Università di Pisa

Dept. of Information Engineering

Course Wireless Networks - 2021/2022

Virtualization (LAB)

Alessandra Fais – PhD Student

email: alessandra.fais@phd.unipi.it
web page: for.unipi.it/alessandra_fais/

LAB organization

- **PART I (theoretical)**
  - Introduction to SDN, NFV, MEC * concepts
  - Cloud computing and service-based architectures

* SDN = Software Defined Networking,
NFV = Network Function Virtualization,
MEC = Multi-access Edge Computing

- **PART II**
  - OpenStack cloud computing platform
  - OpenStack and NFV
  - Live session: OpenStack platform of the DII CrossLab project

LAB organization

* VM = Virtual Machine

- **PART III**
  - Virtualization overview and different approaches
    - VMs* on hypervisors, containers, alternative solutions

- **PART IV**
  - Containers -> Docker
  - Orchestrators -> Kubernetes
  - Hands-on session: Docker, docker-compose, Kubernetes

PART III

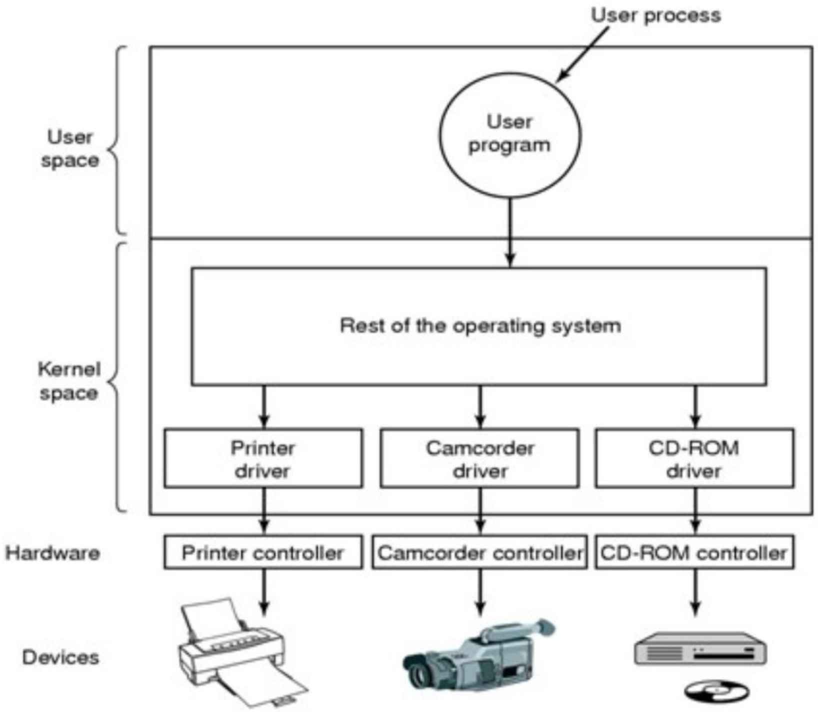
Outline of Part III

1) Virtualization overview
2) Virtualization approaches
    - Hypervisor-based solutions
    - Container-based solutions
    - Alternative solutions
3) Considerations on performance

Virtualization overview

Hardware vs Software

- **Hardware (HW)**: physical components of a computer
  - Monitor, CPUs, GPUs, hard drive, NICs, …
- **Software (SW)**: set of instructions written to alter the state of the hw
  - Device Drivers
  - Operating System
  - Application Software



Virtualization: an overview

- Run different **services** in the cloud

- Keep up with the **growth of exchanged data**
  - Increase the capability of *data centers* by means of *server virtualization*

Virtualization: an overview

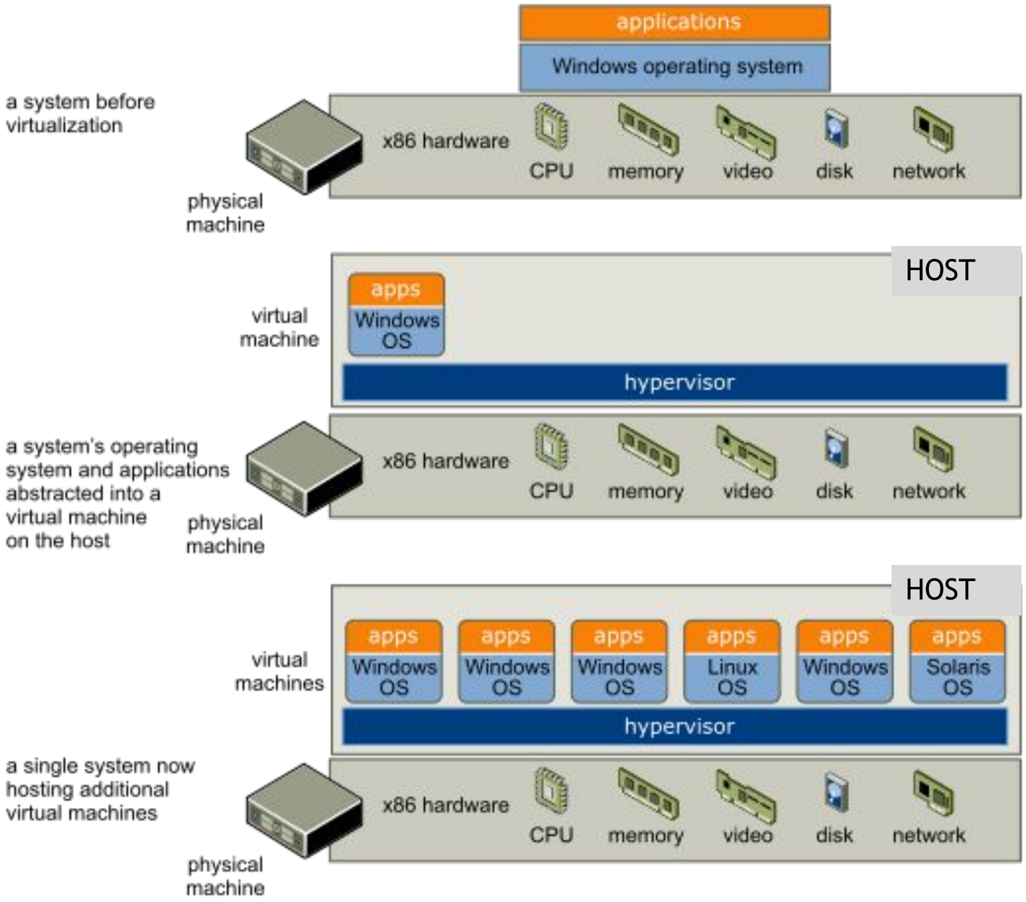
Considerations on **availability** and **cost**

