Docker 101

educative

Introduction to Docker

Docker is a platform that helps us create, deploy, and run applications using portable containers.

Docker Terminology

Dockerfile

A Dockerfile is a configuration file containing instructions to build a Docker image.

Docker image

A Docker image is a read-only template for creating containers.

Base image

A base image is the Docker image used as a template for building other images.

Container

A container is a runnable instance of a Docker image, and an image can have multiple instances. Docker containers are lightweight and can be deployed on any physical or virtual machine or cloud.

Layer

A layer in a Docker image is a distinct component in the build process. Each layer builds on the previous ones, forming the complete Docker image.

Docker engine

A Docker engine is a core component of Docker that runs and manages Docker containers.

Docker host

A Docker host is a physical or virtual machine running the engine.

Docker daemon

A Docker daemon is a background service that listens to Docker client commands and manages Docker images, containers, networks, and volumes.

Docker client

A Docker client is a command-line interface or graphical user interface that interacts with the Docker daemon.

Volume

Volume in Docker is a persistent data storage mechanism shared between the host and the containers.

Docker registry

A Docker registry is a server for storing and distributing Docker images.

Features of Docker

Application isolation

Docker allows running multiple applications in separate containers independent of each other.

Portability

Docker allows running containers consistently across different environments.

Scalability

Docker allows scaling applications up or down by adjusting the number of containers running the application.

Resource efficiency

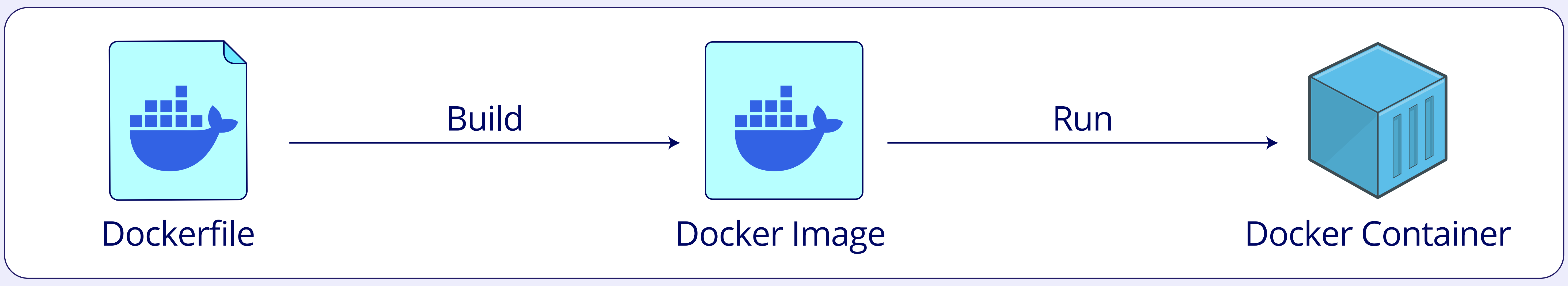
Docker allows using system resources efficiently by sharing the host OS kernel among containers.

Performance

Docker provides enhanced performance due to lightweight and fast container execution.

Dockerfile Creation

A Dockerfile is a text document containing instructions for building a Docker image, which is then used to run a container.



Dockerfile Instructions

Command	Purpose	Syntax	Example
`FROM`	Sets the base image.	`FROM <image>:<tag>`	`FROM ubuntu:20.04`
`WORKDIR`	Sets the working directory for subsequent RUN, CMD, ENTRYPOINT, COPY, and ADD commands.	`WORKDIR <path>`	`WORKDIR /app`
`COPY`	Copies files/directories from the host machine to the image.	`COPY <src> <dest>`	`COPY . /app/`
`ADD`	Similar to COPY, but can also handle remote URLs and extract tar files.	`ADD <src> <dest>`	`ADD app.tar.gz /app/`
`RUN`	Executes commands.	`RUN <command>`	`RUN apt-get update -y && apt-get install -y nginx`
`CMD`	This is the default command to run when the container starts.	`CMD ["executable", "param1", "param2"]`	`CMD ["nginx", "-g", "daemon off;"]`

