Docker: A Beginner-Friendly Guide

Simple concepts and hands-on exercises for beginners

What is Docker?

• A platform for running applications in containers

• Containers are lightweight packages containing:

• Your application code

• Libraries and dependencies

• Configuration files

• Think of containers as "shipping boxes" for software

Why Use Docker?

• Consistency: Works the same everywhere (no more "it works on my machine")

• Lightweight: Uses fewer resources than virtual machines

• Portable: Easily move between development, testing, and production

• Scalable: Great for growing applications

Real-World Docker Uses

• Web Applications: Running websites and web services

• Development Environments: Same setup for all developers

• Microservices: Running many small, independent services

• Continuous Integration: Testing code in consistent environments

Installing Docker

Windows & macOS:

1. Download Docker Desktop from Docker's website

2. Run the installer and follow instructions

Linux (Ubuntu):

sudo apt update

sudo apt install docker.io

sudo systemctl start docker

sudo systemctl enable docker

Check Installation:

docker --version

Expected output:

Docker version 24.0.5, build abc123

Your First Docker Command

Try running:

docker run hello-world

Expected output:

Hello from Docker!

This message shows that your installation appears to be working correctly.

...

This pulls the small "hello-world" image and runs it. It's a simple test to confirm Docker is working correctly.

Basic Docker Commands

# Download (pull) an image

docker pull nginx

# List all downloaded images

docker images

# Run a container

docker run -d --name mynginx -p 8080:80 nginx

# List running containers

docker ps

# List all containers (including stopped ones)

docker ps -a

# Stop a container

docker stop mynginx

# Remove a container

docker rm mynginx

Understanding Docker Run Options

docker run -d --name mynginx -p 8080:80 nginx

• -d: Run in background (detached mode)

• --name mynginx: Give the container a name

• -p 8080:80: Map port 8080 on your computer to port 80 in the container

• nginx: The image to use

Exercise: Run a Web Server

1. Pull the nginx image:

docker pull nginx

2. Run a container:

docker run -d --name mynginx -p 8080:80 nginx

3. Open your browser and visit: http://localhost:8080

4. You should see the Nginx welcome page!

5. Check that the container is running:

docker ps

6. Stop and remove when done:

docker stop mynginx

docker rm mynginx

Basic Docker Volumes

Volumes let you store data outside the container. This means:

• Data persists even if the container is removed

• Data can be shared between containers

• Changes to files happen in real-time

Two main ways to use volumes:

1. Bind mounts: Link a folder on your computer to a folder in the container

2. Named volumes: Docker-managed storage area

Bind Mounts: Sharing Folders

Create a folder on your computer with an index.html file:

<html>

<body>

<h1>My First Docker Website!</h1>

<p>This is running in a container.</p>

</body>

</html>

Run a container with this folder mounted:

docker run -d --name mywebsite -p 8080:80 -v $(pwd):/usr/share/nginx/html nginx

Visit http://localhost:8080 to see your site!

Edit the index.html file on your computer, refresh the page, and see the changes instantly!

Named Volumes: Persistent Storage

Create a named volume:

docker volume create mydata

List all volumes:

docker volume ls

Run a container with the named volume:

docker run -d --name mysql-db -v mydata:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=secret mysql:5.7

Even if you remove the container, the data remains:

docker stop mysql-db

docker rm mysql-db

# Create a new container using the same volume

docker run -d --name new-mysql-db -v mydata:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=secret mysql:5.7

# Your data is still there!

Exercise: Working with Volumes

1. Create a named volume:

docker volume create webdata

2. Run a container with the volume:

docker run -d --name webserver -p 8080:80 -v webdata:/usr/share/nginx/html nginx

3. Connect to the container and add a file:

docker exec -it webserver bash

echo "<h1>Hello from a volume!</h1>" > /usr/share/nginx/html/index.html

exit

4. Visit http://localhost:8080 to see your page

5. Stop and remove the container:

docker stop webserver

docker rm webserver

6. Create a new container using the same volume:

docker run -d --name webserver2 -p 8080:80 -v webdata:/usr/share/nginx/html nginx

7. Visit http://localhost:8080 again - your data is still there!

Connecting to Running Containers

Execute commands in a running container:

# Run a shell inside an Nginx container

docker exec -it mynginx bash

Expected output:

root@7f2569ed9ce0:/# ls

bin boot dev etc ...

root@7f2569ed9ce0:/# cd /usr/share/nginx/html

root@7f2569ed9ce0:/usr/share/nginx/html# ls

index.html

View application logs:

docker logs mynginx

Copy files to/from containers:

# Copy file FROM container TO host

docker cp mynginx:/etc/nginx/nginx.conf ./nginx.conf

# Copy file FROM host TO container

docker cp ./custom-config.conf mynginx:/etc/nginx/conf.d/

Environment Variables in Containers

Environment variables let you configure containers without changing their code.

