1 Virtualization – Motivation

Motivation

- Sharing of Computers
  - mainframes – share powerful/expensive server hardware
  - network computing – run cheap user machines
- Running Different OS – required OS, multiple OSs
- Simulate Different HW – no need to have specific hardware
- Simplify deployment – unified environment!!!
- Clouds
  - give up skills to cloud provider, who have the specialists, security certificates...

Types of Virtualization

- Virtual Machines, Hardware Virtualization
  - Native virtualization: z/VM (OS360, mainframes), KVM, VMWare ESXi, Xen
  - Hosted: VMWare Server/Workstation(Player), VirtualBox
- OS containers: categorized by level of isolation of user space
  - containers: (Docker, Podman)
  - jails: chroot (Linux) or jail (FreeBSD)
  - other: zones (Solaris), partitions, virtual environments, virtual kernels (DragonFly BSD)
- Desktop virtualization: next slide

Key Problem
Performance?

Desktop Virtualization

- Thin clients, network computing (Sun Microsystems, using ssh and X forward), diskless computers, booted from SFTP (my own reason to switch to Linux)
- Multiseat configuration (native in Unixes, Citrix, Win Server)
- Remote desktop (VNC, RDP)
- Recently – 3D games streaming

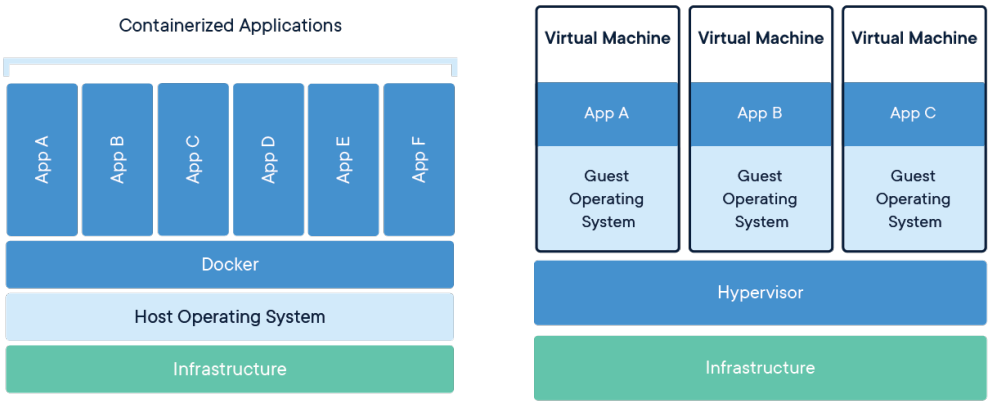

Figure 1: `www.docker.com/resources/what-container/`

Containers vs. Virtual Machines

Virtualization in Clouds

Cloud Services

- On-prem: software that is installed in the same building as your business.
- IaaS: cloud-based services, pay-as-you-go for services such as storage, networking, and virtualization.
- PaaS: hardware and software tools available over the internet.
- SaaS: software that is available via a third-party over the internet.
- Where is SpringBoot???
- SpringBoot – between IaaS and PaaS, requires Operating system, provides Middleware (built-in Tomcat)

Features

- Snapshots
- Migration
- Failover
- Licensing (e.g. Windows in VM requires license, Oracle licenses...)

