Week 05 — Intro to YAML and Docker

Lukáš Grolig et al.

Outline

- YAML Ain't Markup Language
- Infrastructure
- Virtualization & Containerisation
- Container engine: Docker
- Orchestration: Compose

YAML Ain't Markup Language

- Valid extension `.yaml`
- YAML is case sensitive
- Uses spaces instead of tabs

```yaml
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

YAML: Data types

- **scalars** (strings, numbers, booleans)
- **sequences** (arrays / lists)
- **mappings** (hashes / dictionaries)

YAML: Scalars

```yaml
integer: 25
string: "25"
float: 25.0
boolean: true

foo: .inf
bar: -.Inf
plop: .NAN

foo: ~
bar: null

- true           # boolean
- "true"         # string, because it's quoted
- !!str true     # string, because of !!str
- !!bool "true" # boolean, because of !!bool
```

> Yaml is supporting single ' or double " quotation marks. Double quotation marks allow you to use escapings to represent ASCII and Unicode characters.

YAML: Sequences

Lists - single dimensional and multidimensional

```yaml
# One dimensional
- Cat
- Dog
- Goldfish

# Multidimensional
-
  - Cat
  - Dog
  - Goldfish
-
  - Python
  - Lion
  - Tiger
```

YAML: Mappings

```yaml
# Simple dictionary
animal: pets

# Mapping with sequence
units:
 - Footman
 - Grunt
 - Knight
 - Ogre

# Inline with sequence
npcflag: [GOSSIP, VENDOR]

# Complex type
? ["Tower", "Tier 1 Unit"]
: "Gremlin"

# Inline
foo: { thing1: huey, thing2: louie, thing3: dewey }
```

YAML: Sets

```yaml
set:
  ? item1
  ? item2
  ? item3
```

YAML: Special use-cases for mappings

```yaml
# List with mapping
-
  name: Mark McGwire
  points:   65
  grade:  "A"
-
  name: Sammy Sosa
  points:   63
  grade:  "B"

# Mapping of mapping
Mark McGwire: {hr: 65, avg: 0.278}
Sammy Sosa: {
  hr: 63,
  avg: 0.288
}
```

YAML: Refferences, Divider marker

```yaml
--- # Start of document
hr:
  - Mark McGwire
  # Following node labeled SS
  - &anchor Sammy Sosa
rbi:
  - *anchor # Subsequent occurrence
  - Ken Griffey
```

YAML: Complex example

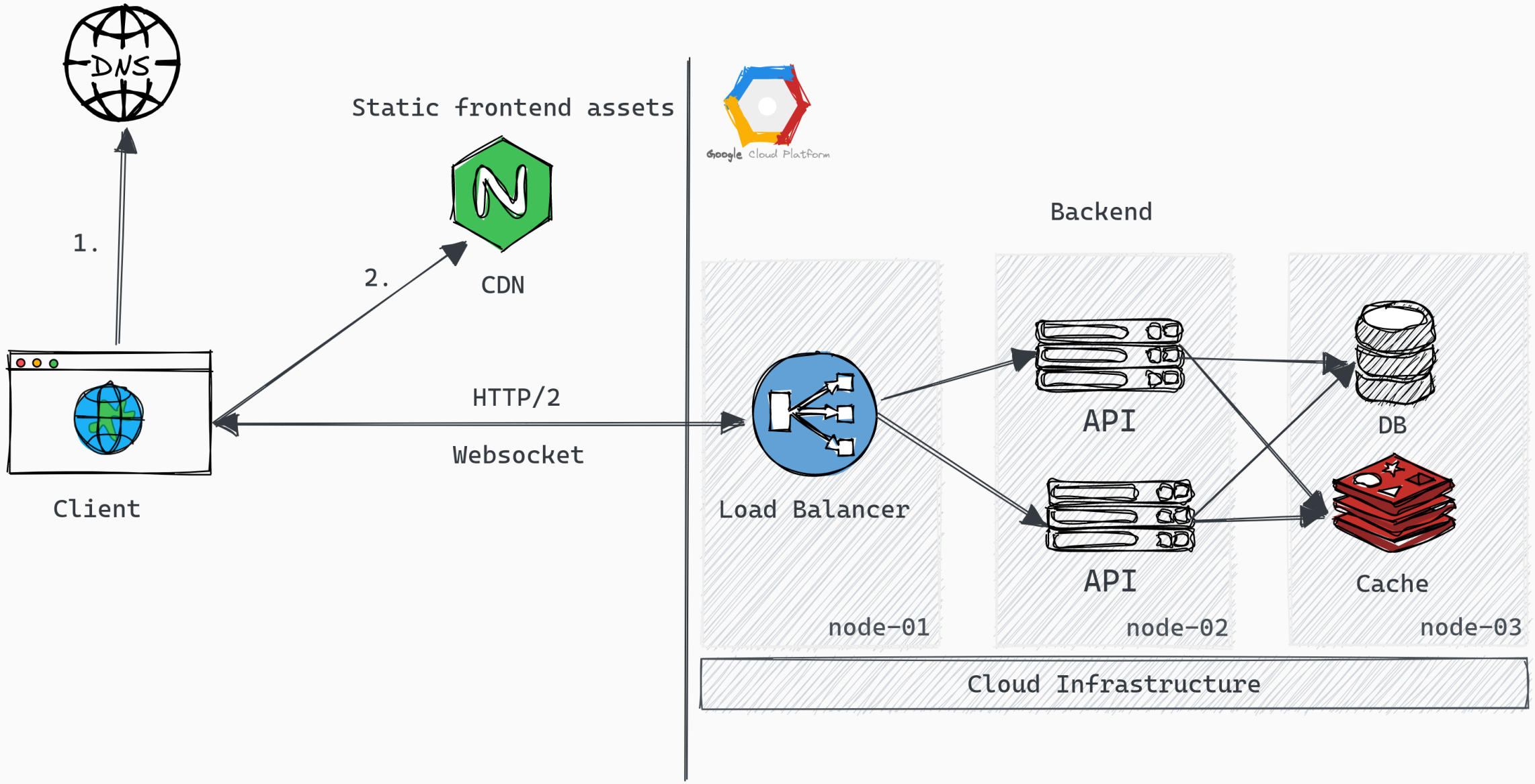
```yaml
---
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    given  : Chris
    family : Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
         city    : Royal Oak
        state   : MI
        postal  : 48046
ship-to: *id001
comments:
    Late afternoon is best.
    Backup contact is Nancy
    Billsmer @ 338-4338.
```

YAML: Typing

