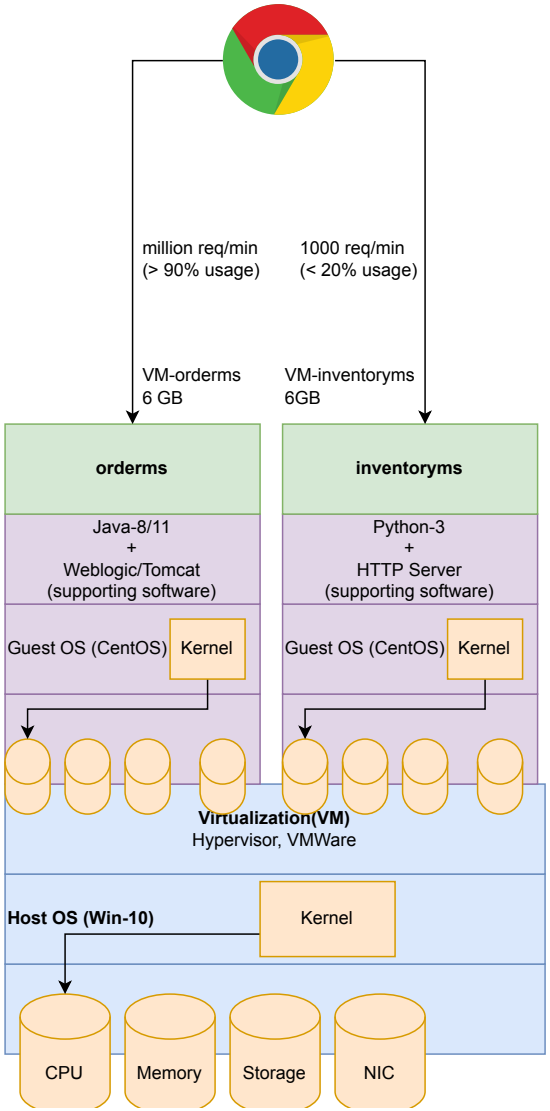
VM vs Docker

Issues With VM:

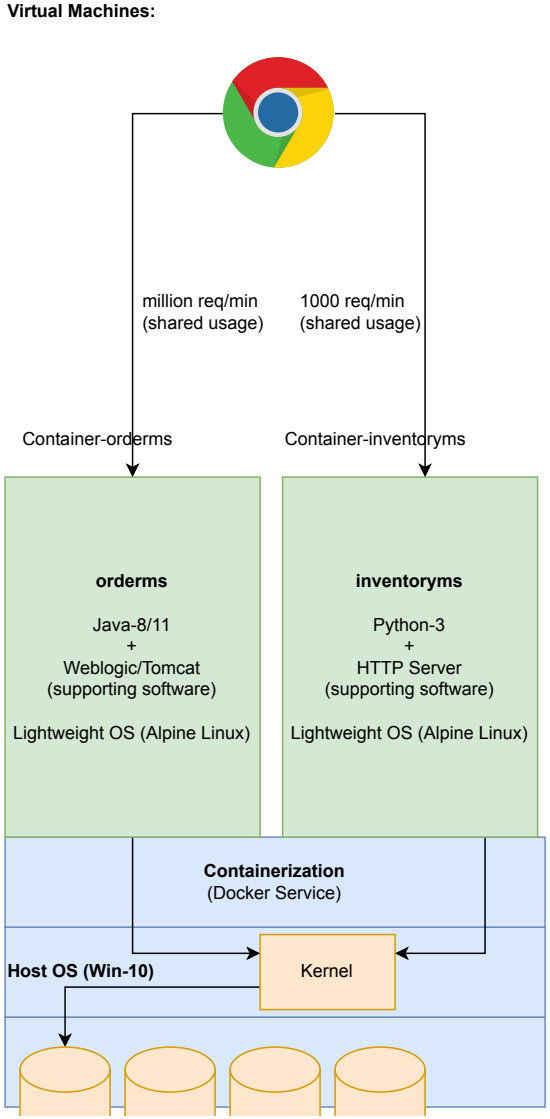
1. Resource sharing not possible
2. Usage of resource is not optimal
3. Software version mismatch
4. Longer startup time

