SERVICE ORIENTED ARCHITECTURES

ACIT3855 – WINTER 2024

AGENDA – LESSON 10

- Quick Review
- Quiz 9
- Topics:
  - Review
  - Externalized Configurations
  - Log Aggregation
  - Persistent Data in Docker
- Lab 10 – Central Configuration, Logging and Volumes

REVIEW

- Docker Logs
- Docker Volumes

QUIZ 9

- Online on D2L, Open Book
- Your have 15 minutes to complete it
- We’ll review any “problem” questions at the end

COURSE SCHEDULE

Week	Topics	Notes
1	• Services Based Architecture Overview • RESTful APIs Review	Lab 1
2	• Microservices Overview • Edge Service	Lab 2, Quiz 1
3	• Database Per Service • Storage Service (SQLite)	Lab 3, Quiz 2
4	• Logging, Debugging and Configuration • Storage Service (MySQL)	Lab 4, Quiz 3
5	• RESTful API Specification (OpenAPI) • Processing Service	Lab 5, Quiz 4
6	• Synchronous vs Asynchronous Communication • Message Broker Setup, Messaging and Event Sourcing	Lab 6, Quiz 5
7	• Deployment - Containerization of Services *Note: At home lab for Monday Set*	Lab 7, Quiz 6 (Sets A and B) , Assignment 1 Due
8	• Midterm Week	Midterm Review Quiz
9	• Dashboard UI and CORS	Lab 8, Quiz 7
10	• Spring Break	No Class
11	• Issues and Technical Debt	Lab 9, Quiz 8
12	• Deployment – Centralized Configuration and Logging	Lab 10, Quiz 9
13	• Deployment – Load Balancing and Scaling *Note: At home lab for Monday Set*	Lab 11, Quiz 10 (Sets A and B)
14	• Final Exam Preview	Quiz 10 (Set C), Assignment 2 Due
15	• Final Exam	

REVIEW

Now that we've built and deployed all our services…

- What do you think of microservices development? What's the good and the bad?
- What do you think of containerization of microservices? What are the benefits? What is hard about it?
- What could we do to improve it?

TECHNICAL DEBT AND ISSUES - REVIEW

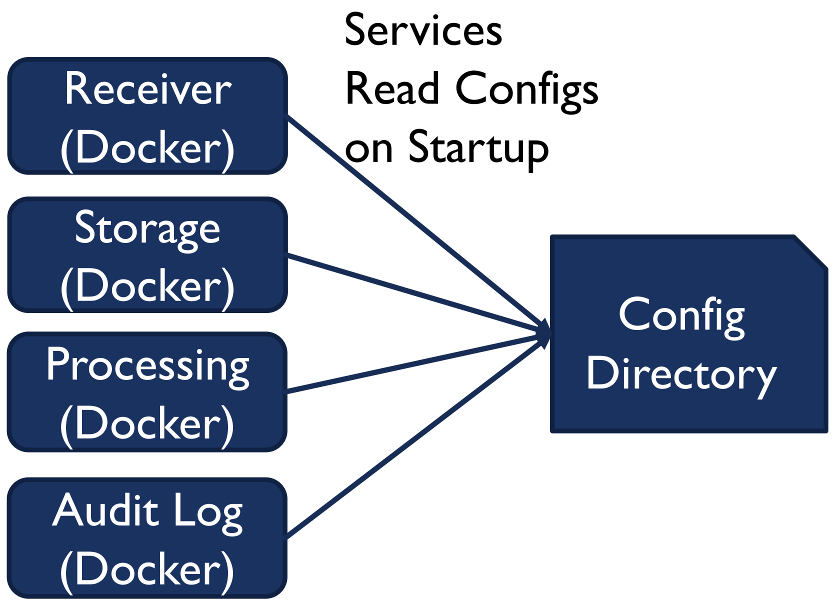
Last lab you addressed issues and technical debt we accumulated in our software…

- What are Software Issues?
- What is Technical Debt?
- What are some examples of Software Issues and Technical Debt in our system so far?

EXTERNALIZED CONFIGURATIONS

- We have already created external configuration files – app_conf.yml, log_conf.yml
- But they are built into are Docker images, so it’s hard to modify them – you need to rebuild the image
- So we also need to externalize them from the Docker container
- We will put them in a separate directory on the Host VM (i.e., the Cloud VM) and access them from our containers

EXTERNALIZED CONFIGURATIONS



Files on Host VM or Network:
/config/receiver/app_conf.yml
/config/receiver/log_conf.yml
/config/storage/app_conf.yml
/config/processing/log_conf.yml
/config/processing/app_conf.yml
/config/audit_log/log_conf.yml
/config/audit_log/app_conf.yml

Looks like:
/config/app_conf.yml
/config/log_conf.yml

This could be a Git repo so we can track changes to the files

We'll be using a Docker **bind mount** for this.
This is where a directory on the host VM is mounted into a container

EXTERNALIZED CONFIGURATIONS

There are other ways we can provide configuration values to our services. For example:

- Environment Variables
  - Docker: docker run -d -p 8080:8080 **-e ENV_VAR=value** receiver
  - Docker Compose:
    - Code them into the docker-compose.yml for each service (as per today's lab)
    - File - .env file (if exists) is loaded by default otherwise specify --env-file <filepath>

- Configuration Service – Infrastructure with a RESTful API that allows services to retrieve their configuration values.

