Tampereen yliopisto
Tampere University

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

Report on Exercise1

Saurav Paul

Email: saurav.paul@tuni.fi

Student ID: **153060573**

Repository

Medium: Github

Repository url: https://github.com/Saurav-Paul/DevOps_Course/tree/exercise1#

Platform Specifications

- → **Hardware:** Desktop PC
- → **OS:** Windows 11, 64-bit
- → **Docker Version:** 28.3.3
- → **Docker Compose Version:** v2.39.2-desktop.1
- → **Testing Platform:** Alphine linux vm (in proxmox)

Run Instructions

Following instructions was tested on a linux virtual environment,

Clone the repositoy

git clone -b exercise1 https://github.com/Saurav-Paul/DevOps_Course.git

cd DevOps_Course

Start the Stack

docker compose up --build –d

Check Health Status

curl localhost:8199/health

Hit Service Endpoints

curl localhost:8199/status

curl localhost:8199/log

Verify Storage Sync

diff -u <(curl -s localhost:8199/log) <(cat ./vstorage)

Cleanup Logs

clean up with api,

curl -X POST localhost:8199/cleanup

clean up manually

docker compose down -v

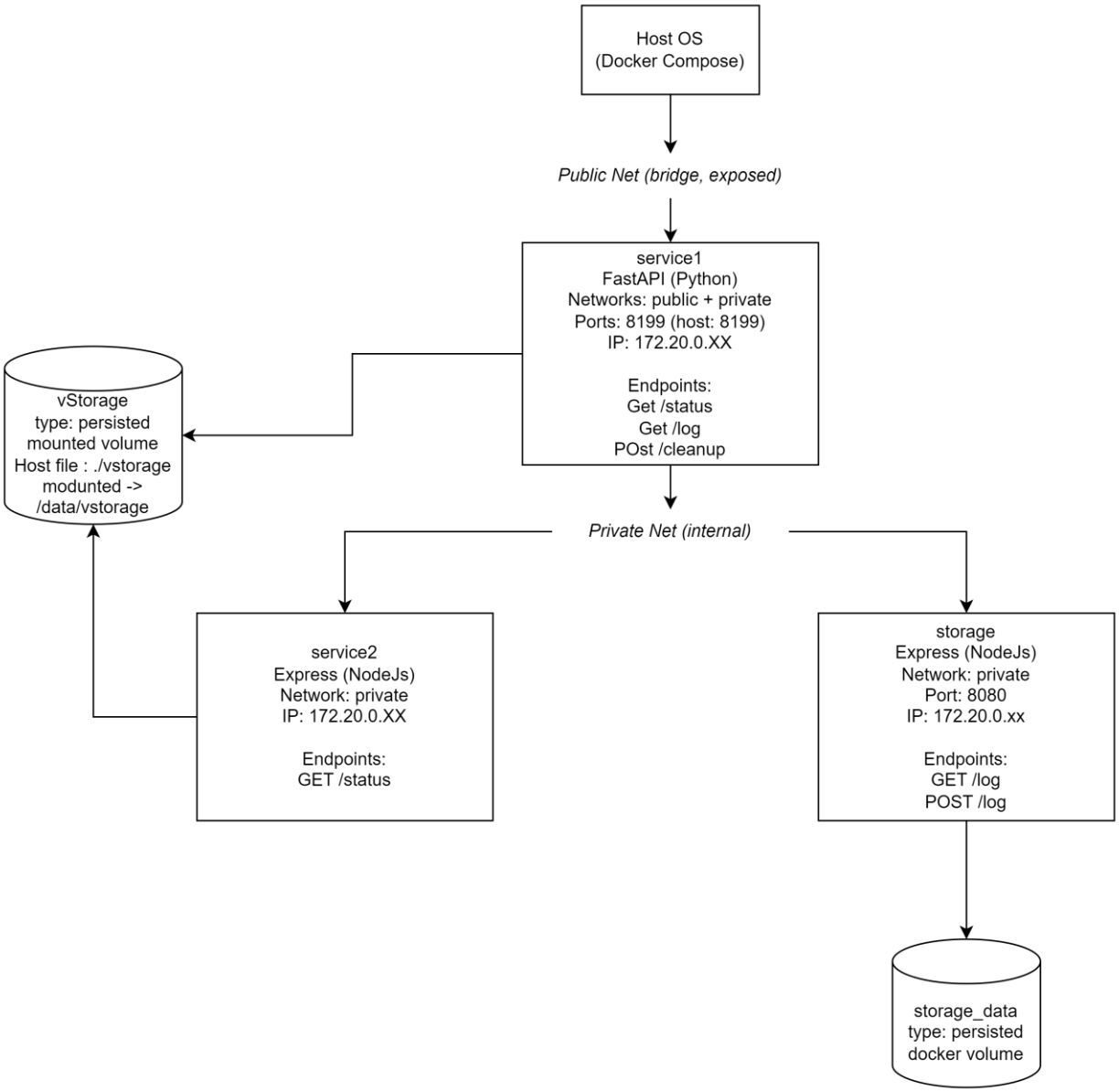
: > ./vstorage

Shutdown Stack

docker compose down

Note: Instruction also can be found on README.md file inside the repository.

Diagram


Host OS
(Docker Compose)
Public Net (bridge, exposed)
service1
FastAPI (Python)
Networks: public + private
Ports: 8199 (host: 8199)
IP: 172.20.0.XX
Endpoints:
Get /status
Get /log
POst /cleanup
vStorage
type: persisted
mounted volume
Host file : ./vstorage
modunted ->
/data/vstorage
Private Net (internal)
service2
Express (NodeJs)
Network: private
IP: 172.20.0.XX
Endpoints:
GET /status
storage
Express (NodeJs)
Network: private
Port: 8080
IP: 172.20.0.xx
Endpoints:
GET /log
POST /log
storage_data
type: persisted
docker volume

Analysis of Status Records

For each **/status** request in the system produces a line of text with two key measurements: uptime and free disk space.

Example of request and response

Request: *curl localhost:8199/status*

Response:

2025-09-29T11:49:26Z: uptime 0.00 hours, free disk in root: 922213 MBytes

2025-09-29T11:49:26Z: uptime 0.00 hours, free disk in root: 922213 MBytes

What we measured

Uptime: We are reporting how long the service process(container) has been running. It resets whenever the container restarts. That means it reflects container lifetime.

Free disk: We call statvfs("/") in python and fs.statfsSync("/") in Nodejs, so the measurement is the available space on the container's root filesystem.

We write the log with these two values alongside the timestamp.

Relevance

