Cloud Computing Applications and Services
(Aplicações e Serviços de Computação em Nuvem)

Virtualization
Part II

University of Minho
2022/2023

Containers

- Lightweight virtual environment that groups and isolates a set of processes and resources (memory, CPU, disk, …), from the host and any other containers

Containers

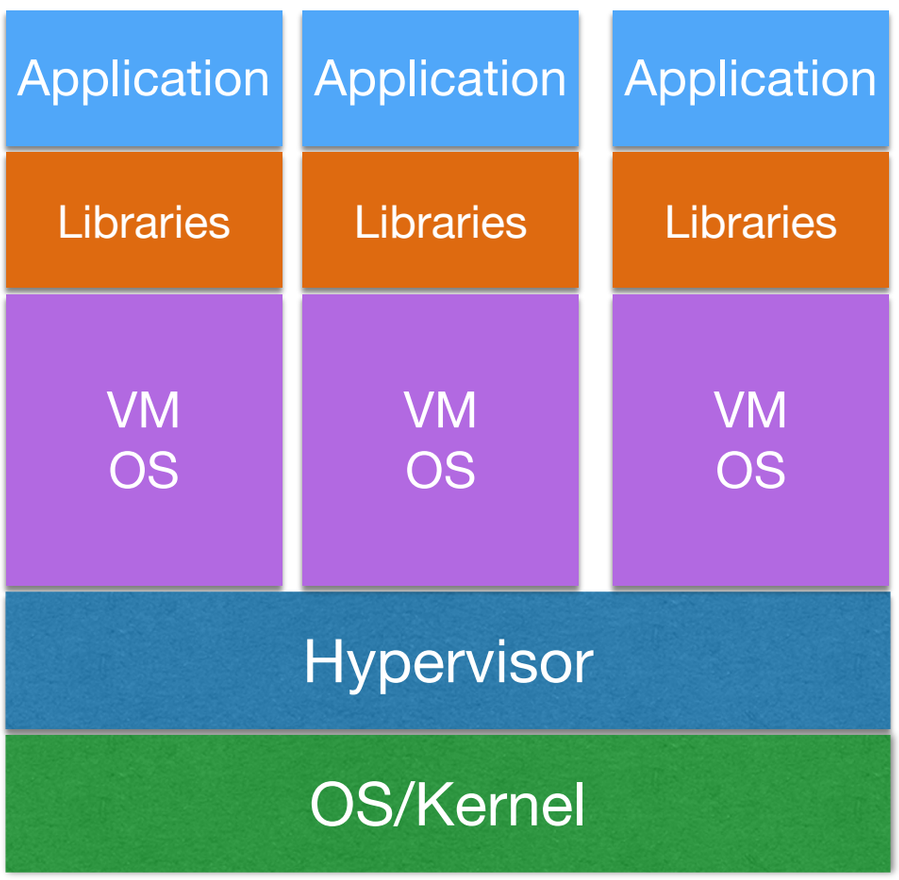
- Why are containers useful?
  - Running different isolated versions of the same software/application (e.g., database) in a shared OS/Kernel environment
  - Portability/migration across servers
  - Easy packaging of software, applications and their dependencies

