COMP.SE.140 – Docker-compose an microservices hands-on

Platform information

Hardware and environment

- **Hardware/VM**: Apple Mac (local hardware)
- **Operating System**: macOS
- **Architecture**: ARM64 (Apple Silicon)

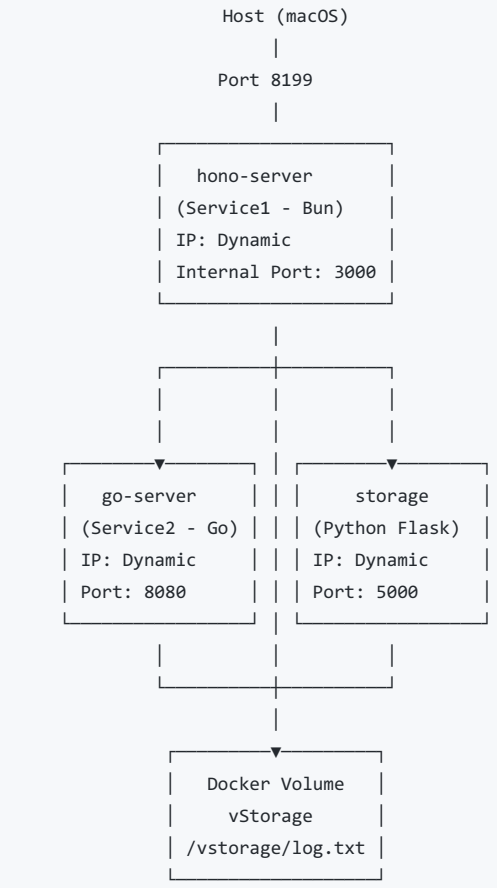
Software versions

- **Docker Version**: 27.4.0, build bde2b89
- **Docker Compose**: Using `docker compose` (integrated, not legacy `docker-compose`)

Project structure

```
comp-se-140-containers/
├── service1/          # Bun + Hono TypeScript service
├── service2/          # Go HTTP server
├── storage/           # Python Flask storage service
├── vstorage/          # Host-mounted shared storage
└── docker-compose.yml # Service orchestration
```

Architecture diagram




```
Network: comp-se-140-containers_default (Bridge)
- Network ID: c07d88f00d9918ecf0e1bfdddf0c66eae540d8f55e52ae09380787e66a310c8f
- Subnet: 172.20.0.0/16
- Gateway: 172.20.0.1
- Services communicate internally via service names (DNS resolution)
- Container IPs are dynamically assigned from the subnet range
- Only hono-server exposes external port 8199 which is internally mapped to port 3000
```

Status record analysis

What is measured

1. **Uptime Measurement**:
    - **hono-server (Bun/TypeScript)**: Uses `Bun.nanoseconds()` to measure process uptime since container start
    - **go-server (Go)**: Uses `time.Since(startTime)` where `startTime` is set when the Go application starts
    - Both measure container/process uptime, not host system uptime
    - Relevant for process uptime, doesn't provide info on the host itself
    - This is pretty ok, could be improved by also measuring the host uptime so we get info on possible restarts
2. **Disk Space Measurement**:
    - **hono-server**: Returns a hardcoded mock value of 9999 MB since Bun does not provide method for getting the hosts disk space
    - **go-server**: Uses `syscall.Statfs("/")` to get actual free disk space of the container's root filesystem
    - Not as relevant as the uptime due to the limits of Bun
    - This could be improved by using a different JavaScript runtime like Node or Dino. I was not aware of this constraint before implementing the software

Persistent storage solutions

This project implements two different storage approaches, each with distinct characteristics:

1. Host directory bind mount (`./vstorage:/vstorage/log.txt`)

Implementation: Services mount local `./vstorage` directory and write directly to shared file.

Advantages: Direct host access, simple backup, persistent data, development-friendly

Disadvantages: Race conditions, platform dependency, no file coordination, security risks

2. Storage service with snternal container Storage

Implementation: Flask-based Python service with REST API endpoints (GET/POST `/log`) for centralized log management.

Advantages: Clean API interface, centralized control, serialized access, service isolation

Disadvantages: Network dependency, single point of failure, ephemeral storage, network overhead

Comparison and recommendations

Aspect	Host Bind Mount	Storage Service
Persistence	High	Low (without volume)
Performance	Direct I/O	Network overhead
Concurrency Safety	No protection	Serialized access

Best Practice Recommendation:

1. Use the **Storage Service API** for write operations (centralized, safe)
2. Use **Docker Named Volumes** for actual storage persistence instead of mounting to a local file
3. Remove direct file access from other services to prevent race conditions

Teacher's instructions for cleaning up persistent storage

Complete environment cleanup

Stop all services and remove all persistent data:

```
 # Stop containers
docker compose down

# Remove the host-mounted directory and all logs
rm -rf ./vstorage
```

Notes for teachers

- Host bind mount data in `./vstorage` must be manually removed as it's outside Docker's management
- Usually this isn't the best practice and Docker named volumes are used instead. These can be cleaned with the `-v` -flag in the compose down command.
- The file `vstorage` needs to exists locally before composing or Docker will create a directory called `/vstorage` and cause an error.

Difficulties and main problems

- Understanding what was actually wanted for persistent Docker volumes. Having worked with volumes before, this approach seemed unintuitive
- It took a bit of time to correctly configure the mounts to work the way described in the instructions.