Modern machines come with a lot of hardware resources

(many CPUs, large memories, …)

- Partition the resources of a **physical server** among several virtual machines, each running a **virtual server**
- Less expensive than having a separate machine for each server
  - Operating **costs** (space, energy, heat, failures, …) **reduced**
- Running inside a virtual machine has a performance cost

Virtualization: an overview



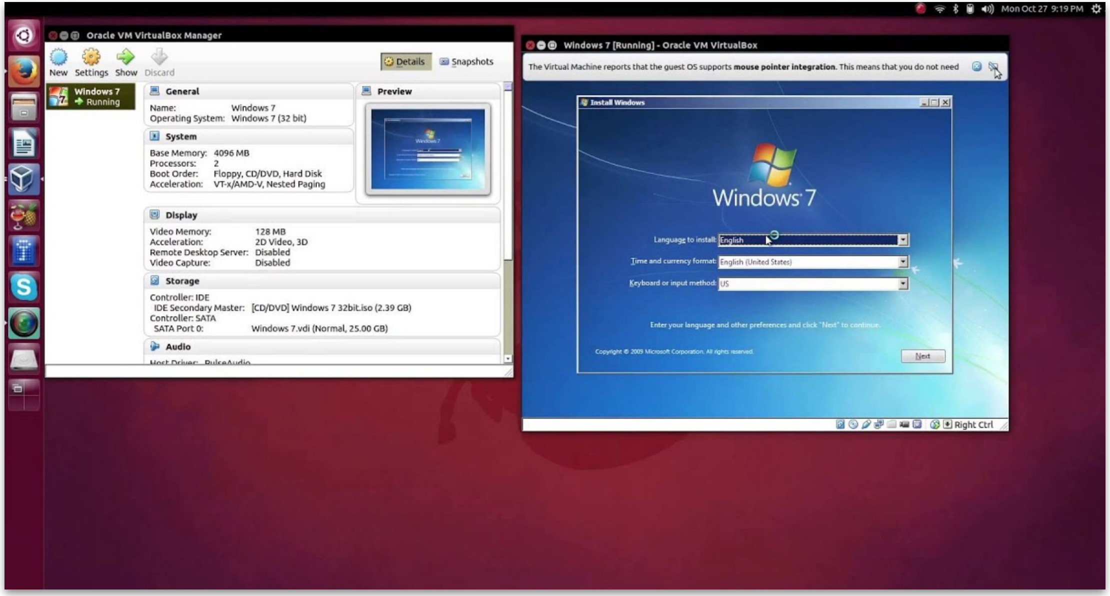
Virtualization: an overview

□ Considerations on **flexibility**

- More flexible **management of resources**
- **Dynamic provisioning**
  - Easily add/remove persistent storage, memory, processors
- Improved **utilization of resources**
  - Available resources multiplexed among the active users
- **Live migration**
  - Move an entire running system from a physical machine to another
- **Ease of deployment**

Virtualization: an overview

- Considerations on **flexibility**



- Use of applications that run on different OSes on the same desktop

Virtualization: where and why

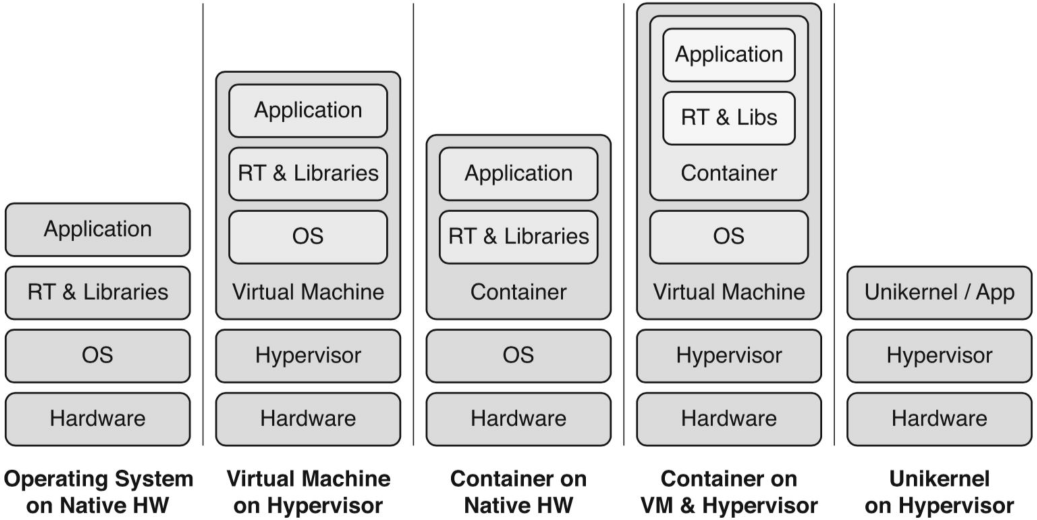
- Context
  - Cloud environment
  - Internet of Things (IoT)
  - Network Function Virtualization (NFV)

- Benefits
  - Hardware independence
  - Isolation
  - Secure user environment
  - Increased scalability

Virtualization approaches

Virtualization: different approaches

- Traditional virtualization
  - Whole system virtualization
  - *Hypervisor-based*
- Lightweight virtualization
  - *Container-based* (OS level)
  - *Unikernel-based* (library OS on a hypervisor level)



Virtualization approaches

Hypervisor-based virtualization architecture

Hypervisor-Based Virtualization Architecture

The **Hypervisor** or **Virtual Machine Manager** (VMM) level provides:

- **Hardware abstraction**
  - Virtual HW and virtual device drivers
- Standalone **VM instances** independent and isolated from the Host OS
- **Full Guest OS** (e.g. Linux) runs on top of the virtualized HW in each VM instance
- **Large disk images**