Linux Containers

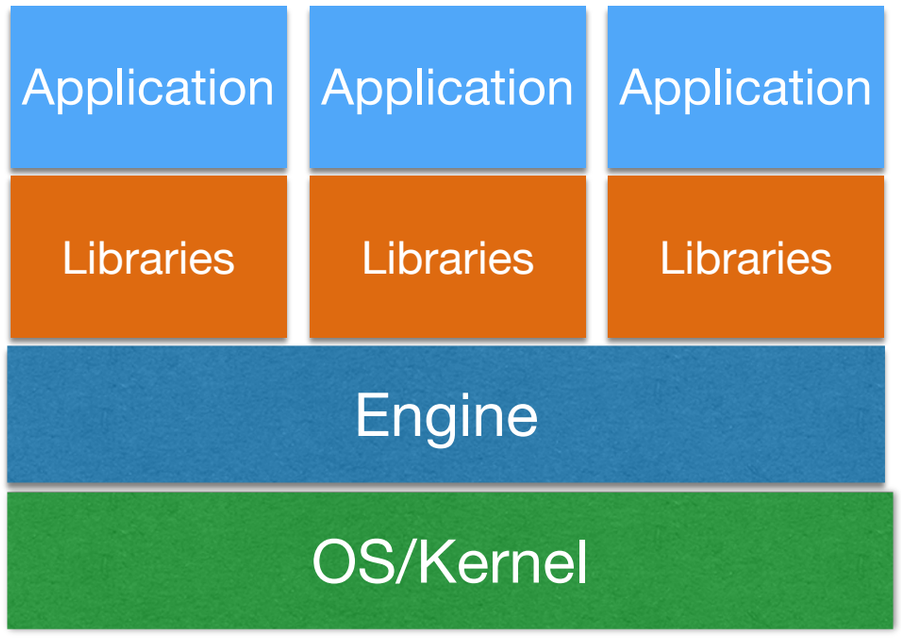
- Containers are lightweight, while not requiring full virtualization of CPU, RAM, network, and storage requests (as in VMs)

- The engine isolates and configures resources (see next slide)

VMs



Containers



Linux Containers

- The host system is compartmentalised for each container in terms of CPU and I/O (memory, disk, network)

- Each container shares the hardware and kernel/OS with the host system

- Containers are isolated from each other

Linux Containers
Building Blocks

- Namespaces (Isolation)
  - Component of the Linux Kernel that makes a global host resource appear as a dedicated resource for a group of processes running in the same namespace
  - Allows sharing of host resources across different containers without conflicts
    - Network, storage (filesystem), ...

Linux Containers
Building Blocks

- Control Groups (Resource Management)
  - Allows allocating resources (CPU, RAM, Disk I/O, network bandwidth) among groups of processes
  - In other words, it limits the amount of resources used by each container (e.g., storage space)

Linux Containers
Building Blocks

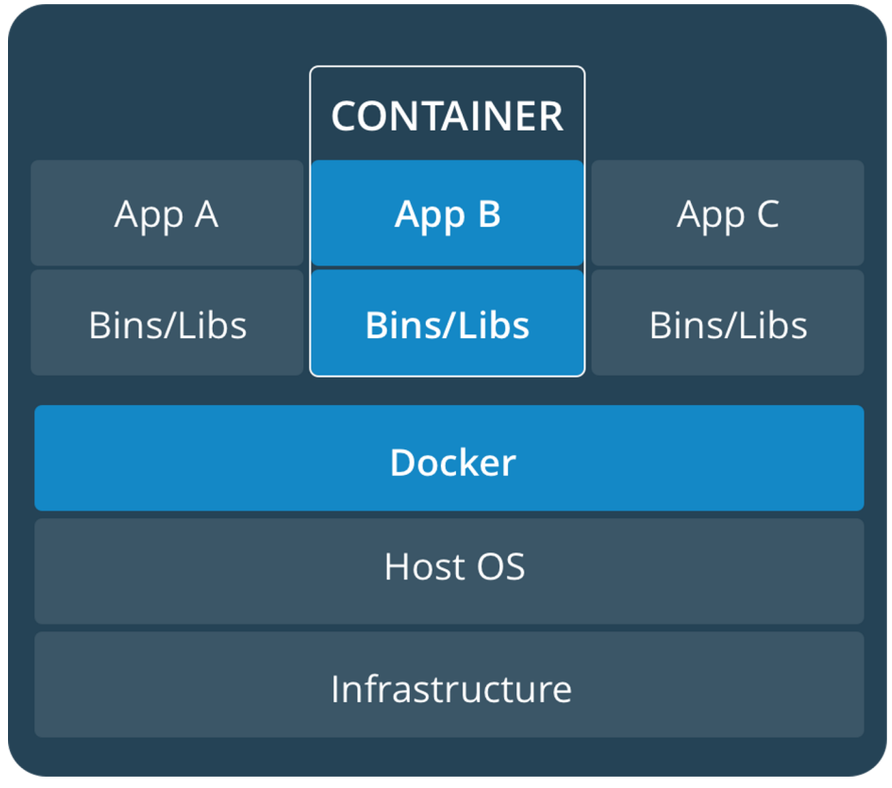
- SELinux (Security)
  - Provides additional security over namespaces so that a container is not able to compromise the host system and other containers
  - Enforces access control and security policies

