Microservices exercise (Service1, Service2, Storage) – Arash Ghasemzadeh Kakroudi

This setup implements the required three services and two persistent storage mechanisms.

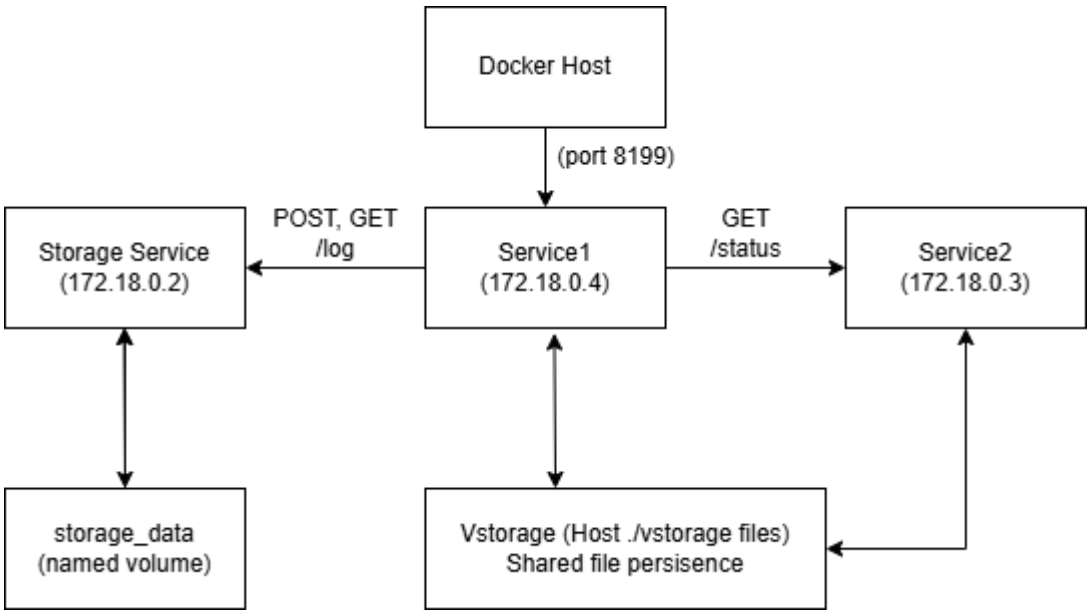
Platform Information

- **Hardware/VM: VirtualBox virtual machine**
- **Operating System: Linux Ubuntu**
- **Docker Version: Docker version 24.0.6**
- **Docker Compose Version: v2.22.0-desktop.2**

System Architecture

Services:

- **Service1 (Node.js/Express) — the only externally exposed service on port 8199. IP: 172.18.0.4**
- **Service2 (PHP) — internal only. Provides /status and logs to both storages. IP: 172.18.0.3**
- **Storage (Node.js/Express) — simple REST service with IP: 172.18.0.2**
    - **POST /log (text/plain) — append one line persistently**
    - **GET /log (text/plain) — return full log**



Persistence:

- **vStorage: host file ./vstorage mounted into Service1 and Service2 at /app/vstorage and appended per request. This is the simple shared file method.**
- **storage_data: a named Docker volume mounted only into Storage at /app/data and used to persist its internal log.txt.**

Networking:

- **A single user-defined backend network connects all three services. Only Service1 publishes a host port.**

How to run

- **cd devops-practices**
- **Build and start: docker compose up --build -d**
- **Wait ~10 seconds**
- **Test status flow: curl localhost:8199/status**
- **Fetch storage log via gateway: curl localhost:8199/log**
- **Stop: docker compose down**

Teacher cleanup instructions

- **Remove the simple host-file storage: rm -f ./vstorage**
- **Remove the named volume for Storage: docker volume rm docker-microservices_storage_data (volume name may be storage_data when not namespaced)**

Analysis of Status Records

- **Timestamp format: Using ISO 8601 UTC (e.g., 2025-09-22T13:05:35Z) as required.**
- **Uptime measurement: Both services report container uptime from /proc/uptime (for Service1 via Node's os.uptime() and for Service2 by parsing the file) and convert to hours with two decimal places.**
- **Disk space measurement:**
    - **Service1: Reports free disk space on the root filesystem (/) in MB via the df command.**
    - **Service2: Reports free disk space on the root filesystem (/) in MB via the df command.**

Measurement relevance:

- **The uptime measurements show how long the container has been running, not the host system.**
- **The disk space measurements reflect the container's root filesystem, which may be a different view than the host system.**
- **Improvements could include monitoring total system resources, memory usage percentages, or container health metrics.**

Persistent Storage Comparison

1. **Host file binding (./vstorage)**
    - **Pros:**
        - **Simple to implement**

        - **Easy to inspect directly from the host**
        - **Persists between container restarts**
    - **Cons:**
        - **Not portable across hosts**
        - **Less secure (host file exposed)**
        - **No built-in isolation**
        - **Could cause permission issues between containers and host**
        - **Considered bad practice in production**

2. **Docker named volume (storage_data)**
    - **Pros:**
        - **Fully managed by Docker**
        - **Portable between different environments**
        - **Better isolation and security**
        - **Data persists beyond container lifecycle**
    - **Cons:**
        - **Requires explicit management commands to inspect or clean**
        - **More complex to access from outside Docker**

Expected Behavior

1. **GET localhost:8199/status:**
    - **Service1 creates Timestamp1 ... line, appends to vStorage, posts to Storage.**
    - **Service1 calls Service2 /status; Service2 creates and logs Timestamp2 ..., posts to Storage and returns the line.**
    - **Service1 returns record1\nrecord2 as text/plain.**
2. **GET localhost:8199/log:**
    - **Service1 proxies to Storage GET /log and returns text/plain.**

Challenges and Solutions

One of my main challenges was ensuring proper isolation of services while allowing needed communication. I have researched a bit more and as a result, I defined custom backend network and only exposed Service1 port externally. Since I am not using Linux for my personal use, I had to use virtualbox as mentioned before. I created a shared file in the vm and my PC to develop it in a more convenient way. That being said, my main problem was Linux configuration.