ChatGPT

Docker Mastery Notes

Author: Srinath K – DevOps & Container Specialist

What is Docker?

Docker is a containerization platform that packages applications and their dependencies together to run reliably across environments.

Why use Docker?

- Isolation: Runs apps in containers (like mini virtual machines)
- Portability: Works the same on any OS
- Consistency: No more "works on my machine"
- Lightweight: Shares host OS kernel

Basic Docker Concepts

Concept	Description
Image	Read-only blueprint to create containers
Container	Running instance of an image
Dockerfile	Script that defines how to build a Docker image
Docker Hub	Cloud registry to host Docker images
Tag	Label to identify versions of images
Port Mapping	Connect container ports to host ports

Essential Docker Commands

```
docker version            # Check Docker version
docker images             # List downloaded images
docker ps -a              # List all containers
docker pull nginx         # Download nginx image
docker run -d -p 8080:80 nginx   # Run nginx in detached mode
docker stop <id>          # Stop container
docker rm <id>            # Remove container
docker exec -it <id> bash # Enter container shell
docker logs <id>          # View container logs
```

🏠 Dockerfile Explained (Python App Example)

```
FROM python:3.9-slim           # Base image with Python
WORKDIR /app                  # Set working directory
COPY requirements.txt .      # Copy dependency list
RUN pip install -r requirements.txt  # Install dependencies
COPY . .                     # Copy project files
CMD ["python", "app.py"]     # Start app when container runs
```

Analogy: Like baking a cake layer by layer:

- FROM: base flavor
- WORKDIR: kitchen setup
- COPY: add ingredients
- RUN: bake the layers
- CMD: serve the cake

🌎 What Happens After Dockerfile is Ready?

1. **Build Image**:

```
docker build -t my-python-app .
```

1. **Run Container**:

```
docker run -d -p 8080:80 my-python-app
```

1. **Verify**:

```
docker ps
```

1. **Push to Docker Hub (optional)**:

```
docker tag my-python-app myusername/my-python-app
docker push myusername/my-python-app
```

What is `requirements.txt`?

Text file listing Python packages your app depends on.

Example:

```
flask==2.3.2
requests==2.31.0
```

Generate it:

```
pip freeze > requirements.txt
```

⚖ Detached vs Attached Mode

Mode	Description	Command Example
Attached	Runs in foreground, tied to terminal	`docker run -it ubuntu bash`
Detached	Runs in background, frees terminal	`docker run -d -p 8080:80 nginx`

Analogy:

- Attached = Live Theater
- Detached = TV Recording

⛏ Port Mapping Explained

```
docker run -p 2025:80 nginx
```

Meaning: Map port 80 in the container to port 2025 on host

Analogy:

- Container = Room with intercom (port 80)
- Host port = Extension number on the street (2025)

Docker Tag Command

```
docker tag my-python-app myusername/my-python-app
```

Purpose: Give your image a new name to prepare for pushing to Docker Hub

Analogy: Like labeling a box with your name and address before shipping

---

What Happens Inside a Container?

- Uses host OS kernel
- Runs your app and dependencies in isolation
- Acts like a mini-computer

Even if Python is already on your PC, Docker makes sure:

- You always get the same version
- You don’t need to install anything on the server
- No dependency conflicts

Docker + MLOps: Real-World Practice Example

Let’s say you’re a data scientist working with a dataset from Kaggle and want to containerize your training script:

Step 1: Sample Project Structure

```
mlops-project/
├── data/
│   └── sample.csv
├── app.py
├── requirements.txt
└── Dockerfile
```

Step 2: app.py (Minimal Example)

```python
import pandas as pd

df = pd.read_csv("data/sample.csv")
print("Rows:", len(df))
```

Step 3: requirements.txt

```
pandas==2.2.2
```

Step 4: Dockerfile for MLOps

```
FROM python:3.10-slim
WORKDIR /mlapp
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

Step 5: Build and Push to Docker Hub

```
# Build Docker image
docker build -t srinathk/mlops-dataset-app .

# Run locally to verify
docker run srinathk/mlops-dataset-app

# Push to Docker Hub
docker push srinathk/mlops-dataset-app
```

Now your data pipeline is containerized — and can run anywhere: cloud, CI/CD, or orchestration platforms.

Summary: Docker Image Lifecycle

1. **Write Dockerfile**
2. **Build Image**
3. **Run Container**
4. **Test & Debug**
5. **Push to Registry**
6. **Deploy Anywhere**

Maintained and written by Srinath K – DevOps Specialist