```yaml
- true           # boolean
- "true"         # string, because it's quoted
- !!str true     # string, because of !!str
- !!bool "true" # boolean, because of !!bool
```

Infrastructure

Infrastructure

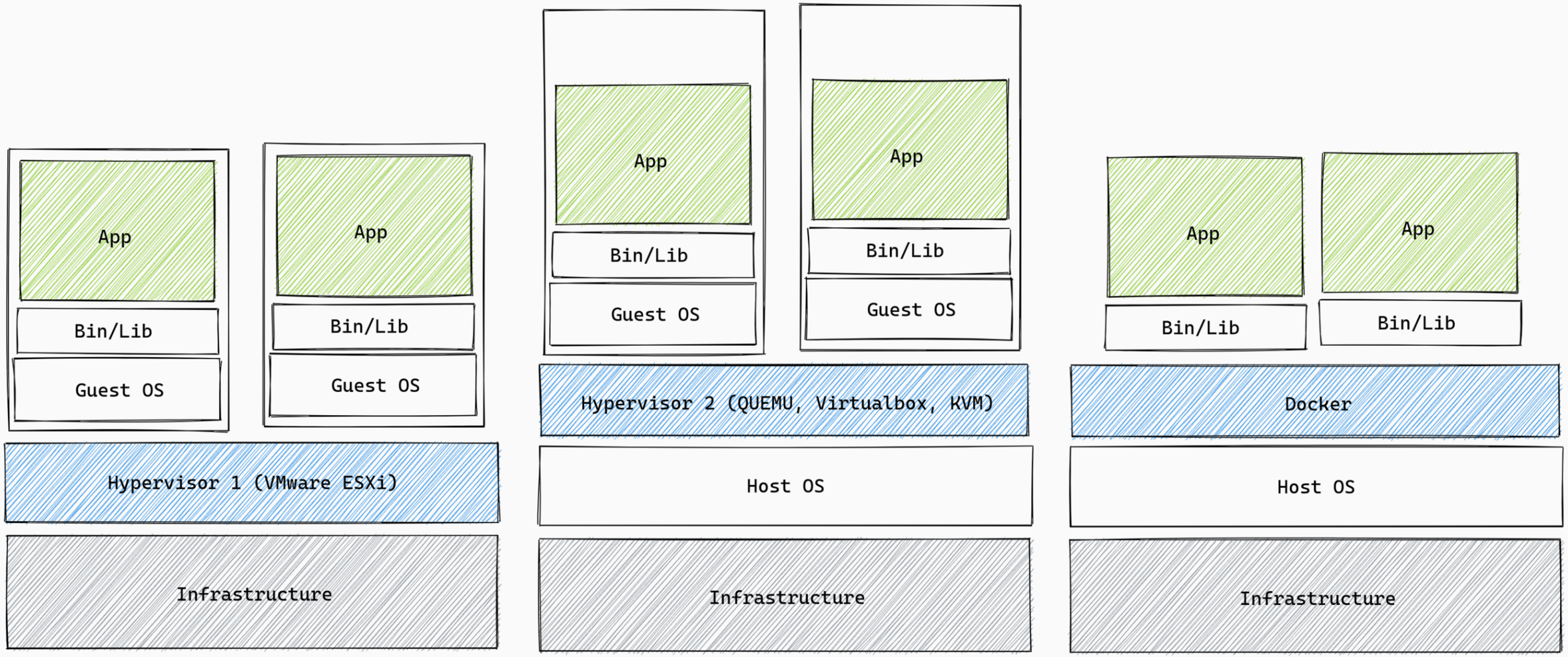


What is a server?

- Environment where your application lives
- Types of environments:
  - Baremetal
  - Virtual machine
  - **Container**
  - Serverless
- Baremetal, Virtual machines often use Stacks (LEM*, LAM**)
- **Environment is choosen by many aspects, there is no clear path to take**