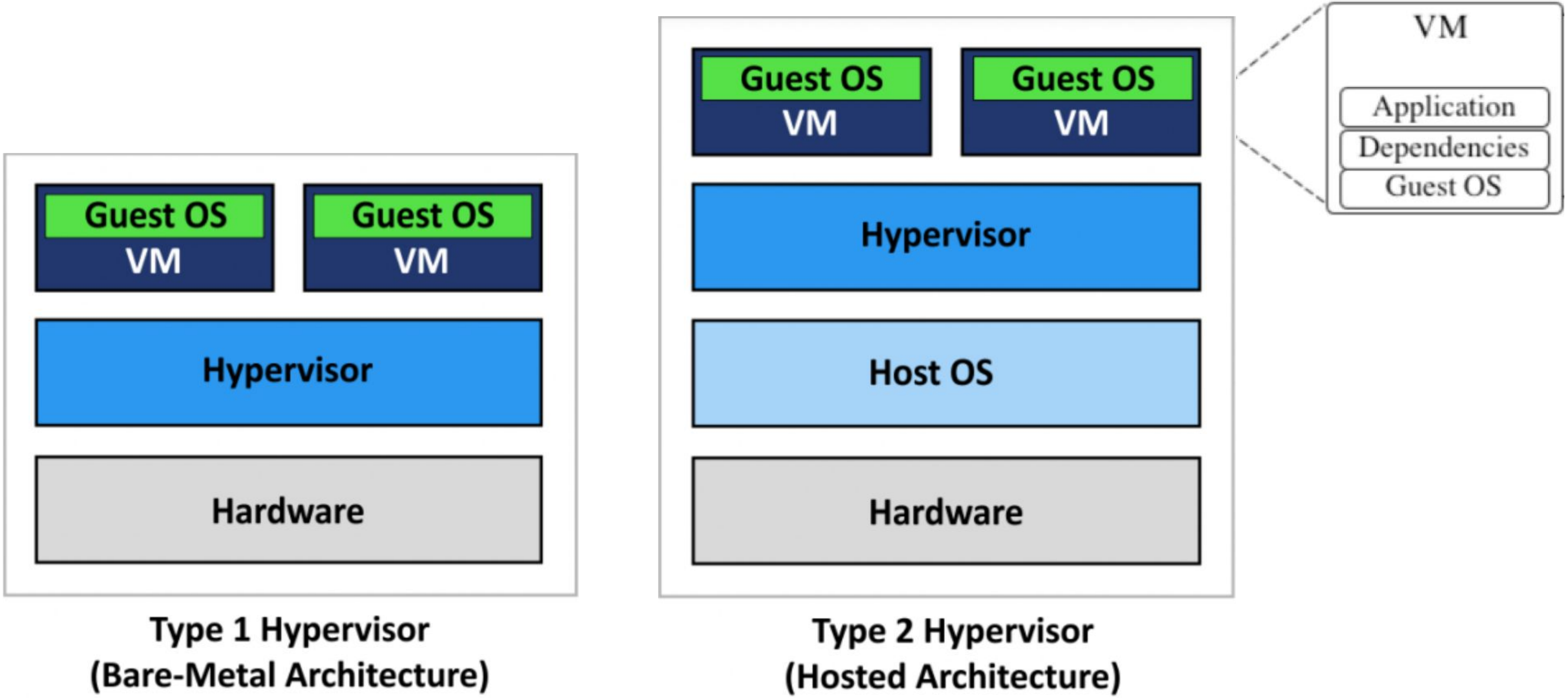
vmware

Oracle VM VirtualBox

KVM

Kernel-based Virtual Machine

Hypervisor-Based Virtualization Architecture



Hypervisor-Based Virtualization Architecture

- **Type 1 Hypervisor (native or bare-metal)**
  - Operate on top of the Host's HW

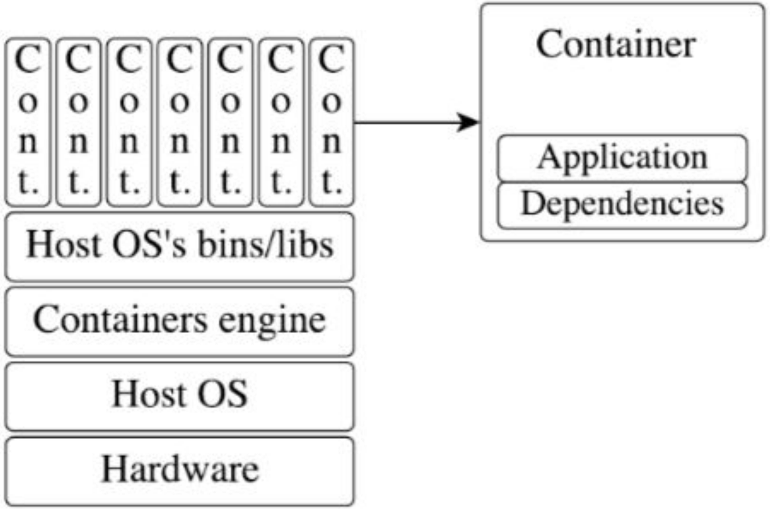
- **Type 2 Hypervisor (client or hosted)**
  - Operate on top of the Host's OS

Virtualization approaches

Container-based virtualization architecture

Container-Based Virtualization Architecture

- Isolation of processes at the **OS level**
  - Avoid overhead for HW virtualization
- Containers run on top of the same **shared Host OS kernel** and libraries
- One or more processes can be run within a container



Container-Based Virtualization Architecture

- **High-density** of virtualized instances
- **Small disk images**
- Multi-tenant **security** issues
  - Host kernel exposed to all the containers
  - Worse resource isolation w.r.t. hypervisors
- Claim to offer **superior performance** with respect to hypervisor-based solutions
- **Very popular** virtualization solution!

Linux Containers

Docker

Virtualization approaches

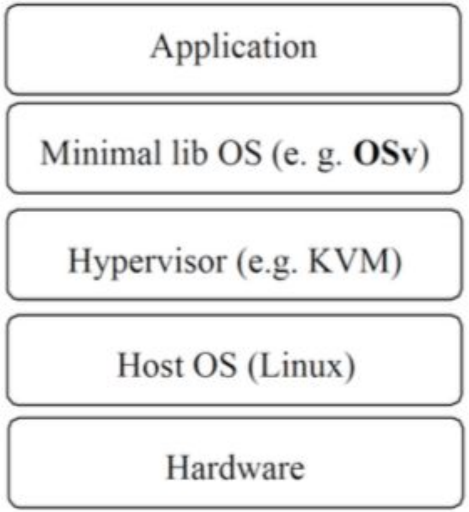
Alternative solutions: unikernels

Alternative Virtualization Architecture

- **Unikernels** are an emerging solution

- Library Operating System
  - Run single applications
  - Avoid the overhead and configuration of a complete Guest OS

