Cloud and Deployment

Prepared By: Aatiz Ghimire, for Herald Center for AI.

Summer, 2025

Learning Objectives.

- Understand the fundamentals of containerization and Docker architecture.
- Build, run, and manage containers using Docker CLI and Dockerfiles.
- Use Docker Compose to define and deploy multi-container applications.
- Manage data persistence with Docker volumes and networks.

1 Introduction

Docker is a platform that uses containerization to package applications and their dependencies into portable, lightweight containers. Containers ensure consistent behavior across environments, simplifying development and deployment. This workshop introduces Docker fundamentals using the Play with Docker platform, a browser-based environment for learning Docker.

Key Concepts:

- **Images**: Read-only templates for creating containers.
- **Containers**: Running instances of images, isolated from the host.
- **Docker Hub**: A public registry for sharing Docker images.

Play with Docker Note: The environment provides a pre-configured Docker host. You'll work in a browser terminal, and sessions last up to 4 hours.

2 Prerequisites

- Access to Play with Docker (requires a Docker Hub account).
- Basic familiarity with the command line.
- A modern web browser (e.g., Chrome, Firefox).

3 Workshop Agenda

1. Introduction to Docker
2. Basic Docker Commands
3. Running Your First Container
4. Managing Docker Images
5. Docker Networking Basics
6. Docker Volumes and Data Persistence
7. Writing a Dockerfile
8. Hands-On Exercise: Building a Custom Image

4 Basic Docker Commands

The Docker Command Line Interface (CLI) is the primary tool for interacting with Docker. Below are essential commands to explore in the Play with Docker terminal.

4.1 Exercise: Exploring the Docker CLI

1. **Open Play with Docker**: Go to labs.play-with-docker.com, log in with your Docker Hub account, click `Start`, and select `Add New Instance`.

2. **Check Docker version**:

```
docker version
```

Listing 1: Check Docker version

Displays the Docker client and server versions.

3. **Display system-wide information**:

```
docker info
```

Listing 2: Display Docker info

Shows details about the Docker setup.

4. **List available commands**:

```
docker
```

Listing 3: List Docker commands

Lists all Docker commands.

Task: Run the commands above in the Play with Docker terminal and note the output.
Play with Docker Note: Use the browser terminal to copy-paste commands.

5 Running Your First Container

Let's run a container using the Ubuntu image from Docker Hub.

5.1 Exercise: Launch an Ubuntu Container

1. **Pull the Ubuntu image**:

```
docker pull ubuntu:latest
```

Listing 4: Pull Ubuntu image

2. **Run an interactive container**:

```
docker run -it ubuntu:latest /bin/bash
```

Listing 5: Run Ubuntu container

Options: `-i` (interactive mode), `-t` (allocates a terminal), `/bin/bash` (runs a bash shell).

3. **Explore the container**: Inside the container, run:

```
ls
cat /etc/os-release
```

Listing 6: Explore container

4. **Exit the container**: Type `exit` or press `Ctrl+D`.

Task: Run the container, execute the commands, and exit. Note the Ubuntu version displayed.
Play with Docker Note: If the terminal becomes unresponsive, start a new container.

6 Managing Docker Images

Docker images are the foundation of containers. Let's manage them in Play with Docker.

6.1 Exercise: Working with Images

1. **List local images**:

```
docker images
```

Listing 7: List images

2. **Search for an image**:

```
docker search nginx
```

Listing 8: Search for Nginx image

3. **Pull an Nginx image**:

```
docker pull nginx:alpine
```

Listing 9: Pull Nginx image

Uses the lightweight `alpine` tag due to Play with Docker's resource limits.

4. **Remove an image**:

```
docker rmi ubuntu:latest
```

Listing 10: Remove image

Note: Use `-f` if the image is in use.

Task: Pull the Nginx image and verify it's listed with `docker images`.
Play with Docker Note: Image pulls may take time due to network speed.

7 Docker Networking Basics

Docker networking enables container communication. Play with Docker supports basic networking with port mapping.

7.1 Exercise: Exploring Docker Networks

1. **List available networks**:

```
docker network ls
```

Listing 11: List networks

2. **Create a custom network**:

```
docker network create --driver bridge my-app-network
```

Listing 12: Create custom network

3. **Run an Nginx container on the custom network**:

```
docker run -d --name web --network my-app-network -p 80:80 nginx:alpine
```

Listing 13: Run Nginx on custom network

4. **Access the Nginx server**: Click the `80` link above the Play with Docker terminal.

5. **Inspect the network**:

```
docker network inspect my-app-network
```

Listing 14: Inspect network

Task: Run the Nginx container and access it via the provided link. Verify the network configuration.
Play with Docker Note: Public URLs for mapped ports appear above the terminal.

8 Docker Volumes and Data Persistence

Containers are ephemeral, so we use volumes to persist data. Play with Docker supports temporary volumes.

8.1 Exercise: Using Docker Volumes

1. **Create a volume**:

```
docker volume create my-vol
```

Listing 15: Create volume

2. **Run a container with a volume**:

```
docker run -d --name my-nginx -v my-vol:/app/data nginx:alpine
```

Listing 16: Run container with volume

3. **List volumes**:

```
docker volume ls
```

Listing 17: List volumes

4. **Inspect the volume**:

```
docker volume inspect my-vol
```

Listing 18: Inspect volume

Task: Create a volume, attach it to a container, and verify its creation.
Play with Docker Note: Volumes persist only for the session's duration (4 hours).