Virtual Machine:
orderms: http://1.1.1.1:8082/orders
inventoryms: http://1.1.1.1:8083/inventories

Virtual Machines:



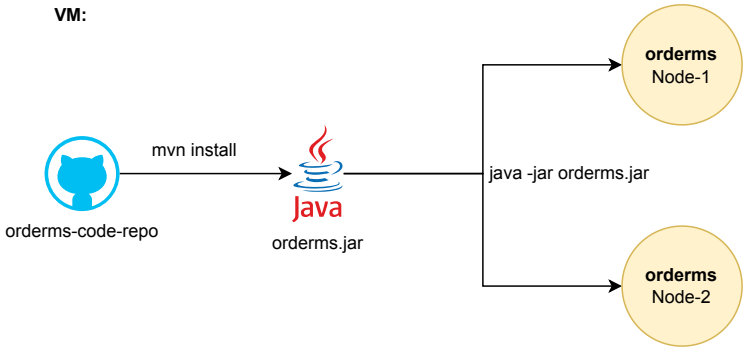
Virtual Machines:



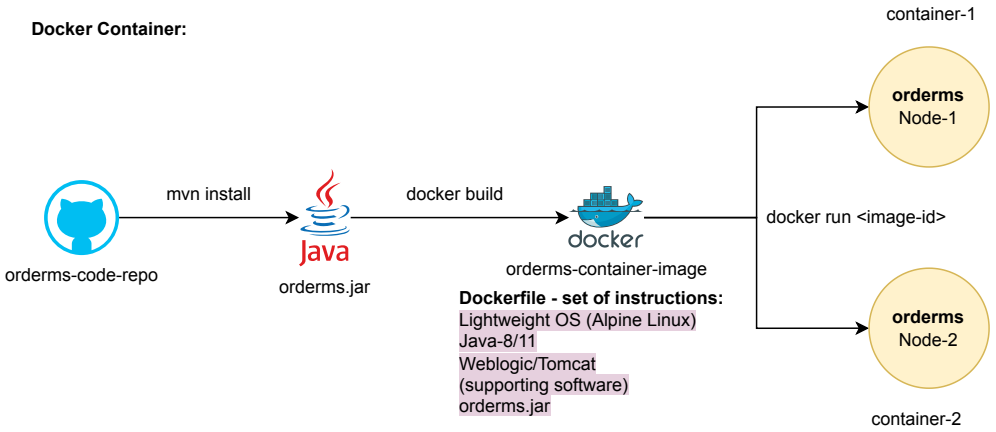
16 GB

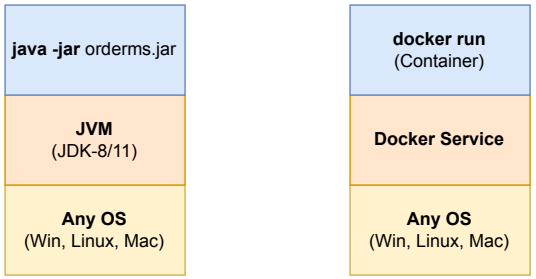
16 GB

VM:

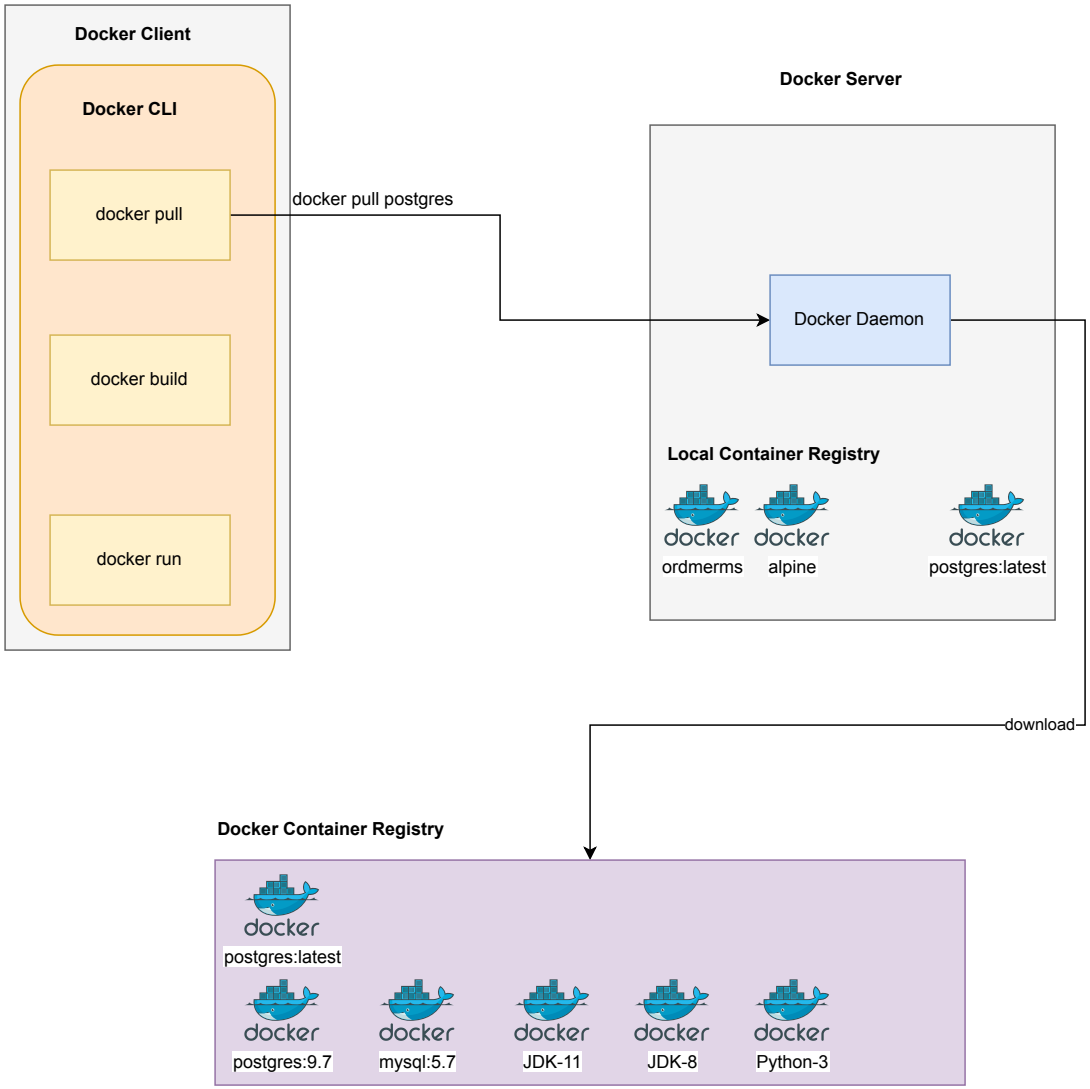


Docker Container:





Docker (Client-Server)

