Empowering The Future Today

Docker Tutorials

By Engr Mbachan Fabrice Tanwan
Email: fabricembachan@gmail.com
Website: mbachanfabrice.com
Msc in Security and Network Engineering @innopolis University, Russia.

Agenda

- What is Docker
- What are Containers
- Why Use Containers
- Getting started with Containers
- Webapps with Docker
- Multi Container Environment

What is Docker?

What is Docker ?

An **open-source** project that automates software application deployment within containers by offering an extra layer of abstraction and automation for OS-level virtualization on Linux.

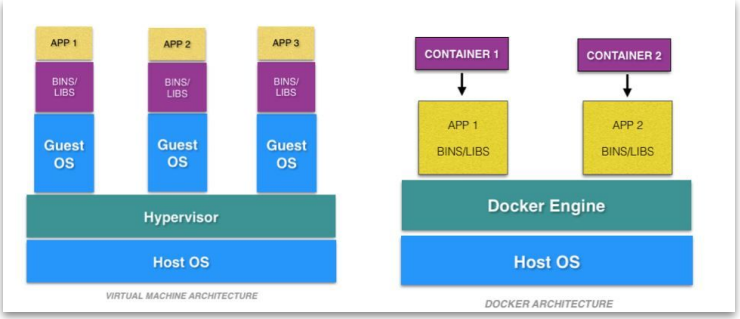
What is Docker?

What is Docker ?

Docker is a tool that allows developers and system administrators to easily deploy applications within isolated environments called containers, which run on the host operating system, usually Linux.

The **key benefit** of Docker is that it bundles an application together with all its dependencies into a standardized unit for software development.

Unlike virtual machines, containers have very little overhead, making it possible to use system resources more efficiently.



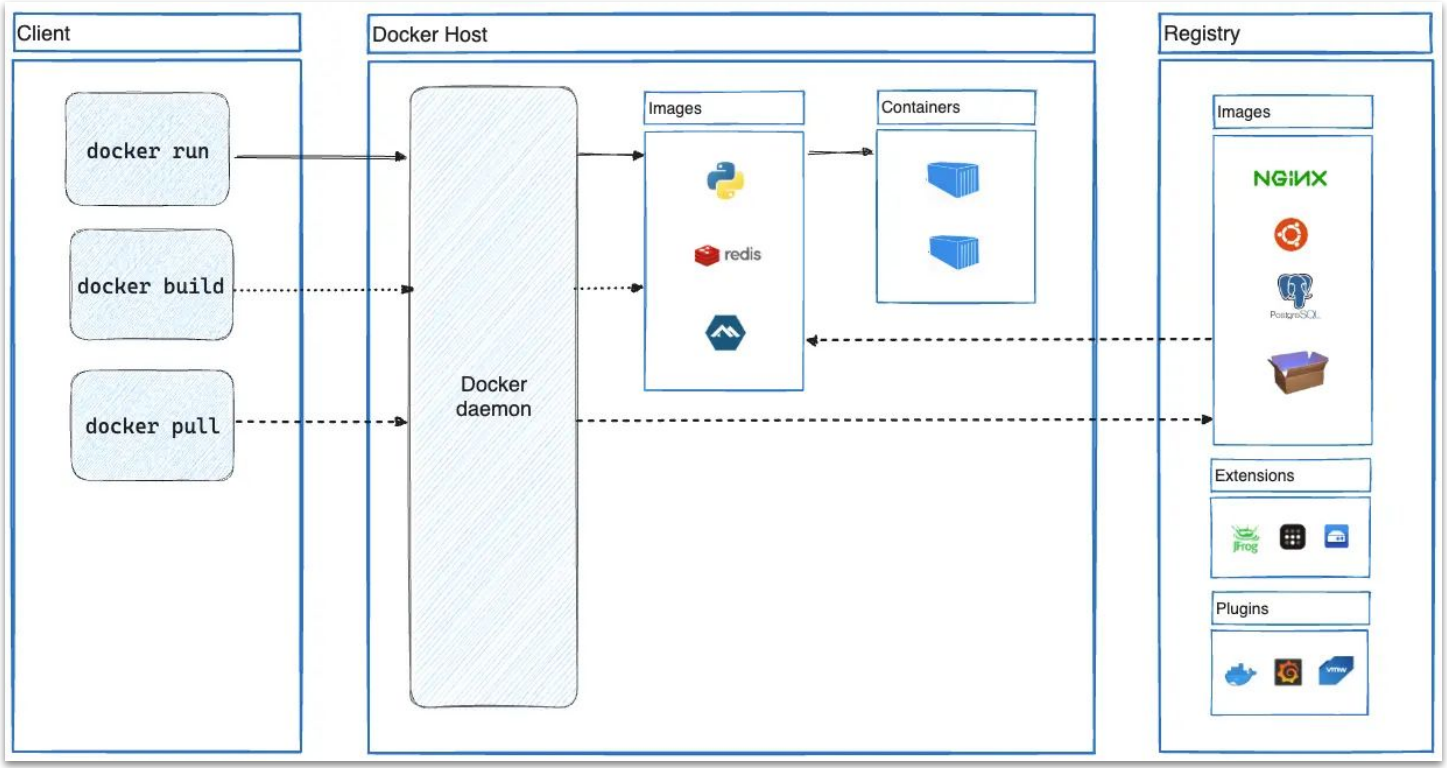
What is Docker?

