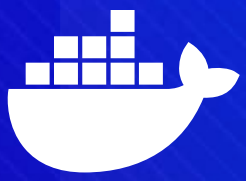


Docker Image

Docker Image

1. Build an image from a Dockerfile:

Terminal

```
> docker build -t image_name path_to_dockerfile

# EXAMPLE
> docker build -t myapp .
```

2. List all local images:

Terminal

```
> docker images

# EXAMPLE
> docker image ls
```

3. Pull an image from Docker Hub:

Terminal

```
> docker pull image_name:tag

# EXAMPLE
> docker pull nginx:latest
```

Docker Image

4. Remove a local image:

Terminal

```
> docker rmi image_name:tag

# EXAMPLE
> docker rmi myapp:latest
```

Or

Terminal

```
> docker rm [image_name/image_id]

# EXAMPLE
> docker rm fd484f19954f
```

5. Tag an image:

Terminal

```
> docker tag source_image:tag new_image:tag

# EXAMPLE
> docker tag myapp:latest myapp:v1
```

Docker Image

6. Push an image to Docker Hub:

Terminal

```
> docker push image_name:tag

# EXAMPLE
> docker push myapp:v1
```

7. Inspect details of an image:

Terminal

```
> docker image inspect image_name:tag

# EXAMPLE
> docker image inspect myapp:v1
```

8. Save an image to a tar archive:

Terminal

```
> docker save -o image_name.tar image_name:tag

# EXAMPLE
> docker save -o myapp.tar myapp:v1
```

Docker Image

9. Load an image from a tar archive:

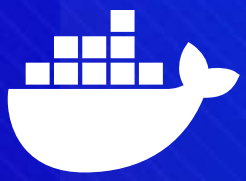
Terminal

```
> docker load -i image_name.tar

# EXAMPLE
> docker load -i image_name.tar
```

10. Prune unused images:

Terminal

```
> docker image prune
```



Docker Container

Docker Container

1. Run a container from an image:

Terminal

```
> docker run container_name image_name

# EXAMPLE
> docker run myapp
```

2. Run a named container from an image:

Terminal

```
> docker run --name container_name image_name:tag

# EXAMPLE
> docker run --name my_container myapp:v1
```

3. List all running containers:

Terminal

```
> docker ps
```

Docker Container

4. List all containers (including stopped ones):

Terminal

```
> docker ps -a
```

5. Stop a running container:

Terminal

```
> docker stop container_name_or_id

# EXAMPLE
> docker stop my_container
```

6. Start a stopped container:

Terminal

```
> docker start container_name_or_id

# EXAMPLE
> docker start my_container
```

Docker Container

7. Run container in interactive mode:

Terminal

```
> docker run -it container_name_or_id

# EXAMPLE
> docker run -it my_container
```

8. Run container in interactive shell mode

Terminal

```
> docker run -it container_name_or_id sh

# EXAMPLE
> docker run -it my_container sh
```

9. Remove a stopped container:

Terminal

```
> docker rm container_name_or_id

# EXAMPLE
> docker rm my_container
```

Docker Container

10. Remove a running container (forcefully):

Terminal

```
> docker rm -f container_name_or_id

# EXAMPLE
> docker rm -f my_container
```

11. Inspect details of a container:

Terminal

```
> docker inspect container_name_or_id

# EXAMPLE
> docker inspect my_container
```

12. View container logs:

Terminal

```
> docker logs container_name_or_id

# EXAMPLE
> docker logs my_container
```

Docker Container

13. Pause a running container:

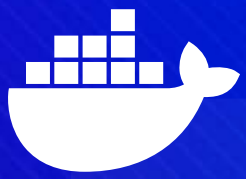
Terminal

```
> docker pause container_name_or_id

# EXAMPLE
> docker pause my_container
```

14. Unpause a paused container:

Terminal

```
> docker unpause container_name_or_id

# EXAMPLE
> docker unpause my_container
```



Docker Volumes and Network

Docker Volumes and Network

VOLUMES:

1. Create a named volume:

Terminal

```
> docker volume create volume_name

# EXAMPLE
> docker volume create my_volume
```

2. List all volumes:

Terminal

```
> docker volume ls
```

Docker Volumes and Network

3. Inspect details of a volume:

Terminal

```
> docker volume inspect volume_name

# EXAMPLE
> docker volume inspect my_volume
```

4. Remove a volume:

Terminal

```
> docker volume rm volume_name

# EXAMPLE
> docker volume rm my_volume
```

Docker Volumes and Network

5. Run a container with a volume (mount):

Terminal

```
> docker run --name container_name -v volume_name:/path/in/
  container image_name:tag

# EXAMPLE
> docker run --name my_container -v my_volume:/app/data myapp:v1
```