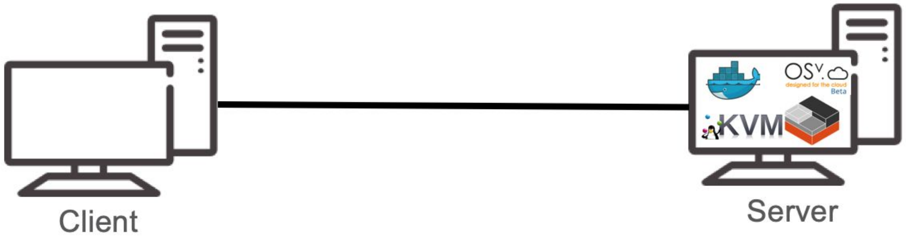
- Run on top of a hypervisor
  - Virtual hardware interface
  - Isolation benefits



Considerations on performance

A word on performance comparison

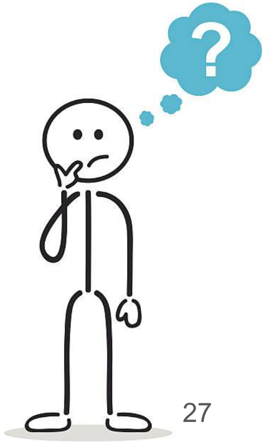
- Interesting metrics
  - **Instance startup time**
  - **Image size**
  - **Image footprint** (memory usage)
  - **Memory throughput** (Disk I/O)
  - **CPU** (e.g., execution time, FLOPs)
  - **Network latency** (Round Trip Time)
  - **Network I/O** (e.g., number of UDP/TCP transactions – request, response – between client and server)



Summary: how to choose the most suited technology for the intended purpose?

- Container-based solutions and emerging ones (like unikernels) are challenging the traditional hypervisor-based approach in cloud computing

- Light-weight technologies offer
  - Shorter instance startup time
  - Tiny image sizes
  - Smaller amount of memory required
  - Dense deployment of instances



Summary: how to choose the most suited technology for the intended purpose?

- Containers introduce almost negligible overhead
  - Tradeoff between versatility and ease of use and security aspects

- Hypervisors performance continuously improve

- Network efficiency is a delicate aspect for all the solutions

About **performance** and **overheads**:

- https://www.backblaze.com/blog/vm-vs-containers/
- https://www.brightcomputing.com/blog/containerization-vs.-virtualization-more-on-overhead

Useful references

Useful references

- Hypervisors vs Lightweight Virtualization: a Performance Comparison
  - http://faculty.washington.edu/wlloyd/courses/tcss562/research_papers/T3_Hypervisors_vs_Lightweight_Virtualization_A_Performance_Comparison.pdf
- A Performance Survey of Lightweight Virtualization Techniques
  - https://www.researchgate.net/publication/319407909_A_Performance_Survey_of_Lightweight_Virtualization_Techniques
- Kernel-based Virtual Machine (KVM)
  - https://www.redhat.com/en/topics/virtualization/what-is-KVM
- Oracle VM VirtualBox
  - https://download.virtualbox.org/virtualbox/6.1.4/UserManual.pdf
- Linux Containers
  - https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/overview_of_containers_in_red_hat_systems/introduction_to_linux_containers
- Docker
  - https://docs.docker.com/get-started/
- vmware glossary
  - https://www.vmware.com/topics/glossary/

PART IV

Outline of Part IV

1) Containers characteristics
2) Docker
    - Objects
    - Architecture
    - Deployment modes
        - Single host VS Cluster
3) Kubernetes

Outline of Part IV

Hands-on session:

Docker

- Installation
- Execution flow
- Commands
- Dockerfile
- Persistent storage management (volumes)
- Docker Compose

Kubernetes

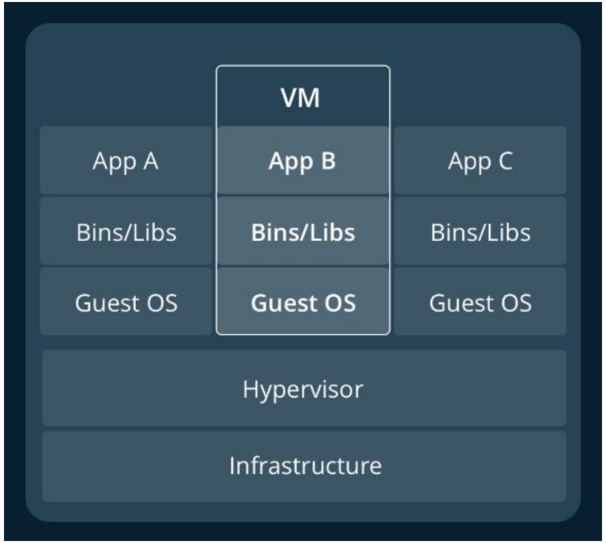
- Installation
- Complete application

Containers and Docker

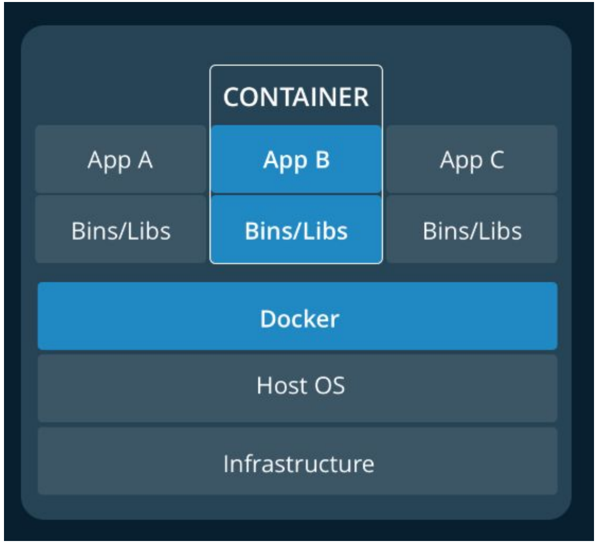
Container characteristics

Containers vs Virtual Machines

- A **VM hosts a whole Operating System** (*guest*), separated from the Host OS, over an **emulated hardware**



- A **container shares the OS kernel** with the host, avoiding hardware emulation (gain efficiency but lose isolation)