What Can Use Docker For?

- Application Deployment
- Microservices Architecture
- Scaling Applications
- Continuous Integration/Continuous Deployment (CI/CD)
- Testing and Debugging
- Simplified Configuration Management
- Version Control for Environments
- Isolated Sandboxes for Security Testing

Docker

Docker Architecture



Docker Components

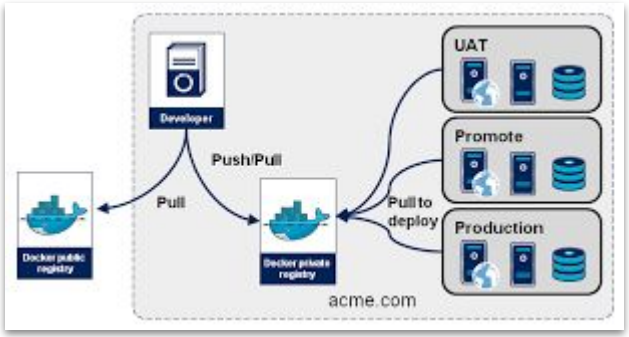
The Docker Clients:

The **Docker Client** is the primary interface used to interact with the Docker Engine. It provides the command-line tools for users to communicate with the Docker daemon (server) and manage Docker containers, images, volumes, networks, and more.

Common Docker Client Commands:

- **docker run:** Runs a container from an image.
- **docker build:** Builds a Docker image from a Dockerfile.
- **docker pull:** Downloads images from Docker Hub or another repository.
- **docker push**: Uploads images to a repository.
- **docker ps**: Lists running containers.
- **docker stop:** Stops running containers.
- **docker logs**: Displays logs for a container.

Docker Components



The Docker Registries:

Docker registries are storage and distribution systems for Docker images. The most common public registry is Docker Hub, but private registries can also be created for custom images.

- **Commands**:
  - List images in a registry: docker search <image_name>
  - Push an image to a registry: docker push <image_name>

Docker

Docker Objects

What is Docker Image?

Docker Image

Docker images are read-only templates used to create containers. They contain everything needed to run an application, including the code, runtime, libraries, environment variables, and configuration files. Images can be built from a Dockerfile or downloaded from a registry like Docker Hub.

Commands:

- List images: `docker images`
- Pull an image: `docker pull <image_name>`
- Build an image: `docker build -t <image_name> .`

Containers

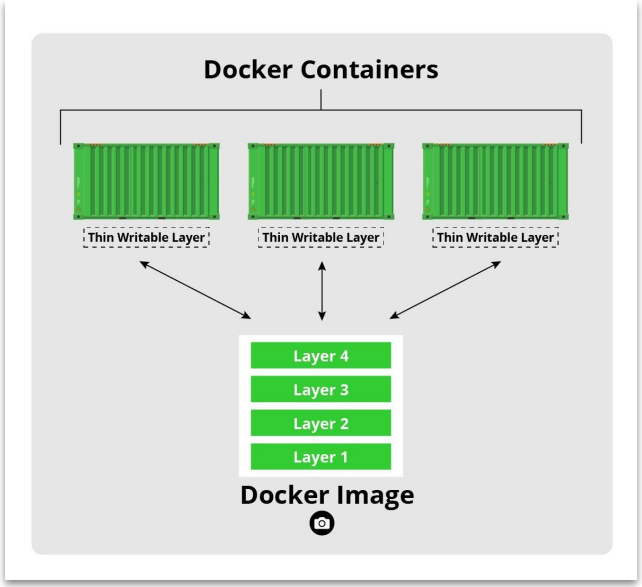
Docker Container

Containers are a **lightweight alternative** to virtual machines (VMs) for running software applications.

While VMs operate inside a **guest operating system** on virtual hardware provided by the host OS, containers rely on the host OS's underlying mechanisms to run applications.

This allows containers to offer a similar level of **isolation** as VMs but with significantly less **computational overhead,** making them much more efficient.

NB: A container is a running instance of an image. So, the container is created from the image, and the image provides the blueprint or template for the container.

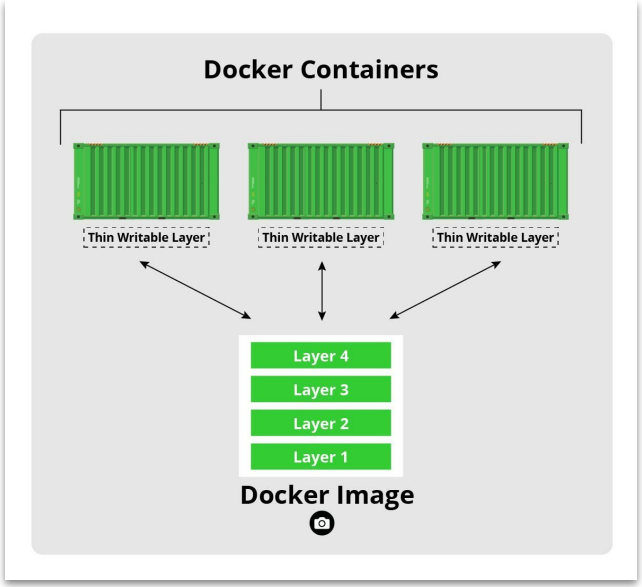


Containers

Docker Container

Commands:

- List running containers: `docker ps`
- List all containers (including stopped): `docker ps -a`
- Run a container: `docker run <image_name>`
- Stop a container: `docker stop <container_id>`
- Remove a container: `docker rm <container_id>`



Why Use Containers

1. Consistency Across Environments
2. Lightweight and Fast
3. Isolation
4. Portability
5. Scalability
6. Faster Development and Deployment
7. Improved Resource Utilization
8. Easier Application Management
9. Enhanced Security
10. Simplified Dependency Management
11. Better Collaboration Across Teams
12. Support for Microservices Architecture
13. Rapid Scaling and Orchestration

Volumes

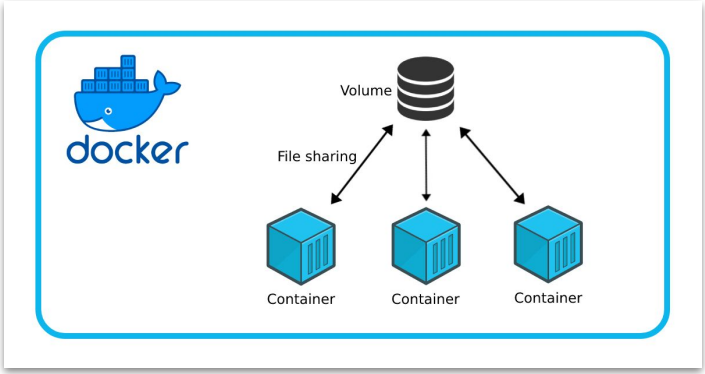
Volumes

Volumes are persistent storage objects used by containers to store and share data. They allow data to persist even when containers are stopped or removed.

Volumes can be shared between multiple containers and are managed by Docker outside of the container's lifecycle

Commands:

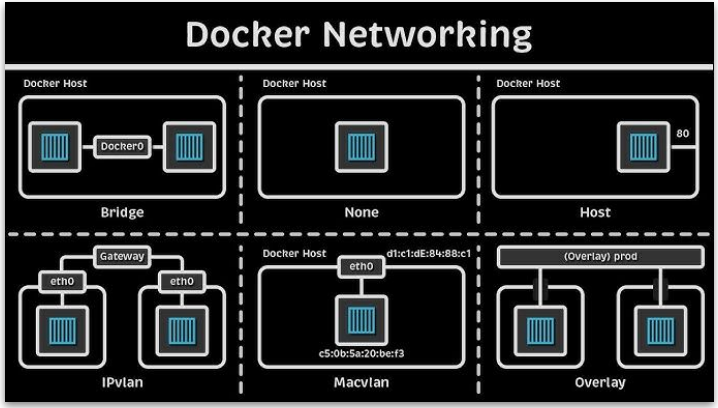
- List volumes: docker volume ls
- Create a volume: docker volume create <volume_name>
- Attach a volume to a container: docker run -v <volume_name>:/path/in/container <image_name>
- Remove a volume: docker volume rm <volume_name>



Networks

Networks

Docker networks allow containers to communicate with each other and with external systems. Docker provides different types of networks such as bridge, host, overlay, and macvlan.



Commands:

- List networks: docker network ls
- Create a network: docker network create <network_name>
- Connect a container to a network: docker network connect <network_name> <container_name>
- Inspect network details: docker network inspect <network_name>

Registries & Docker File

Docker registries are storage and distribution systems for Docker images. The most common public registry is Docker Hub, but private registries can also be created for custom images.

- **Commands**:
  - List images in a registry: `docker search <image_name>`
  - Push an image to a registry: `docker push <image_name>`

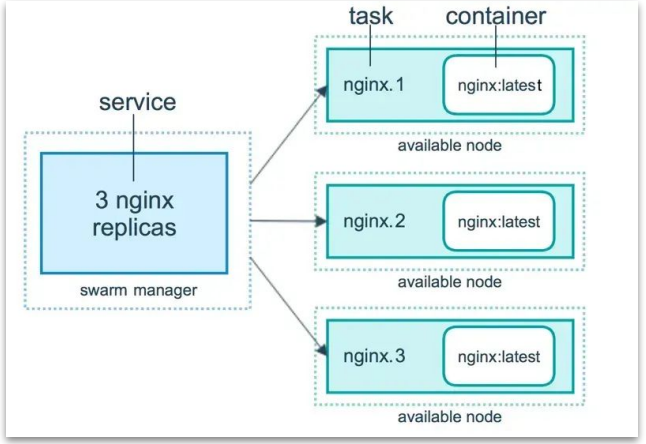
A Dockerfile is a text file containing instructions on how to build a Docker image. It defines the base image, the installation of software packages, and configurations needed to run the application.

- **Basic Commands in a Dockerfile**:
  - `FROM` – Specifies the base image.
  - `RUN` – Executes commands in a new layer on top of the base image.
  - `COPY` or `ADD` – Copies files or directories into the image.
  - `CMD` – Specifies the default command to run when the container starts.

Services

Services

A **Docker service** is an abstraction that allows you to define how a containerized application should run in a distributed environment, such as Docker Swarm. It specifies details like the number of replicas, the image to use, and how tasks are distributed across nodes.