```
* LEM (Linux, Nginx, MySQL) + PHP, NodeJS...
** LAM (Linux, Apache, MySQL) + PHP, NodeJS...
```

Virtualisation



Why containerisation matters?

- Own user space but share host's kernel - minimal and safe (if rootless)
- Uses a fraction of computing power compared to Virtual Machines
- Increased portability (Versioning, OS Independent, Architecture dependent)
- Consistent operation (Same environment on deployment machines as on dev machines)
- Rapid startup, deployment and scalling

> It works on my machine ¯_(ツ)_/¯
> – Developer

Container runtimes & engines

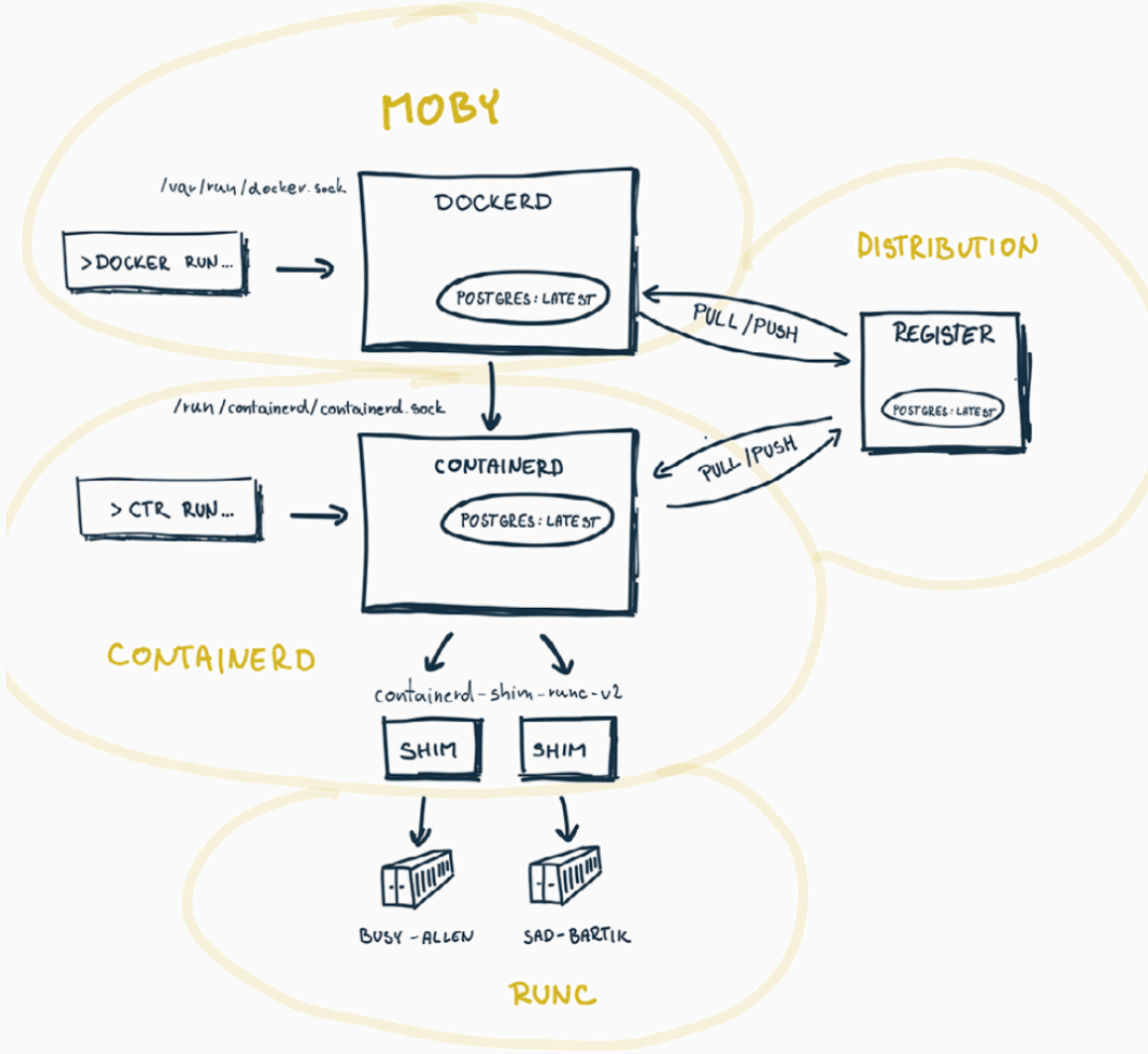
- Runs, Manages containers for operating system

Low level

- runC - runner (Go)
- youki - runner (Rust)
- containerd - daemon (Go)

High level

- **Docker**
- **Podman**
- CRI-O (Kubernetes)
- LXC



Key concepts

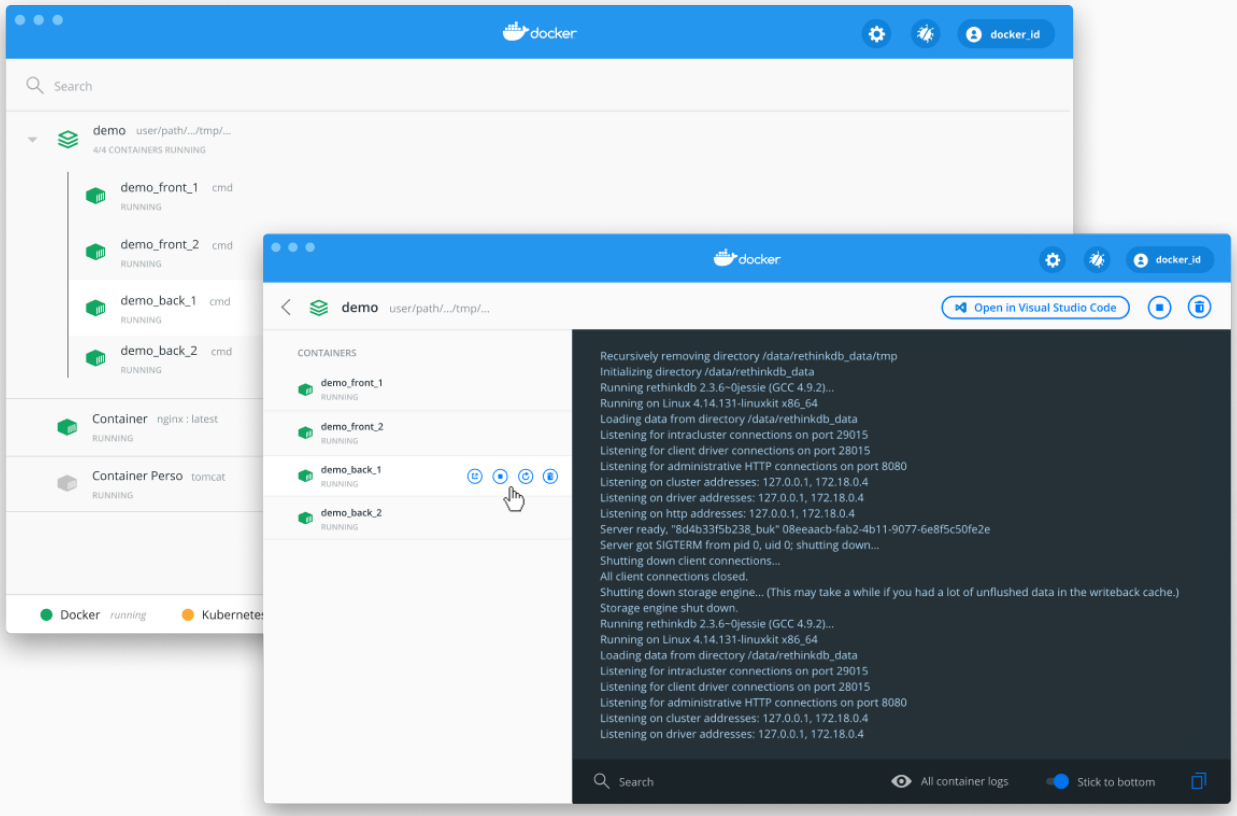
- **Host** - Place where containers run
- **Engine** - Docker, Podman
- **Containerfile** - Definition, set of instructions for Image (eg. Dockerfile)
- **Image** - Template for creating a container
- **Registry** - Storage for versioned images (eg. Dockerhub, Github, Gitlab)
- **Container** - Running image within user space

Installing the engine

- Docker desktop (User friendly: Win, Mac)
- Docker CLI[1]
- Podman[1]

The following slides depends on engine, but they share same concepts

podman



[1]Linux geek friendly

Running images

Image is either pulled from registry or kept locally

