# COMP.SE.140-2025-2026-1 CONTINUOUS DEVELOPMENT AND DEPLOYMENT – DEVOPS

## Report on Exercise1

Pranab Sarker
Student Number: 152768261
Email: pranab.sarker@tuni.fi

## Repository URL:

https://github.com/pranabumal/centralLogService-for-microservice-applications.git

## Basic information about the platform:

=== HARDWARE ===
Model: MacBookAir10,1
CPU Cores: 8
Memory: 16 GB

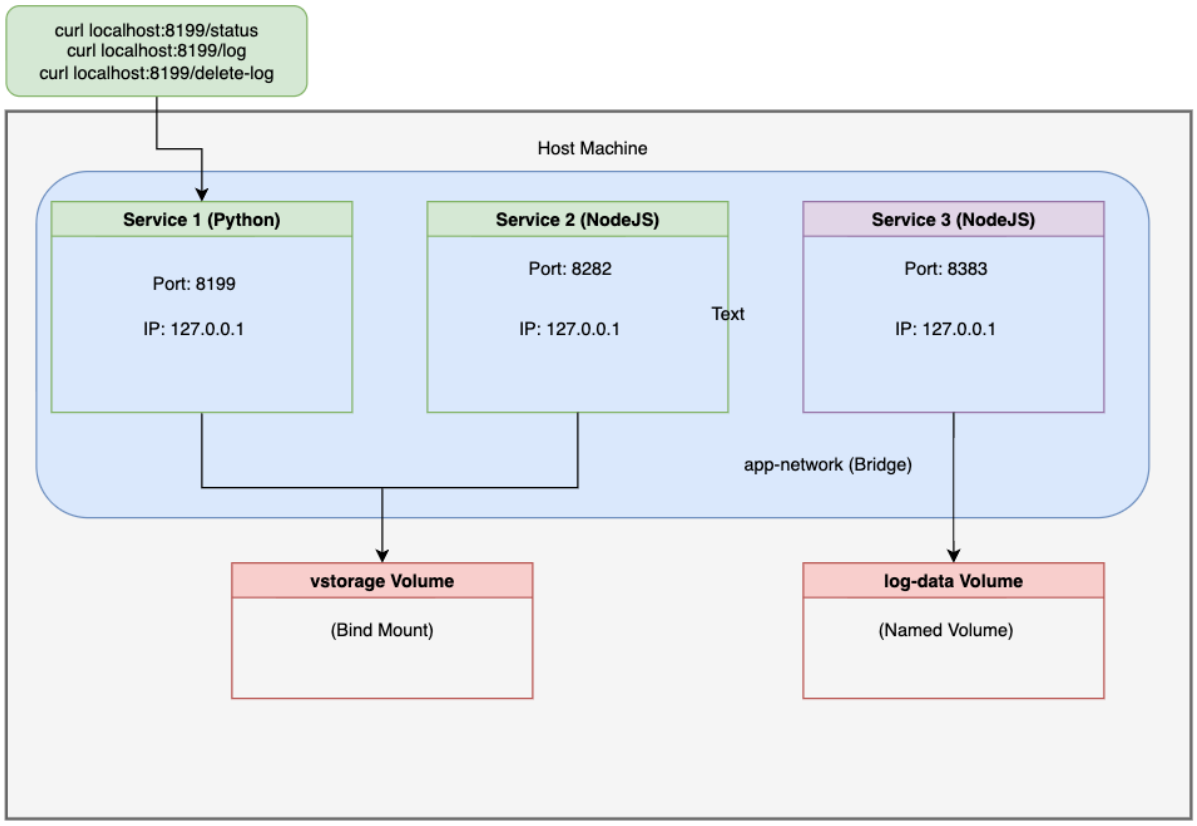
=== OPERATING SYSTEM ===
ProductName: macOS
ProductVersion: 26.0
BuildVersion: 25A354
Kernel: Darwin DIGITALs-MacBook-Air-2.local 25.0.0 Darwin Kernel Version 25.0.0: Mon Aug 25 21:17:45 PDT 2025; root:xnu-12377.1.9~3/RELEASE_ARM64_T8103 arm64

=== DOCKER ===
Docker version 23.0.5, build bc4487a

=== DOCKER COMPOSE ===
Docker Compose version v2.17.3

=== DOCKER DESKTOP===
Version: 4.19.0

## Diagram showing the services:




## Analysis of the content of the status records:

**Server 1**: 2025-09-28T01:15:42.472605: uptime 6.91 hours, free disk in root: 18877.62 MBytes

**Server 2:**2025-09-28T01:15:42.531Z: uptime 6.91 hours, free disk in root: 18877.62 MBytes

**Disk Space:** Here I tried to measure the available disk space, like -
/app # df /

| Filesystem | 1K-blocks | Used | Available | Use% | Mounted on |
|---|---|---|---|---|---|
| overlay | 61202244 | 38730248 | 19330672 | 67% | / |

With node js tried to run directly ' df /' command and with pyhton used ‘psutil.disk_usage('/')’ function to measure the available free space.

Uptime: Measured Host system uptime, not container runtime. [os.uptime()]

Regarding the relevance, could be also measure the applications storage volume. For now measured the container root filesystem. And also uptime is showing only host machine uptime not the container runtime. Which does not actually measure the application’s resource usage.

Lots of metrics could be used for better sight of the application health. Like , including the volume storage used space and free space, memory usage, API request count, APIs are up or down status.

## Comparison of persistence storages

Here I used two types of volume mounting, one is Bind mounts [ ./vstorage → /vstorage ], another one is named volumes [log-data → /log-data].

Bind mounts is very simple and cleanup implementation. Which is also easy to backup with standard OS tools. But it has some bad impact, such as permission issue between host and container, and its not suitable for production use, and also not suitable for portability.

Named volumes is a storage location managed by docker itself. Its optimized and portable. This one also suitable for production use. But its has some complexity for maintainance. And also need to learn perfectly that how its working, and what is it’s architecture.

## Log Cleaning:

For cleaning up the logs, I have added an extra API, ‘localhost:8199/delete-logs’ which will delete the logs from both types of storage.
After adding the logs through the status api, you can run the delete-logs api to delete the logs. Command ex:

curl localhost:8199/delete-logs

## Difficultu faced during this project work:

Main difficulties were choosing the second language to implement the server 1, As I was know to Node.js, but I didn’t do any application with the python, there was a learning curve and also took help from the AI to get the server status.

Also faced issue with the use of different persistence storage usages, at first got confused which one could be a central volume to access from both server to storing logs.

## Regarding the Main Problems:

Main problem was calculating the free spaces with different language. And also the different types of volume storage.