Report

This is the report for the first exercise of the "Continuous Development and Deployment – DevOps" course at Tampere University in Finland 🇫🇮.

1. Platform used

The development platform is a `Macbook Pro 16-inch 2021`. It uses an `Apple M1 Pro` CPU and provides `16 GB` of memory storage. The software runs on `MacOS Tahoe`, version `26.0`.

The version of Docker used is `28.0.4`, build `b8034c0`.

```
docker --version
```

The version of Docker Compose is `v2.34.0-desktop.1`

```
docker-compose --version
```
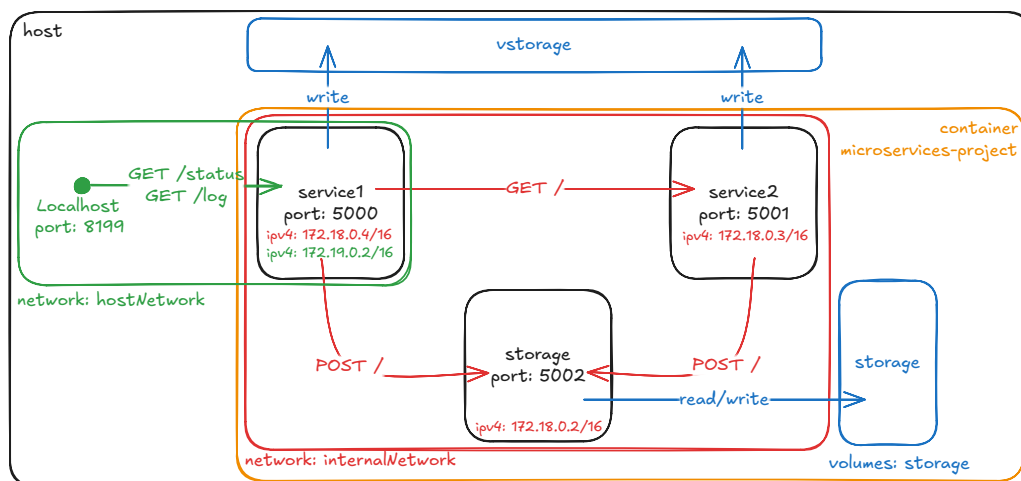
2. Graphical Representation

Diagram Showing the Services, Networks and Storage.[1]

Color / Style	Description
Black text	Service name & internal port (inside each container)
Red text	IPv4 addresses (internal Docker networking)
Red arrows/lines	Internal connections between services via `internalNetwork` Text on arrows = HTTP method & endpoint
Green text/arrows	Communication between host and service1 via `hostNetwork` (e.g., `localhost:8199`)
Green border/line	Network: `hostNetwork` (Docker network accessible from the host machine)
Red border/line	Network: `internalNetwork` (isolated, internal-only Docker network)
Blue lines/arrows	Shared/Internal volumes and file operations (read/write access between services/host)
Orange border	Container boundary (the `microservices-project` Docker Compose project)
Host (also black)	Host boundary

2.1. Components

This image shows the architecture of the Docker-based microservices project and illustrates the relationships between the host, the three services (`service1`, `service2` and `storage`), their networks and the shared storage volumes.
The outermost black rectangle represents the `host` machine. It interacts with the `microservices` container via the `hostNetwork`. The docker software, which hosts our `microservices` container, runs on it. The `hostNetwork` is accessible to the `host` and `service1`, which is one of the three services hosted by the Docker container. However, there is also an `internalNetwork` that connects all the services and is completely isolated from the host.

2.2. Networking

`service1`: Runs on port `5000`, with two internal IPv4 addresses (`172.18.0.4/16` and `172.19.0.2/16`). It is accessible from the host via port mapping (`localhost:8199`).

`service2`: Runs on port `5001`, with internal IPv4 address `172.18.0.3/16`.

`storage`: Runs on port `5002`, with internal IPv4 address `172.18.0.2/16`.

The networks were inspected by:

```
docker network ls
```

Then the IPv4 addresses were extracted by running the following command:

```
docker network inspect microservices-project_internalNetwork
```

```
docker network inspect microservices-project_hostNetwork
```

2.3. Communication Flows
Host to service1: The `host` sends HTTP GET requests (`GET /status`, `GET /log`) to `service1` via the `hostNetwork` (green arrow).

service1 to service2: `service1` sends HTTP GET requests to `service2` (red arrow labeled `GET /`).

service1 to storage: `service1` sends HTTP POST requests to `storage` (red arrow labeled `POST /`).

service2 to storage: `service2` also sends HTTP POST requests to `storage` (red arrow labeled `POST /`).

2.4. Volumes
vstorage (blue lines/arrows): A shared volume mounted on both `service1` and `service2`, allowing both services to write to the same persistent storage on the host.

storage volume (blue line): The storage service has its own dedicated volume for persistent data, accessible only by `storage`.

3. Analysis of Records

4. Analyses of Storage Solutions

5. Instructions for Cleaning Up

6. Difficulties

7. Main Problems

---

1. used tool: https://excalidraw.com