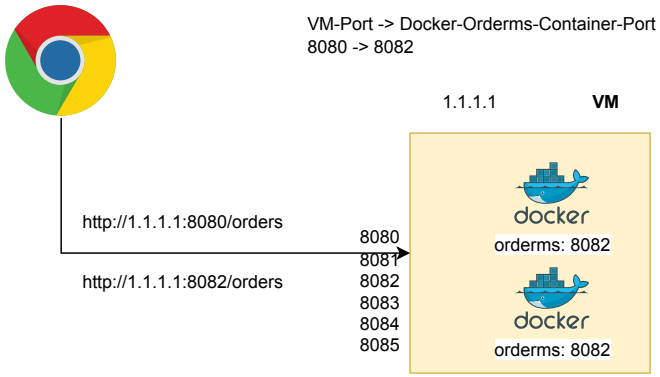


Port Forwarding

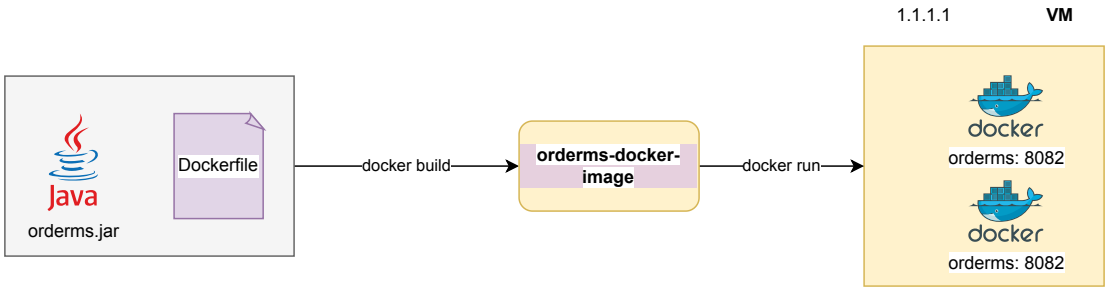
docker run -p **8082:8082** <orderms-image>

docker run -p **8080:8082** <orderms-image>

VM-Port -> Docker-Orderms-Container-Port
8080 -> 8082

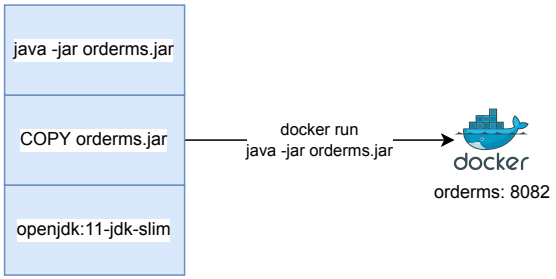


Dockerfile

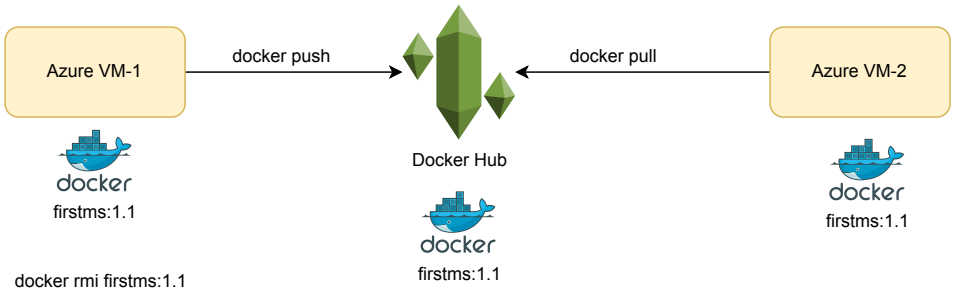


Dockerfile:

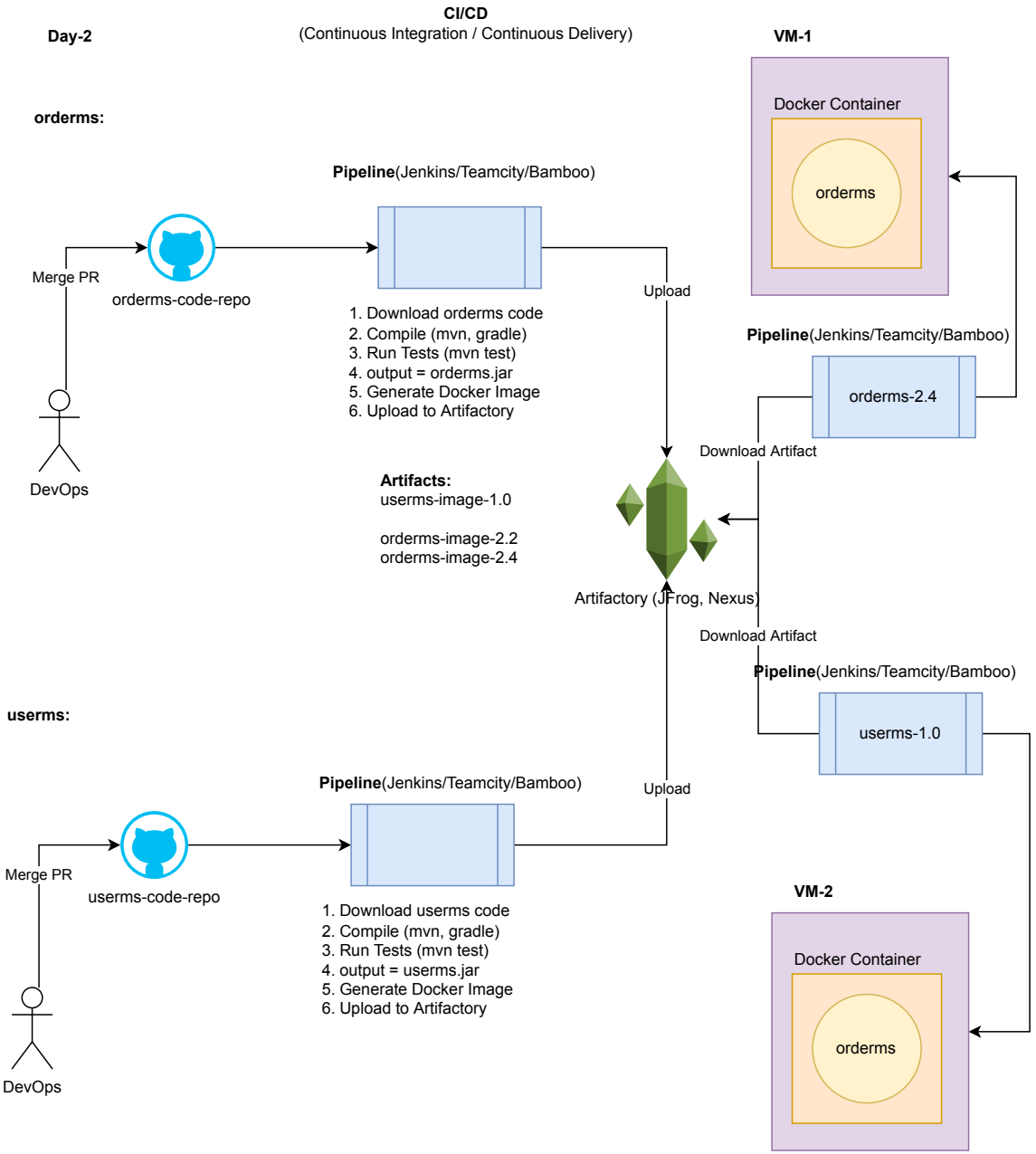
FROM openjdk:11-jdk-slim
COPY orderms.jar orderms.jar
ENTRYPOINT ["java", "-jar", "/orderms.jar"]



Push/Pull from Docker Registry



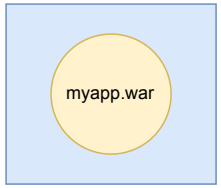
docker rmi firstms:1.1


Day-2
CI/CD
(Continuous Integration / Continuous Delivery)
VM-1
orderms:
Docker Container
orderms
Pipeline(Jenkins/Teamcity/Bamboo)
Merge PR
orderms-code-repo
1. Download orderms code
2. Compile (mvn, gradle)
3. Run Tests (mvn test)
4. output = orderms.jar
5. Generate Docker Image
6. Upload to Artifactory
Upload
Pipeline(Jenkins/Teamcity/Bamboo)
orderms-2.4
Download Artifact
DevOps
Artifacts:
userms-image-1.0
orderms-image-2.2
orderms-image-2.4
Artifactory (JFrog, Nexus)
Download Artifact
Pipeline(Jenkins/Teamcity/Bamboo)
userms:
userms-1.0
Pipeline(Jenkins/Teamcity/Bamboo)
Upload
Merge PR
userms-code-repo
1. Download userms code
2. Compile (mvn, gradle)
3. Run Tests (mvn test)
4. output = userms.jar
5. Generate Docker Image
6. Upload to Artifactory
VM-2
Docker Container
orderms
DevOps

