Introduction to Containers

December 2023

Agenda

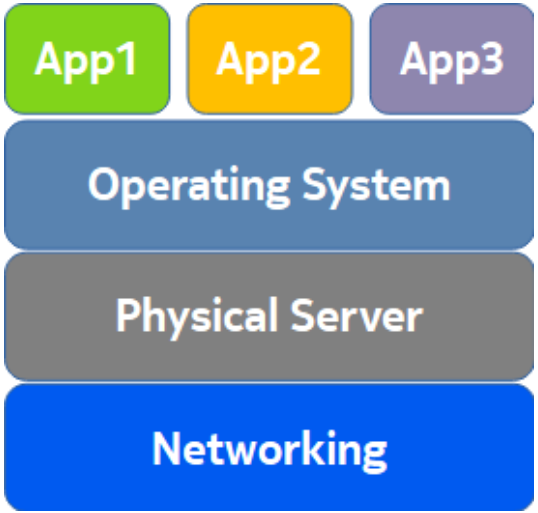
1. From bare metal to serverless
2. Container ecosystem glossary
3. Tools
4. Technology behind
5. Use-cases and demo

From bare metal to serverless

From bare metal to serverless

Bare metal

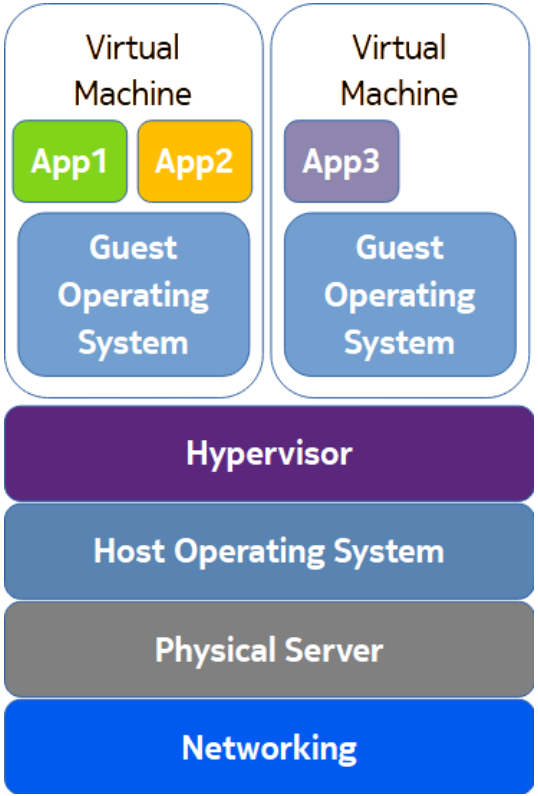
- Physical server
- Single tenant
- Maximal control
- Physical isolation
- Expensive
- Hard to manage
- Hard to scale
- Optionally bare metal as a service



From bare metal to serverless

Virtualized

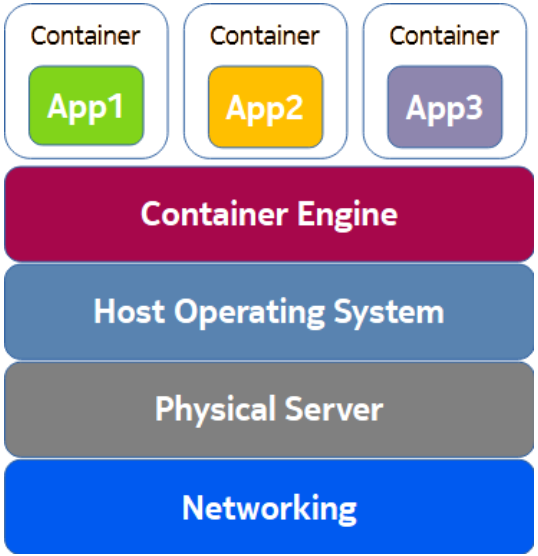
- Emulation of physical computer
  - Abstraction layer over the hardware
- We can select VM size (CPU, RAM, Storage,...)
- Cheaper to run
- Share the same hardware
- Better resource utilization
- Vertical/horizontal scaling
  - Migrate without VM shutdown
- Vulnerable to noisy neighbour problem
- Side-channel attacks like spectre and meltdown