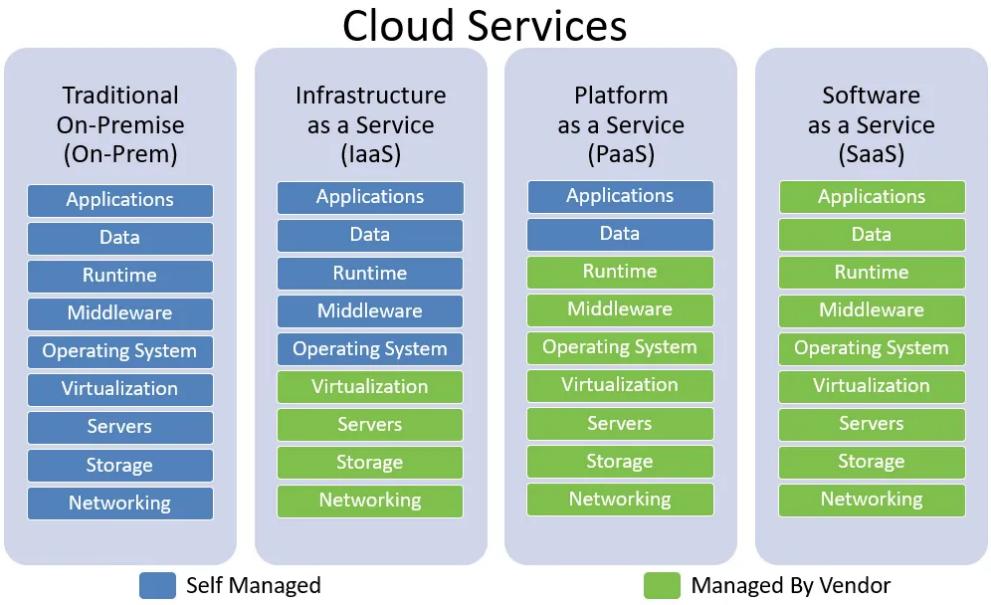

Figure 2: `medium.com/swlh/iaas-vs-paas-vs-saas-dfece8fd6ca`

2 Docker

Containers

- Rise of containers
  - fast, very little overhead
  - safe enough, separates users/customers
  - simple to configure
- Provide only fence between processes
  - share kernel
  - share memory
  - share CPU
  - process from inside doesn't see outside
  - virtualizes network, volumes (disk space)
- Docker became the most popular

Limits

- Memory limits
- CPU quotas

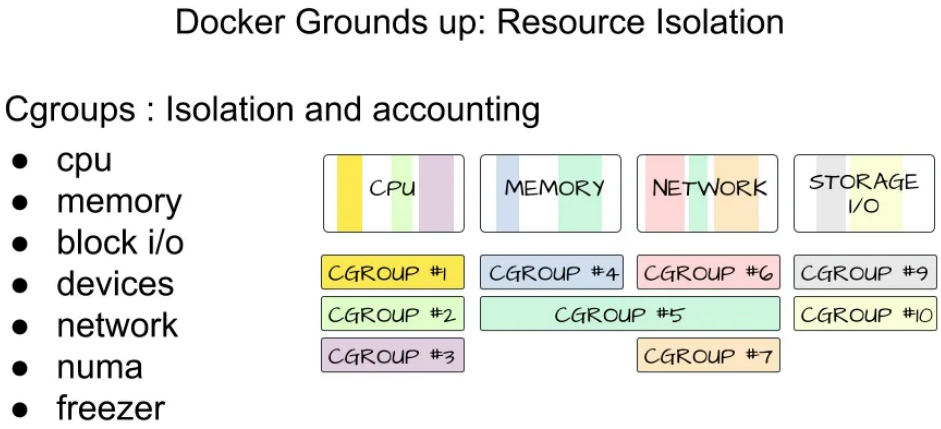

Figure 3: `medium.com/@BeNitinAgarwal/understanding-the-docker-internals-7ccb052ce9fe`

- Network isolation
- File system isolation (copy on write, disk quotas, I/O rate limits)
- Live migration
- Nested virtualization
- Based on Linux's cgroups

Docker: Resource Isolation

2.1 Basics

Basics

- image – read-only template
- container – encapsulated environment based on
- layers – transparent union filesystem
  - dependencies between images are layers (download independently)
  - line in Dockerfile is a layer (cache between rebuilds)
- registry – keeps list of named images
- volume – link to hosting filesystem

