Worksheet: Git - Docker - Kubernetes

`Martin Forell` `October, 17th 2024`

> Remember to always commit your code when changing a file or creating a new one

Building a Flask App and Deploying with Docker & Kubernetes

Objective: By the end of this worksheet, you'll have a simple Flask app that you can dockerize, push to Git, and deploy using Kubernetes.

Part 1: Introduction to Git

Task 1: Set Up Git

1. **Install Git**: If you haven't already, check if Git is already installed by running:

```
git --version
```

If it's not installed, use the appropriate command based on your OS:

- **Linux**: `sudo apt-get install git`
- **macOS**: `brew install git`
- **Windows**: Download and install from https://git-scm.com/download/win`

```
sudo apt-get install git
```

2. **Configure Git**: Set up your Git username and email address:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Task 2: Create a Git Repository

1. Create a new directory for your Flask project:

```
mkdir flask-k8s-app
cd flask-k8s-app
```

2. Initialize a new Git repository:

```
git init
```

3. Create a Python file called `app.py` and add the following Flask code:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Kubernetes!"

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

4. Add and commit the file to Git (remember to commit after each change or creation of a new file to keep your project versioned and organized):

```
git add app.py
git commit -m "Initial commit: Add Flask app"
```

Task 3: Push to GitHub (Optional)

1. Create a new repository on GitHub.
2. Link your local repository to GitHub:

```
git remote add origin https://github.com/yourusername/flask-k8s-app.git
git branch -M main
git push -u origin main
```

Part 2: Dockerize the Flask App

Task 4: Create a Dockerfile

1. Create a file named `Dockerfile` in your project directory with the following content:

```
# Use the official Python image from the Docker Hub
FROM python:3.9

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container
COPY . /app

# Install the required packages
RUN pip install flask

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define the command to run the app
CMD ["python", "app.py"]
```

Task 5: Build and Run the Docker Image

1. **Build the Docker Image**:

```
docker build -t flask-k8s-app .
```

2. **Run the Docker Container**:

```
docker run -p 5000:5000 flask-k8s-app
```

3. Open your browser and go to `http://localhost:5000` to see your Flask app in action. If you cannot access it, make sure Docker is running, and that the port is not being blocked by a firewall or already in use.

Task 5.1: Use Docker Compose

1. Create a file named `docker-compose.yml` with the following content:

```
version: '3'
services:
  flask-app:
    build: .
    ports:
      - "5000:5000"
```

2. **Run with Docker Compose**:

```
docker-compose up
```

3. Open your browser and go to `http://localhost:5000` to see your Flask app running using Docker Compose.

Task 6: Push Docker Image to Docker Hub

1. Log in to Docker Hub:

```
docker login
```

2. Tag your Docker image:

```
docker tag flask-k8s-app yourdockerhubusername/flask-k8s-app
```

3. Push the image to Docker Hub:

```
docker push yourdockerhubusername/flask-k8s-app
```

Part 3: Deploy with Kubernetes

Task 7: Enable Kubernetes in Docker Desktop

1. If you are using Docker Desktop, ensure that Kubernetes is enabled. Go to Docker Desktop settings and check the Kubernetes option.

Task 8: Create a Kubernetes Deployment

1. Create a file called `deployment.yaml` with the following content:

```
apiVersion: apps/v1  # This specifies the version of the Kubernetes API being used for this deployment. 'apps/v1' is commonly used for deployments.
kind: Deployment  # Defines the type of Kubernetes object. In this case, it's a 'Deployment', which is used to manage stateless applications.
metadata:
  name: flask-k8s-app  # The name of the deployment, which uniquely identifies it in the Kubernetes cluster.
spec:
  replicas: 2  # Specifies the number of instances of the application to run.
  selector:
    matchLabels:
      app: flask-k8s-app  # Selects the pods that match these labels for management under this deployment.
  template:
    metadata:
      labels:
        app: flask-k8s-app  # Labels applied to the pods created by this deployment.
    spec:
      containers:
      - name: flask-k8s-app  # The name of the container running in each pod.
        image: flask-k8s-app  # Specifies the Docker image to use for the container.
        ports:
        - containerPort: 5000  # The port that the container listens on, matching the Flask app's port.
```

Task 9: Deploy to Kubernetes

1. Apply the deployment configuration:

```
kubectl get deployments
```

```
kubectl apply -f deployment.yaml
```

2. Verify that the pods are running:

```
kubectl get pods
```

Task 10: Expose the Deployment

1. Create a file named `service.yaml` with the following content:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-k8s-service
spec:
  selector:
    app: flask-k8s-app
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
  type: LoadBalancer
```

2. Apply the service configuration:

```
kubectl apply -f service.yaml
```

3. Verify the service is running:

```
kubectl get services
```

Part 4: Summary

Congratulations! You've successfully:

- Created a Flask app and committed it to Git
- Dockerized your app
- Pushed the Docker image to Docker Hub
- Deployed your app using Kubernetes

Extra Task: Try modifying the Flask app and observe how you can update your deployment by rebuilding the Docker image and applying the changes to Kubernetes.

Questions:

1. Why do we use Docker to containerize applications?
2. What are the advantages of using Kubernetes for deploying applications?
3. How would you scale your Flask app to handle more traffic using Kubernetes?