From bare metal to serverless

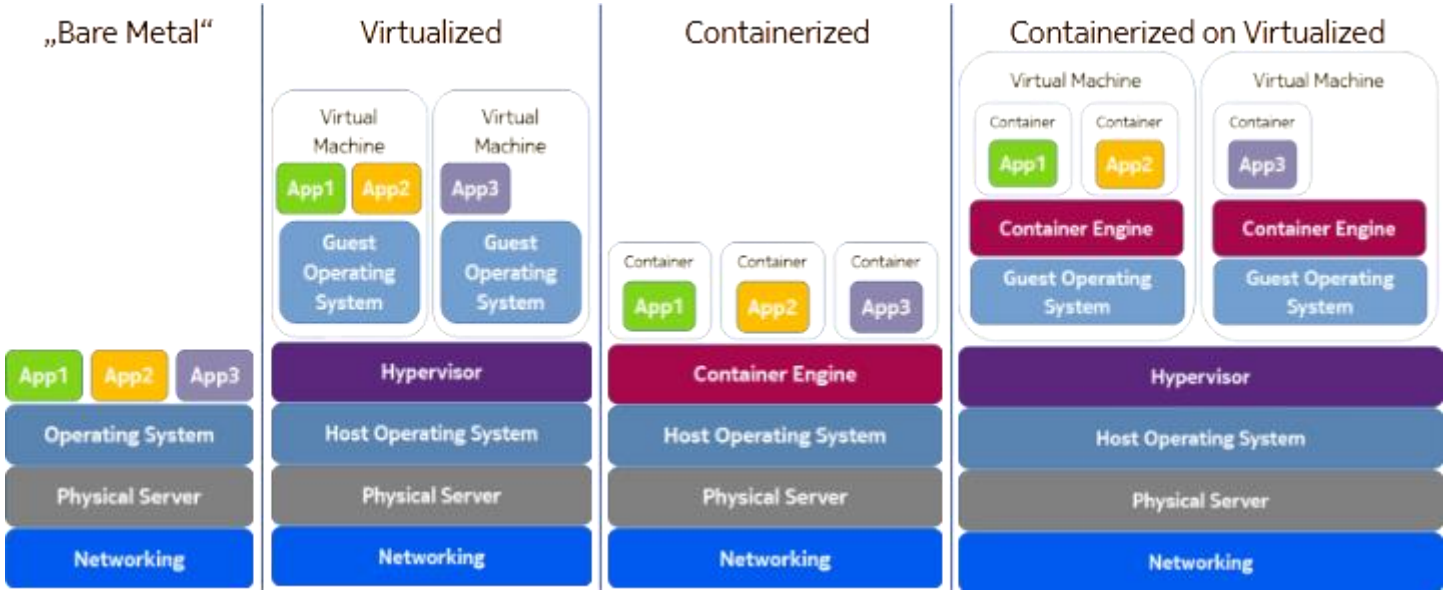
Containerized

- Lightweight and standalone package of application with all its dependencies
- Quick resource provisioning
- Scalable and portable
- Potentially less secure
  - Shared underlying OS
  - Isolation relies on the OS-level primitives



From bare metal to serverless

Summary



Container ecosystem glossary

Container ecosystem glossary

Basics

- **Containerization** - a process of encapsulating an application and its dependencies into a container image for deployment
- **Container** - isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging. It's a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another
- **Image** - a lightweight, standalone, executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files
- **Volume** - a directory or file in a container that bypasses the Union File System to provide access to persistent storage.
- **Namespace** – a Linux kernel feature which can isolate processes from each other
- **CGroups** - a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, etc.) of a collection of processes
- **Microservices** - an architectural style that structures an application as a collection of loosely coupled services, which are independently deployable and scalable

- **Docker** - a platform for developing, shipping, and running applications using containerization
- **Container Registry** - a repository for storing and managing container images, allowing for version control and sharing of images

