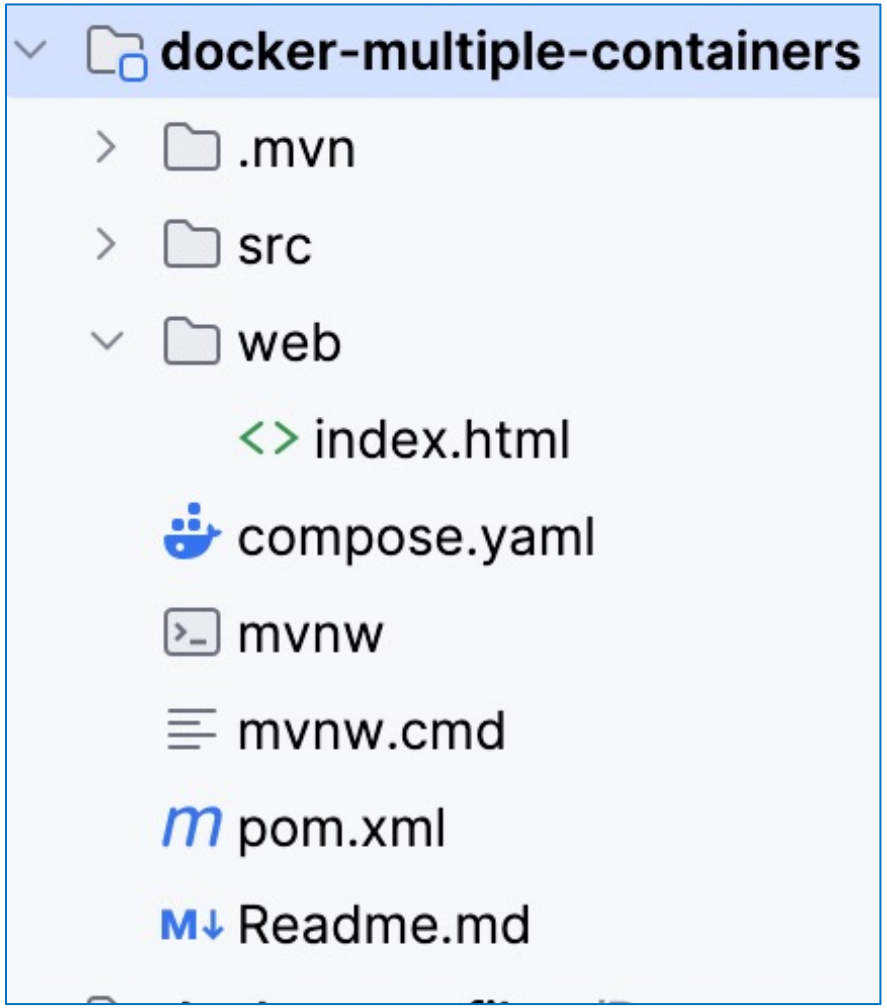
7. **Step 7**: Sử dụng connect network để Springboot có thể kết nối được với DB mariaDB (Step 1)

docker network connect docker-mariadb_db user-service

```
user-service  | 2025-03-20T07:26:40.275Z  INFO 1 --- [user-service] [           main] o.s.b.w.embedded.tomcat.TomcatWebSer
ver  : Tomcat started on port 8080 (http) with context path '/'
user-service  | 2025-03-20T07:26:40.288Z  INFO 1 --- [user-service] [           main] i.f.s.u.UserServiceApplication
     : Started UserServiceApplication in 2.327 seconds (process running for 2.543)
user-service  | 2025-03-20T07:26:49.453Z  INFO 1 --- [user-service] [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]
     : Initializing Spring DispatcherServlet 'dispatcherServlet'
user-service  | 2025-03-20T07:26:49.453Z  INFO 1 --- [user-service] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet
     : Initializing Servlet 'dispatcherServlet'
user-service  | 2025-03-20T07:26:49.454Z  INFO 1 --- [user-service] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet
     : Completed initialization in 1 ms
user-service  | Hibernate: select t1_0.id,t1_0.created_at,t1_0.created_by,t1_0.expiry_date,t1_0.revoked,t1_0.token,t1_0.up
dated_at,t1_0.updated_by,t1_0.user_id from tokens t1_0 where t1_0.token=?
```

Example – Multiple containers

Docker Compose



compose.yaml

```yaml
services:
  # Web container (nginx)
  mul-web: <6 keys>

  mul-mariadb: <8 keys>

  mul-adminer: <4 keys>

  mul-redis: <6 keys>

networks: multiple-containers

volumes: mariadb-data, redis-data
```

Example – Multiple containers (cont.)

Docker Compose

```yaml
services:
  # Web container (nginx)
  mul-web:
    image: nginx:alpine
    container_name: web-container
    volumes:
      - ./web:/usr/share/nginx/html:ro
    networks:
      - multiple-containers
    ports:
      - "8080:80"
    depends_on:
      - mul-mariadb
```

```yaml
mul-mariadb:
  image: mariadb:11.7.2
  container_name: mariadb-container
  restart: unless-stopped
  networks:
    multiple-containers:
      ipv4_address: 172.19.0.20
  volumes:
    - ./mariadb-data:/var/lib/mysql:rw
  ports:
    - "3307:3306"
  environment:
    - MYSQL_ROOT_PASSWORD=123456
    - MYSQL_DATABASE=test
    - MYSQL_USER=user
    - MYSQL_PASSWORD=123456
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
    interval: 30s
    retries: 3
```

Example – Multiple containers (cont.)

Docker Compose

```yaml
mul-adminer:
  image: adminer:latest
  container_name: adminer-container
  ports:
    - "8081:8080"
  networks:
    - multiple-containers
```

```yaml
mul-redis:
  image: redis:latest
  container_name: redis-container
  ports:
    - "6379:6379"
  volumes:
    - ./redis-data:/data
  restart: always
  networks:
    multiple-containers:
      ipv4_address: 172.19.0.40
```

Example – Multiple containers (cont.)

Docker Compose

```yaml
networks:
  multiple-containers:
    driver: bridge
    ipam:
      config:
        - subnet: 172.19.0.0/16

volumes:
  mariadb-data:
    driver: local
  redis-data:
    driver: local
```

Example – Multiple containers (cont.)

Docker Compose

1. Command: docker compose up -d
2. Truy cập web:
   http://localhost:8080/
3. Truy cập adminer:
   http://localhost:8081
   Với thông số:
   - Server: mul-mariadb
   - Username: root
   - Password: 123456
   - Database: test
4. Truy cập redis:
   - Trong container:
     docker exec -it redis-container redis-cli
   - Từ bên ngoài (sử dụng redis-cli trên máy chủ):
     redis-cli -h localhost -p 6379
   - Từ container khác trong Docker: Sử dụng tên service trong Docker Compose
     redis-cli -h mul-redis -p 6379

Docker Swarm

- Swarm mode is an advanced feature for managing a cluster of Docker daemons.
- Use Swarm mode if you intend to use Swarm as a production runtime environment.
- If you're not planning on deploying with Swarm, use Docker Compose instead. If you're developing for a Kubernetes deployment, consider using the integrated Kubernetes feature in Docker Desktop.

Ref: https://docs.docker.com/engine/swarm/#feature-highlights

Q&A