Basic Commands for Docker Services:

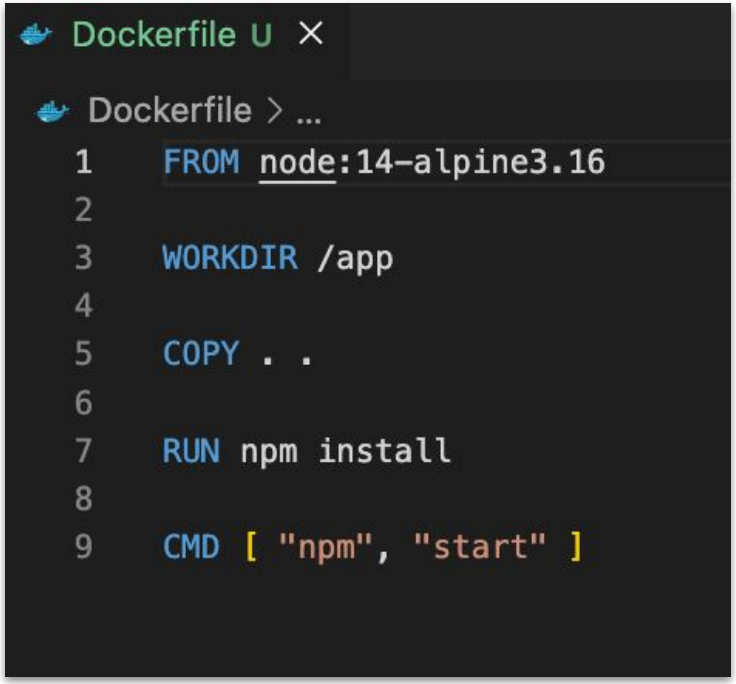
- **docker service create** – Creates a new service, specifying the image and how the service should run (e.g., number of replicas, port mapping).
- **docker service ls** – Lists all the running services.
- **docker service scale** – Scales the number of running instances (replicas) of a service.
- **docker service update** – Updates the configuration or image of an existing service.
- **docker service rm** – Removes an existing service.

Dockerfile

Dockerfile

While not strictly an object, a Dockerfile is the script that defines how to build a Docker image.

It includes commands for assembling an image from various layers.



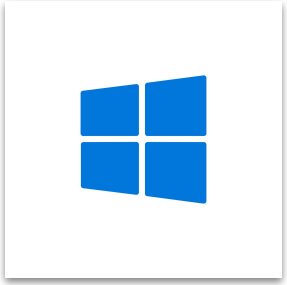
Docker Prerequisites

Getting Started

"There are no specific skills needed for this tutorial beyond a basic comfort with the command line and using a text editor"

Installing Docker

Getting Started



Installing Docker

Installation Option

Docker Desktop

is an all-in-one solution that includes Docker Engine, Docker CLI, Docker Compose, and Kubernetes (optional). It's designed for use on developer workstations and provides a more comprehensive, user-friendly experience.

Features:

- GUI
- Docker Compose
- Integrated WSL
- Kubernetes
- Platform (Windows & MacOS)
- FIle Sharing

Docker Engine

is the core Docker component responsible for running containers. It includes the Docker daemon, CLI, and APIs that allow users to interact with the Docker service.

Features:

- No GUI
- Lightweight
- Customizable

Installing Docker

Installation Option

Choose Docker Desktop if:

- You are a developer on Windows or macOS and need a simple, easy-to-use Docker environment with a GUI.
- You want an integrated tool that provides Docker Engine, Docker Compose, and Kubernetes for local development.

Choose Docker Engine if:

- You are working on Linux or deploying Docker in a production environment where you don’t need a GUI.
- You prefer lightweight, customizable setups, especially on servers or cloud environments.

Installing Docker

Installation On Linux (Debian x64)

Step 1: Update the apt package index

Open your terminal and run the following command to ensure your package list is up to date:

sudo apt-get update

```
root@devops:~# sudo  apt-get   update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:2 http://gb.archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://gb.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/main i386 Packages [554 k
B]
Get:5 https://download.docker.com/linux/ubuntu jammy InRelease [48.8 kB]
```

Installing Docker

Installation On Linux (Debian x64)

Step 2: Install dependencies and Setup Docker repository

Before installing Docker Engine for the first time on a new host, you need to configure the Docker APT repository. Once it's set up, you can install and update Docker directly from the repository

1. Setup Dockers apt Repository using the following commands

- *sudo apt-get update*
- *sudo apt-get install ca-certificates curl*
- *sudo install -m 0755 -d /etc/apt/keyrings*
- *sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc*
- *sudo chmod a+r /etc/apt/keyrings/docker.asc*

Installing Docker

Installation On Linux (Debian x64)

Step 2: Install dependencies and Setup Docker repository

Before installing Docker Engine for the first time on a new host, you need to configure the Docker APT repository. Once it's set up, you can install and update Docker directly from the repository

2. Install the Docker Packages with the following command

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Installing Docker

Installation On Linux (Debian x64)