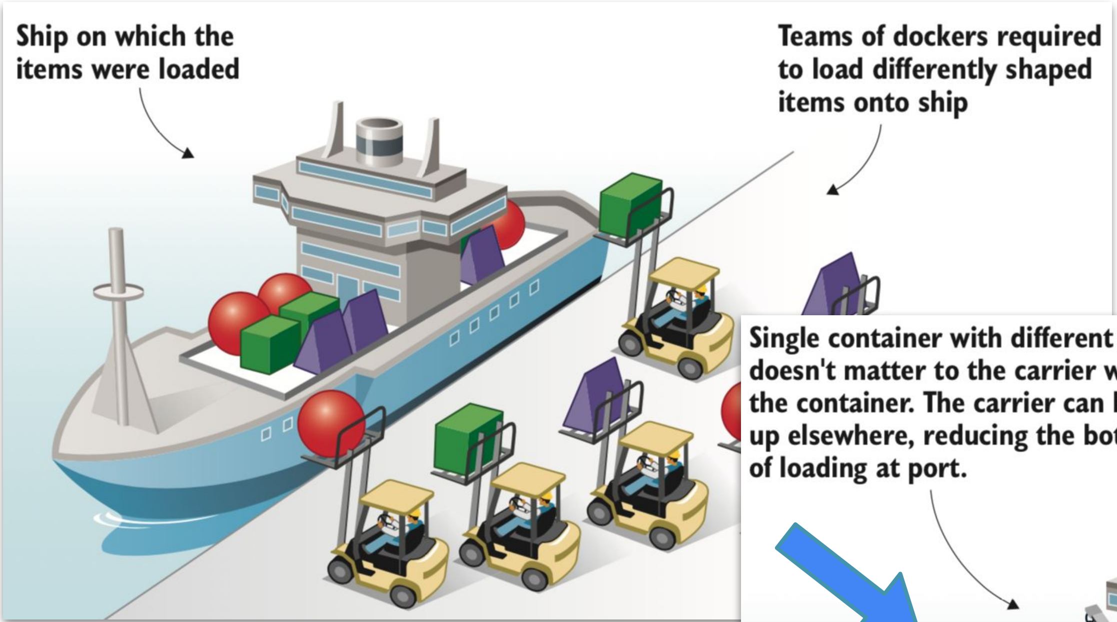


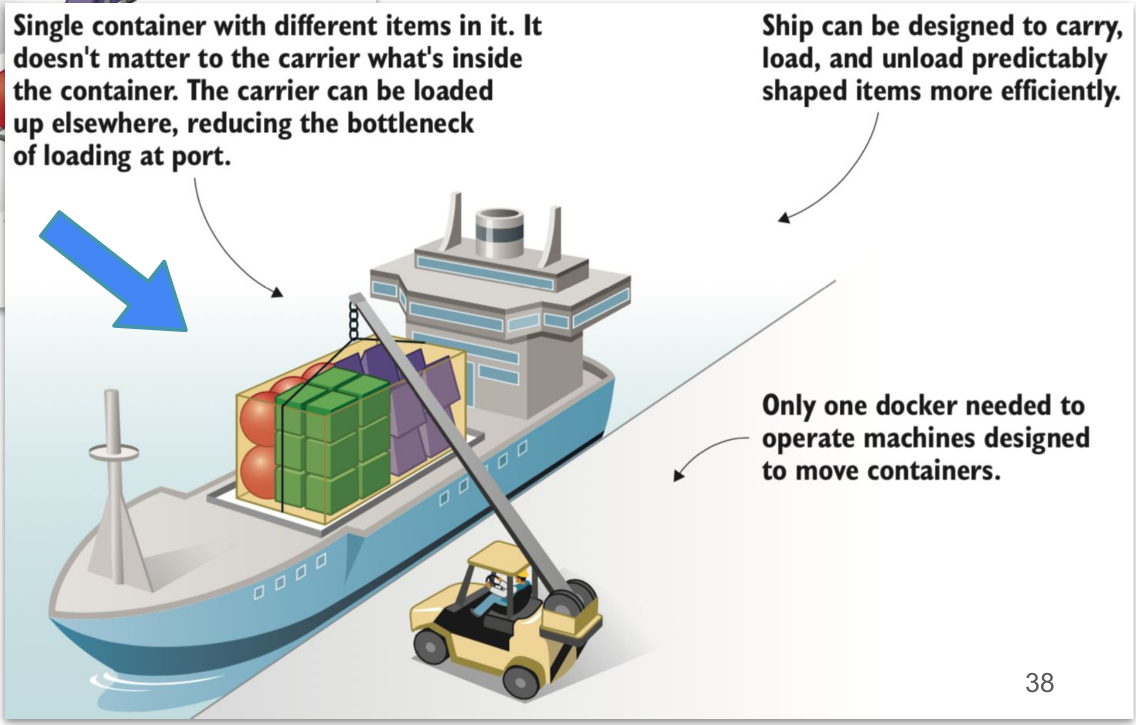
Containers: characteristics

- **Resources are shared** with the Host OS
  - Efficiency, overhead reduced
- **Portability**
  - **Build once, run anywhere!**

- Lightweight virtualization
  - Run dozens of instances at the same time (**high-density**)
- **Dependencies are embedded**
  - No need to configure and install