```
# Starts container in with interative session
docker run -it alpine sh
```

```
# Starts container in background and bind port to host
docker run -d -p 127.0.0.1:8888:5000 application:v1
```

Basic commands

```
# Shows currently running containers
docker ps
```

```
# List images or volumes
docker images or volumes
```

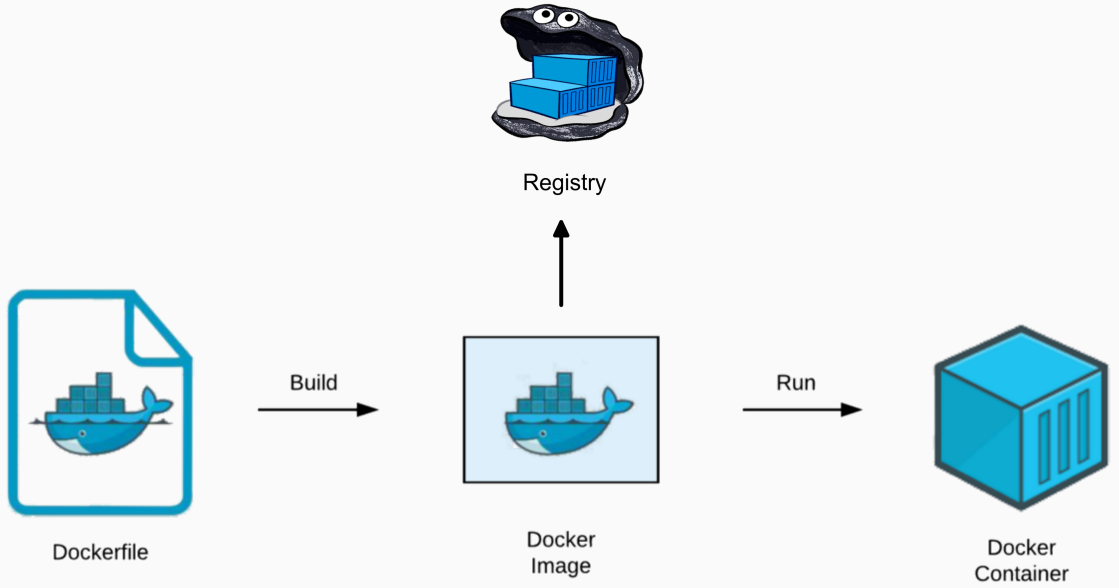
```
# Kills running container
docker kill <container-id>
```