Step 2: Install dependencies and Setup Docker repository

3. Verifying Installations and Docker Engine

- *sudo docker systemctl status/start/stop docker*
- *sudo docker run hello-world*
- *sudo docker images*
- *sudo docker ps*

```
root@devops:~# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset>
     Active: active (running) since Tue 2024-10-29 18:44:14 CDT; 16min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 845 (dockerd)
      Tasks: 10
     Memory: 106.0M
        CPU: 1.107s
     CGroup: /system.slice/docker.service
             └─845 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/conta>

Oct 29 18:44:09 devops dockerd[845]: time="2024-10-29T18:44:09.633972225-05:00">
Oct 29 18:44:10 devops dockerd[845]: time="2024-10-29T18:44:10.242621094-05:00">
Oct 29 18:44:10 devops dockerd[845]: time="2024-10-29T18:44:10.470077226-05:00">
Oct 29 18:44:12 devops dockerd[845]: time="2024-10-29T18:44:12.540404731-05:00">
Oct 29 18:44:13 devops dockerd[845]: time="2024-10-29T18:44:13.205271089-05:00">
Oct 29 18:44:13 devops dockerd[845]: time="2024-10-29T18:44:13.205559988-05:00">
Oct 29 18:44:13 devops dockerd[845]: time="2024-10-29T18:44:13.702682055-05:00">
```

Uninstalling Docker

Installation On Linux (Debian x64)

Step 3: Uninstalling Docker

1. Stop all running containers (optional but recommended)

Before uninstalling Docker, you may want to stop all running containers:

- *sudo docker ps -q | xargs -r sudo docker stop*

2. Uninstall Docker Engine, CLI, and associated packages

Run the following command to remove Docker Engine and all its components:

- *sudo apt-get purge docker-ce docker-ce-cli containerd.io*

3. Remove Docker data (optional)

To delete all Docker-related data (containers, images, volumes, networks, etc.), remove the Docker directory:

- *sudo rm -rf /var/lib/docker*
- *sudo rm -rf /var/lib/containerd*

Creating Objects

Creating the above Objects

Lets create a Simple NGINX Image Using a Dockerfile

1. Set up a Directory for the Project

```
root@devops:~# cd  nginx-project/
root@devops:~/nginx-project# ls
root@devops:~/nginx-project# nano  index.html
root@devops:~/nginx-project# ls
index.html
root@devops:~/nginx-project#
```

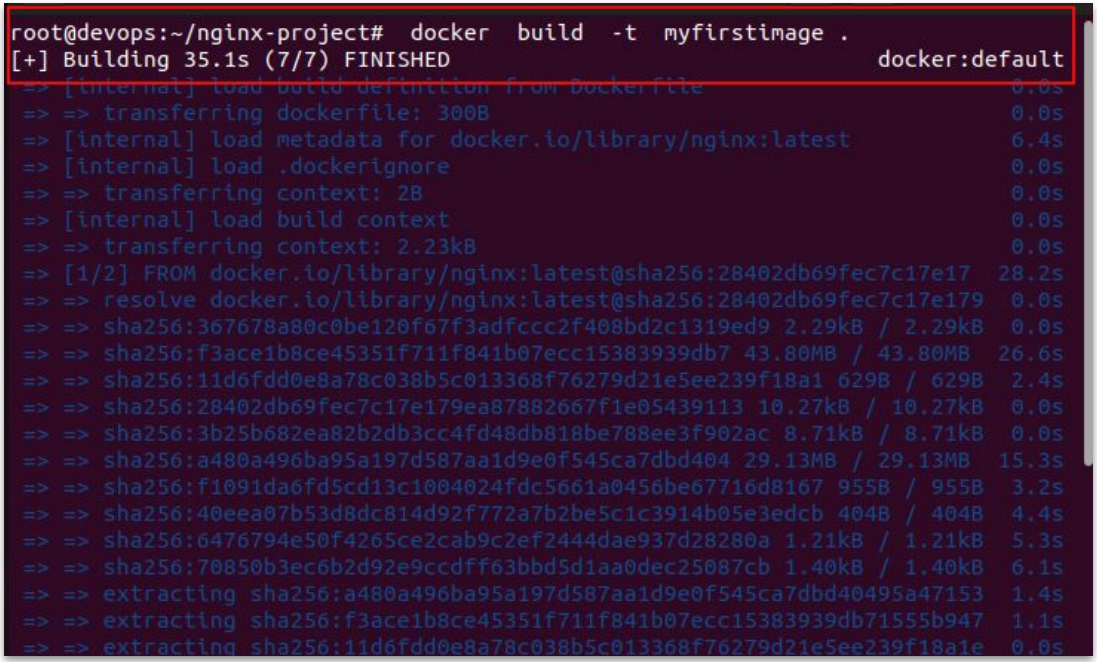
```
root@devops:~/nginx-project# nano  Dockerfile
root@devops:~/nginx-project# ls
Dockerfile  index.html
root@devops:~/nginx-project#
```

Creating Objects

Creating the above Objects

Lets create a Simple NGINX Image Using a Dockerfile

2. Building the image



Creating Objects

Creating the above Objects

Lets create a Simple NGINX Image Using a Dockerfile

2. Building the image (Verify Image)

```
root@devops:~/nginx-project# docker  iamges
docker: 'iamges' is not a docker command.
See 'docker --help'
root@devops:~/nginx-project# docker  images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
myfirstimage    latest    8721d2d1a9b7   2 minutes ago    192MB
web-server      latest    192d1c632a5a   8 weeks ago      1.24GB
grpc-manager    latest    52b69e0283b6   8 weeks ago      1.09GB
eqalpha/keydb   latest    33724420d9a5   12 months ago    137MB
hello-world     latest    d2c94e258dcb   18 months ago    13.3kB
root@devops:~/nginx-project#
```