Linux Containers
Types

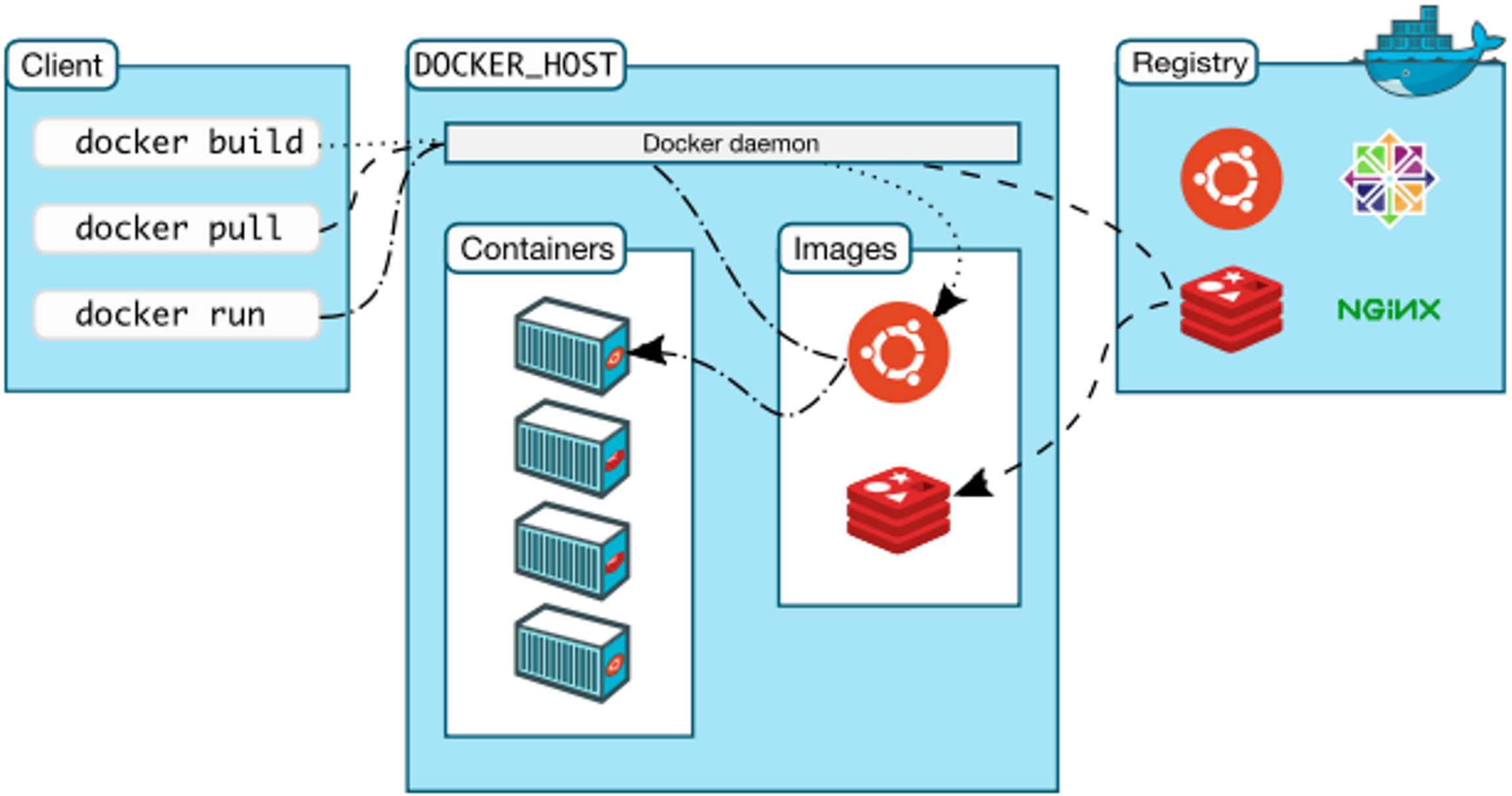
- OS Containers (e.g., LXC)
  - Containers run multiple processes and simulate a “lightweight” operating system
- Application Containers (e.g., Docker)
  - Focused on deploying applications
  - Each application is seen as an independent process

