Fix & Deploy Go App with Redis/ DevOps

EYouth X DEPI Tech Challenge

Submitted by:
EYouth Limited

Go App Challenge

Challenge Description:

Your task is to troubleshoot, fix, and deploy a Go web application that uses Redis for caching. There are some issues in the Dockerfile or the Go code. After fixing these issues, you'll deploy the app to a Kubernetes cluster with Redis.

Implementation in Steps:

Part 1: Fix the Dockerfile and Go Application

- You’re given a Dockerfile & Go source code & dependencies.
- The Go app uses Redis to cache the number of visitors.
- Review and fix the issues in the Dockerfile or Go code to be able to build and run the app successfully.
- Important: For the Go app to run, you must have a running Redis container.
- Bonus: optimize the Go app image size.

Part 2: Deploy to Kubernetes

- Create Kubernetes YAML files to deploy Go (as stateless workload) and Redis (as stateful workload).
- Use separate namespaces: "app" & "db".
- Redis should have persistent storage & network ID.
- Manage variables using Kubernetes native capabilities.
- One pod per each workload is fine, no need for over provisioning.
- Expose the Go app using nodeport or loadbalancer service.
- Redis should be exposed with the appropriate service type that is suitable for internal communications.

Supporting Material:

https://github.com/Mostafa-Yehia/docker-k8s-challenge

Time Frame:

- Duration: 1 Week
- Deadline: Friday, 28/2/2025.

Student Deliverables:

- A Github repository contains:
  - Fixed Dockerfile & Go App code.
  - Working Kubernetes YAML files.
- Evidence of the working Docker container & image build (status & logs & screenshots).
- Evidence of the working Kubernetes YAML files (status & logs & screenshots).
- **Bonus:** Evidence of optimized image size.

This challenge will help you:

- Troubleshoot Dockerfiles and Go applications.
- Build and run Docker containers locally.
- Deploy stateful & stateless workload to Kubernetes.
- Manage configurations and variables in Kubernetes environments.
- Use storage persistence in Kubernetes environments.
- Manage traffic across namespaces & persistent network IDs.
- Optimize Docker images size.

Evaluation Criteria:

1. **Correctness of Fix (50%)**
   - Did the participant correctly identify and resolve the issue in the Dockerfile?
   - Does the container run the Flask app successfully after the fix?
2. **Explanation (30%)**
   - Is the explanation of the changes clear and concise?
   - Does it include logical reasoning for the fixes?
3. **Testing Evidence (20%)**
   - Did the participant provide proof of successful container execution (e.g., screenshot, logs)?
   - Is the evidence clear and relevant?