Complete Docker Command and Example Manual

Prepared by Abdo Wael

October 15, 2025

Contents

1. Introduction

Docker is an open-source platform that automates the deployment of applications inside software containers. Containers provide isolation, portability, and consistency across different environments.

1.1. Why Docker?

- Works on any platform (Linux, Windows, macOS).
- Reduces dependency conflicts.
- Fast deployment and scalability.
- Integrates with CI/CD and cloud services.

2. Docker Architecture Overview

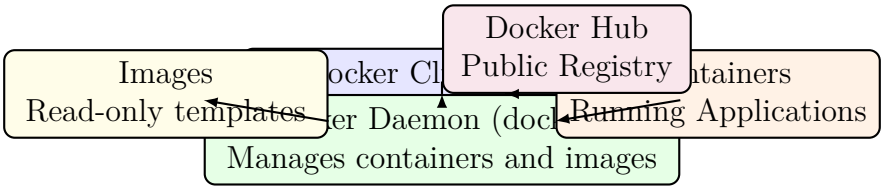

Figure 1: Docker Architecture

3. Docker Lifecycle

Figure 2: Docker Container Lifecycle

4. Dockerfile Explained

A Dockerfile is a script containing instructions to build an image.

4.1. Example Dockerfile

```
# Start from a base image
FROM python:3.12

# Set working directory
WORKDIR /app

# Copy files to container
COPY . .

# Install dependencies
```

```
RUN pip install -r requirements.txt

# Expose a port
EXPOSE 5000

# Command to run application
CMD ["python", "app.py"]
```

4.2. Dockerfile Commands Summary

- **FROM** – Specifies the base image.
- **WORKDIR** – Sets the working directory.
- **COPY/ADD** – Copies files into image.
- **RUN** – Executes commands during build.
- **CMD** – Sets the command to run.
- **EXPOSE** – Opens network ports.
- **ENV** – Defines environment variables.

5. Image Management

5.1. Build and Tag Image

```
docker build -t myapp:1.0 .
```

5.2. View Image History

```
docker history myapp:1.0
```

5.3. Save and Load Image

```
docker save -o myapp.tar myapp:1.0
docker load -i myapp.tar
```

5.4. Push and Pull from Registry

```
docker login
docker tag myapp:1.0 username/myapp:1.0
docker push username/myapp:1.0
docker pull username/myapp:1.0
```

6. Advanced Container Commands

6.1. Monitoring and Logging

```
docker logs myapp          # Show logs
docker stats               # Real-time resource usage
docker top myapp           # Running processes inside container
```

6.2. Commit and Export Container

```
docker commit mycontainer myimage:v2
docker export mycontainer > container_backup.tar
docker import container_backup.tar newimage:latest
```

6.3. Attach and Inspect

```
docker attach myapp
docker inspect myapp
```

7. Networking Visualization

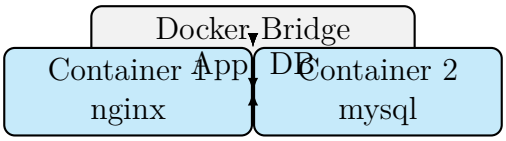

Figure 3: Container Communication via Bridge Network

8. Docker Security Best Practices

- Use minimal base images like `alpine`.
- Run applications as non-root users.
- Regularly update images.
- Use Docker Bench for Security.
- Restrict container privileges with `-cap-drop`.

9. Troubleshooting Common Issues

9.1. Container Fails to Start

```
docker logs <container_id>
docker inspect <container_id>
```

9.2. Remove All Stopped Containers and Dangling Images

```
docker container prune
docker image prune -a
```

9.3. Fix Permission Issues

```
sudo chmod 666 /var/run/docker.sock
```

10. Example: Deploy Flask Web App

10.1. Project Structure

```
app/
        app.py
        requirements.txt
        Dockerfile
```

10.2. Dockerfile

```
FROM python:3.12
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

10.3. Build and Run

```
docker build -t flaskapp:v1 .
docker run -d -p 5000:5000 flaskapp:v1
```

11. Docker in DevOps Pipeline

- Docker images are used for CI/CD testing.
- Containers ensure consistent environments.
- Integrated into Jenkins, GitHub Actions, and Kubernetes.

12. Conclusion

Docker revolutionizes software deployment by ensuring consistency, speed, and reliability. With these commands and practices, developers can efficiently manage containerized applications in modern DevOps environments.

"Build, Ship, and Run — Anywhere with Docker."