Docker

- Most widely-known container platform
- Supports Ubuntu, Fedora,
  RHEL, CentOS, Windows, etc
- https://www.docker.com



Docker


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

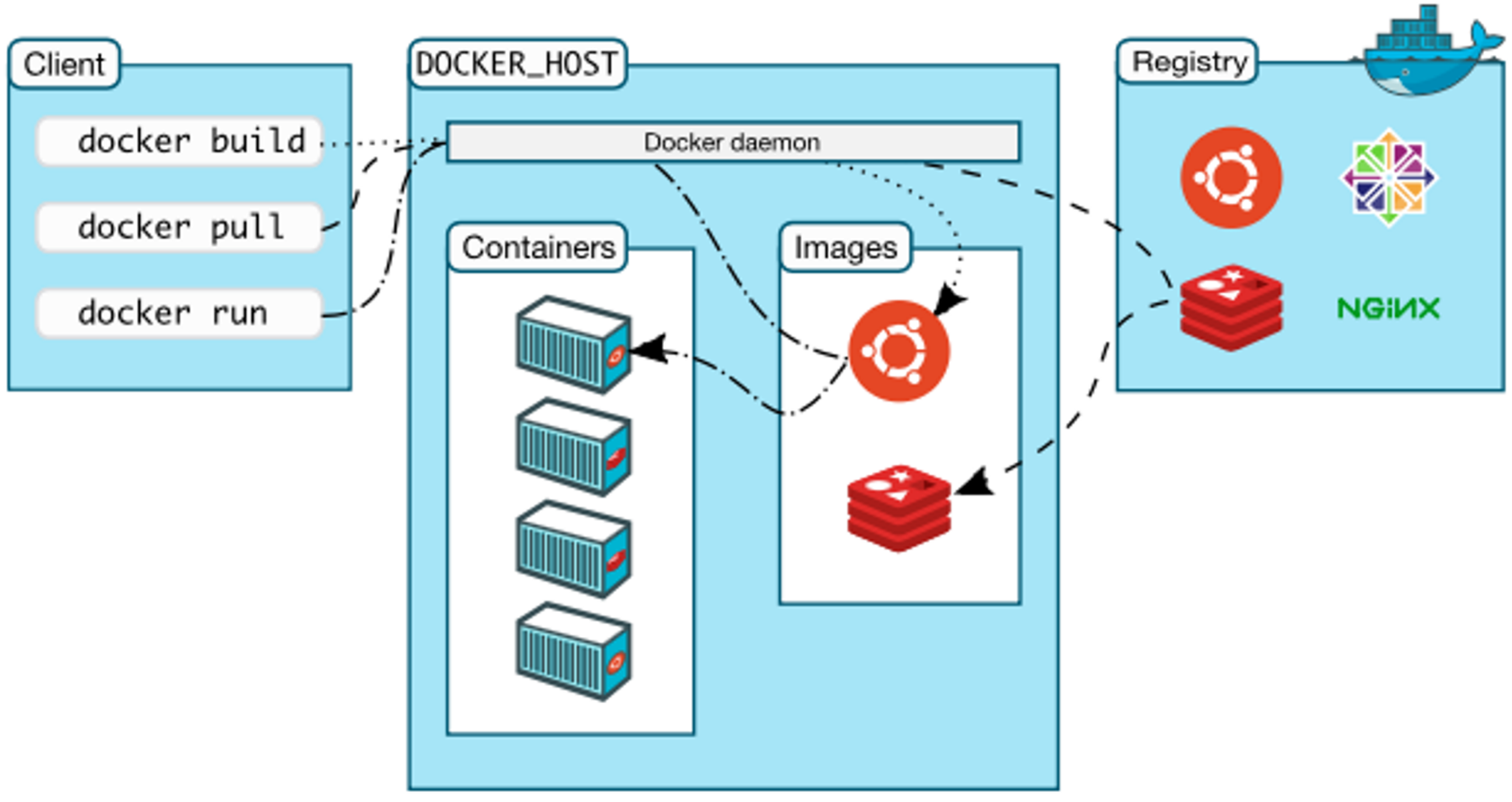
Docker
Docker Client

- Component used by users to interact with the Docker platform