Containers: Docker

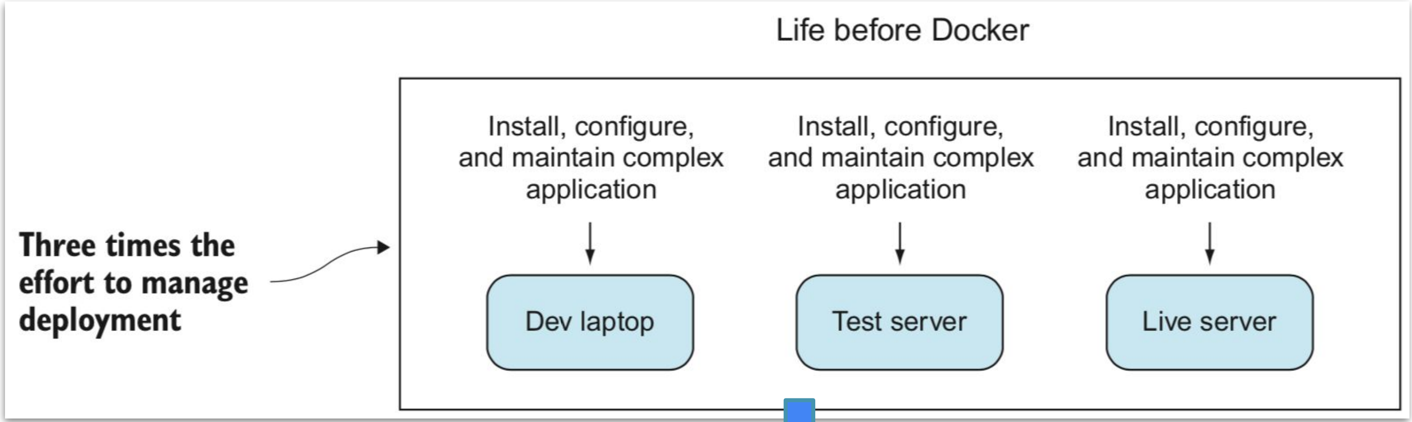


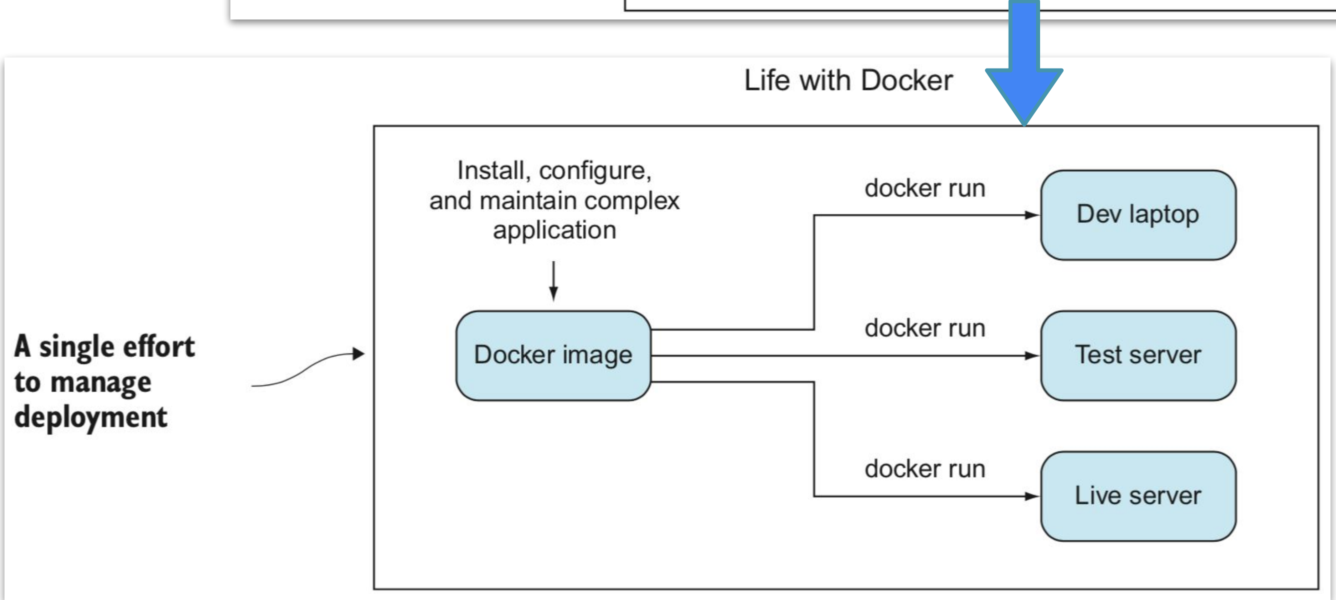


Docker

Introduction

Docker

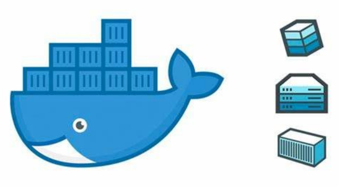

Life before Docker
Three times the effort to manage deployment
Install, configure, and maintain complex application
Install, configure, and maintain complex application
Install, configure, and maintain complex application
Dev laptop
Test server
Live server


Life with Docker
A single effort to manage deployment
Install, configure, and maintain complex application
Docker image
docker run
docker run
docker run
Dev laptop
Test server
Live server

Docker

- A definition:
  ''Docker is an open source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere''

- What can you do with Docker?
  Docker allows to **create**, **manage** and **orchestrate** application containers



  - Each application component is packed in a separate container
  - Optimization of development, testing, delivery and deployment of applications

Docker

- Design oriented to **software development steps**
  - Local **development environment**
  - **Testing**
    - Containers isolate tests into their own environment
      → no need to clean up the environment after each test execution!
    - Parallelize tests across multiple machines
    - Create different system configurations to test against
  - **Delivery**
  - **Deployment**

Docker

Objects

Docker objects

Image

- **read-only template** with instructions for creating a Docker container
- can be based on another image (extend the base image through a list of instructions defined in a *Dockerfile*)
- example: build an image based on the **ubuntu** image which also installs our application and the configuration details required to run it

Container

- **runnable instance** of an image
- defined by the image and the configuration options provided to it when created or started
- unit for distributing and testing our application, along with its dependencies

Docker objects

Network

- **bridged network** : new containers on a single host are connected by default to it and can refer each other by IP address
- **host network** : containers connected to this network share the host machine's network (remove network isolation between containers and host)
- **none network** : the container is not connected to any network
- **overlay network** : allow connectivity among containers on different hosts

Volume

- **persistent storage** for containers
- can be associated to one or more containers
- can be shared among several containers
- its lifespan is completely independent of the containers that use it

Docker objects

Service

- set of containers which are replicas of the same image
  - together they provide a load balanced service
  - scale up or down depending on the input load
- **deploy containers in production**

Stack

- set of **interdependent services that interact to implement an application**
- example: a voting application could be composed by (i) a service for the web interface which allows users to vote; (ii) a service to collect the votes of the users and store them in a Docker volume; (iii) a service for the web interface which shows the results of the voting in real time

Docker

Architecture

Docker ecosystem and deployment modes

Docker platform

- Docker Engine
  - Create and run containers
- Docker Hub
  - Cloud service (database) for storing and distributing images

Single host mode

Deploy containers on a single host machine

Cluster mode

Deploy containers of a Docker stack on all the nodes of a cluster (configuration with manager node + set of workers nodes)

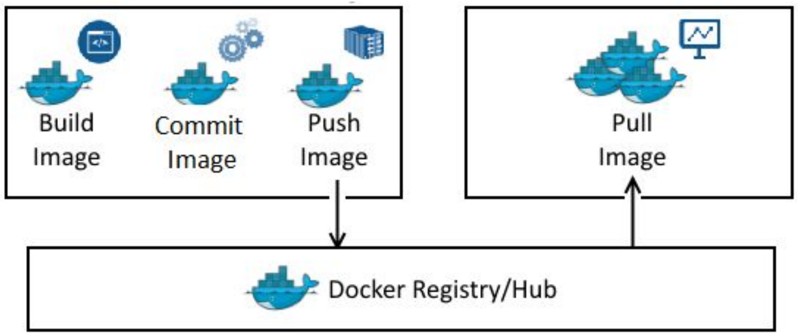
- Docker Swarm
- Mesos
- Kubernetes

Docker Engine

Docker guide sections here: https://docs.docker.com/engine/docker-overview/

- Build and containerize applications!
- **client-server architecture**
  - Server runs the daemon process **dockerd**
    - **dockerd** creates and manages images, containers, networks, and volumes
  - API exposed to programs to instruct the **dockerd**
  - Command line interface (CLI) client **docker**
    - Uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands

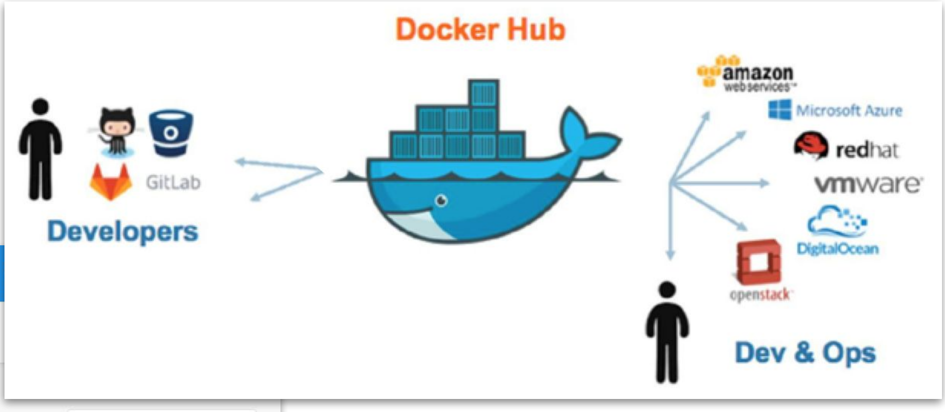
Docker Hub



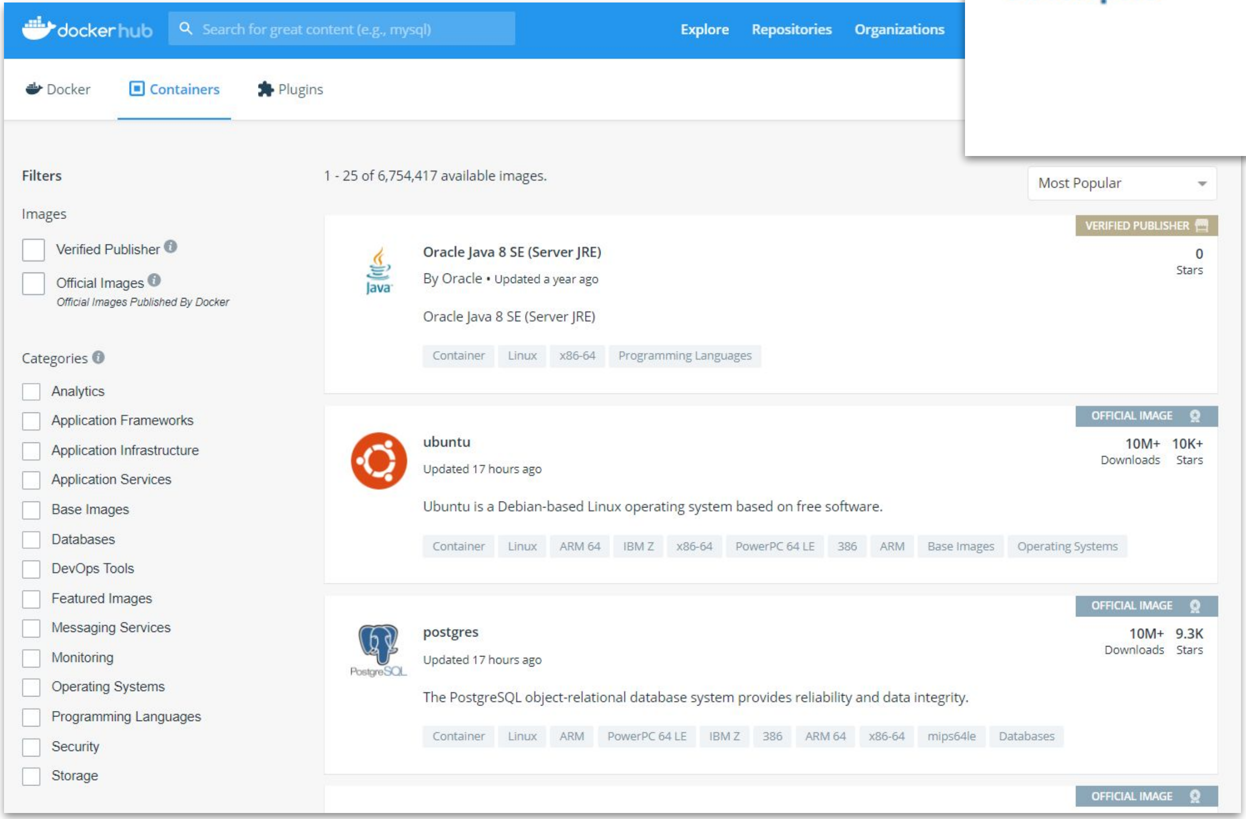
- Service provided by Docker for finding and sharing container images
- **Repositories** allow sharing container images with the Docker community
- Container images can be *pushed* to a repository or *pulled* from it
  - Official images (provided by Docker)
    - Clear documentation, best practices, design for most common use cases, scanned for security vulnerabilities
  - Publisher images (provided by external vendors)