Docker: Image and Container Layers

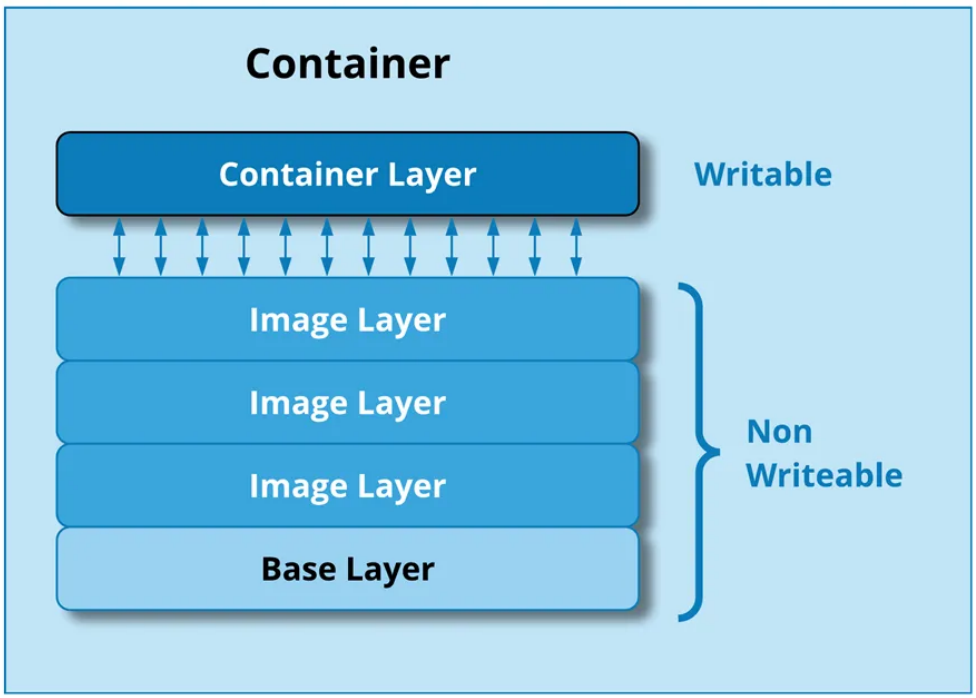

Figure 4: medium.com/@kuldeepkumawat195/understanding-docker-layered-architecture-06695

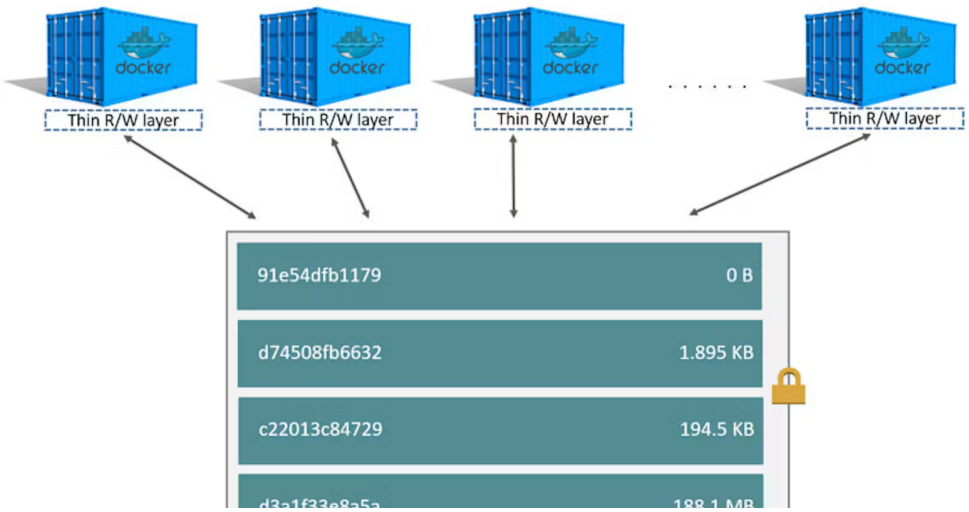

Figure 5: frenchymike.hashnode.dev/understand-the-docker-layers

2.2 Make Your Own Docker

SpringBoot/Docker Guide

- https://spring.io/guides/topicals/spring-boot-docker/

Docker Your App

```
FROM eclipse-temurin:17-jdk-alpine
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

- Dockerfile (default name of the image will be the directory name)
- Based on eclipse-temurin:17-jdk-alpine (https://hub.docker.com/layers/library/eclipse-temurin/17-jdk-alpine/images/sha256-595dfc1148baa2a6d632cfa7ec5c793191290957551be4a46dde6d6e6
- COPY copies file into image
- ENTRYPOINT – what is executed (container ends with this process)
- `docker build -t kbss/e-shop .`

Docker Your App

```
FROM eclipse-temurin:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["sh", "-c", "java ${JAVA_OPTS} -jar /app.jar"]
```

- Dockerfile (default name of the image will be the directory name)
- Based on eclipse-temurin:17-jdk-alpine (https://hub.docker.com/layers/library/eclipse-temurin/17-jdk-alpine/images/sha256-595dfc1148baa2a6d632cfa7ec5c793191290957551be4a46dde6d6e6
- VOLUME makes persistent storage
- ARG specifies argument with default value
- COPY copies file into image
- ENTRYPOINT – what is executed (container ends with this process)
- `docker build --build-arg JAR_FILE=target/*.jar -t kbss/e-shop .`

Build on a Clean Machine

- multi-stage build
- requires nothing on build machine, on front-end developer's machine
- copying data between containers

```
FROM maven:3.9.5-eclipse-temurin-17-alpine as build
WORKDIR /workspace/app
COPY pom.xml .
COPY src src