```
# Removes stopped container
docker rm <container-id>
```

Creating own images

There are multiple builders for images: Buildah, Kaniko for creating OCI Images

1. Define Containerfile or Dockerfile
2. Build image & Tag image
3. (Optionally) Push image to registry

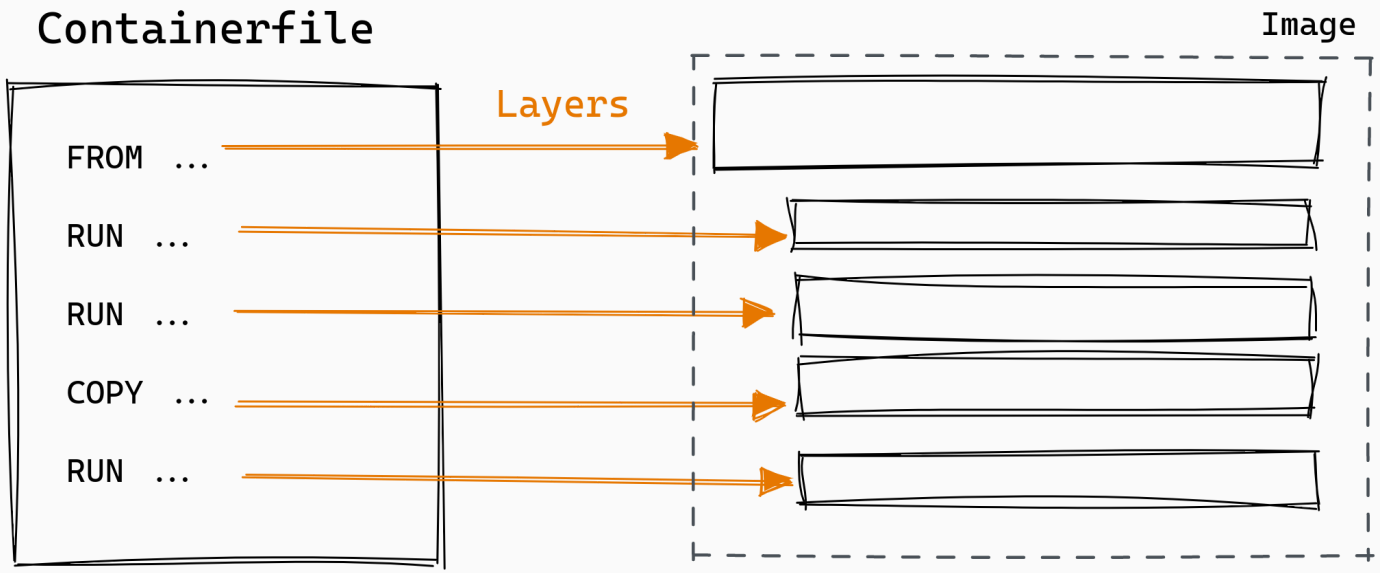


Containerfile