- Exposes the Docker API for:
  - Running and managing containers, and networks
  - Reading logs and metrics
  - Pulling and managing images

Docker
Docker Daemon

- Uses the Docker API for receiving requests from the Docker Client

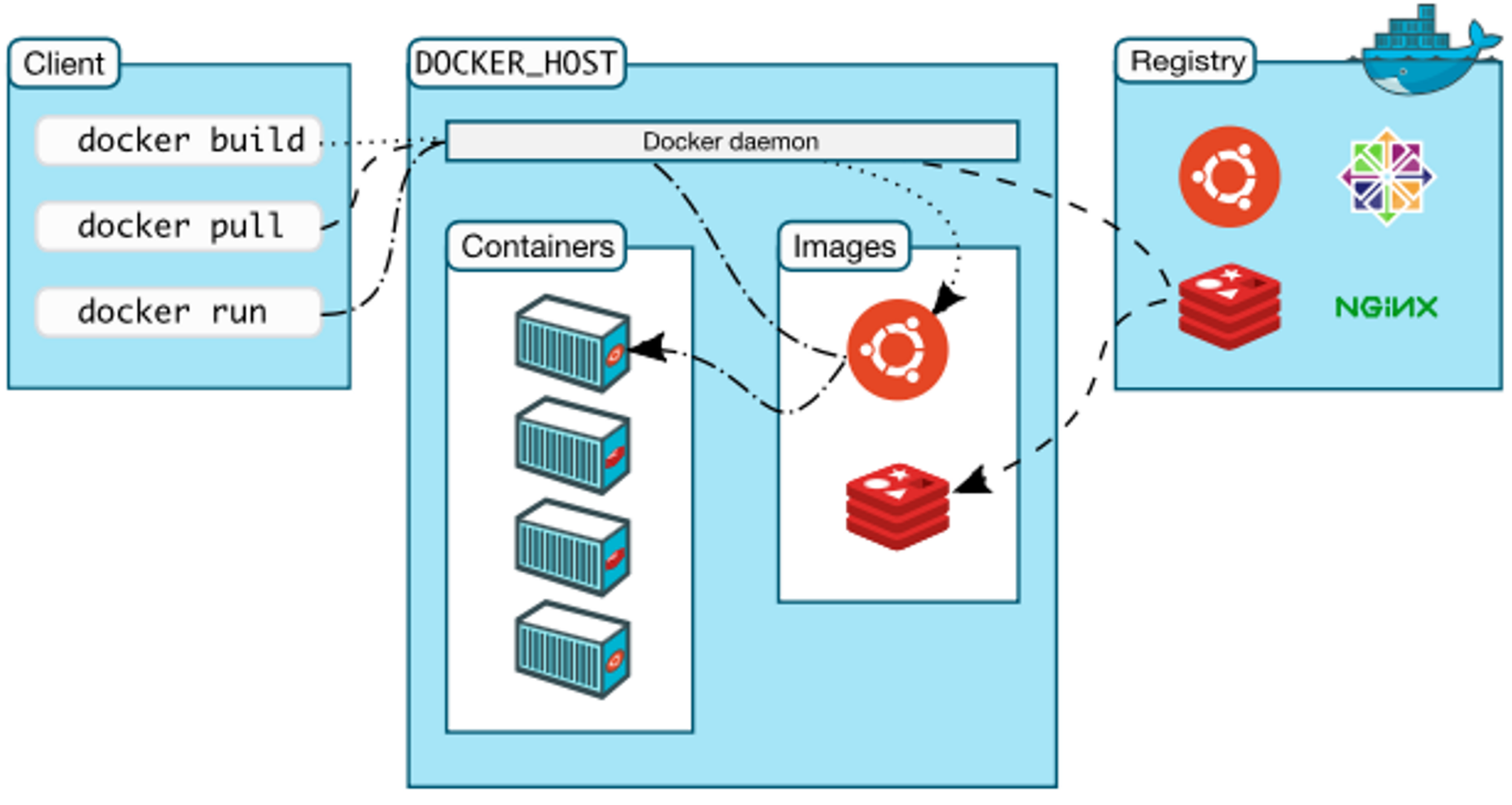
- Manages Docker Images, Containers, Networks