Simple example with environment variables:

docker run -d --name mycontainer -e NAME=John -e GREETING="Hello there" nginx

Checking environment variables inside the container:

docker exec -it mycontainer bash

echo $NAME

echo $GREETING

Expected output:

John

Hello there

Environment Variables: Real Example

Let's run a MySQL database with environment variables:

docker run -d --name mydb \

-e MYSQL\_ROOT\_PASSWORD=rootpass \

-e MYSQL\_DATABASE=myapp \

-e MYSQL\_USER=appuser \

-e MYSQL\_PASSWORD=apppass \

mysql:5.7

Connect to the database:

docker exec -it mydb mysql -u appuser -papppass myapp

Inside MySQL:

CREATE TABLE users (id INT, name VARCHAR(50));

INSERT INTO users VALUES (1, 'Alice');

SELECT \* FROM users;

The database and user were created using environment variables!

Docker Networking Basics

List all networks:

docker network ls

Create a custom network:

docker network create mynetwork

Run containers in the network:

# Run first container

docker run -d --name container1 --network mynetwork nginx

# Run second container

docker run -d --name container2 --network mynetwork nginx

Containers can find each other by name:

# Connect to container2

docker exec -it container2 bash

# Install ping

apt-get update && apt-get install -y iputils-ping

# Ping container1 by name

ping container1

Expected output:

PING container1 (172.18.0.2) 56(84) bytes of data.

64 bytes from container1.mynetwork (172.18.0.2): icmp\_seq=1 ttl=64 time=0.075 ms

64 bytes from container1.mynetwork (172.18.0.2): icmp\_seq=2 ttl=64 time=0.055 ms

Simple Node.js Application in Docker

1. Create a folder for your project:

mkdir node-docker

cd node-docker

2. Create a simple app.js file:

const http = require('http');

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello from Node.js in Docker!\n');

});

server.listen(3000, '0.0.0.0', () => {

console.log('Server running at http://0.0.0.0:3000/');

});

3. Run the application in Docker:

docker run -d --name nodeapp -p 3000:3000 -v $(pwd):/app -w /app node:14 node app.js

4. Visit http://localhost:3000 in your browser

Customizing the Node.js App with Environment Variables

Modify your app.js:

const http = require('http');

// Use environment variables with defaults

const port = process.env.PORT || 3000;

const message = process.env.MESSAGE || 'Hello from Node.js in Docker!';

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end(message + '\n');

});

server.listen(port, '0.0.0.0', () => {

console.log(Server running at http://0.0.0.0:${port}/);

});

Run with environment variables:

docker run -d --name nodeapp -p 3000:3000 \

-e PORT=3000 \

-e MESSAGE="Welcome to my custom Docker app!" \

-v $(pwd):/app -w /app \

node:14 node app.js

Building Your Own Image with Dockerfile

Create a file named Dockerfile:

FROM node:14

WORKDIR /app

COPY app.js .

EXPOSE 3000

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

Build and run:

docker build -t mynodeapp .

docker run -d -p 3000:3000 mynodeapp

This creates a reusable image with your application.

Practical Exercises for Beginners

1. First Steps:

• Run docker run hello-world

• Run docker run -it ubuntu and try some Linux commands

• Run docker run -d -p 8080:80 nginx and visit the webpage

2. Volumes Exercise:

• Create a volume: docker volume create myvolume

• Run a container with the volume: docker run -d -v myvolume:/data --name vol-test ubuntu sleep 1000

• Connect to the container: docker exec -it vol-test bash

• Create a file in the volume: echo "test" > /data/testfile.txt

• Exit and remove the container: docker stop vol-test && docker rm vol-test

• Run a new container with the same volume and verify the file exists

3. Simple Web Server:

• Create an HTML file with your name and a message

• Run Nginx with your HTML file mounted

• Visit your custom web page

Docker Commands Cheat Sheet

Images:

• docker pull image - Download an image

• docker images - List all images

• docker rmi image - Remove an image

• docker build -t name . - Build an image from Dockerfile

Containers:

• docker run image - Run a container

• docker ps - List running containers

• docker ps -a - List all containers

• docker stop container - Stop a container

• docker rm container - Remove a container

• docker exec -it container command - Run command in container

Volumes:

• docker volume create name - Create a volume

• docker volume ls - List volumes

• docker volume rm name - Remove a volume

Next Steps in Your Docker Journey

• Learn to write better Dockerfiles

• Run multi-container applications

• Study container orchestration with Kubernetes

• Practice by containerizing your own projects

Additional Resources

• Official Docker Documentation: https://docs.docker.com/

• Docker Hub: https://hub.docker.com/

• Play with Docker: https://labs.play-with-docker.com/ (practice in browser)