**Dockerfile Instructions Workshop (45 Minutes)**

**Understanding RUN, CMD, and ENTRYPOINT Differences**

**Part 1: Basic Dockerfile Creation (10 minutes)**

**Exercise 1.1: Create Basic Web Server Image**

# Create working directory

mkdir dockerfile-workshop

cd dockerfile-workshop

**Create Dockerfile using vi:**

vi Dockerfile

**Dockerfile content:**

FROM debian:stable

LABEL authors="your-name"

RUN apt-get update && apt-get install -y --force-yes apache2

EXPOSE 80

**Build and inspect the image:**

# Build the image

docker build -t mydeb .

# List images

docker images

# Get image ID and inspect

docker inspect <image-id>

# Run the container

docker run -d -p 8080:80 mydeb

**Part 2: Understanding RUN vs CMD vs ENTRYPOINT (25 minutes)**

**Exercise 2.1: Create the Log Event Script**

**Create the test directory and script:**

mkdir test

cd test

**Create date.sh script:**

vi date.sh

**Script content:**

#!/bin/sh

echo `date` $@ >> log.txt;

cat log.txt;

**Make it executable:**

chmod +x date.sh

**Exercise 2.2: RUN Command - Build Time Execution**

**Create Dockerfile with RUN:**

vi Dockerfile

**Dockerfile content:**

FROM alpine

ADD date.sh /

RUN chmod +x /date.sh

RUN ["/date.sh", "image created"]

**Build and test:**

docker build -t myimage .

docker run myimage cat log.txt

docker run myimage cat log.txt # Run again

**Observation:** The timestamp remains the same because RUN executes at **build time**, not runtime.

**Exercise 2.3: Adding CMD - Runtime Execution**

**Modify Dockerfile to include CMD:**

FROM alpine

ADD date.sh /

RUN chmod +x /date.sh

RUN ["/date.sh", "image created"]

CMD ["/date.sh", "container started"]

**Build and test:**

docker build -t myimage1 .

docker run myimage1

docker run myimage1 # Run multiple times

**Observation:**

* "image created" timestamp stays the same (RUN at build time)
* "container started" timestamp updates each run (CMD at runtime)

**Exercise 2.4: CMD Can Be Overridden**

**Test CMD override:**

# This ignores CMD and runs cat instead

docker run myimage1 cat log.txt

**Observation:** CMD is ignored when you provide arguments to docker run.

**Exercise 2.5: Multiple CMD Instructions**

**Create Dockerfile with multiple CMDs:**

FROM alpine

ADD date.sh /

RUN chmod +x /date.sh

RUN ["/date.sh", "image created"]

CMD ["/date.sh", "container started"]

CMD ["/date.sh", "container running"]

**Build and test:**

docker build -t myimage2 .

docker run myimage2

**Observation:** Only the last CMD executes. Previous CMDs are ignored.

**Exercise 2.6: ENTRYPOINT - Always Executes**

**Create Dockerfile with ENTRYPOINT:**

FROM alpine

ADD date.sh /

RUN chmod +x /date.sh

RUN ["/date.sh", "image created"]

ENTRYPOINT ["/date.sh"]

**Build and test:**

docker build -t myimage3 .

docker run myimage3

docker run myimage3 "welcome to container"

**Observation:** ENTRYPOINT always executes and receives additional arguments.

**Exercise 2.7: CMD + ENTRYPOINT Combination (Assignment)**

**Create Dockerfile with both:**

FROM alpine

ADD date.sh /

RUN chmod +x /date.sh

RUN ["/date.sh", "image created"]

ENTRYPOINT ["/date.sh"]

CMD ["default message"]

**Build and test:**

docker build -t myimage4 .

# Test with default CMD

docker run myimage4

# Test with override (CMD gets replaced)

docker run myimage4 "custom message"

# Test with multiple arguments

docker run myimage4 "arg1" "arg2" "arg3"

**Assignment Observation:**

* ENTRYPOINT always runs the script
* CMD provides default arguments to ENTRYPOINT
* Arguments to docker run replace CMD, not ENTRYPOINT

**Part 3: Docker Registry Operations (10 minutes)**

**Exercise 3.1: Tag and Push Images**

**Login to Docker Hub:**

docker login

# Enter your Docker Hub credentials

**Tag your image:**

# Replace 'yourusername' with your Docker Hub username

docker image tag myimage3 yourusername/myimage:latest

docker image tag myimage3 yourusername/myimage:v1.0

**Push to Docker Hub:**

docker image push yourusername/myimage:latest

docker image push yourusername/myimage:v1.0

**Verify the push:**

# List your local images

docker images | grep yourusername

# Pull from a different location to test

docker rmi yourusername/myimage:latest

docker pull yourusername/myimage:latest

**Cleanup Commands:**

# Remove all created images

docker rmi mydeb myimage myimage1 myimage2 myimage3 myimage4

# Remove all stopped containers

docker container prune

# Clean up test directory

cd .. && rm -rf test