Exercise 1: Docker-compose and microservices hands-on

Student: Tuong Hoang

1. Basic Informations

Hardware Platform

- **Device**: Apple MacBook Pro (2021 model)
- **Chip**: Apple M1 chip
- **Architecture**: ARM64 (Apple Silicon)

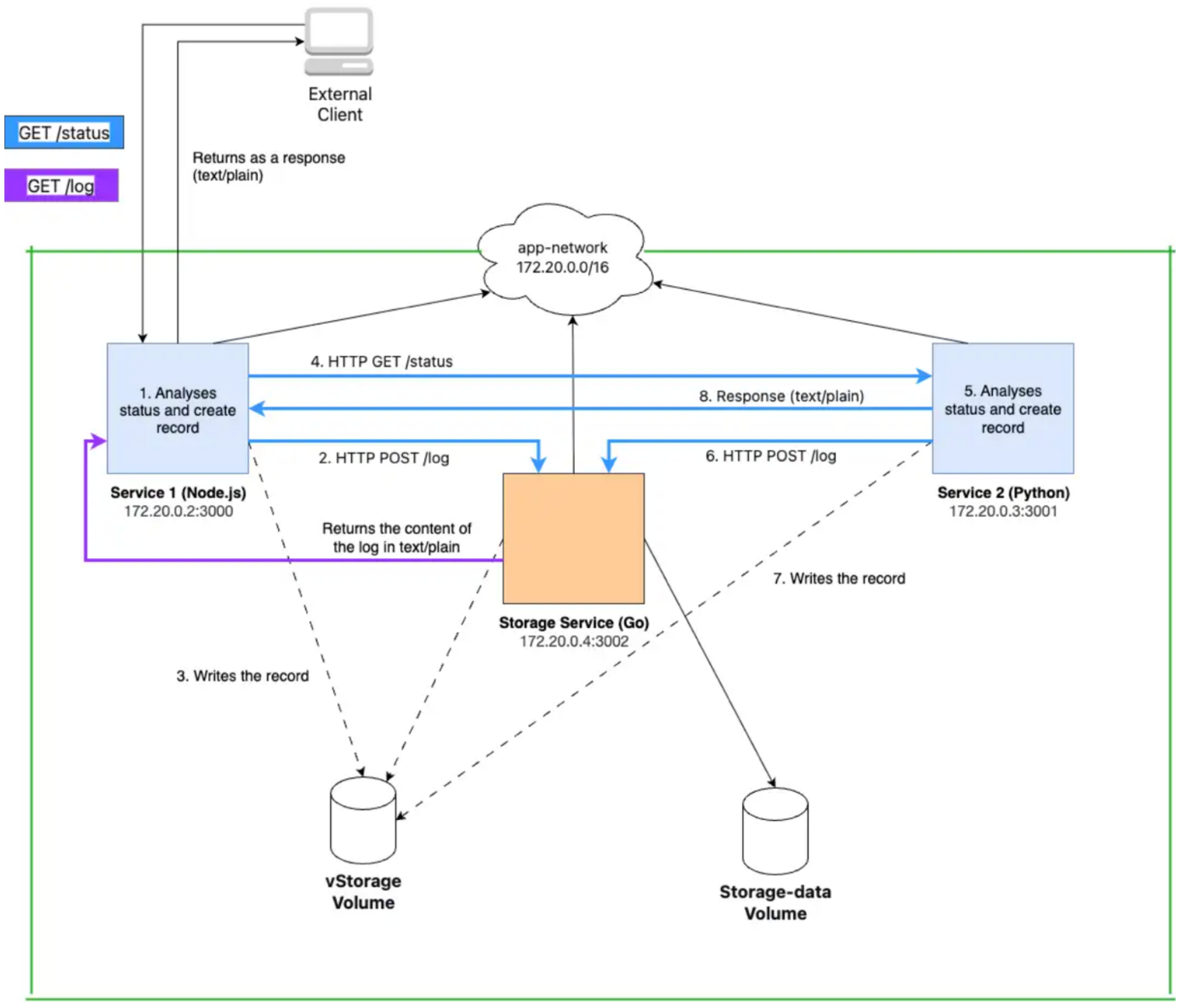
Operating System

- **OS**: macOS Sequoia 15.6.1
- **Type**: Unix-based operating system
- **Kernel**: Darwin

Docker Environment

- **Docker Version**: 28.4.0
- **Docker Compose Version**: 2.39.2
- **Docker Desktop**: Version 4.46.0

2. System Architecture Diagram


External Client
GET /status
GET /log
Returns as a response (text/plain)
app-network 172.20.0.0/16
4. HTTP GET /status
8. Response (text/plain)
1. Analyses status and create record
5. Analyses status and create record
2. HTTP POST /log
6. HTTP POST /log
Service 1 (Node.js) 172.20.0.2:3000
Service 2 (Python) 172.20.0.3:3001
Returns the content of the log in text/plain
7. Writes the record
Storage Service (Go) 172.20.0.4:3002
3. Writes the record
vStorage Volume
Storage-data Volume

Analysis of Status Records

Disc Space

The disk space determined during the running of containers is the free space in each container. This is one of the important data and is commonly used to reflect the usage status of containers when storing files and folders. This is not the actual disk capacity of the server (host) and does not show any status of the host.