Docker
Docker Objects

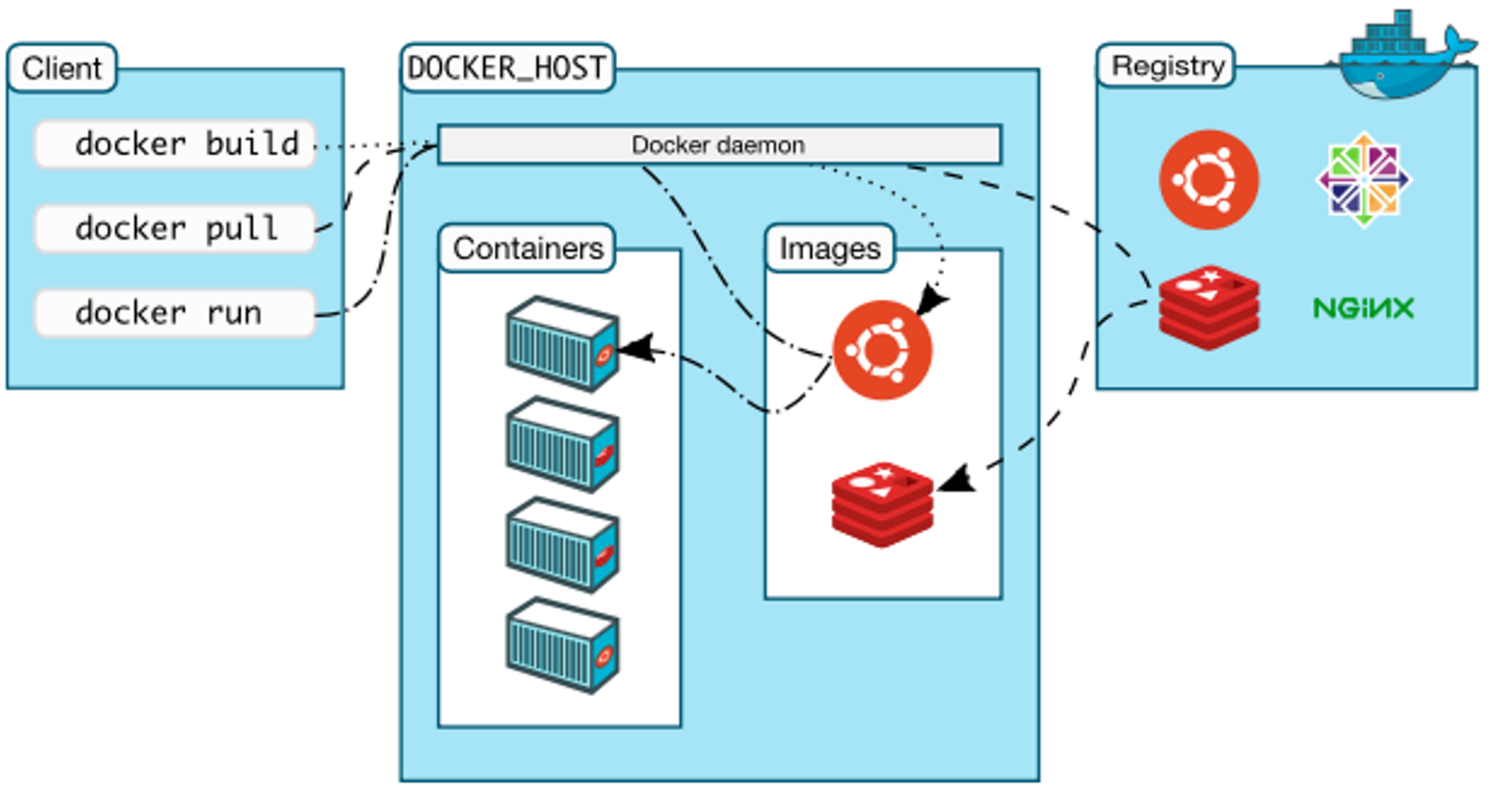
- Image - immutable (unchangeable) file that contains the source code, libraries, and other files needed for an application to run

