COMP.SE.140-2025-2026-1-TAU

Continuous Development and Deployment – DevOps

Exercise 1 - Basics of containers and microservices

Tuukka Salonen

20.09.2025

1. Platform

The containers were run in two different environments while doing the exercise. The main system was a local Windows PC, but the functionality of the containers was also tested on an Ubuntu virtual machine using VMware software.

Windows PC

- Windows 11 version 10.0.22631
- Docker version: 28.3.2, build 578ccf6
- Docker Compose version: v2.39.1-desktop.1

Ubuntu virtual machine

- Ubuntu 24.04.3 LTS
- Docker version 28.1.1, build 4eba377
- Docker Compose version v2.33.1

2. Services, network and storage

Service 1: the main container which runs NodeJs + Express.js. It exposes port 8080 inside the network and is bound to the host machine port 8119, allowing access at `localhost:8119`. The requests are forwarded to the other services through this service.

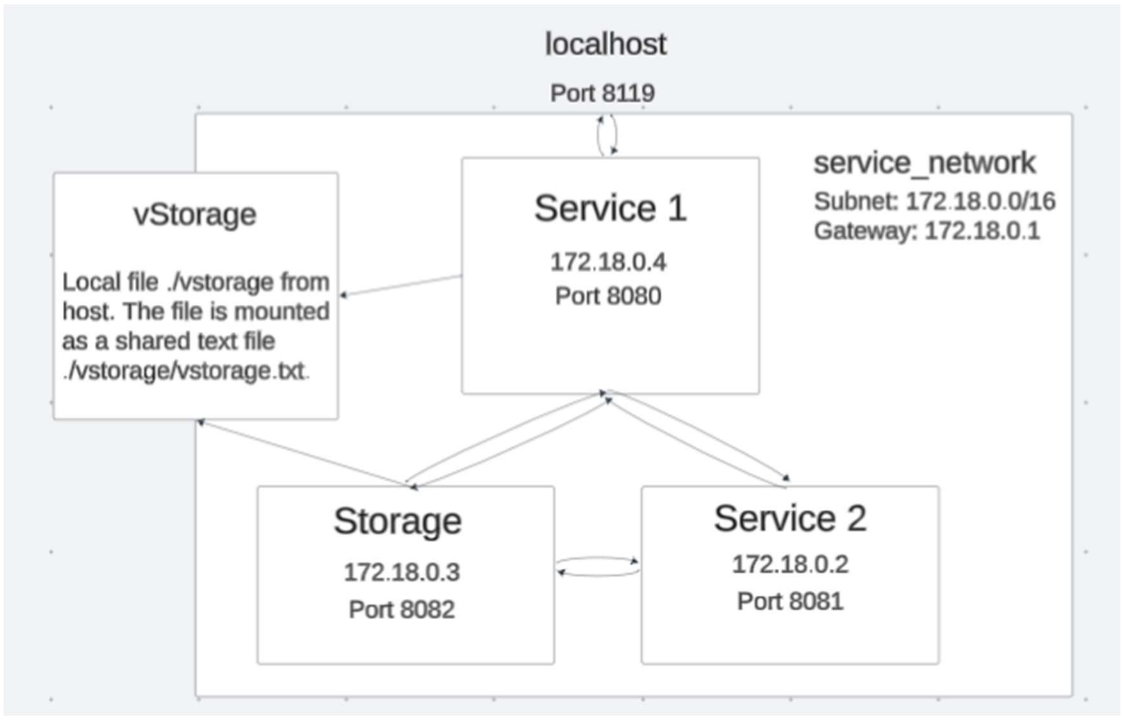
Service 2: container which runs Flask. It exposes port 8081 inside the network. It isn't bound the port to the host machine, meaning it is only accessible inside the network.

Storage: container which runs C#. It exposes port 8082 inside the network. It isn't bound the port to the host machine, meaning it is only accessible inside the network. It uses a log.txt file to store the log data. The file is stored inside /data folder which is mounted to a Docker volume `storage_data` to keep the log data persistent if the container is removed.

vStorage: a local file from the host which is mounted to Service 1 and Service 2.

service_network: internal network for the containers.

Diagram:



3. Analysis of content of status records

Inside Service 1, I used check-disk-space npm package to fetch the data to make it simple. It offers functionality for Linux, macOS and Windows. For Service 2, I used shutil module to get the amount of free disk space.

When the containers were running on my Windows 11 computer, the available space didn't match the free space from the containers. The reason is due to Docker desktop running through WSL2. The containers saw the size of the WSL2 virtual disk. On the Ubuntu virtual machine, it displayed the correct free disk space that was available on the allocated disk. As Docker ran directly on the virtual machine and not through WSL2 virtualization, correct results were obtained.

The uptime of was the uptime of the kernel and not the container. On Windows 11, it is the uptime of WSL and on the Ubuntu virtual machine, it is the uptime of the virtual machine, since Docker shares the host's kernel.

These measurements were relevant when the containers were run on the Ubuntu virtual machine, since they were accurate. On Windows, these values were disrupted by WSL running between Docker and the system. The measurements were not relevant data about the system, however having that data about the WSL could be useful as well.

The measurements could be improved, especially on Windows, by having metrics for the different levels (container, WSL, host) or mounting host drives as volumes for more accurate disk space measurements.

4. Analysis and comparison of storage solutions

Shared mount to a local host file

This option isn't really recommended as was said in the instructions for this exercise. Since multiple containers used and updated the same file, the data is not isolated. Also, using the file directly from the host makes the containers dependent on the host. If the file is renamed or deleted, the container may face problems if these are not considered.

However, this option also has some positives. It is an easy way to share a resource between containers. This is also a way to keep the data persistent without a volume if containers are removed.

Storage container

This option was much better in my opinion. The solution can be done multiple ways inside the container. This solution isolates the data better. In my solution, the data can be shared the same way between the containers using the data volume, but since it is not as dependent on the host machine, it's more isolated and safer.

This way is more difficult and laborious to set up since a completely new container is required. In addition, isolation may be a problem rather than a positive in some cases.

5. Cleanup instructions

The persistent volume of Storage can be deleted after the container has been removed. You can remove the containers by first shutting them down. Then run `*docker compose down -v*` to remove the containers and the volume. You can also remove the containers and volume separately by running `*docker compose down* ` and then `*docker volume rm exercise1_storage_data*`.

The mounted vStorage local file can be erased simply by deleting it or deleting the contents.

6. Difficulties and problems

I wanted to try a new programming language since it was encouraged to have multiple different ones. I ended up trying C# since I have wanted to try it for a while, but I have never gotten to it. It was used in the Storage service, which itself didn't require much code. However, since it was a new language to me, it ended up causing some minor problems. I also had some permission problems with Docker when testing on the Ubuntu virtual machine. However, I was able to solve them.