SOFE 4790U Distributed Systems
Lab 3: Deploying a Circuit Breaking ambassador and a Function-as-a-Service (FaaS)
23 October 2022
Group 1- Michelle Cheng 100696572

Introduction

This Lab introduces how to deploy a circuit breaking ambassador and a function as a service (Faas)
Main Objectives:

1. Learn how to create a service using NodeJS.
2. Learn how to create a Docker image.
3. Learn how to configure and use a circuit breaker using Nginx.
4. Get Familiar with FaaS
5. Learn how to configure and use OpenFaas on GCP
6. Get Familiar with Decorator Pattern

Discussion

Part 1

Problem

Businesses are required to monitor applications to ensure availability and correct performance. However, cloud-hosted applications can be more complicated to complete this. Factors such as network latency, performance and availability of the systems can cause full and partial failure in these cloud-hosted applications. Verification must be done in regular intervals to ensure the requirements are met.

Solution

Health monitoring sends requests to an endpoint on the application to complete checks and return indication of status. Checks for latency, response time, cloud storage, availability and other resources located in the application may all be carried by the health monitoring code. Results are evaluated against a set of configurable rules and checks are performed by monitoring tools. Examples include validating response code, checking content of response to detect errors, measuring response time, checking expiration of SSL certificates and more.

Requirements

- Checks performed by application in response to request to health verification endpoint
- Analysis of results that perform health verification check
- Run checks from on-premises or hosted locations close to customers
- Run test against service instances that customers use
- Periodically check the system health and caches status
- Security for monitoring endpoints
- Ensure monitoring agent is performing correctly

Part 2 and 3

In part two of this lab, one pod was used for the deployment and the scripts has instructions on the check the application is to make on the status of the service. This is done by the /alive endpoint. This is done periodically and repeated every 10 seconds. In part two the decorator pattern was used and functions can be used to add values to test the input and output response from the HTTP RESTful API. In this case the main function stores shapes defined by a name, color, and dimensions into a database. The function returns the received JSON which mimics it storing the object. The decorator function is able to extend the behavior and add missing attributes for the shape JSON object. Test were run for both of these services by adding a JSON shape object and checking that is returned as expected. An incomplete JSON object was also tested with the decorator function and the missing attribute colour was set to default transparent.

Design

Some help from https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/ was used to complete this section of the lab.

Persistent volumes (PV) are a resource within a cluster that is used for storage. They are volume plugins like Volumes but have independent lifecycle not dependent on the pod. PersistentVolumeClaim(PVC) are similar to pods and consume PV resources. They can request for different size and access modes as needed. Pods can be configured to use PVC for storage. This is implemented by setting a PVC that is bound to the PV and then creating a pod that uses the PVC. An example of persistent volume use can be to store and share data amongst pods.

For example the PVC created below called task-pv-claim is bound to the PV task-pv-volume.

```
michelle0109cheng@cloudshell:~ (agile-skyline-362514)$ kubectl get pv task-pv-volume
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
task-pv-volume   10Gi       RWO            Retain           Bound    default/task-pv-claim   manual                  72s
michelle0109cheng@cloudshell:~ (agile-skyline-362514)$ kubectl get pvc task-pv-claim
NAME            STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Bound    task-pv-volume   10Gi       RWO            manual         53s
```

A pod can then be configured to use the PVC task-pv-claim as seen below in the yaml file.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        Created an hour ago
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: task-pv-storage
```

Video Link:

https://drive.google.com/drive/folders/1xgc_tz8b0dCA3KTDVDksw7GwbgYliRxA?usp=sharing

Resources

- https://github.com/GeorgeDaoud3/SOFE4790U-lab3
- https://learn.microsoft.com/en-us/azure/architecture/patterns/health-endpoint-monitoring
- https://martinfowler.com/bliki/CircuitBreaker.html
- https://www.nginx.com/blog/microservices-reference-architecture-nginx-circuit-breaker-pattern/
- https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/