Container ecosystem glossary

Orchestration

- **Container Orchestration** - refers to the automated management of containerized applications, including deployment, scaling, and scheduling

- **Kubernetes** - an open-source platform designed to automate deploying, scaling, and operating application containers
- **Deployment** - a resource that represents a set of multiple, identical Pods with no unique identities, all running the same application
- **Pod** - a group of one or more containers that are deployed and managed together on the same host
- **Node** - a worker machine, part of a cluster, that may be a VM or physical machine, depending on the cluster
- **Replica Set** - ensures that a specified number of pod replicas are running at any given time, and allows for scaling up or down
- **Ingress** - provides HTTP and HTTPS routing to services in a cluster, typically providing load balancing, SSL termination, and name-based virtual hosting
- **Egress** - refers to the traffic that flows out of a cluster, from a pod to an external endpoint
- **Secret** - an object that is used to store sensitive information, such as passwords, OAuth tokens, and ssh keys, in a cluster

Technology behind

Technology behind

Overview

- Docker is not a container
- Container is a technology
- Built from a few new features of the Linux kernel
- Two main kernel features are “namespaces” and “cgroups”

Technology behind

Namespaces

- The **PID namespace** allows us to create separate processes.
- The **networking namespace** allows us to run the program on any port without conflict with other processes running on the same computer.
- **Mount namespace** allows you to mount and unmount the filesystem without affecting the host filesystem.
- Linux kernel feature

Technology behind

Control Groups (CGroups)

- CGroups are used to **limit the usage of CPU and Memory** that a process or collection of processes can use
- Linux kernel feature

Technology behind

Docker

- Docker helps us easily create containers instead of having to do many things
- It’s one of many tools helping us to control the underlying container technology

Tools

Tools

Lifecycle management

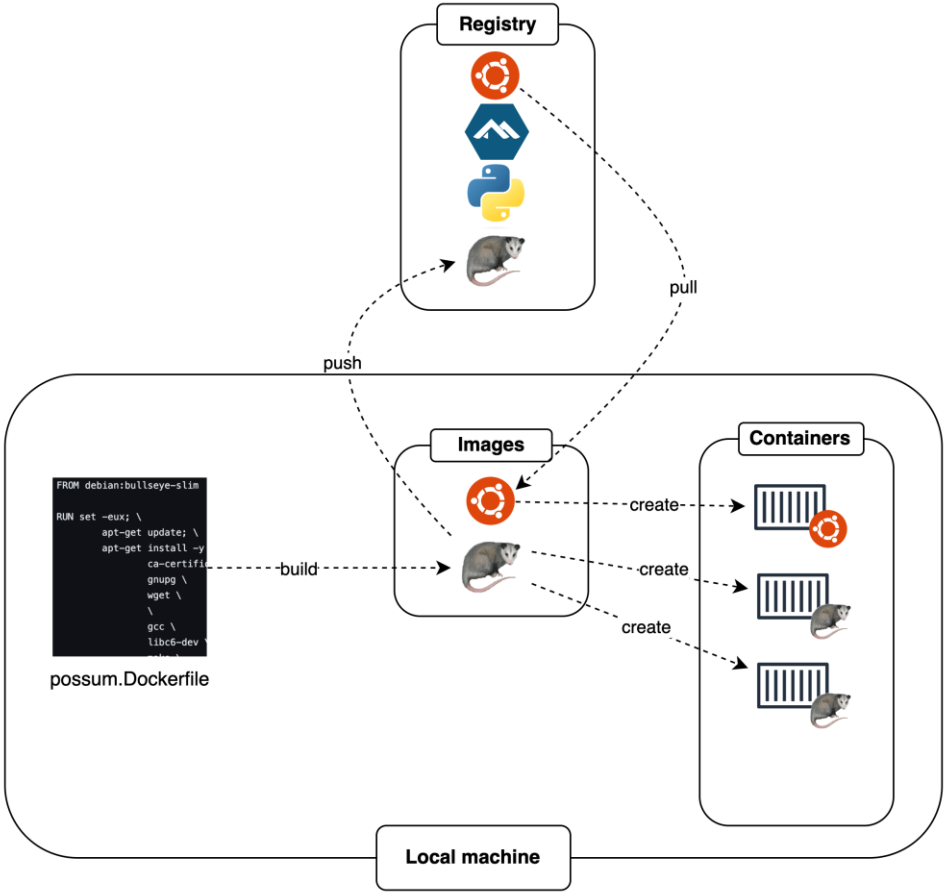
- Build image
- Push image
- Pull image
- Create container