EXTERNALIZED CONFIGURATION - SECURITY

What might be a security issue in our external configuration files?

- Database usernames and passwords

How can we deal with it?

No perfect solution. Some options:

- Environment variable so it's not stored on file. But this still can be visible if someone has access to the VM.
- Encrypt the password in the configuration file. Then the service needs to know how to encrypt it. But someone could look at the code, especially if it's an interpreted language like Python.
- Centralized keystore (i.e., Vault – another infrastructure services where services with the correct permissions can retrieve credentials). But there still has to be trust between the service and the keystore.

DOCKER LOGGING

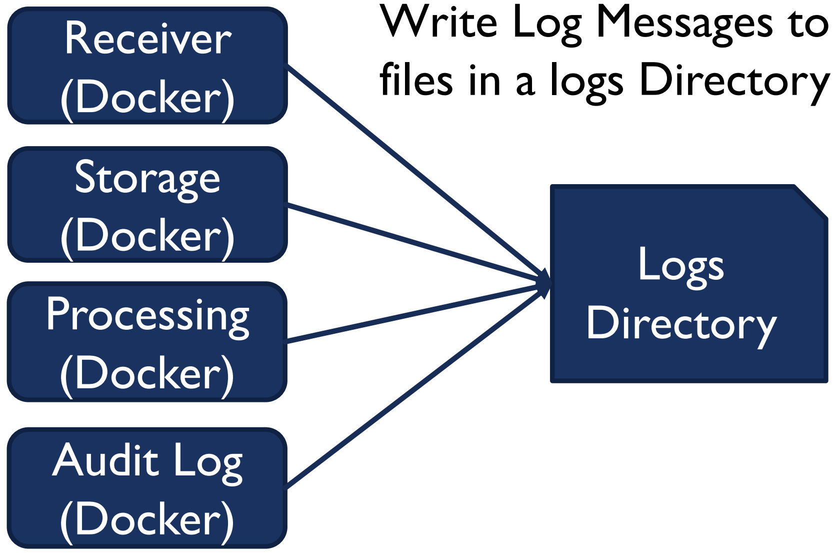
- We can see the logs written to stdout in the following ways:
  - Running our Docker containers in the foreground (shows up on the console)
  - Using the docker logs <container id> command (with an optional –f to follow)
  - Use docker-compose logs
- Or we can run a bash terminal on the container and view them directly in the app.log file:
  - docker exec –it <container id> bash

LOG AGGREGATION

- But if we want to monitor or analyze the logs, we need access to the actual files. Ideally in one location.

- So we want the log message written to a central place, such as:
  - A shared directory (i.e., on the Host VM or network)
  - A message queue
  - A log aggregation service

LOG AGGREGATION – SHARED DIRECTORY



Services
Write Log Messages to .log files in a logs Directory

Files on Host VM or Network:
/logs/app_receiver.log
/logs/app_storage.log
/logs/app_processing.log
/logs/app_audit_log.log

We'll be using a Docker **bind mount** for this.
This is where a directory on the host VM is mounted into a container

Looks like:
/logs
Writes to a file called:
App_<service name>.log

PERSISTENT DATA IN DOCKER

- A scalable service is one that allows us to have multiple instances of that service (i.e., container) running at the same time
- A fault tolerant service is one that can crash and be easily brought back up again and pick from where it left off
- In both those cases, we want any configurations and data used or updated by that service to be stored outside of the container in a central place where it can be:
  - Easily assessed by all running containers for that service
  - Shared by all running containers for that service
  - Backed up or maintained in a high-available manner in case of data loss or corruption