Creating Objects

Creating the above Objects

Lets create a Simple NGINX Image Using a Dockerfile

3. Create a container from the running images

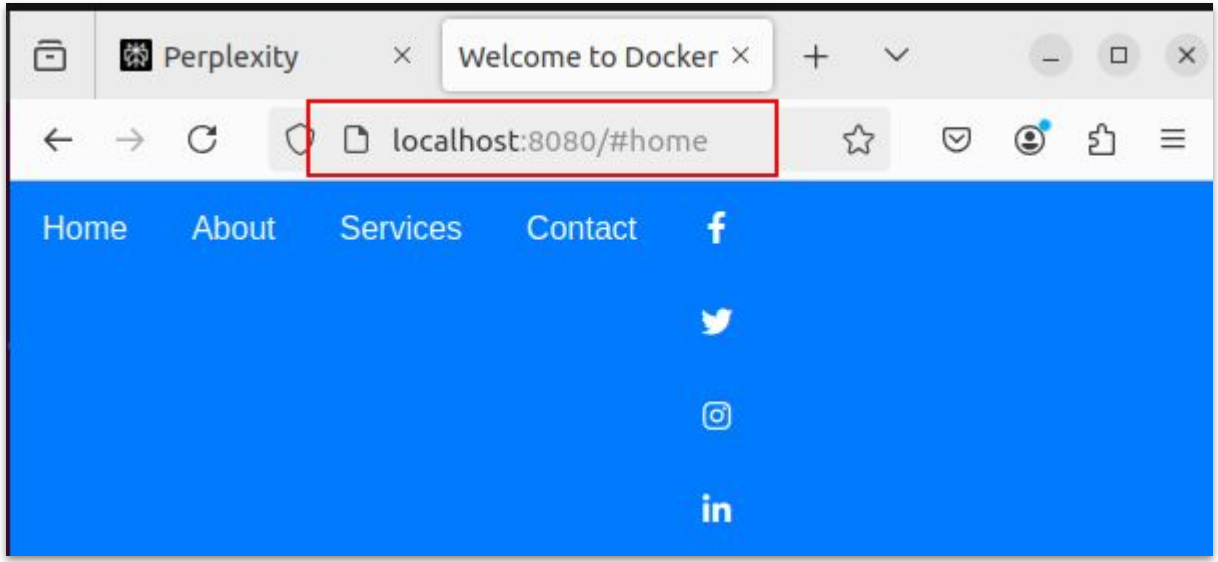
```
root@devops:~/nginx-project# docker  run -d -p 8080:80 --name myfirstcontainer m
yfirstimage
b8e8ff26661bb3958fb80eecdef6cbb5dbca4c327f3d2600b04ae1388b5d30cf
root@devops:~/nginx-project# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
      PORTS                                  NAMES
b8e8ff26661b   myfirstimage   "/docker-entrypoint.…"   9 seconds ago   Up 9 seco
nds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   myfirstcontainer
root@devops:~/nginx-project#
```

Creating Objects

Creating the above Objects

Lets create a Simple NGINX Image Using a Dockerfile

3. Testing the container using configured port



Creating Objects

Creating the above Objects

Lets create a Simple NGINX Image Using a Dockerfile

4. Stopping the container

```
root@devops:~/nginx-project# docker  stop myfirstcontainer
myfirstcontainer
root@devops:~/nginx-project# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@devops:~/nginx-project#
```

Creating Objects

Multi-Container Application

If your application consists of more than one service (e.g., a web server, a database, and a caching layer), Docker Compose is useful to manage the whole stack. Instead of running each service manually with individual **docker run** commands, Docker Compose allows you to define and orchestrate all services in one place.

Example: You have a web app using NGINX, a Redis cache, and a MySQL database. Compose lets you define these services and manage them together.

When Not to Use Docker Compose:

- **Production at Scale**: For complex production deployments, you would use tools like Kubernetes or Docker Swarm to orchestrate services across multiple servers. Docker Compose is more suited for development or small-scale production environments.
- **Single Container Applications**: If your application only consists of one service (e.g., a static site served by NGINX), using Compose might be overkill. In this case, a simple **docker run** command might suffice.

Multi-Co
ntainer

Steps to Build and Use Docker Compose for a Multi-Container Project

Let’s walk through an example where we create a simple project with:

- **Nginx** as a frontend server.
- **Node.js** for backend processing.
- **MongoDB** as the database.