Relevance: Although it is important to determine the resources available in containers, this measurment does not clarify the current state of the host system. However, in certain cases, this disc space measurment is useful for predicting and increasing the disk capacity of containers when the system scales up.

Improvements:
In fact, there are still many other convenient and accurate ways to get data on the free disk of containers without having to build services:

- Use the command container stats with container's name can container ID to get the status information of containers such as CPU usage in percentage, memory USAGE and LIMIT, MEM usage in % and so on.
- Prometheus is very suitable for Microservices systems like this exercise and has Listening services. A proactive monitoring system will help administrators detect early warning signs of bad things including managing containers as well as their information such as disk space.

Uptime

Uptime is a measure of container initialization time and can be obtained using /proc/uptime. Besides providing important information about the startup, restart, and running time of each container, it also reflects the presence and availability of services running in the container environment.

Relevance: This is a very important metric in managing the health of containers as well as existing services running in many different environments. In larger systems or applications with many services running concurrently, the time spent on initial setup and configuration will be more and so optimizing and managing uptime will help save time as well as make services run smoother.

Improvements:
Getting uptime data and managing services can be done better in this exercise:

- Add counters to track the number of restarts of containers, making it easier and more convenient to manage the health and availability of services.
- Add logging for container lifecycle events to easily capture and manage as well as easily troubleshoot issues related to operational status

Comparison and Analysis of the persistent storage solutions

	vStorage Volume	**Storage Service** **(Containerized)**
Advantages	- Easy setup and debugging - Access directly to host data - Suitable for	- Better containers and services management and security with microservice architecture approach - Scalable and best practice for many environments (staging and production)

	vStorage Volume	**Storage Service (Containerized)**
	development and testing	
Disadvantages	- Security problem (access file permission) - Only suitable for development and testing	- Complex approach and initial setup - Require more resources (Cloud services, platforms,...) - Need to manage network communication, subnet systems

In general, creating a persistent storage service as a separate container is complex and time-consuming initially to set up and initialize. But as applications and services scale up and are added, having a unified microservice system combined with effective management and monitoring capabilities will make the whole system run more smoothly and seamlessly. And that is why this approach is more practical and suitable for environments such as staging and production. However, the approach of mounting vStorage in a container to a local file is also effective when working in a development environment or for learning purposes because of its quickness and ease of setup. In short, both solutions above have pros and cons and are suitable for different types of situations and purposes, and programmers must consider carefully based on the technical requirements as well as the scale of the system.

Teacher's Cleaning Instructions

Complete Cleanup

```
# Stop and remove containers
docker-compose down

# Remove volumes
docker volume rm exercise1_storage-data exercise1_vstorage 2>/dev/null || true

# Remove local vstorage files
rm -f vstorage/log.txt

# Remove Docker images
docker rmi service1 exercise1-service1 exercise1-service2 exercise1-storage 2>/dev/null || true
```

Technical Difficulties and Main Problems

1. psutil compilation issues

Initially, when getting container uptime information built with Python and Flask, I used psutil, a popular Python library for process and system monitoring in Python. However, during compilation on Alpine Linux, missing build dependencies caused the containerization application to fail

Solution: I decided to replace psutil with using Linux's /proc/uptime and the df command to get container uptime data. This is a good alternative, as I don't need to add any external libraries while still ensuring the service runs properly.

2. ES6 module import syntax

This is not really a challenging problem and just a small mistake when I worked with Service 1. Initially, I planned to import the libraries needed to use the ES6 module import syntax to perform the necessary functions in Service 1 with JavaScript and Express. However, after completing Service 1, I forgot to add the ""type": "module"" config in the package.json file to be able to use the import syntax. And when compiling the application, the process was stopped because it could not find the module that I imported.

Solution: I added the ""type": "module"" config in the package.json file and the compilation process went smoothly.

Learning Outcomes

This exercise not only helped me sharpen my skills in working with Express, Flask, Go applications but also provided me with great insights into the best practices and practical challenges in building microservices architecture such as managing CPU and memory usage efficiently, optimizing uptime of services for better performance and understanding more deeply about logging, debugging and monitoring for the services system. So for me, this was a very useful and extremely interesting exercise.