RUN mvn install

FROM eclipse-temurin:17-jdk-alpine
VOLUME /tmp
COPY --from=build /workspace/app/target/eshop-0.0.1.jar app.jar
ENTRYPOINT ["sh", "-c", "java ${JAVA_OPTS} -jar /app.jar"]
```

Few More Options

- `https://docs.docker.com/engine/reference/builder/`
- Security
  - USER – run as a different user
- why alpine
  - small
  - security – small attack vector, few CVEs!!!
  - but no ping, traceroute...
- docker repositories (company-local)
- CI/CD, jenkins, automatic upload to nexus, docker repo

Docker – Beyond One App

- docker compose
  - run several docker images together
  - provide configuration for run
  - share network
  - "Infrastructure as a code"
  - `docker-compose up`
  - `docker-compose up -d`
  - `docker-compose down`

Docker-compose

```
version: '3'
services:
  spring:
    image: kbss/e-shop
    restart: unless-stopped
    depends_on:
      - db
    ports:
      - "8080:8080" # web
  db:
    image: postgres
    restart: always
    volumes:
      - db:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=ear
      - POSTGRES_USER=ear
      - POSTGRES_PASSWORD=ear
    ports:
      - "8254:5432"
volumes:
    db:
```

Demo

- Build eshop
- Prepare Docker file
- multistage to build
- Prepare docker-compose with PG

Further Topics

- testcontainers
  - testing using Docker containers
  - perfect separation, complete environments
  - slower
- kubernetes
  - orchestarate many images on many nodes
  - yes, cloud

Real Deployment – Reverse Proxy

- Java application server is usually not publicly accessible
- Reverse proxy is used for performance and security

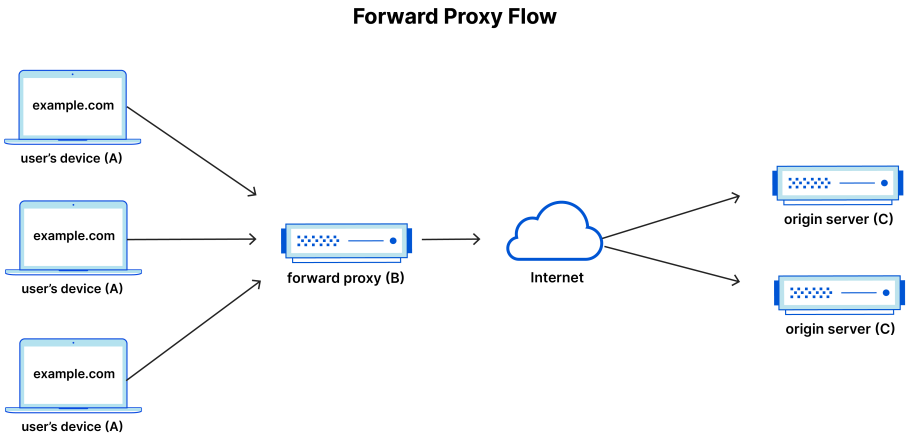

Figure 6: `https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/`

Forward Proxy

Reverse Proxy

Reverse Proxy

- **Performance**
  - web servers work faster than Java application server
  - perfect for static files (images, script)
  - faster SSL handling (ideally with HW support)
- **nginx/apache** – either one
- **certificates** – When separated computer, can have separate admin and customer's certificate.
- **security** – simple servers have less vulnerabilities

The End

Thank You

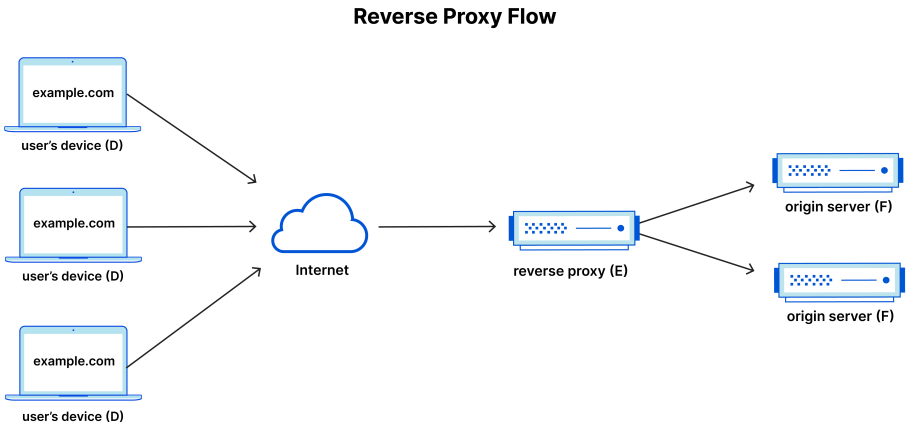

Figure 7: `https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/`

References

Dockerfile Documentation `https://docs.docker.com/engine/reference/builder/`

Spring Boot Docker `https://spring.io/guides/topicals/spring-boot-docker/`

Reverse Proxy `https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/`