DATA PERSISTENCE IN DOCKER

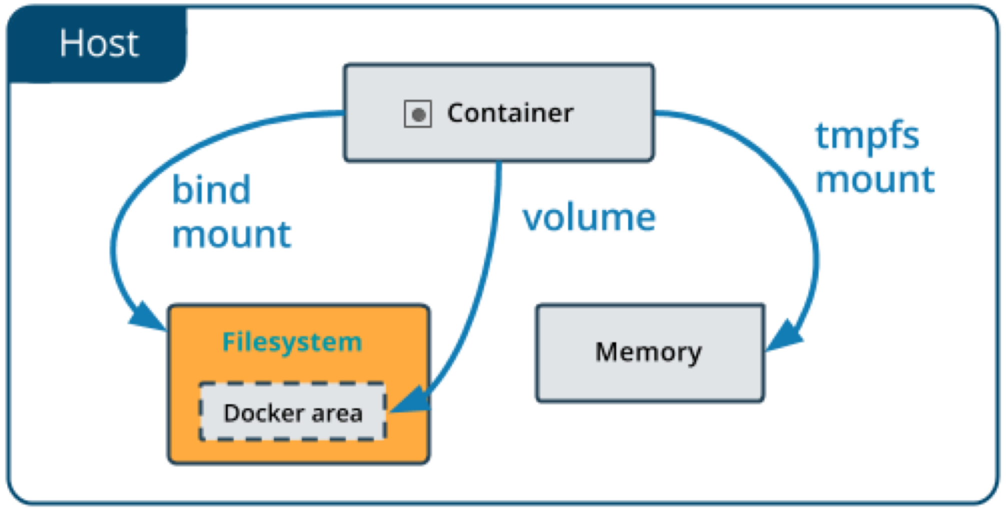
- We can persist application data in a database service, a caching service or message broker

- But sometimes we need to store data like:
  - Configurations (Externalized Configurations)
  - Logs (Aggregrated Logs)
  - File based data stores (i.e., like SQLite or JSON)
  - File based data (i.e., image files, Word Documents, PDF files, etc.)

- In Docker we have three options built-in: Bind Mounts, Volumes, tmpfs

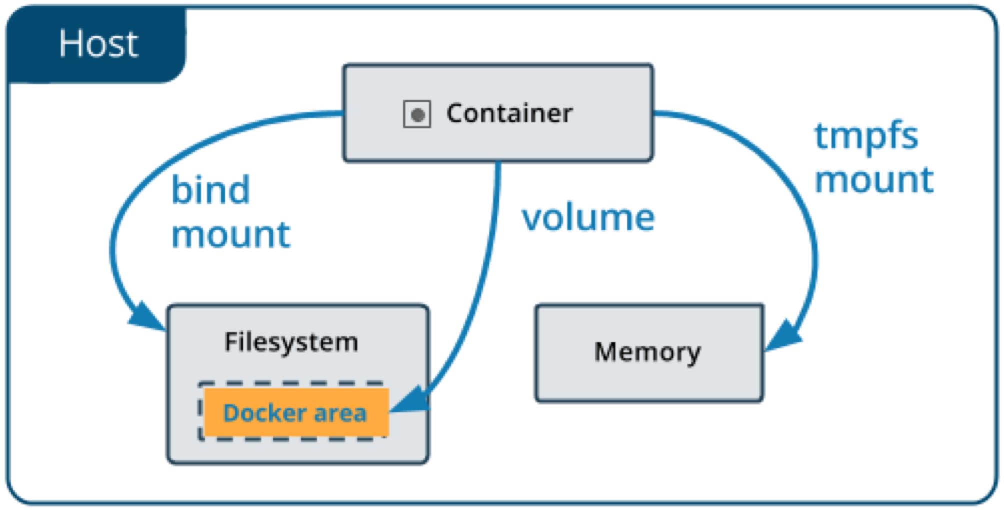
DOCKER BIND MOUNTS

- File or directory on the Host VM is mounted into a container
- Application accesses it like any other file or directory inside the container
- But it mapped to the Host VM
- Multiple containers can be mounted to the same file or directory (to share or aggregate data)