Command	Purpose	Syntax	Example
`ENTRYPOINT`	Configures the primary executable for a Docker container.	`ENTRYPOINT ["executable", "param1"]`	`ENTRYPOINT ["nginx"]`
`EXPOSE`	It documents the ports exposed by a Docker container.	`EXPOSE <port>`	`EXPOSE 80`
`LABEL`	Adds metadata.	`LABEL <key>=<value>`	`LABEL version="1.0" description="My app"`
`VOLUME`	Creates a mount point with the specified path.	`VOLUME ["/path/to/volume"]`	`VOLUME ["/data"]`
`USER`	Sets the username or UID to use when running the container.	`USER <username>`	`USER appuser`
`ARG`	Defines a build-time variable.	`ARG <name>[=<default_value>]`	`ARG APP_VERSION=1.0`
`ONBUILD`	Adds an instruction that will be triggered when the image is used as a base for another image.	`ONBUILD <instruction>`	`ONBUILD RUN echo 'Image built!'`
`STOPSIGNAL`	Sets the system call signal that will be used to stop the container.	`STOPSIGNAL <signal>`	`STOPSIGNAL SIGTERM`
`HEALTHCHECK`	Used to test if the container is healthy.	`HEALTHCHECK [<options>] CMD <command>`	`HEALTHCHECK CMD curl --fail http://localhost:8080/`
`SHELL`	Sets shell for RUN commands in Dockerfile.	`SHELL ["<executable>", "<param1>", "<param2>"]`	`SHELL ["/bin/bash", "-c"]`

Sample Dockerfile

```
# Use Ubuntu 20.04 as the base image
FROM ubuntu:20.04

# Set the working directory inside the container
WORKDIR /app

# Copy the application files from the host to the container
COPY package.json /app/

# Run a command to update the package list and install curl
RUN apt-get update -y && apt-get install -y curl

# Add a file from a URL to the container
ADD https://example.com/file.tar.gz /app/

# Set an environment variable
ENV NODE_ENV=production

# Expose port 8080 to the outside world
EXPOSE 8080

# Create a volume for persistent data storage
VOLUME ["/data"]

# Set the default command to run when the container starts
CMD ["echo", "Hello, Docker!"]
```

Images

A Docker image is a read-only template for creating containers. It can be built from a Dockerfile or pulled from a Docker registry.

Command	Purpose	Syntax	Example
`docker build`	Builds an image from a Dockerfile.	`docker build -t <image-name>:<tag> <Dockerfile path>`	`docker build -t my_app:1.0 .`
`docker images` or `docker image ls`	Lists all images on the local system.	`docker images` or `docker image ls`	`docker images`
`docker rmi`	Removes an image from the local system.	`docker rmi <image>:<tag>`	`docker rmi my_app:1.0`
`docker tag`	Tags an image with a new name and tag.	`docker tag <image>:<tag> <new_image_name>:<new_tag>`	`docker tag my_app:1.0 theapp:2.0`
`docker history`	Shows the history of an image.	`docker history <image>:<tag>`	`docker history ubuntu:20.04`

Container

A Docker container is a lightweight, standalone, and executable environment with everything needed to run software. Containers are created from Docker images and can be started, stopped, and managed independently.

Command	Purpose	Syntax	Example
`docker run`	Starts a new container from an image. Optionally, you can specify a command to run when the container starts.	`docker run <options> <image> [<command> [<args>...]]`	`docker run -d -p 80:80 --name my_container nginx`
`docker ps` or `docker container ls`	Lists all running containers with their IDs, names, status, and other details.	`docker ps <options>` or `docker container ls <options>`	`docker ps`
`docker stop`	Stops a running container by name or ID.	`docker stop <options> <container> [<container>...]`	`docker stop my_container`
`docker start`	Starts a stopped container.	`docker start <options> <container> [<container>...]`	`docker start my_container`
`docker restart`	Restarts a running or stopped container.	`docker restart <options> <container> [<container>...]`	`docker restart my_container`
`docker rm`	Removes a stopped container.	`docker rm <options> <container> [<container>...]`	`docker rm my_container`
`docker exec`	Executes a command inside a running container.	`docker exec <container> <command> [<args>...]`	`docker exec -it my_container ls`

Options

Some of the common options you can use with `docker run` are given in the following table.

