Hands-on Exercise 5: Dockerfile Mastery & Image Creation

Overview

These exercises focus on creating efficient Dockerfiles and analyzing existing ones for optimization and security improvements.

Prerequisites

- Docker Engine installed
- Text editor
- Terminal or command prompt

Exercise 5.1: Creating Efficient Dockerfiles

Objective

Create and optimize a multi-stage Dockerfile for a Python application following best practices.

Task 1: Setup Basic Python Application

```
# Create project directory
mkdir -p python-app
cd python-app

# Create a simple Flask application
cat > app.py << 'EOF'
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from containerized Python application!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
EOF

# Create requirements file
cat > requirements.txt << 'EOF'
flask==2.0.1
gunicorn==20.1.0
EOF
```

Task 2: Create an Initial Single-Stage Dockerfile

```
cat > Dockerfile.single << 'EOF'
FROM python:3.9

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000
CMD ["python", "app.py"]
EOF

# Build the image
docker build -f Dockerfile.single -t python-app:single .

# Check image size
docker images python-app:single
```

Task 3: Create a Multi-Stage Dockerfile

```
cat > Dockerfile.multi << 'EOF'
# Build stage
FROM python:3.9 AS builder

WORKDIR /build

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Runtime stage
FROM python:3.9-slim

WORKDIR /app

# Copy dependencies from builder
COPY --from=builder /usr/local/lib/python3.9/site-packages
/usr/local/lib/python3.9/site-packages
COPY --from=builder /usr/local/bin /usr/local/bin

# Copy application code
COPY . .

EXPOSE 5000
CMD ["python", "app.py"]
EOF

# Build the image
docker build -f Dockerfile.multi -t python-app:multi .

# Check image size
docker images python-app:multi
```

Task 4: Create an Optimized Dockerfile

```
cat > Dockerfile.optimized << 'EOF'
# Build stage
FROM python:3.9-slim AS builder

WORKDIR /build

COPY requirements.txt .
RUN pip install --no-cache-dir --user -r requirements.txt

# Runtime stage
FROM python:3.9-alpine

# Add metadata
LABEL org.opencontainers.image.authors="workshop@example.com"
LABEL org.opencontainers.image.version="1.0.0"
LABEL org.opencontainers.image.description="Optimized Python application"

WORKDIR /app

# Copy dependencies from builder
COPY --from=builder /root/.local/lib/python3.9/site-packages
/usr/local/lib/python3.9/site-packages

# Copy application code
COPY . .

# Create non-root user
RUN addgroup -S appgroup && \
    adduser -S appuser -G appgroup -h /app && \
    chown -R appuser:appgroup /app

# Switch to non-root user
USER appuser

# Add health check
HEALTHCHECK --interval=30s --timeout=3s CMD wget --no-verbose --tries=1 --
spider http://localhost:5000/ || exit 1

EXPOSE 5000
CMD ["python", "app.py"]
EOF

# Build the image
docker build -f Dockerfile.optimized -t python-app:optimized .

# Check image size
docker images python-app:optimized
```

Task 5: Compare Image Sizes and Security

```
# Compare all three images
docker images | grep python-app

# Check user privileges
docker run --rm python-app:single id
docker run --rm python-app:optimized id

# Check layer count
docker history python-app:single | wc -l
docker history python-app:optimized | wc -l
```

Exercise 5.2: Dockerfile Analysis and Improvement

Objective

Analyze sample Dockerfiles for common issues and improve them using best practices.

Task 1: Review Sample Dockerfiles

```
# Create directory for analysis
mkdir -p dockerfile-analysis
cd dockerfile-analysis

# Sample problematic Dockerfile
cat > Dockerfile.problematic << 'EOF'
FROM python:latest

RUN apt-get update
RUN apt-get install -y vim git curl
RUN pip install flask==2.0.1
RUN pip install gunicorn==20.1.0
RUN pip install requests==2.26.0

WORKDIR /app

COPY . /app/

ENV SECRET_KEY="development_key_123456"
ENV DEBUG=True

EXPOSE 5000

CMD ["python", "app.py"]
EOF

# Create sample app file
echo 'print("Hello from Python container")' > app.py
```

Task 2: Identify Issues

Review the Dockerfile and identify issues related to:

- Security
- Efficiency
- Best practices

Issues to identify include:

- Using `latest` tag
- Multiple RUN commands creating unnecessary layers
- Running as root
- Installing unnecessary packages
- Hardcoded secrets in ENV
- No `.dockerignore` file
- Copying all files without filtering

Task 3: Create Improved Dockerfile

```
# Create .dockerignore
cat > .dockerignore << 'EOF'
.git
.gitignore
.dockerignore
Dockerfile*
__pycache__
*.pyc
*.pyo
*.pyd
.env
*.md
EOF

# Create improved Dockerfile
cat > Dockerfile.improved << 'EOF'
FROM python:3.9-slim

# Add metadata
LABEL org.opencontainers.image.authors="workshop@example.com"
LABEL org.opencontainers.image.version="1.0.0"

WORKDIR /app

# Combine RUN commands to reduce layers
RUN apt-get update && \
    apt-get install -y --no-install-recommends curl && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* && \
    pip install --no-cache-dir flask==2.0.1 gunicorn==20.1.0 requests==2.26.0

# Copy only necessary files
COPY app.py .

# Create non-root user
RUN useradd -m appuser && \
    chown -R appuser:appuser /app

USER appuser

# Secrets should be passed at runtime, not in Dockerfile
# ENV SECRET_KEY will be provided at runtime

EXPOSE 5000

HEALTHCHECK --interval=30s --timeout=3s CMD curl -f http://localhost:5000/ ||
exit 1

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
EOF
```

Task 4: Build and Compare

```bash
# Build the original image
docker build -f Dockerfile.problematic -t python-app:problematic .

# Build the improved image
docker build -f Dockerfile.improved -t python-app:improved .

# Compare the images
docker images | grep python-app
docker history python-app:problematic | wc -l
docker history python-app:improved | wc -l
```

Task 5: Security Verification

```bash
# Check if running as root
docker run --rm python-app:problematic id
docker run --rm python-app:improved id

# Check if unnecessary software is installed
docker run --rm python-app:problematic which vim
docker run --rm python-app:improved which vim
```

Expected Results

1. Optimized image should be significantly smaller than the original
2. Optimized image should run as a non-root user
3. Optimized image should have fewer layers
4. Improved security through proper practices

Cleanup

```bash
# Remove all created images
docker rmi python-app:single python-app:multi python-app:optimized
docker rmi python-app:problematic python-app:improved
```