- Container - runnable instance of an Image



Docker
Docker Registry

- Repository of Docker Images
- E.g., https://store.docker.com



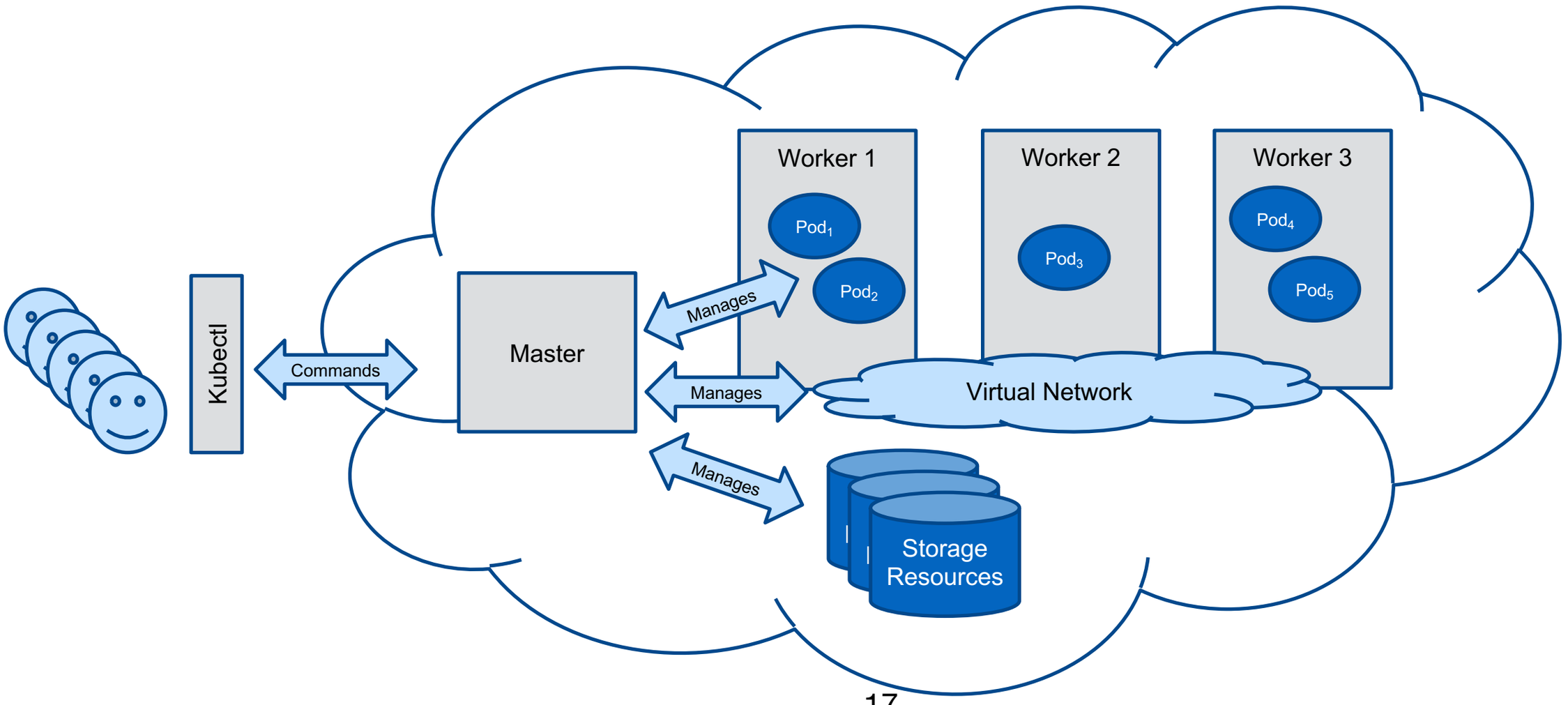
Kubernetes

- Automates the deployment, scaling and management of containers
- Interesting features:
  - Network management
    (e.g., service discovery, load balancing)
  - Modular storage orchestration