Option	Description
`-d, --detach`	Runs the container in the background and prints the container ID in the terminal.
`-p, --publish`	Maps a host port to a container port (e.g., -p hostPort:containerPort).
`-e, --env`	Sets environment variables (e.g., -e ENV_VAR=value).
`--name`	Assigns a specific name to the container (e.g., --name my_container). This helps in managing and referencing containers easily.
`-v, --volume`	Binds mount a volume (e.g., -v host_path:container_path).
`--network`	Connects the container to a specific network.
`--restart`	Defines a restart policy (e.g., --restart always).
`--rm`	Automatically removes the container when it exits.
`-it`	Runs the container in interactive mode with a terminal (e.g., -it ubuntu bash). `-i` keeps `stdin` open for command input, while `-t` allocates a pseudo-terminal for a better interactive experience.
`--init`	Runs an init process inside the container to manage zombie processes.
`--user`	Specifies the user to run the container as (e.g., --user user:group).
`--cpus`	Limits the number of CPUs available to the container (e.g., --cpus="2").
`--memory`	Limits the memory usage (e.g., --memory="256m").
`--gpus`	Specifies GPU resources (e.g., --gpus all).
`--read-only`	Mounts the container's filesystem in read-only mode.
`--entrypoint`	Overrides the default entry point of the image.
`--workdir, -w`	Sets the working directory inside the container (e.g., -w /app).
`--tmpfs`	Mounts a tmpfs (temporary filesystem) (e.g., --tmpfs /tmp).

Docker Registries

A Docker registry is a server-side application that allows you to host private or public repositories for Docker images. You can set up your own Docker registry server using the official Docker Registry image. This is useful for organizations that need to keep their images private or have specific requirements for image storage.

Docker Hub

Docker Hub is a cloud-based service provided by Docker Inc. It is the default public Docker registry. Docker Hub hosts public repositories but also supports private repositories for a fee (one private repository is free). The following commands interact with Docker Hub through Docker CLI commands for pushing, pulling, and managing Docker images.

Command	Purpose	Syntax	Example
`docker login`	Used to log in to Docker Hub.	`docker login <options> [<server>]`	`docker login -u my_user -p my_password`
`docker logout`	Used to log out from Docker Hub.	`docker logout [<server>]`	`docker logout`
`docker search`	Searches for images on Docker Hub.	`docker search <options> TERM`	`docker search nginx`
`docker pull`	Pulls an image from a Docker registry (from Docker Hub by default).	`docker pull <options> NAME[:tag|@digest]`	`docker pull ubuntu:20.04` or `docker pull myregistry.example.com/my_image:my_tag`

General Docker Management

Docker environments can be managed with commands for help, versioning, and system resource usage.

Docker Help Commands

Command	Purpose	Syntax	Example
`docker help`	Used to get help on Docker commands.	`docker help [<command>]`	`docker help run`

Version and Information Commands

Command	Purpose	Syntax	Example
`docker version`	Shows Docker version information.	`docker version <options>`	`docker version`
`docker info`	Shows detailed information about your Docker setup.	`docker info <options>`	`docker info`

System and Disk Usage Commands

Command	Purpose	Syntax	Example
`docker system df`	Shows disk usage for Docker images, containers, volumes, and build the cache.	`docker system df <options>`	`docker system df`
`docker system prune`	Removes unused data (containers, images, volumes, networks).	`docker system prune <options>`	`docker system prune -a`

Docker Volumes

Volumes are used for handling persistent data in Docker. Using them ensures that your containers can access and store data even as they stop, start, or are recreated.