9 Writing a Dockerfile

A Dockerfile defines a custom image. Play with Docker provides a terminal-based editor.

9.1 Exercise: Building a Custom Image

1. **Create a directory**:

```
mkdir my-nginx-app
cd my-nginx-app
```

Listing 19: Create directory

2. **Create a Dockerfile**: Use `nano`:

```
nano Dockerfile
```

Listing 20: Create Dockerfile

Add:

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Listing 21: Dockerfile for Nginx

Save and exit (`Ctrl+O`, `Enter`, `Ctrl+X`).

3. **Create an HTML file**:

```
nano index.html
```

Listing 22: Create index.html

Add:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Welcome to My Docker Workshop!</h1>
    <p>This is a custom Nginx page running in a container.</p>
</body>
</html>
```

Save and exit.

4. **Build the image**:

```
docker build -t my-nginx-app:v1 .
```

Listing 23: Build image

5. **Run the container**:

```
docker run -d -p 8080:80 --name my-web-app my-nginx-app:v1
```

Listing 24: Run container

6. **Access the web page**: Click the `8080` link above the terminal.

Task: Build and run the custom Nginx image. Verify the webpage displays.
Play with Docker Note: Use `nano` for file editing. Port links appear automatically.

10 Hands-On Exercise: Building a Custom Flask App

Build a Python Flask web application in a container.

10.1 Scenario

Create a simple Flask app running in a Docker container.

1. **Create a project directory**:

```
mkdir flask-app
cd flask-app
```

Listing 25: Create Flask directory

2. **Create a Dockerfile**:

```
nano Dockerfile
```

Listing 26: Create Flask Dockerfile

Add:

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]
```

Listing 27: Dockerfile for Flask

3. **Create a requirements.txt**:

```
nano requirements.txt
```

Listing 28: Create requirements.txt

Add:

```
flask==2.0.1
```

4. **Create a Flask app (app.py)**:

```
nano app.py
```

Listing 29: Create app.py

Add:

```
from flask import Flask
app = Flask(__name__)

@app-route('/')
def hello():
    return '<h1>Hello from Flask in Docker!</h1>'

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

5. **Build and run the image**:

```
docker build -t my-flask-app:v1 .
docker run -d -p 5000:5000 --name my-flask my-flask-app:v1
```

Listing 30: Build and run Flask app

6. **Test the application**: Click the `5000` link above the terminal.

Task: Complete the Flask app setup, build the image, and verify it runs at the provided URL.

Play with Docker Note: Building the Python image may take a few minutes. Ensure the port link appears.

11 Conclusion

You've completed the Docker Basics Workshop on Play with Docker! You've learned to:

- Run and manage containers.
- Work with images and networks.
- Persist data with volumes.
- Build custom images with Dockerfiles.

Next Steps:

- Experiment with Docker Compose (create a `docker-compose.yml` file).
- Explore Kubernetes or Docker Swarm.
- Practice containerizing other applications.
- Save Dockerfiles externally, as Play with Docker sessions are temporary.

12 Exploring Swarm with Templates

The Play with Docker platform provides templates to create Swarm clusters with different node configurations: 3 managers + 2 workers, 5 managers + no workers, and 1 manager + 1 worker. This exercise compares how services behave across these templates.

12.1 Exercise: Comparing Templates

1. **3 Managers + 2 Workers (Current Template)**:
   - On `node1`, run:

     ```
     docker node ls
     ```

     Listing 31: List nodes

     Confirm 3 managers and 2 workers.
   - Deploy an Nginx service:

     ```
     docker service create --name web-test -p 8081:80 --replicas 3 nginx:alpine
     ```

     Listing 32: Deploy Nginx service
   - Check task distribution:

     ```
     docker service ps web-test
     ```

     Listing 33: List service tasks
2. **5 Managers + No Workers**:
   - Start a new PWD session with the **5 Managers + No Workers** template.

- Run `docker node ls` on `node1`. All nodes are managers.
- Deploy the Flask service from Section 8:

```
docker service create --name my-flask -p 5000:5000 my-flask-app:v1
```

Listing 34: Deploy Flask service

- Access via the `5000` port link and check task distribution.

3. **1 Manager + 1 Worker**:

- Start a new session with the **1 Manager + 1 Worker** template.
- Run `docker node ls` on `node1`.
- Deploy an Nginx service:

```
docker service create --name web-min -p 8082:80 nginx:alpine
```

Listing 35: Deploy minimal Nginx service

- Check task placement with `docker service ps web-min`.

Task: Compare task distribution across templates using `docker service ps`.
Swarm Note: The 3 Managers + 2 Workers template is ideal for high availability, tolerating one manager failure [1]. The 5 Managers template maximizes fault tolerance but increases overhead [2]. The 1 Manager + 1 Worker template is minimal and not fault-tolerant [3].

References

[1] Docker Docs, "How nodes work," `https://docs.docker.com`, 2024.

[2] Stack Overflow, "Pros and Cons of running all Docker Swarm nodes as Managers?," `https://stackoverflow.com`, 2018.

[3] Docker Community Forums, "Can you create a swarm with two manager nodes only?," `https://forums.docker.com`, 2018.

Play with Docker Tip: Copy important files to a local machine before the session expires.
———————— The - End ————————-