Day-3

Multistage Build

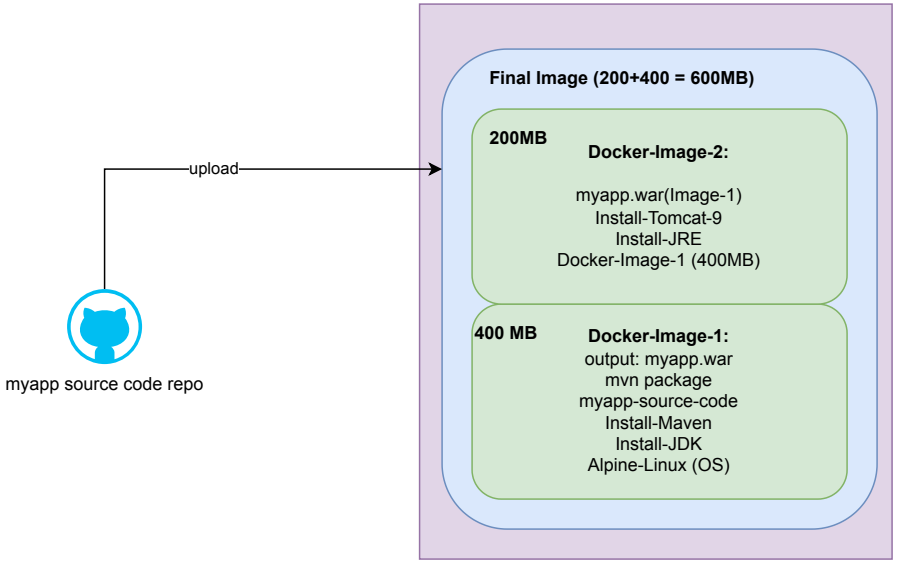
External Tomcat



Normal Build

ubuntu machine

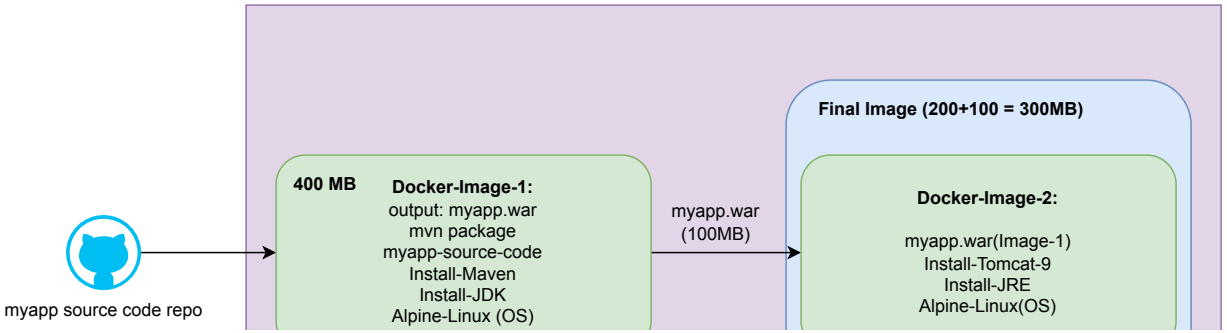
Huge Final Docker Image size



MultiStage Build

ubuntu machine

Small Final Docker Image

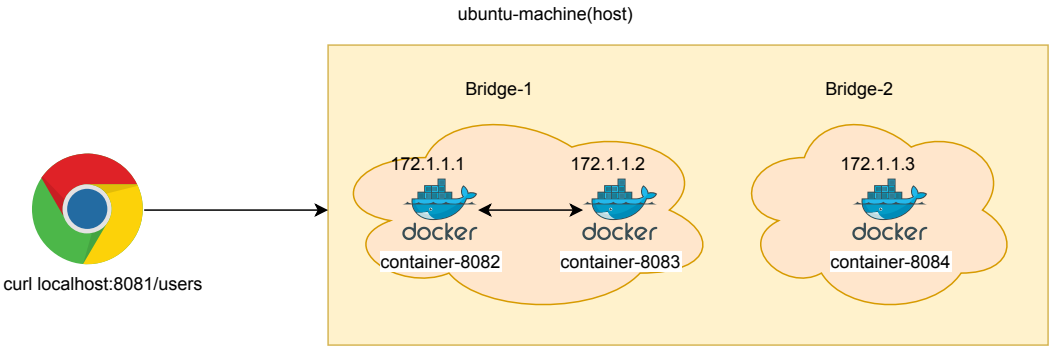


Docker Network

1. Bridge (default)
2. Host
3. Overlay