Container uptime is relevant if knowing how long the process remained healthy is a criterion. However, it misses failed restarts and provides no insight into the node's stability.

Root filesystem free space is only marginally relevant. It ignores the persistent volumes or bind mounts that hold the real data, so it might report “plenty of space” even if “/data/vstorage” or “/var/storage” is full.

What could be improved

1. Track the file system usages where the actual log live
2. Add counters for recent failures or the last restart time, rather than just logging uptime.
3. Consider exposing both container uptime and host uptime.
4. Record additional health signals, e.g: latency of storage calls, pending write errors etc.

Analysis of Storage Solution

There are two different persistence strategies in this stack:

1. **Storage service**: storage service writes to /var/storage/log.txt which is backed by the named Docker volume storage_data.
   Good: Volume is managed by Docker, survives container rebuilds, can live on remote storage and also isolate service data from the repo checkout.
   Less Good: It is harder to inspect/edit without docker run or docker volume commands, lifecycle is tied to Docker.

2. **Service1/ Service2**: both append to vstorage which is mounted(bind) to the literal file ./vstorage on the host.

   Good: Very simple to inspect, diff or edit from the host.

   Less Good: Careless clean or rm might wipes the data and bind mounts don’t translate well to clustered or cloud deployments.

Cleaning Up Instruction

Cleaing up can be done in two ways,

1. **Using the API**:

Calling the api ` localhost:8199/cleanup` in POST method would remove the logs from both volumes.

e.g.: ***curl -X POST localhost:8199/cleanup***

2. **Manual cleaning**:

Remove the docker volume with,

Docker compose down –v

And then run,

: > ./vstorage

This will remove the logs from vstorage(tested on linux)

Difficulties Faced

1. I faced some issues regarding writing the logs, it was inconsistent, and the reason was I was writing the logs parallelly.

Main Problems

1. Keeping services isolated and only making service1 accessible
2. I am new to NodeJs so I had some difficulties regarding that. But I learned a lot of things while implementing.
3. Calculating container uptime was a bit problematic because of my initial methodology. I was calculating host uptime instead of the container uptime, I was able to fix it by keeping the starting time in the memory.

Conclusion

Although I had previous knowledge about docker systems and the backend, I learned lots of things while implementing these microservices.

Overall, this was a very good learning for me.