```
> cat Containerfile

# Define image base for the image
# This image has already node-js and npm installed
FROM node:16-alpine

# Create and change active working directory
WORKDIR /app

# Install runtime dependency into image
RUN apk add -u chromium

# Copy files from local filesystem to CWD
COPY hello.js .

# When running the container will expose port 8080
EXPOSE 8080

# Define the single process to run inside of container
CMD ["hello.js"]
```

Building images

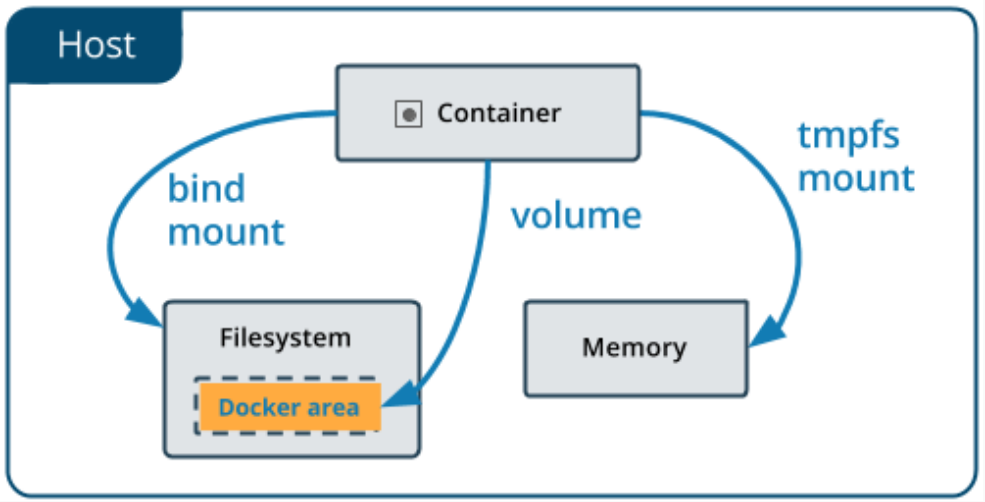
```
# Build image in current context (CWD)
# Specify tag - container name and version
docker build -f Containerfile -t ghcr.io/pb138/app:1.0.0 .
```



Volumes and mounting

During lifetime of container, files can be produced or accessed by container.

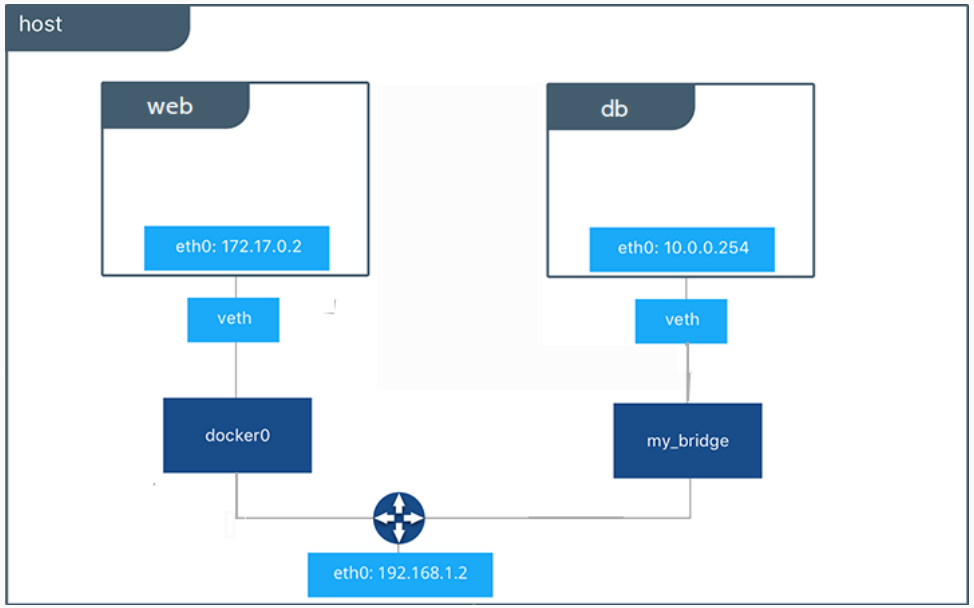
- Only files required to run the container should be the part of image
- Three types of mounts
  - Bind mount to host filesystem
  - Volume - Managed by Engine
  - tmpfs - Temporary filesystem mount



> eg. Store photos generated by container or uploaded by users

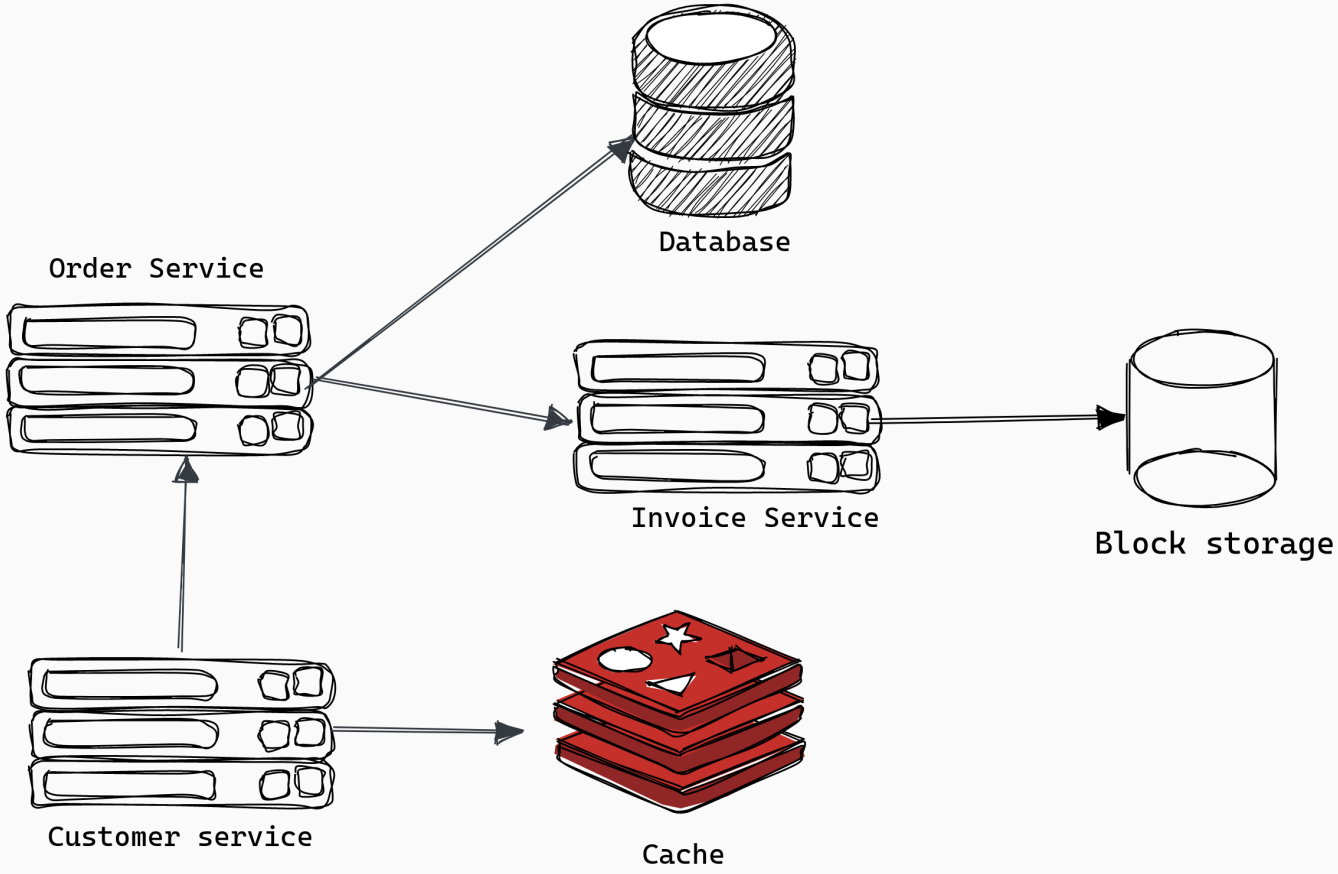
Networking and ports

- Under the hood it manipulates `iptables` rules
- Containers have their own network interfaces
- And following network drivers:
  - host (Removed isolation)
  - bridge (Default network driver)
  - overlay (Cross docker daemon communication)
  - ipvlan, macvlan (Allows to controll addressing)
  - none
- Networking of container has following functionallities:
  - Connect multiple containers within the network
  - Expose container ports to network interface



Rewind: What is application?

Multiple services (Database, Cache, Application runtime, Webserver, Storage)
Need for multiple containers -> container orchestrator



Compose

- Simplest orchestrator
- Used for local development and smaller environments (few servers - swarm)
- Defined in YAML
- Stored in `compose.yml`, `docker-compose.yml`

Composefile

- **Version** definition
- **Services** - Definition of services to orchestrate
- **Volumes** - Persistent volume definition
- **Secrets** - Sensitive data storage
- **Networks** - Networks used for containers to communicate
- **Configs** - Configuration files exposed to container