4. None

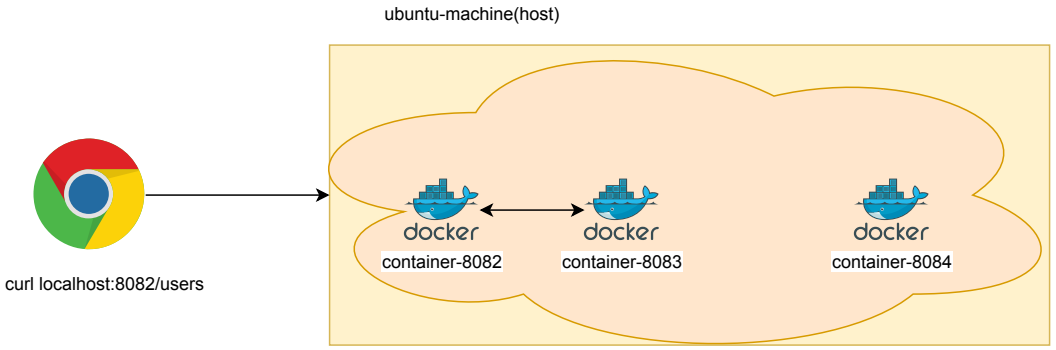
1. Bridge

(Single-House Networking): It's the default Network
docker run -p 8081:8082 my-image
curl localhost:8081/users



2. Host

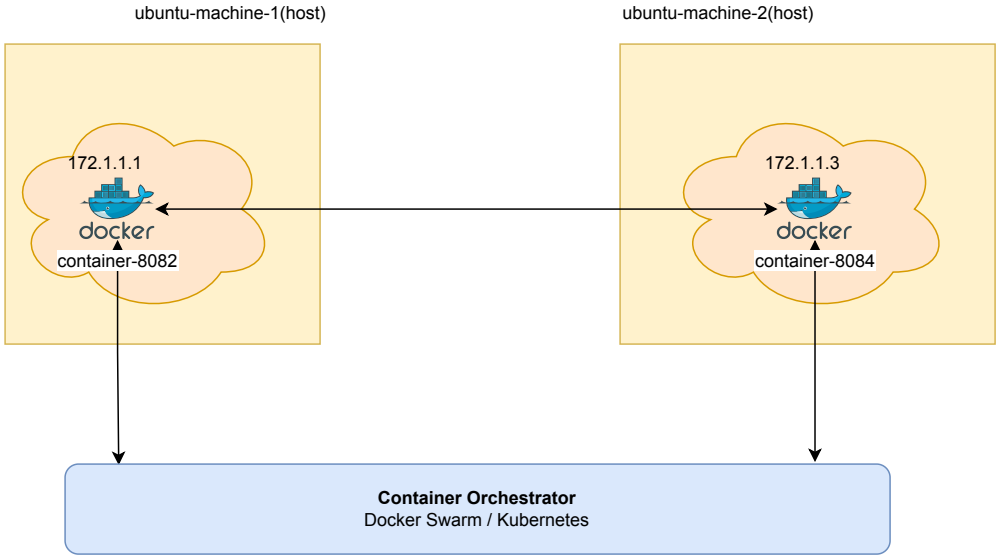
(No Port mapping required)
It uses host machine network/port
docker run --network host my-image
curl localhost:8082/users