Multi-Co
ntainer

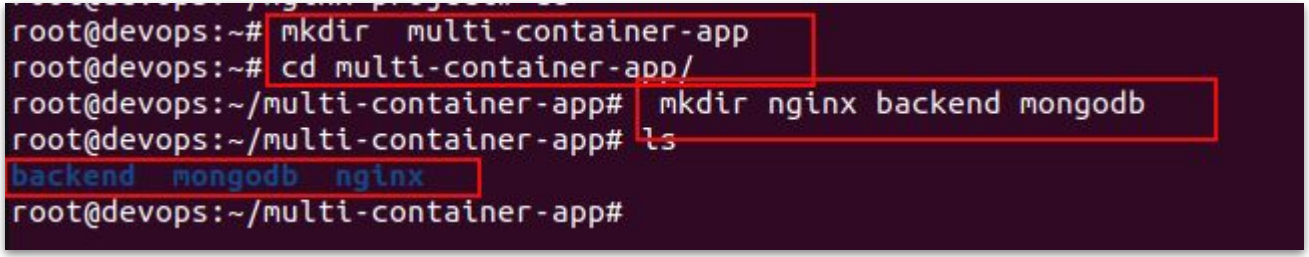
Steps to Build and Use Docker Compose for a Multi-Container Project

Step 1: Set Up the Project Directory

Create a directory for your project:

```
mkdir multi-container-app
cd multi-container-app
```

1. Inside this directory, you'll have subdirectories for each service (e.g., `nginx`, `backend`, and `mongodb`).



Multi-Co
ntainer

Steps to Build and Use Docker Compose for a Multi-Container Project

Step 2: Create a Backend Service (Node.js)

Create a `backend` directory for the Node.js app:

```
mkdir backend
```

1. `cd backend`
2. Create a `server.js` file in the `backend` directory:

```
  GNU nano 6.2                                server.js
const express =require('express');
const app = express();
const PORT = process.env.PORT || 5000;

app.get('/', (req,res) => {
  res.send('Hello from the  Node.js backend!');
});

app.listen(PORT,() => {
  console.log('server running  port ${PORT}');
});
```

Multi-Co
ntainer

Steps to Build and Use Docker Compose for a Multi-Container Project

Step 2: Create a Backend Service (Node.js)

Create a `backend` directory for the Node.js app:
`mkdir backend`

3. Create a `Dockerfile` for the Node.js service in the `backend` directory:

```
#base  image
FROM node:16

#set the  working directory
WORKDIR /app

#copy  package.json   and  install  dependencies
COPY package*.json ./

RUN npm  install

#copy the rest  of the  application  code
COPY . .

#Expose  port  5000
EXPOSE 5000

#start   the  application
 CMD ["node", "server.js"]
```

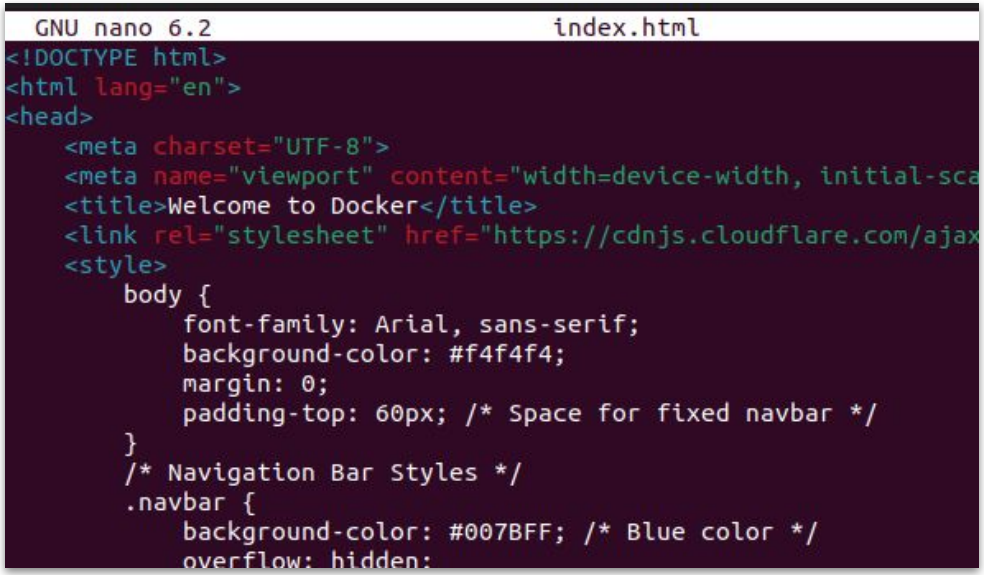
Steps to Build and Use Docker Compose for a Multi-Container Project

Multi-Co
ntainer

Step 3: Create a Frontend Service (Nginx)

In the root directory of your project (`multi-container-app`), create an `nginx` directory:
`mkdir nginx cd nginx`

- Create a simple `index.html` file in the `nginx` directory:



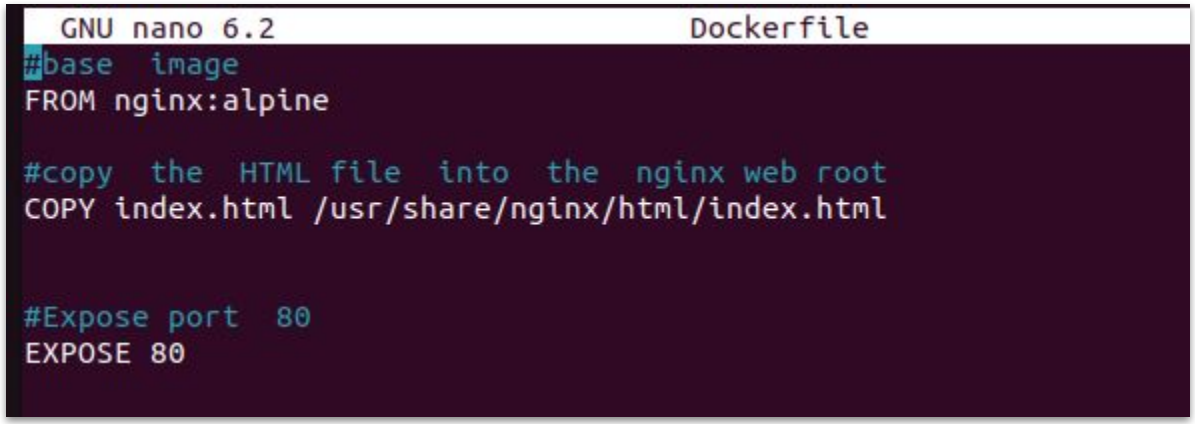
Multi-Container

Steps to Build and Use Docker Compose for a Multi-Container Project

Step 3: Create a Frontend Service (Nginx)

In the root directory of your project (`multi-container-app`), create an `nginx` directory:
`mkdir nginx cd nginx`

- Create a `Dockerfile` for Nginx:

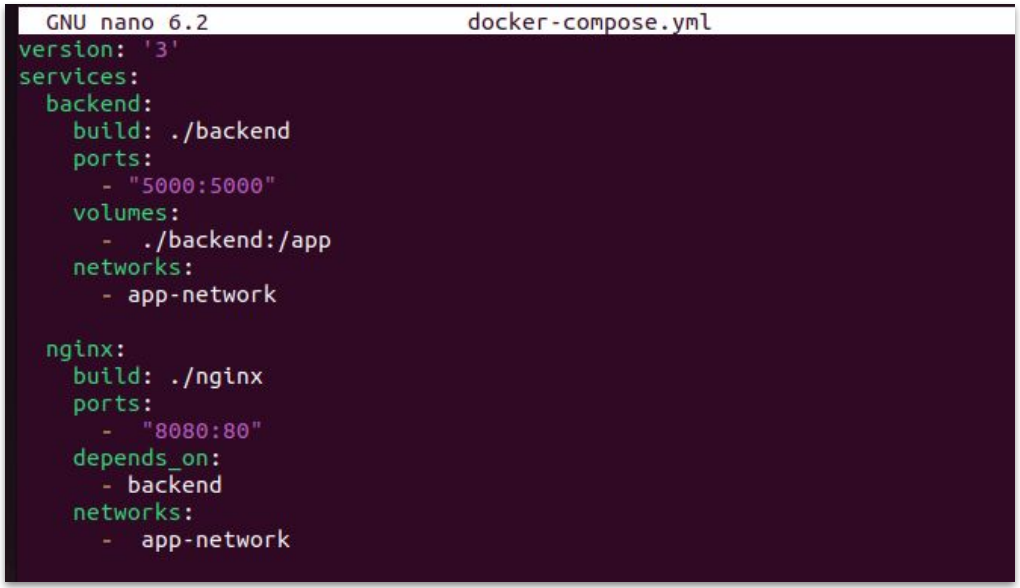


Multi-Container

Steps to Build and Use Docker Compose for a Multi-Container Project

Step 4: Define Services in Docker Compose

1. Create a `docker-compose.yml` file in the root directory (`multi-container-app`):
   `touch docker-compose.yml`



Multi-Container

Steps to Build and Use Docker Compose for a Multi-Container Project

Step 5: build the images

```
root@devops:~/multi-container-app# docker-compose build --no-cache
/usr/lib/python3/dist-packages/paramiko/transport.py:237: CryptographyDepr
nWarning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
mongodb uses an image, skipping
Building backend
[+] Building 586.0s (10/10) FINISHED                                docker:de
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 340B
 => [internal] load metadata for docker.io/library/node:14
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c
 => => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c59
 => => sha256:1d12470fa662a2a5cb50378dcdc8ea228c1735747db 7.51kB / 7.51kB
 => => sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f8451 50.45MB / 50.45MB
 => => sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff 7.86MB / 7.86MB
```

```
root@devops:~/multi-container-app# docker  images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
multi-container-app_nginx     latest    153dba3405ed   8 minutes ago    47MB
multi-container-app_backend   latest    f95e63c48da2   10 minutes ago   912MB
```

Multi-Container

Steps to Build and Use Docker Compose for a Multi-Container Project

Step 6: Run the container

```
root@devops:~/multi-container-app# docker  compose  up
WARN[0000] /root/multi-container-app/docker-compose.yml: the attribute `version`
 is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 7/9
 ⠸ mongodb [▒▒▒▒▒▒▒▒▒▒] 63.19MB / 274.6MB Pulling                          51.0s
   ✔ ff65ddf9395b Pull complete                                           36.3s
   ✔ 458feb307082 Pull complete                                           36.4s
   ✔ f59af5df8253 Pull complete                                           36.6s
   ✔ 145c7b6ccdb9 Pull complete                                           36.8s
   ✔ 35cc527541fc Pull complete                                           36.8s
   ✔ 076d157aff57 Pull complete                                           36.9s
   ⠸ 197a30480327 Downloading        30.8MB/242....                       45.6s
   ✔ 3736af050cc0 Download complete                                       13.8s
```