```yaml
services:
  frontend:
    image: awesome/webapp
    ports:
      - "443:8043"
    networks:
      - front-tier
      - back-tier
    configs:
      - httpd-config
    secrets:
      - server-certificate

  backend:
    image: awesome/database
    volumes:
      - db-data:/etc/data
    networks:
      - back-tier

volumes:
  db-data:
    driver: flocker
    driver_opts:
      size: "10GiB"

configs:
  httpd-config:
    external: true

secrets:
  server-certificate:
    external: true

networks:
  # The presence of these objects is sufficient to define them
  front-tier: {}
  back-tier: {}
```

```yaml
version: "3.7"

services:
  frontend:
    build: frontend
    ports:
      - 3000:3000
    volumes:
      - ./frontend:/usr/src/app
    networks:
      - frontend

  backend:
    build: backend
    volumes:
      - ./backend:/usr/src/app
    networks:
      - backend
      - frontend

networks:
  frontend:
  backend:
```

Compose commands

```
# Start services in background
docker-compose -f compose.yml up -d
```

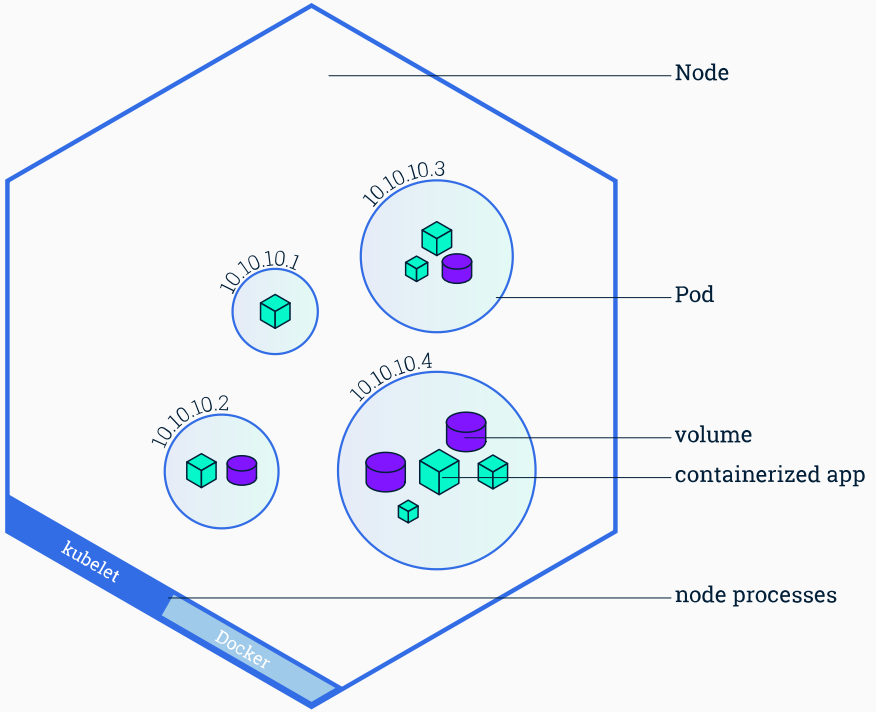
```
# Show logs and follow them
docker-compose -f compose.yml logs -f
```

Kubernetes: orchestration

- Used for large scale production environments, where scallability and isolation plays significant role in architecture.
- Provides:
  - Service discovery and load balancing
  - Self-healing
  - Storage orchestration

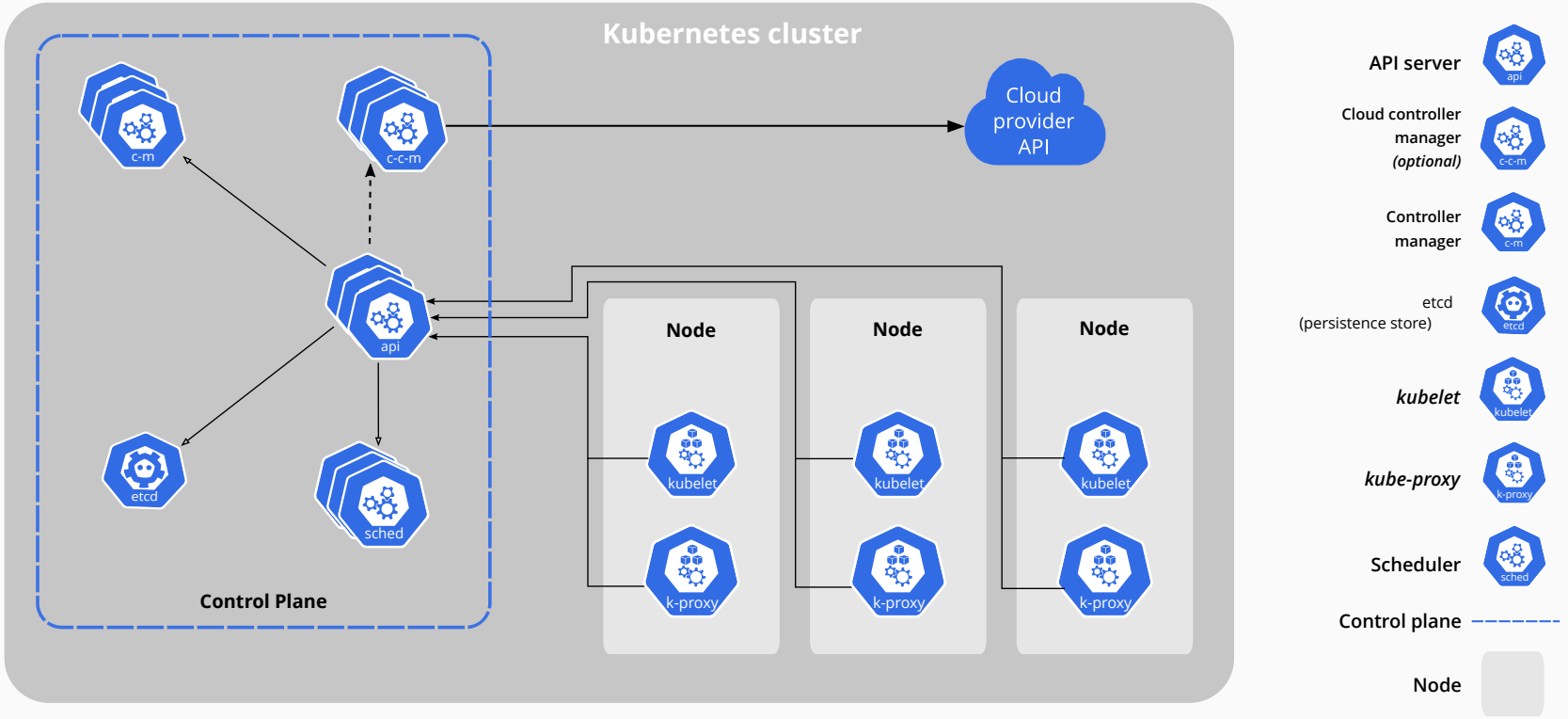
Kubernetes: concepts

- **Pod** - Group of cohesive containers (Smallest deployable unit, optionally, they can use shared volumes and network resources)
