Docker Compose for Multi-Container Applications

Step 1: Install Docker Compose

Docker Compose is usually bundled with Docker Desktop. To verify:

```
docker-compose --version
```

Or for Linux:

```
sudo apt install docker-compose
```

Step 2: Create a Project Directory

```
mkdir my-multicontainer-app && cd my-multicontainer-app
```

Add your app.py, requirements.txt, and other files here.

Step 3: Create a Dockerfile

Example: Dockerfile

```
FROM python:3.9
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

Step 4: Create a docker-compose.yml File

Example: Python App + Redis

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/app
    depends_on:
      - redis
  redis:
    image: redis:alpine
```

Step 5: Run the Multi-Container App

docker-compose up -d

- -d runs it in detached mode.

Step 6: Check Running Containers

docker ps

You should see both the web and redis containers.

Step 7: Stop and Remove Containers

docker-compose down

Benefits of Docker Compose

- Simplifies multi-container setup
- Version control with docker-compose.yml
- Easier scaling with docker-compose up --scale web=3

Docker Security Best Practices

1. Use Official and Minimal Base Images

Use images from trusted sources (like alpine, python:slim)

Dockerfile:

FROM python:3.9-slim

2. Avoid Running as Root in Containers

Create a non-root user in the Dockerfile:

Dockerfile:

RUN useradd -m appuser

USER appuser

3. Limit Container Capabilities

Run containers with least privilege:

docker run --cap-drop=ALL --cap-add=NET_BIND_SERVICE myapp

4. Use .dockerignore to Limit Build Context

Example .dockerignore:

.git

node_modules

*.log

This prevents sensitive or unnecessary files from being added to the image.

5. Scan Images for Vulnerabilities

Use Docker's built-in scanning:

```
docker scan <image-name>
```

6. Keep Docker Up to Date

Regularly update Docker Engine and Docker Compose:

```
sudo apt update && sudo apt upgrade docker-ce
```

7. Use Read-Only Filesystems

Run containers in read-only mode:

```
docker run --read-only myapp
```

8. Restrict Networking

Use custom bridge networks and limit exposed ports:

```
docker network create secure-net
docker run --network secure-net ...
```

9. Use Secrets Management (Avoid ENV for Secrets)

Use Docker secrets (in Swarm) or volume-mounted secrets.

Summary: Docker Security Checklist

Best Practice	**Command or Tip**
Use trusted base images	FROM python:3.9-slim
Don’t run as root	USER appuser in Dockerfile
Limit container capabilities	--cap-drop=ALL --cap-add=...
Ignore sensitive files	Use .dockerignore
Scan for vulnerabilities	docker scan
Run as read-only	--read-only
Use secure networks	docker network create
Avoid exposing all ports	Expose only needed ports