**Docker Fundamentals Workshop (60Minutes)**

**Prerequisites**

* Docker installed on your system
* Basic command line knowledge
* Text editor of choice

**Part 1: Essential Docker Commands (15 minutes)**

**1.1 Container Lifecycle Commands**

**Pull and Run Your First Container**

# Pull an image from Docker Hub

docker pull nginx:latest

# Run a container

docker run nginx

# Run container in detached mode (background)

docker run -d nginx

# Run with custom name and port mapping

docker run -d --name my-web-server -p 8080:80 nginx

**Managing Running Containers**

# List running containers

docker ps

# List all containers (including stopped)

docker ps -a

# Stop a container

docker stop my-web-server

# Start a stopped container

docker start my-web-server

# Restart a container

docker restart my-web-server

# Remove a container

docker rm my-web-server

# Force remove a running container

docker rm -f my-web-server

**Interactive Containers**

# Run interactive container with terminal

docker run -it ubuntu:20.04 /bin/bash

# Execute command in running container

docker exec -it container\_name /bin/bash

# Exit container without stopping it

# Press Ctrl+P, then Ctrl+Q

**Container Information**

# View container logs

docker logs container\_name

# Follow logs in real-time

docker logs -f container\_name

# Inspect container details

docker inspect container\_name

# View container resource usage

docker stats container\_name

**1.2 Image Management**

# List local images

docker images

# Remove an image

docker rmi nginx:latest

# Remove unused images

docker image prune

**Quick Exercise (3 minutes):**

1. Pull the hello-world image
2. Run it and observe the output
3. List all containers and remove the stopped one

**Part 2: Docker Networking Basics (15 minutes)**

**2.1 Understanding Docker Networks**

**Default Networks**

# List all networks

docker network ls

# Inspect default bridge network

docker network inspect bridge

**Creating Custom Networks**

# Create a custom bridge network

docker network create my-network

# Create network with custom subnet

docker network create --driver bridge --subnet=192.168.1.0/24 custom-network

# Remove a network

docker network rm my-network

**2.2 Connecting Containers to Networks**

**Basic Network Connection**

# Run container on custom network

docker run -d --name web-server --network my-network nginx

# Connect existing container to network

docker network connect my-network existing-container

# Disconnect container from network

docker network disconnect my-network existing-container

**Container Communication Example**

# Create a custom network

docker network create app-network

# Run a database container

docker run -d --name db --network app-network \

-e MYSQL\_ROOT\_PASSWORD=secret \

-e MYSQL\_DATABASE=myapp \

mysql

# Run an application container that can communicate with db

docker run -d --name app --network app-network \

-p 3000:3000 \

-e DB\_HOST=db \

-e DB\_PASSWORD=secret \

node:16-alpine sleep 3600

**2.3 Port Mapping and Exposure**

# Map single port

docker run -d -p 8080:80 nginx

# Map multiple ports

docker run -d -p 8080:80 -p 8443:443 nginx

# Map to specific interface

docker run -d -p 127.0.0.1:8080:80 nginx

# Expose port without mapping (internal only)

docker run -d --expose 80 nginx

**Practical Exercise (5 minutes):**

1. Create a network called workshop-net
2. Run two Alpine containers on this network
3. Test communication between containers using ping

# Solution:

docker network create workshop-net

docker run -dit --name alpine1 --network workshop-net alpine

docker run -dit --name alpine2 --network workshop-net alpine

docker exec alpine1 ping alpine2

**Part 3: Docker Volumes Basics (15 minutes)**

**3.1 Understanding Volume Types**

**Named Volumes**

# Create a named volume

docker volume create my-data

# List volumes

docker volume ls

# Inspect volume details

docker volume inspect my-data

# Remove volume

docker volume rm my-data

# Remove unused volumes

docker volume prune

**Using Named Volumes with Containers**

# Mount named volume to container

docker run -d --name web-server \

-v my-data:/usr/share/nginx/html \

-p 8080:80 \

nginx

# Multiple volume mounts

docker run -d --name app \

-v app-data:/app/data \

-v app-logs:/app/logs \

ubuntu:20.04

**3.2 Bind Mounts**

# Mount host directory to container (absolute path required)

docker run -d --name web-server \

-v /host/path:/container/path \

-p 8080:80 \

nginx

# Mount current directory

docker run -d --name web-server \

-v $(pwd):/usr/share/nginx/html \

-p 8080:80 \

nginx

# Read-only bind mount

docker run -d --name web-server \

-v $(pwd):/usr/share/nginx/html:ro \

-p 8080:80 \

nginx

**3.3 Practical Volume Examples**

**Database with Persistent Storage**

# Create volume for MySQL data

docker volume create mysql-data

# Run MySQL with persistent storage

docker run -d --name mysql-db \

-e MYSQL\_ROOT\_PASSWORD=secret \

-e MYSQL\_DATABASE=myapp \

-v mysql-data:/var/lib/mysql \

-p 3306:3306 \

mysql:8.0

**Development Environment**

# Mount source code for live development

docker run -dit --name dev-env \

-v $(pwd):/workspace \

-w /workspace \

-p 3000:3000 \

node:16-alpine

# Execute commands in development container

docker exec dev-env npm install

docker exec dev-env npm start

**3.4 Volume Backup and Restore**

# Backup volume data

docker run --rm \

-v mysql-data:/data \

-v $(pwd):/backup \

alpine tar czf /backup/backup.tar.gz -C /data .

# Restore volume data

docker run --rm \

-v mysql-data:/data \

-v $(pwd):/backup \

alpine tar xzf /backup/backup.tar.gz -C /data

**Final Exercise :** Create a complete web application stack:

# 1. Create network and volumes

docker network create webapp-net

docker volume create db-data

docker volume create web-content

# 2. Run database

docker run -d --name database \

--network webapp-net \

-v db-data:/var/lib/mysql \

-e MYSQL\_ROOT\_PASSWORD=secret \

-e MYSQL\_DATABASE=webapp \

mysql:8.0

# 3. Create simple HTML file

echo "<h1>Hello Docker Workshop!</h1>" > index.html

# 4. Run web server

docker run -d --name webserver \

--network webapp-net \

-v $(pwd):/usr/share/nginx/html \

-p 8080:80 \

nginx

# 5. Test the setup

curl http://localhost:8080

**Cleanup Commands**

# Stop all containers

docker stop $(docker ps -q)

# Remove all containers

docker rm $(docker ps -aq)

# Remove all images

docker rmi $(docker images -q)

# Remove all volumes

docker volume rm $(docker volume ls -q)

# Remove all networks (except defaults)

docker network rm $(docker network ls -q --filter type=custom)

# Nuclear option - remove everything

docker system prune -a --volumes