Task: Building and Deploying a Golang Application with Docker and Kubernetes

Note: Proper documentation of steps for running the project is required

Description: As part of this task, your goal is to build a simple Golang application, containerize it using Docker, and deploy it to a Kubernetes cluster. The task involves several steps to demonstrate your knowledge of Golang programming, Docker containerization, and Kubernetes deployment.

Task Steps:

Step 1: Golang Application

Microservice1:

Create a new user:

- Accept user details (name, email, etc.) as input.
- Store the user information in the PostgreSQL database.
- Cache the user details in Redis for faster retrieval.

Get a user by ID:

- Accept a user ID as input.
- Check if the user exists in the Redis cache.
- If found, return the user details from the cache.
- If not found, fetch the user details from the PostgreSQL database
- Cache the fetched user details in Redis.
- If the user does not exist in the database return an appropriate response.

Update a user:

- Accept a user ID and updated details as input.
- Update the user information in the PostgreSQL database.
- Update the user details in the Redis cache if the user exists.

Delete a user:

- Accept a user ID as input.
- Delete the user from the PostgreSQL database.
- Remove the user details from the Redis cache if the user exists.

→ microservice 2:

- Make a endpoint with name methods where there will be two function named method 1 and method 2
- Post request for this method will be heaving two data one in method number and other one is for wait time eg : {“method”:1,”waitTime”:2}

- Method 1 execution:
  - Requests for Method 1 should be processed sequentially. If a Method 1 request is already being processed, subsequent Method 1 requests should wait for the ongoing task to complete before executing.
  - checks the number of users in the database and their names.
  - After checking, the microservice should wait for waitTime(in seconds)
 given in post request
  - Finally, it should return the list of users' names.

- *tocommunicatewithbothmicroservicesforgettingdatathere should be use of grpc

- Method 2 execution:

Requests for Method 2 should be processed in parallel. They should not wait for ongoing Method 1 requests.

- checks the number of users in the database and their names.
- After checking, the microservice should wait for waitTime(in seconds) given in post request
- Finally, it should return the list of users' names.

Step 2: Docker Containerization

- Write a Docker filet containerize the Golang application.
- Use the official Golang base image.
- Build the Golang application inside the Docker image.
- Ensure the Docker image is as small and efficient as possible.

Step 3: Kubernetes Deployment

- Setup a Kubernetes cluster(you can use Minikube or any other Kubernetes platform of your choice).
- Write a Kubernetes Deployment manifest to deploy the Golang application as a Pod.
- Use the Docker image you built in Step2 for the Deployment.

Step 4: Kubernetes Service

- Write a Kubernetes Service manifest to expose the Golang application to the outside world.
- Use a Load Balancer type or Node Port type to allow external access.

Step 5: Kubernetes Testing

- Deploy the Golang application to the Kubernetes cluster using kubectl commands.
- Test the deployed application by accessing it through the Service's external Node Port.

Task Submission: Please submit the following as part of your task submission:

- The Golang source code for the application.
- The Docker file used to build the Docker image.
- Kubernetes manifests for the Deployment and Service.
- Any additional documentation or instructions you believe are necessary for others to understand and run your solution.

Task Evaluation: Your task will be evaluated based on the following criteria:

- Correctness and functionality of the Golang application.
- Efficiency and best practices used in the Docker file for containerization.

- Successful deployment and functioning of the Golang application on the Kubernetes cluster.
- Proper use of Kubernetes manifests for the Deployment and Service.