Command	Purpose	Syntax	Example
`docker volume create`	Creates a new volume.	`docker volume create <options> [<volume_name>]`	`docker volume create my_volume`
`docker volume ls`	Lists all volumes.	`docker volume ls <options>`	`docker volume ls`
`docker volume rm`	Removes one or more volumes.	`docker volume rm <options> <volume> [<volume>...]`	`docker volume rm my_volume`
`docker volume prune`	Removes all unused volumes.	`docker volume prune`	`docker volume prune`
`docker volume inspect`	Displays detailed information about a volume.	`docker volume inspect <options> <volume> [<volume>...]`	`docker volume inspect my_volume`

Mounting Docker Volumes

Command	Purpose	Syntax	Example
`docker run -v`	Mounts a volume in a container.	`docker run -v [volume_name]:[path] [image]` or `docker run --volume [volume_name]:[path] [image]`	`docker run -v my_volume:/data my_image`
`docker run --mount`	Allows mounting differently, such as bind mounts, tmpfs, or volumes.	`docker run --mount type=volume,source=[volume_name],target=[path] [image]` or `docker run --mount type=bind,source=[source_path],target=[path] [image]` or `docker run --mount type=tmpfs,target=[path] [image]`	`docker run --mount type=volume,source=my_volume,target=/data my_image`

Note: Use -v for bind mounts to directly link host directories to containers. Use --mount for more options, including bind, volume, and tmpfs mounts for temporary data.

Networking

Networking in Docker allows containers to communicate with each other and the outside world.

Command	Purpose	Syntax	Example
`docker network create`	Creates a new network.	`docker network create <options> <network_name>`	`docker network create my_network`
`docker network ls`	Lists all networks.	`docker network ls <options>`	`docker network ls`
`docker network inspect`	Displays detailed information about a network.	`docker network inspect <options> <network> [<network>...]`	`docker network inspect my_network`
`docker network rm`	Removes one or more networks.	`docker network rm <options> <network> [<network>...]`	`docker network rm my_network`
`docker network prune`	Removes all unused networks.	`docker network prune <options>`	`docker network prune`

Network Driver Option

You can specify the network driver using the --driver <driver> option while creating the network.

Driver	Purpose
`bridge`	It is a default network driver for standalone containers.
`overlay`	It creates a multi-host network.
`macvlan`	It assigns a MAC address to a container.

Attaching a Network to a Docker Container

Command	Purpose	Syntax	Example
`docker network connect`	Connects a container to a network.	`docker network connect <network> <container>`	`docker network connect my_network my_container`
`docker network disconnect`	Disconnects a container from a network.	`docker network disconnect <network> <container>`	`docker network disconnect my_network my_container`
`docker run --network`	Creates and runs a new container connected to a specified network.	`docker run --network <network> <image>`	`docker run --network my_network my_app`

Debugging and Logs

Docker containers are monitored by examining logs and detailed container information.

Command	Purpose	Syntax	Example
`docker logs`	Fetches the logs of a container.	`docker logs <options> <container>`	`docker logs my_container`
`docker inspect`	Views detailed information about a container.	`docker inspect <options> <name>`	`docker inspect my_container`
`docker stats`	Displays real-time resource usage statistics for containers.	`docker stats <options> [<container>...]`	`docker stats`

Docker Compose

Docker Compose simplifies the management of multi-container Docker applications using a `docker-compose.yml` file

Docker Compose File Example

An example of the `docker-compose.yml` file is:

```
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
    volumes:
      - db_data:/var/lib/mysql

volumes:
  db_data:
```

Docker Compose Commands

Command	Purpose	Syntax	Example
`docker-compose up`	Starts up all services defined in the `docker-compose.yml` file.	`docker-compose up <options>`	`docker-compose up`
`docker-compose down`	Stops and removes all services defined in the `docker-compose.yml` file.	`docker-compose down <options>`	`docker-compose down`
`docker-compose start`	Starts services defined in the `docker-compose.yml` file that are stopped.	`docker-compose start [<service>...]`	`docker-compose start web`
`docker-compose restart`	Restarts services defined in the `docker-compose.yml` file.	`docker-compose restart [<service>...]`	`docker-compose restart db`
`docker-compose build`	Builds or rebuilds services defined in the `docker-compose.yml` file.	`docker-compose build <options> [<service>...]`	`docker-compose build`
`docker-compose logs`	Views output from containers.	`docker-compose logs <options> [<service>...]`	`docker-compose logs`