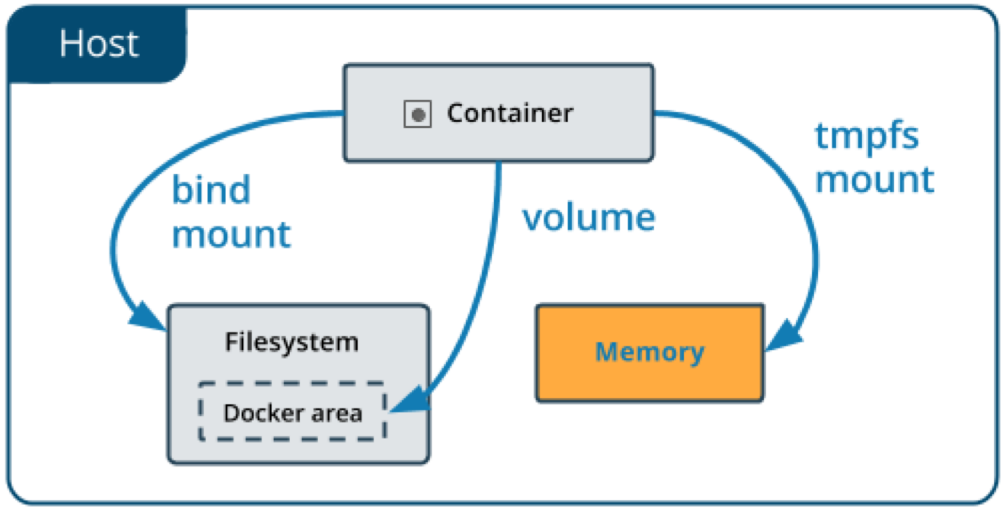
DOCKER VOLUMES

- Docker volume is mounted into a container
- Application accesses it like any other directory inside the container
- But it is actually mapped to a volume managed by Docker on the Host VM
- It is the preferred option since it is independent of the OS of the Host VM
- Can also:
  - Backup and migrate them
  - Use Docker commands to manage them
  - Mount the same volume to multiple containers (to share or aggregate data)
  - Etc.

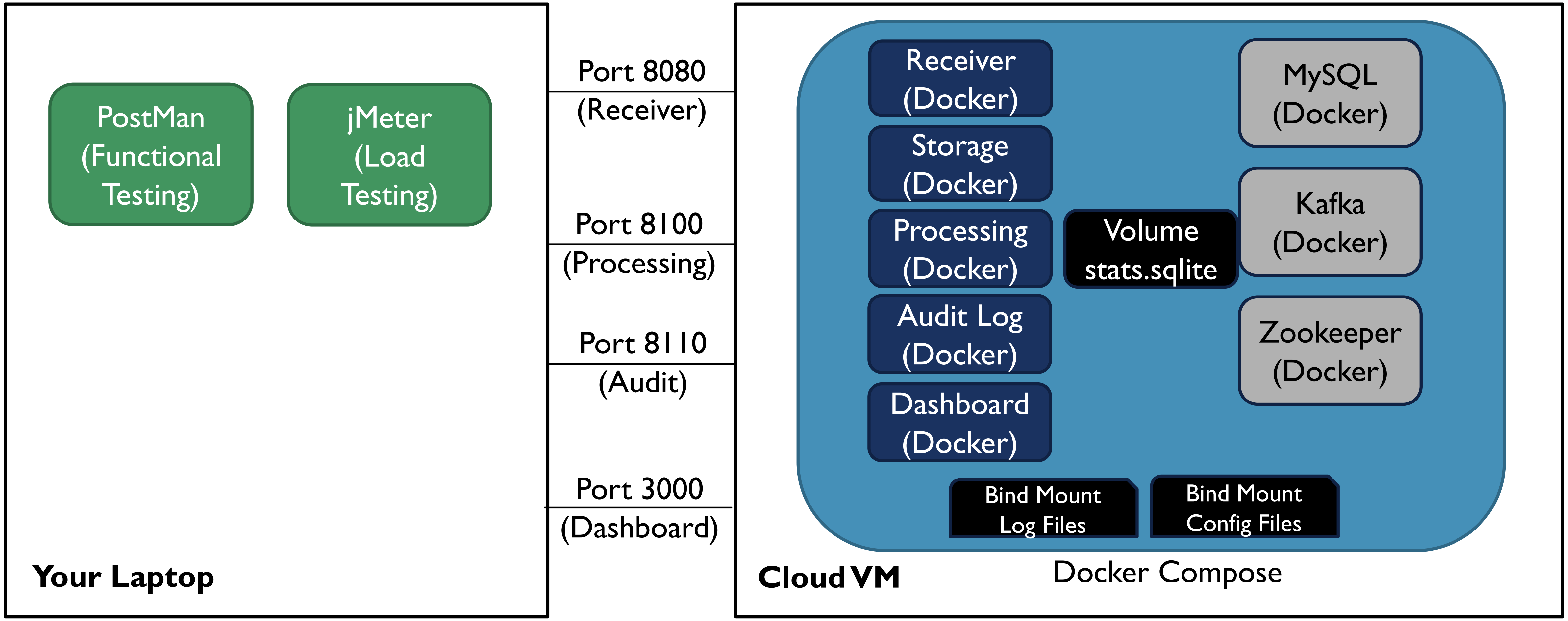


TMPFS MOUNTS

- Similar to Bind Mounts but mapped to memory on the Host VM, rather than a file or directory
- Used for temporary storage only
- Can’t be shared between containers



LAB 10 TEST ENVIRONMENT



DEMO – LAB 10

- There are three parts to the lab:
  - Externalized Configurations (Bind Mount)
  - Log Aggregation (Bind Mount)
  - Persistent Storage (Volume) for Processing Service

TODAY’S LAB

Today you will:

- Demo Lab 9 by the end of class
- Work on Lab 10 – It is due for Demo next class.