Docker guide section, here:
https://docs.docker.com/docker-hub/

Docker Hub

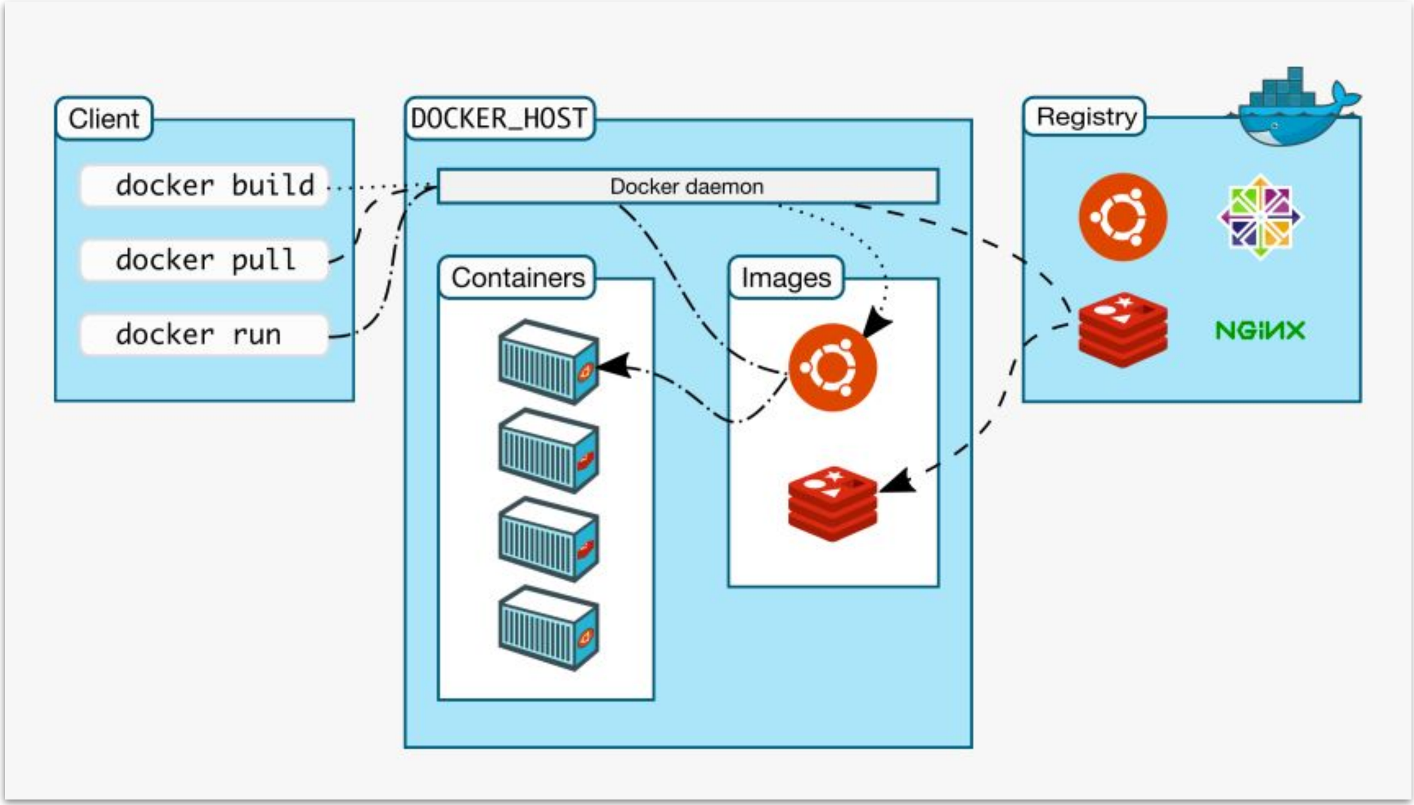


Docker Hub
GitLab
Developers
amazon webservices
Microsoft Azure
redhat
vmware
DigitalOcean
openstack
Dev & Ops



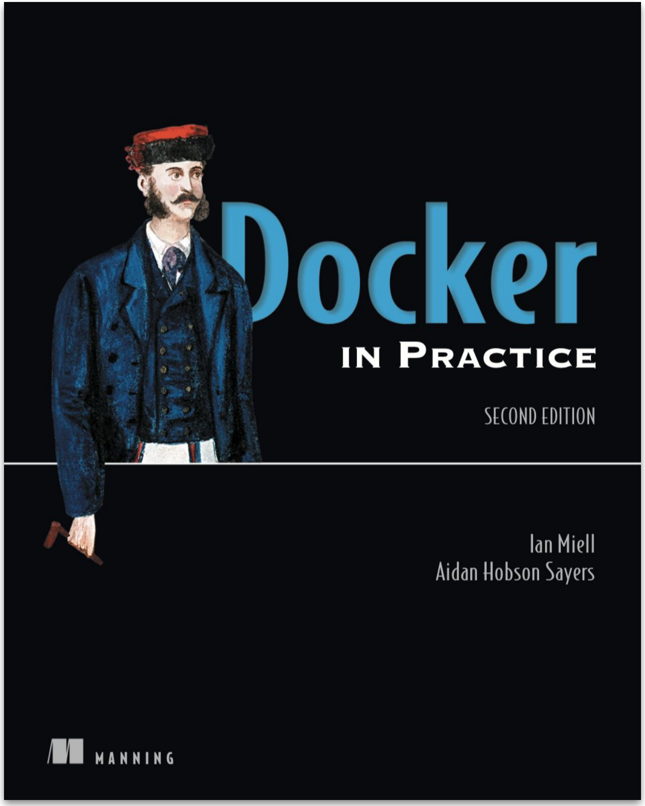
docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Docker
Containers
Plugins
Filters
Images
Verified Publisher
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
Application Infrastructure
Application Services
Base Images
Databases
DevOps Tools
Featured Images
Messaging Services
Monitoring
Operating Systems
Programming Languages
Security
Storage
1 - 25 of 6,754,417 available images.
Most Popular
VERIFIED PUBLISHER
Oracle Java 8 SE (Server JRE)
By Oracle • Updated a year ago
0
Stars
Oracle Java 8 SE (Server JRE)
Container
Linux
x86-64
Programming Languages
OFFICIAL IMAGE
ubuntu
Updated 17 hours ago
10M+
Downloads
10K+
Stars
Ubuntu is a Debian-based Linux operating system based on free software.
Container
Linux
ARM 64
IBM Z
x86-64
PowerPC 64 LE
386
ARM
Base Images
Operating Systems
OFFICIAL IMAGE
postgres
Updated 17 hours ago
10M+
Downloads
9.3K
Stars
PostgreSQL
The PostgreSQL object-relational database system provides reliability and data integrity.
Container
Linux
ARM
PowerPC 64 LE
IBM Z
386
ARM 64
x86-64
mips64le
Databases
OFFICIAL IMAGE

Putting all together: Docker Architecture



Useful references

Useful reference book



Link

Link

Useful references on Docker and Dockerfiles

- Docker quickstart
  - https://docs.docker.com/get-started/
- Dockerfile reference
  - https://docs.docker.com/engine/reference/builder/
- Best practices for writing Dockerfiles
  - https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- A Dockerfile tutorial by examples
  - https://takacsmark.com/dockerfile-tutorial-by-example-dockerfile-best-practices-2018/
- Interesting point of view on containers and VMs
  - https://www.docker.com/blog/containers-are-not-vms/

Useful references on Docker networking

- Docker networking overview
  - https://docs.docker.com/network/
- Container networking
  - https://docs.docker.com/config/containers/container-networking/
- Bridge network description and tutorial
  - https://docs.docker.com/network/bridge/
  - https://docs.docker.com/network/network-tutorial-standalone/
- Host network description and tutorial
  - https://docs.docker.com/network/host/
  - https://docs.docker.com/network/network-tutorial-host/