docker

cri-o

LXC + LXD

buildah + podman



NOKIA

Tools

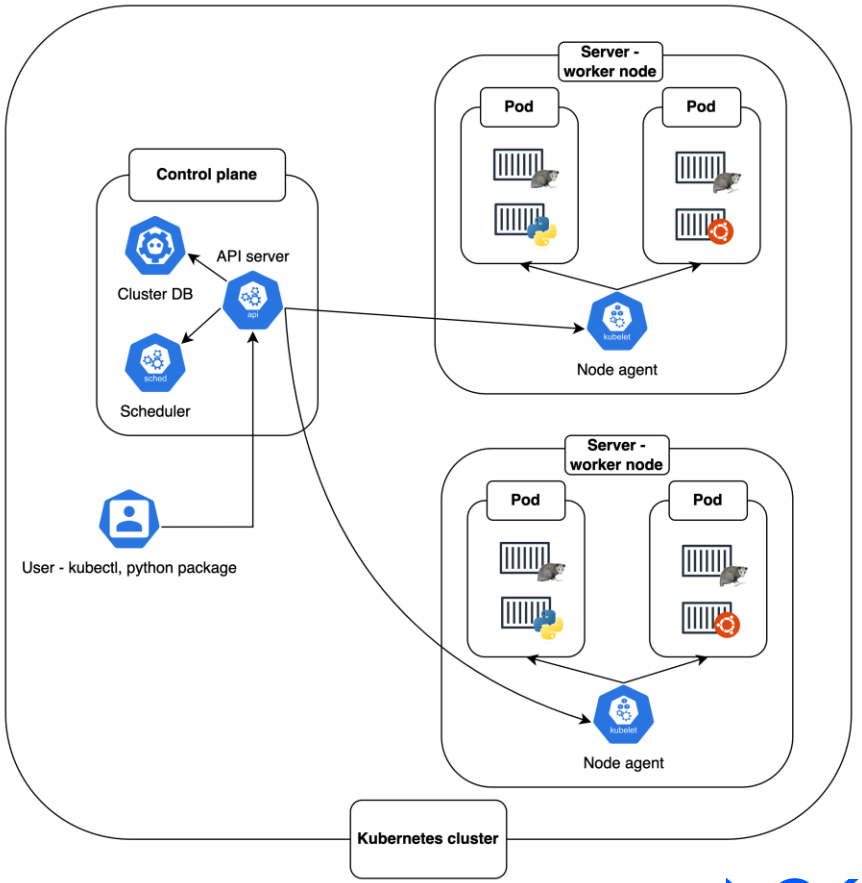
Orchestration

- Targeted at handling deployments at scale
- Exposing or hiding services
- Handling redundancy and high availability
- Rolling upgrades
- Build/Pull image
- Create container