- **Service** - Group of pods providing functionallity to another group of pods (Service)
- **Namespace** - Isolation of resources withing a single cluster
- **Node** - Worker machine, where containers run
- **Cluster** - Group of nodes



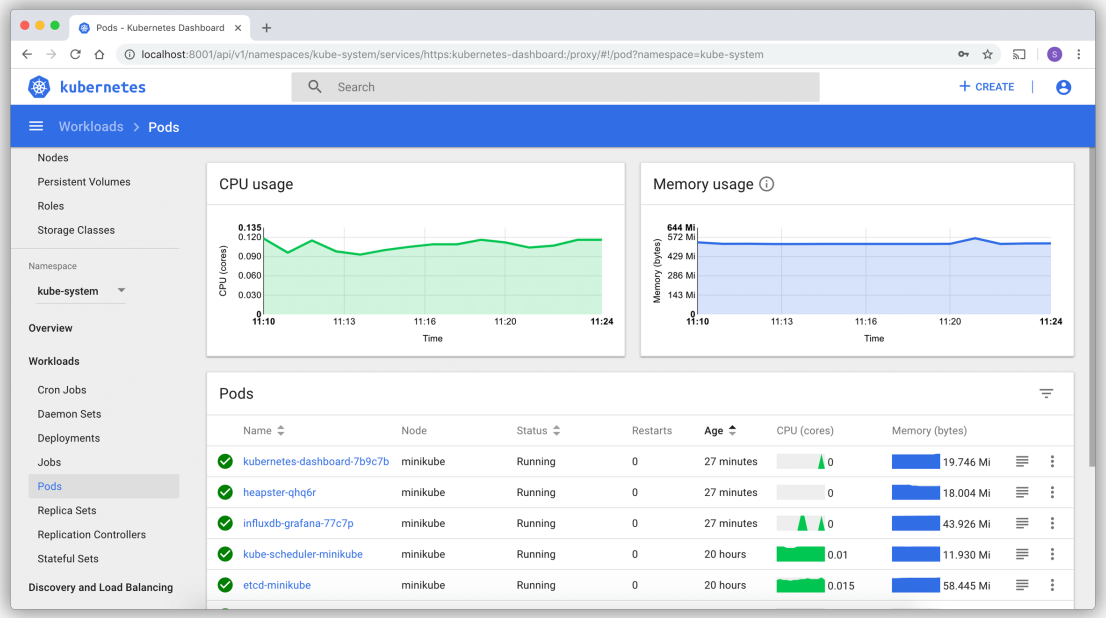
Kubernetes: components

Control plane is orchestration layer that exposes the API and interfaces to define, deploy, and manage the lifecycle of containers.



Kubernetes: management

- Web based user interface
- Provides basic troubleshooting, management of cluster
- Can be used to deploy, scale and restart applications



Courses on FI (Containerisation & Virtualisation)

- PB176 Základy kvality a správy kódu
- PV282 Designing and building infrastructure in public cloud

Resources

- https://github.com/containers/youki
- https://cri-o.io
- https://www.cncf.io
- https://linuxcontainers.org
- https://github.com/JaSei/docker_under_the_hood_talk
- https://blog.ttulka.com/containers-under-the-hood
- https://iximiuz.com/en/posts/you-need-containers-to-build-an-image
- https://okontajneroch.sk
- https://www.docker.com/resources/what-container
- https://medium.com/swlh/understand-dockerfile-dd11746ed183
- https://www.mankier.com/5/Containerfile
- https://docs.docker.com/engine/reference/builder
- https://github.com/compose-spec/compose-spec/blob/master/spec.md
- https://docs.docker.com/compose/compose-file/compose-file-v3/
- https://iximiuz.com/en/posts/containers-vs-pods/
- https://kubernetes.io/docs/concepts/