Intern Assignment Overview

We want to assess your understanding of Kubernetes through a practical task involving a Python Flask application connecting with MongoDB.

Part 1: Local Setup

This task will guide you through the setup of a Python Flask application and how to configure it to connect to MongoDB. The purpose of this task is to help you understand the problem statement and prepare you for the Kubernetes setup in the second task. You do not need to submit anything from Part 1.

Prerequisites

Ensure the following are installed on your system:

- **Python 3.8 or later**
- **Docker**
- **Pip** for managing Python packages

Step-by-Step Instructions

1. Create Project Directory

Create a new directory for your project and navigate into it:

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

2. Set Up Virtual Environment

Create and activate a virtual environment for the Python project:

```
python3 -m venv venv
source venv/bin/activate  # On Windows, use `venv\Scripts\activate`
```

Cookie Points: Can you explain the benefits of using a virtual environment for python applications?

3. Create Flask Application

Create a file named `app.py` with the following content:

```
from flask import Flask, request, jsonify  # Import necessary modules from Flask
from pymongo import MongoClient  # Import MongoClient to interact with MongoDB
from datetime import datetime  # Import datetime to get the current time
import os  # Import os to access environment variables

# Initialize the Flask application
app = Flask(__name__)

# Set up the MongoDB client
# The MongoDB URI is fetched from the environment variable 'MONGODB_URI'
# If not found, it defaults to 'mongodb://localhost:27017/'
client = MongoClient(os.environ.get("MONGODB_URI", "mongodb://localhost:27017/"))

# Connect to the database named 'flask_db'
db = client.flask_db

# Connect to the collection named 'data' within the 'flask_db' database
```

```
collection = db.data

# Define the route for the root URL
@app.route('/')
def index():
    # Return a welcome message with the current time
    return f"Welcome to the Flask app! The current time is: {datetime.now()}"

# Define the route for the '/data' endpoint
# This endpoint supports both GET and POST methods
@app.route('/data', methods=['GET', 'POST'])
def data():
    if request.method == 'POST':
        # If the request method is POST, get the JSON data from the request
        data = request.get_json()
        # Insert the data into the 'data' collection in MongoDB
        collection.insert_one(data)
        # Return a success message with status code 201 (Created)
        return jsonify({"status": "Data inserted"}), 201
    elif request.method == 'GET':
        # If the request method is GET, retrieve all documents from the 'data' collection
        # Convert the documents to a list, excluding the '_id' field
        data = list(collection.find({}, {"_id": 0}))
        # Return the data as a JSON response with status code 200 (OK)
        return jsonify(data), 200

# Run the Flask application
# The application will listen on all available IP addresses (0.0.0.0) and port 5000
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

4. Create Requirements File

Create a `requirements.txt` file with the following content:

```
Flask==2.0.2
Werkzeug==2.0.3
pymongo==3.12.0
```

5. Install Python Dependencies

Install the dependencies from `requirements.txt`:

```
pip install -r requirements.txt
```

6. Set Up MongoDB Using Docker

Pull and run the MongoDB Docker image:

```
docker pull mongo:latest
docker run -d -p 27017:27017 --name mongodb mongo:latest
```

This will start a MongoDB instance accessible on `localhost:27017`.

7. Set Up Environment Variable

Set the `MONGODB_URI` environment variable to point to the local MongoDB instance. Create a `.env` file in the project directory with the following

```
MONGODB_URI=mongodb://localhost:27017/
```

Load the environment variables from the `.env` file:

```
export $(cat .env | xargs)
```

8. Run the Flask Application

Start the Flask application:

```
export FLASK_APP=app.py
export FLASK_ENV=development
flask run
```

If you are on Windows, use:

```
set FLASK_APP=app.py
set FLASK_ENV=development
flask run
```

The Flask application should now be running on `http://localhost:5000`.

9. Accessing the Application

Open your web browser and navigate to `http://localhost:5000` to see the welcome message.

Example Requests Using curl:

GET request to `/` endpoint:

```
curl http://localhost:5000/
```

Response:

```
Welcome to the Flask app! The current time is: <Date and Time>
```

POST request to `/data` endpoint:

```
curl -X POST -H "Content-Type: application/json" -d '{"sampleKey":"sampleValue"}' http://localhost:5000/data
```

Response:

```
{"status": "Data inserted"}
```

GET request to `/data` endpoint:

```
curl http://localhost:5000/data
```

Response:

```
[  {    "sampleKey": "sampleValue"  }]
```

Part 2: Kubernetes Setup

Task: Deploy the provided Python Flask application that interacts with a MongoDB database on a Kubernetes cluster, ensuring proper setup of services, volumes, autoscaling, database authentication, DNS resolution, and resource management.

Requirements:

1. **Python Flask Application:**
    - **Endpoints:**
        - `/`: Returns "Welcome to the Flask app! The current time is: `<Date and Time>`"
        - `/data`: Allows POST requests to insert data into MongoDB and GET requests to retrieve data.
    - Ensure the application is deployed with at least 2 replicas.
2. **Database:**
    - Use MongoDB to store and retrieve data.
    - Ensure MongoDB uses authentication to secure data access. Update the Flask code to use authentication to connect with mongo.
3. **Kubernetes Setup:**
    - Use Minikube or any other local Kubernetes alternatives.

4. **Pod Deployment:**
    - **Flask Application Deployment:**
        - Create a Deployment for the Flask application with at least 2 replicas.
    - **MongoDB StatefulSet:**
        - Figure out how to enable authentication in MongoDB. Create a StatefulSet for MongoDB with authentication enabled.
5. **Services:**
    - **Service for Flask:**
        - Create a Service to expose the Flask application within the cluster and make it accessible from your local machine.
    - **Service for MongoDB:**
        - Create a Service to allow the Flask application to connect to MongoDB, ensuring MongoDB is accessible only within the cluster. Choose the appropriate type of Kubernetes service for this purpose.
6. **Volumes:**
    - **Persistent Volume (PV) and Persistent Volume Claim (PVC):**
        - Configure a PV and PVC for MongoDB to ensure data persistence. Configure the directory where MongoDB stores its data and make it persistent using Kubernetes volume.
7. **Autoscaling:**
    - **Horizontal Pod Autoscaler (HPA):**
        - Set up HPA for the Flask application to scale based on CPU usage. Scale the application if CPU utilization exceeds 70%. Minimum number of replicas are 2 and it can go to a maximum of 5 replicas.
8. **DNS Resolution:**
    - **Explanation of DNS Resolution:**
        - Explain how DNS resolution works within the Kubernetes cluster for inter-pod communication. Describe how to configure the connection from the Flask application to the MongoDB database within the Kubernetes cluster.
9. **Resource Management:**
    - **Resource Requests and Limits:**
        - Configure resource requests and limits for both the Flask and MongoDB pods to ensure efficient resource utilization and stability. Explain the use case for resource requests and limits, and configure both the applications with sample values of limits and requests (request: 0.2cpu, 250M ; limit: 0.5cpu, 500M)

Instructions for Submission:

- Provide the Dockerfile for the Flask application.
- Provide instructions on how to build and push the Docker image to a container registry.
- Provide the Kubernetes YAML files for all resources created.
- Provide a README with detailed steps to deploy the Flask application and MongoDB on a Minikube Kubernetes cluster.
- Include an explanation of how DNS resolution works within the Kubernetes cluster for inter-pod communication.
- Include an explanation of resource requests and limits in Kubernetes.
- Design Choices: Describe why you chose the specific configurations and setups, including any alternatives(if any) you considered and why you did not choose them.
- Cookie point: Testing Scenarios: Detail how you tested autoscaling and database interactions, including simulating high traffic. Provide results and any issues encountered during testing.