kubernetes

HashiCorp Nomad

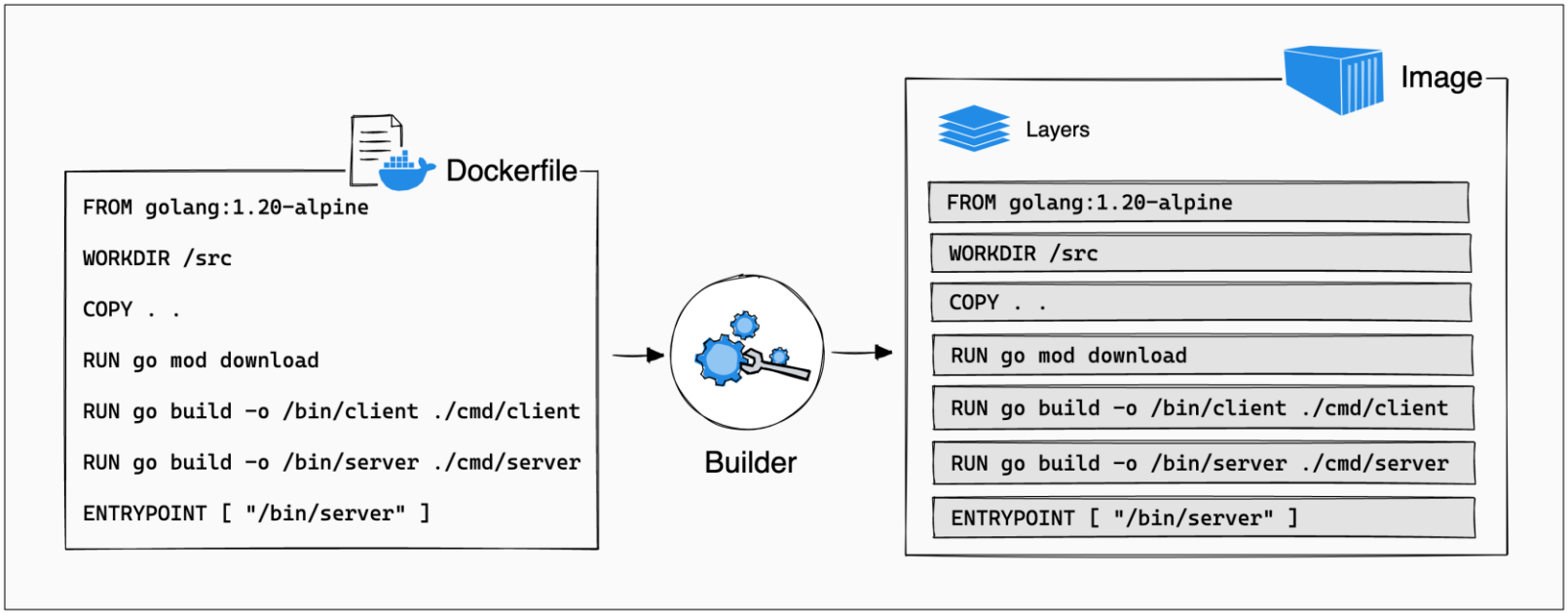
Docker Swarm



Tools

Dockerfile

- A Dockerfile is a text document in which you define the build steps for your application



Credit: https://docs.docker.com/build/guide/layers/

Use-cases and demo

Use-cases and demo

Hello world!

- Our application (`hello-world.py`):
  ```
  # echo 'print("Hello world")' > hello-world.py
  ```
- Our `Dockerfile`:
  ```
  FROM python:3.10.12-alpine
  WORKDIR /app
  COPY . .
  CMD [ "python", "hello-world.py" ]
  ```
- Build container image
  ```
  # docker build -t my-hello-world .
  ```
- Run
  ```
  # docker run my-hello-world
  ```

```
# docker run my-hello-world

Hello world
#
```

Use-cases and demo

Docker

Dockerfile Reference

https://docs.docker.com/engine/reference/builder/

Docker Compose File Reference
https://docs.docker.com/compose/compose-file/compose-file-v3/

The good, the bad and the ugly of containers

The good, the bad and the ugly of containers

- Very easy to spawn container from existing images
- Google, AI Chat saves the day!
- Get & share reproducible build setup, keep host OS clean and lean
- Not so easy to understand how it works behind the scene when problem occurs
- Might get ugly when dealing with specifics
- Find the right image (with right tools) or build own
- Mount project into container
- Setup inner user to match host user if you want modify host files, i.e. build output, git interaction, ...
- Sometimes image doesn't expect to be run as non-root (XDG_CACHE_HOME example)
- Alias it once it works as expected

```
$alias dgo='docker run -it -v $(pwd):/go/ws -w /go/ws -u $(id -u):$(id -g) -e XDG_CACHE_HOME='/tmp/.cache' golang go'
```

Questions?

More resources

https://docs.docker.com/get-started/overview

https://podman.io/get-started

https://linuxcontainers.org/lxc/introduction

https://kubernetes.io


NOKIA