6. Copy files between a container and a volume:

Terminal

```
> docker cp local_file_or_directory container_name:/path/in/
  container

# EXAMPLE
> docker cp data.txt my_container:/app/data
```

Docker Volumes and Network

NETWORK (PORT MAPPING):

1. Run a container with port mapping:

Terminal

```
> docker run --name container_name -p host_port:container_port
  image_name

# EXAMPLE
> docker run --name my_container -p 8080:80 myapp
```

2. List all networks:

Terminal

```
> docker network ls
```

Docker Volumes and Network

3. Inspect details of a network:

Terminal

```
> docker network inspect network_name

# EXAMPLE
> docker network inspect bridge
```

4. Create a user-defined bridge network:

Terminal

```
> docker network create network_name

# EXAMPLE
> docker network create my_network
```

5. Connect a container to a network:

Terminal

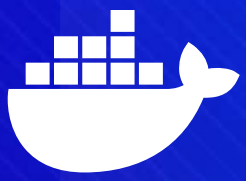
```
> docker network connect network_name container_name

# EXAMPLE
> docker network connect my_network my_container
```

Docker Volumes and Network

6. Disconnect a container from a network:

Terminal

```
> docker network disconnect network_name container_name

# EXAMPLE
> docker network disconnect my_network my_container
```



Docker Compose

Docker Compose

1. Create and start containers defined in a docker-compose.yml file:

This command reads the docker-compose.yml file and starts the defined services in the background.

2. Stop and remove containers defined in a docker-compose.yml file:

This command stops & removes the containers, networks, and volumes defined in the docker-compose.yml file.

Docker Compose

3. Build or rebuild services:

```
Terminal
> docker compose build
```

This command builds or rebuilds the Docker images for the services defined in the docker compose.yml file.

4. List containers for a specific Docker Compose project:

```
Terminal
> docker compose ps
```

This command lists the containers for the services defined in the docker-compose.yml file.

Docker Compose

5. View logs for services:

Terminal

```
> docker compose logs
```

This command shows the logs for all services defined in the docker-compose.yml file.

6. Scale services to a specific number of containers:

Terminal

```
> docker compose up -d --scale
  service_name=number_of_containers

# EXAMPLE
> docker compose up -d --scale web=3
```

Docker Compose

7. Run a one-time command in a service:

Terminal

```
> docker compose run service_name command

# EXAMPLE
> docker compose run web npm install
```

8. List all volumes:

Terminal

```
> docker volume ls
```

Docker Compose creates volumes for services. This command helps you see them.

Docker Compose

9. Pause a service:

```
Terminal
> docker volume pause service_name
```

This command pauses the specified service.

10. Unpause a service:

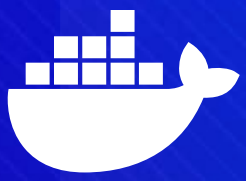
```
Terminal
> docker volume unpause service_name
```

This command unpauses the specified service.

11. View details of a service:

```
Terminal
> docker compose ps service_name
```

Provides detailed information about a specific service.



Latest Docker

Latest Docker

1. Initialize Docker inside an application

Terminal

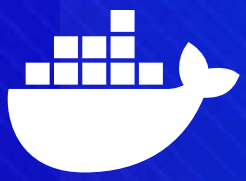
```
> docker init
```

2. Watch the service/container of an application

Terminal

```
> docker compose watch
```

It watches build context for service and rebuild/refresh containers when files are updated



Dockerfile Reference

Dockerfile Reference

What is a Dockerfile?

A Dockerfile is a script that contains instructions for building a Docker image. It defines the base image, sets up environment variables, installs software, and configures the container for a specific application or service.

DOCKERFILE SYNTAX

FROM:

Specifies the base image for the Docker image.

```
FROM image_name:tag

# EXAMPLE
FROM ubuntu:20.04
```

Dockerfile Reference

WORKDIR:

Sets the working directory for subsequent instructions.

```
WORKDIR /path/to/directory

# EXAMPLE
WORKDIR /app
```

COPY:

Copies files or directories from the build context to the container.

```
COPY host_source_path container_destination_path

# EXAMPLE
COPY . .
```

Dockerfile Reference

RUN:

Executes commands in the shell.

```
RUN command1 && command2

# EXAMPLE
RUN apt-get update && apt-get install -y curl
```

ENV:

Sets environment variables in the image.

```
ENV KEY=VALUE

# EXAMPLE
ENV NODE_VERSION=14
```

Dockerfile Reference

EXPOSE:

Informs Docker that the container listens on specified network ports at runtime.

```
EXPOSE port

# EXAMPLE
EXPOSE 8080
```

CMD:

Provides default commands or parameters for an executing container.

```
CMD ["executable","param1","param2"]

# EXAMPLE
CMD ["npm", "start"]
```

Or,

Dockerfile Reference

```
CMD executable param1 param2

# EXAMPLE
CMD npm run dev
```

ENTRYPOINT:

Configures a container that will run as an executable.

```
ENTRYPOINT ["executable","param1","param2"]

# EXAMPLE
ENTRYPOINT ["node", "app.js"]
```

Or,

```
ENTRYPOINT executable param1 param2

# EXAMPLE
ENTRYPOINT node app.js
```

Dockerfile Reference

ARG:

Defines variables that users can pass at build-time to the builder with the docker build command.

```
ARG VARIABLE_NAME=default_value

# EXAMPLE
ARG VERSION=latest
```

VOLUME:

Creates a mount point for external volumes or other containers.

```
VOLUME /path/to/volume

# EXAMPLE
VOLUME /data
```

Dockerfile Reference

LABEL:

Adds metadata to an image in the form of key-value pairs.

```
LABEL key="value"

# EXAMPLE
LABEL version="1.0" maintainer="Adrian"
```

USER:

Specifies the username or UID to use when running the image.

```
USER user_name

# EXAMPLE
USER app
```

Dockerfile Reference

ADD:

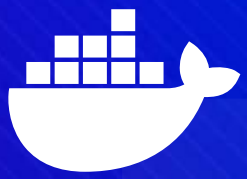
Copies files or directories and can extract tarballs in the process.

```
ADD source_path destination_path

# EXAMPLE
ADD ./app.tar.gz /app
```

Similar to **COPY**, but with additional capabilities (e.g., extracting archives).

Dockerfile Example

```
# Use an official Node.js runtime as a base image
FROM node:20-alpine

# Set the working directory to /app
WORKDIR /app

# Copy package.json and package-lock.json to the
working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the current directory contents to the container
at /app
COPY . .

# Expose port 8080 to the outside world
EXPOSE 8080

# Define environment variable
ENV NODE_ENV=production

# Run app.js when the container launches
CMD node app.js
```



Docker Compose File Reference

Docker Compose File Reference

What is a Docker Compose File?

A Docker Compose file is a YAML file that defines a multi-container Docker application. It specifies the services, networks, and volumes for the application, along with any additional configuration options.

DOCKER COMPOSE FILE SYNTAX

version:

Specifies the version of the Docker Compose file format.

Example:

```
version: '3.8'
```

Docker Compose File Reference

services:

Defines the services/containers that make up the application.

Example:

```
services:
   web:
      image: nginx:latest
```

networks:

Configures custom networks for the application.

Example:

```
networks:
  my_network:
    driver: bridge
```

Docker Compose File Reference

volumes:

Defines named volumes that the services can use.

Example:

```
volumes:
  my_volume:
```

environment:

Sets environment variables for a service.

Example:

```
environment:
  - NODE_ENV=production
```

Docker Compose File Reference

ports:

Maps host ports to container ports.

Example:

```
ports:
  - "8080:80"
```

depends_on:

Specifies dependencies between services, ensuring one service starts before another.

Example:

```
depends_on:
  - db
```

Docker Compose File Reference

build:

Configures the build context and Dockerfile for a service.

Example:

```
build:
  context: .
  dockerfile: Dockerfile.dev
```

volumes_from:

Mounts volumes from another service or container.

Example:

```
volumes_from:
  - service_name
```

Docker Compose File Reference

command:

Overrides the default command specified in the Docker image.

Example:

```
command: ["npm", "start"]
```

Docker Compose File Example

Here's a simple Docker Compose file example for a web and database service:

```yaml
version: '3.8'

# Define services for the MERN stack
services:

  # MongoDB service
  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: admin

  # Node.js (Express) API service
  api:
    build:
   # Specify the build context for the API service
      context: ./api

   # Specify the Dockerfile for building the API service
      dockerfile: Dockerfile
```

Docker Compose File Example

```
    ports:
      - "5000:5000"

  # Ensure the MongoDB service is running before starting
the API
    depends_on:
      - mongo

  environment:
      MONGO_URI: mongodb://admin:admin@mongo:27017/
mydatabase

  networks:
      - mern_network

  # React client service
  client:
    build:
   # Specify the build context for the client service
      context: ./client
   # Specify the Dockerfile for building the client service

      dockerfile: Dockerfile

  ports:
      - "3000:3000"
```

Docker Compose File Example

```
 # Ensure the API service is running before starting the
client
  depends_on:
      - api

    networks:
      - mern_network

# Define named volumes for persistent data
volumes:
  mongo_data:

# Define a custom network for communication between
services
networks:
  mern_network:
```