Docker Network

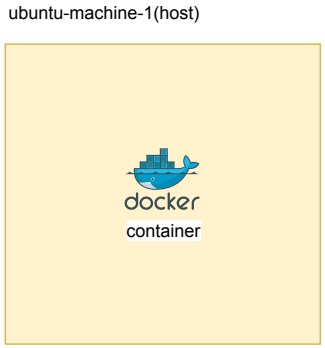
3. Overlay

docker run -p 8081:8082 --network my-overlay-network my-image
curl localhost:8081/users



4. None

It disables the Docker Network



Day-3

1. Build Docker image userms
2. Get postgres DB image
3. New network (custom bridge)
4. Run postgres container
5. attach postgres container to new n/w
6. Run userms container
7. attach userms container to new n/w
8. integrate userms with postgres

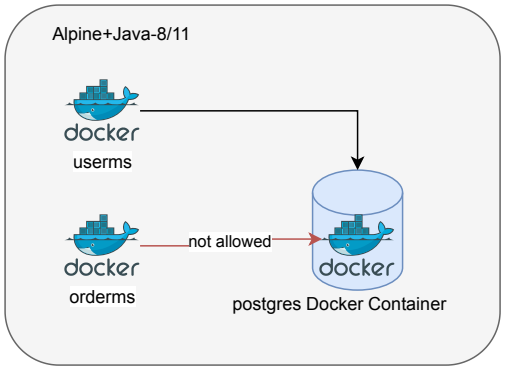
docker inspect <userms-container>

docker-compose.yml

docker-compose up

Docker-Compose

ubuntu machine:



server.port -> SERVER_PORT
spring.application.name -> SPRING_APPLICATION_NAME
spring.datasource.url=jdbc...

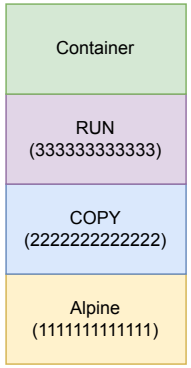
docker-compose.yml

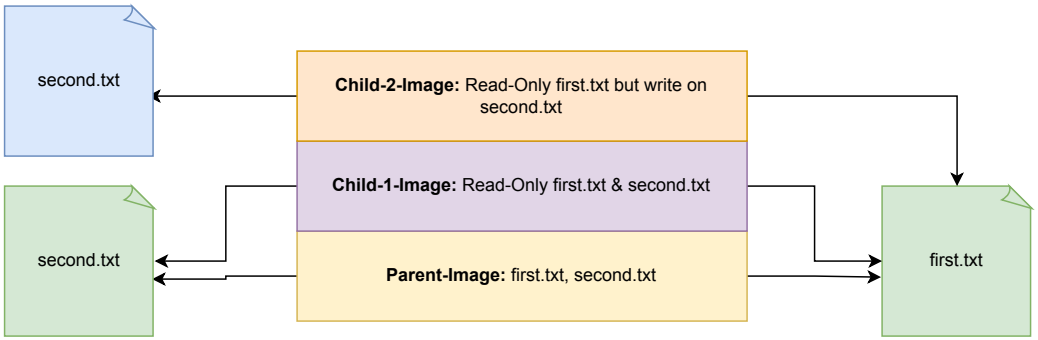
```
version: '3'
networks:
    my-network

services:
    db:
        image: postgres
        environment:
            - POSTGRES_PASSWORD=1234
            - POSTGRES_DB=userdb
        volumes:
            - ./db:/var/lib/postgresql/data
        ports:
            - 5432:5432
        networks:
            - my-network

    userms:
        image: userms-image
        environment:
            - SPRING_DATASOURCE_URL=jdbc:.....

        build:
            context: .
            dockerfile: my.Dockerfile
        ports:
            - 8080:8080
        networks:
            - my-network
        depends_on:
            - db
```

CoW
(Copy-on-write)

Docker Image Layering:

Dockerfile:
FROM alpine:latest
LABEL author=john doe
COPY src /app
RUN rm -r $HOME/cache
ENTRYPOINT ["start"]





Docker Volumes

1. tmpfs
2. Bind Mount
3. Named/Anonymous Volume