National University of Singapore

School of Computing

Summer Workshop **Quiz** 2024

Submission Deadline

26th May 2024 (Sunday) 11:59pm sharp.

Objectives

Welcome to the Summer School Quiz. In this quiz, you will use Kubernetes to implement a simple HTTP Echo Server that can be accessed via an intermediate Proxy Server. This assignment will help you to gain familiarity with programming and configuring Pods on the Kubernetes Cluster.

Submission

Please submit a zip file that contains source programs and submit it to the *Quiz* folder in *Canvas* > *2024_SWS3004* > *Assignments*. The .zip file should contain the files required for your implementation. The name of your project folder should be <STUDENT ID>.zip. Replace <NUSNET ID> (starting with T), for example, T0123456.zip.

The archive should have the following structure:

```
.
├── kube-yaml/
│   ├── proxy-pod.yaml
│   └── server-pod.yaml
├── client/
│   └── client.py
├── proxy/
│   ├── Dockerfile
│   └── proxy.py
└── server/
    ├── Dockerfile
    └── server.py
```

Example Code

We provide example code with a simple HTTP server and client to test the Kubernetes deployment.

The example project has the following structure:

```
.
├── README.md
├── kube-yaml/
│   ├── client-pod.yaml
│   └── server-pod.yaml
├── client/
│   ├── Dockerfile
│   └── client.py
└── server/
    ├── Dockerfile
    └── server.py
```

You can read all the instructions in README.md to complete the Kubernetes environment setup, and test example code on different environments, e.g., on Localhost, Docker, and Kubernetes.

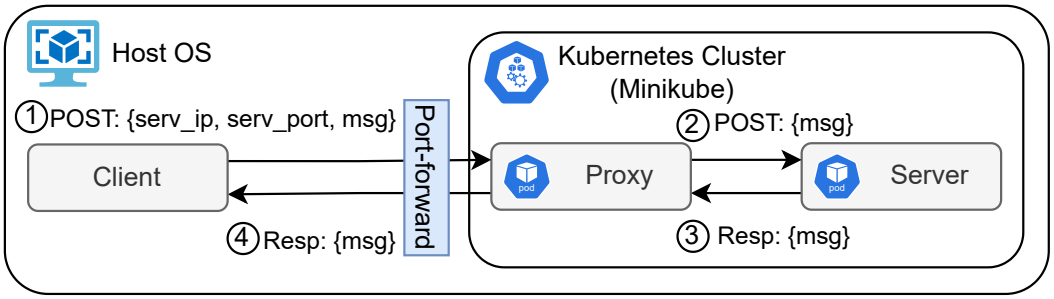

Figure 1: HTTP Echo Server Workflow.

Quiz Overview

In this quiz, you will develop a simple HTTP echo server, a proxy server, and a client. The echo server will respond with any received message. The proxy server will forward requests and responses between the client and the echo server. The client will send requests to the echo server via the proxy. You will containerize these applications, create Docker images, and then deploy them as separate pods in a Kubernetes cluster.

An overview of the application workflow can be shown in Figure 1. The proxy and server will be run as pods in the Kubernetes cluster. The proxy exposes its port externally to the host OS through *port-forward*. The client in the Host OS tries to access the server in the Kubernetes cluster via the proxy. The entire workflow covers 4 steps:

1. The client sends a POST request to the proxy through the forwarded port. The request to proxy contains a) server IP and port, and b) message to echo.
2. The proxy parses the request and connects to the server through the provided server IP and port. It sends a POST request that contains the message to echo to the server.
3. The server responds to the proxy with the message.
4. The proxy responds to the client with the message from the server.

Requirements

Step 1 - Development:

Write the code for server.py, proxy.py, and client.py as described. You can test the output of the program locally with the following commands.

- server.py: Create an HTTP echo server that listens for POST requests and responds with the same message it receives. The server listens to port 8080 by default. Command to run server.py:

```
python3 server.py
```

- proxy.py: Develop a proxy that forwards POST requests from the client to the echo server and returns the response from the echo server back to the client. The proxy listens to port 8888 by default. Command to run proxy.py:

  ```
  python3 proxy.py
  ```

- client.py: Implement a client that reads inputs (i.e., server/proxy IP, server/proxy port, and message to echo) from the command line interface (CLI) and sends POST requests to the proxy server. The request to proxy can be in JSON format and includes the message to be echoed and the echo server's details. Command to run client.py:

  ```
  python3 client.py localhost 8888 localhost 8080 "Hello World"
  # The first "localhost" is the IP address for the proxy.
  # The second "localhost" is the IP address for the server.
  # Expected output: "Hello World".
  ```

Step 2 - Dockerization:

Create a Dockerfile for the echo server and proxy to build Docker images. Build and push the Docker images to a Docker registry (e.g., Docker Hub). Commands to build and push server and proxy images:

```
cd server/
docker build -t ${user}/quiz-server .
docker push ${user}/quiz-server
cd proxy/
docker build -t ${user}/quiz-proxy .
docker push ${user}/quiz-proxy
```

Step 3 - Kubernetes Deployment:

Write YAML configuration files (i.e., server-pod.yaml and proxy-pod.yaml) to deploy each Dockerized application as a separate pod in a Kubernetes cluster. You can run the following commands to test the deployment:

1. Commands to deploy server and proxy pods on Kubernetes Cluster:

```
kubectl apply -f server-pod.yaml # server name "http-server".
kubectl apply -f proxy-pod.yaml # proxy name "http-proxy".
```

2. Use port-forward to expose proxy port 8888 to Host OS port 8881:

```
kubectl port-forward --address 0.0.0.0 http-proxy 8881:8888
```

3. Command to get IP address, i.e., Server_IP, of server pod:

```
kubectl get pod http-server --template "{{.status.podIP}}"
```

4. Command to run client.py:

```
python3 client.py localhost 8881 ${Server_IP} 8080 "Hello World"
# The port 8881 on localhost forwards requests to the proxy.
# Expected output: "Hello World".
```

THE END