    (e.g., iSCSI, NFS, Ceph, AWS, GCP)
  - Simplified **scheduling, self-healing** and **scale out** for containers
- https://kubernetes.io

Kubernetes Cluster

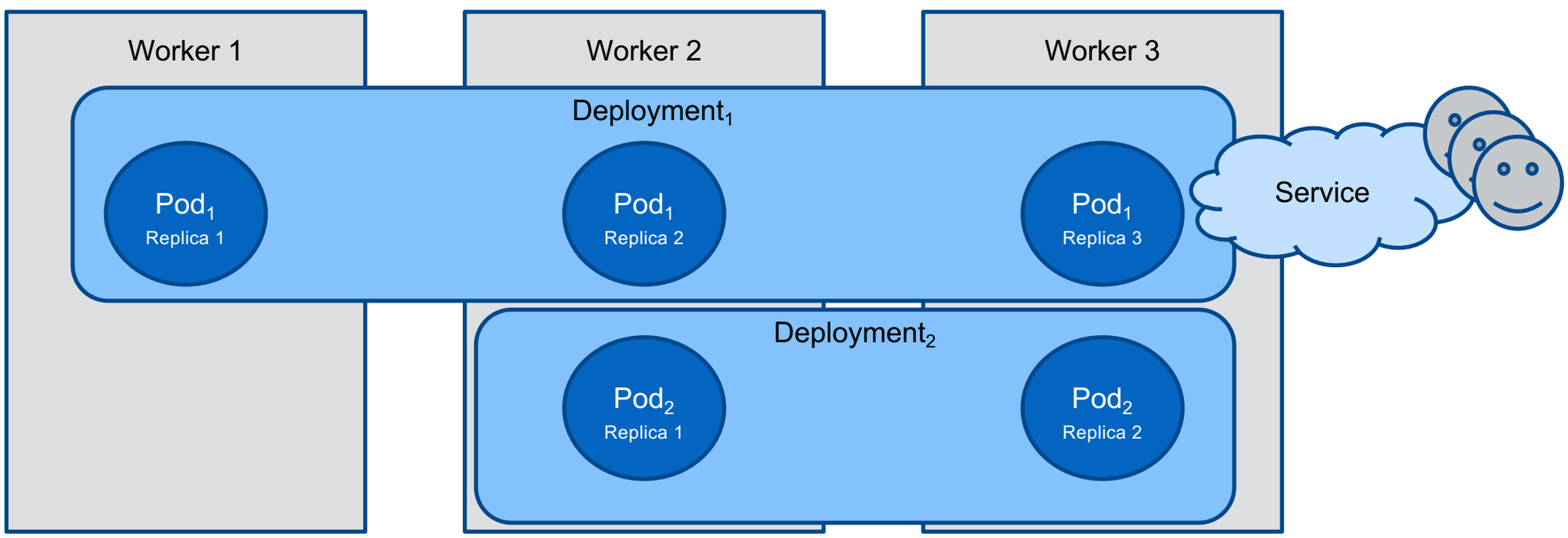
- Clients interact with the Master node
  (e.g., through the *kubectl* command-line tool)

- The Master manages the kubernetes cluster
  (i.e., worker nodes, pods, network, storage, ...)



Pod, Deployment and Service

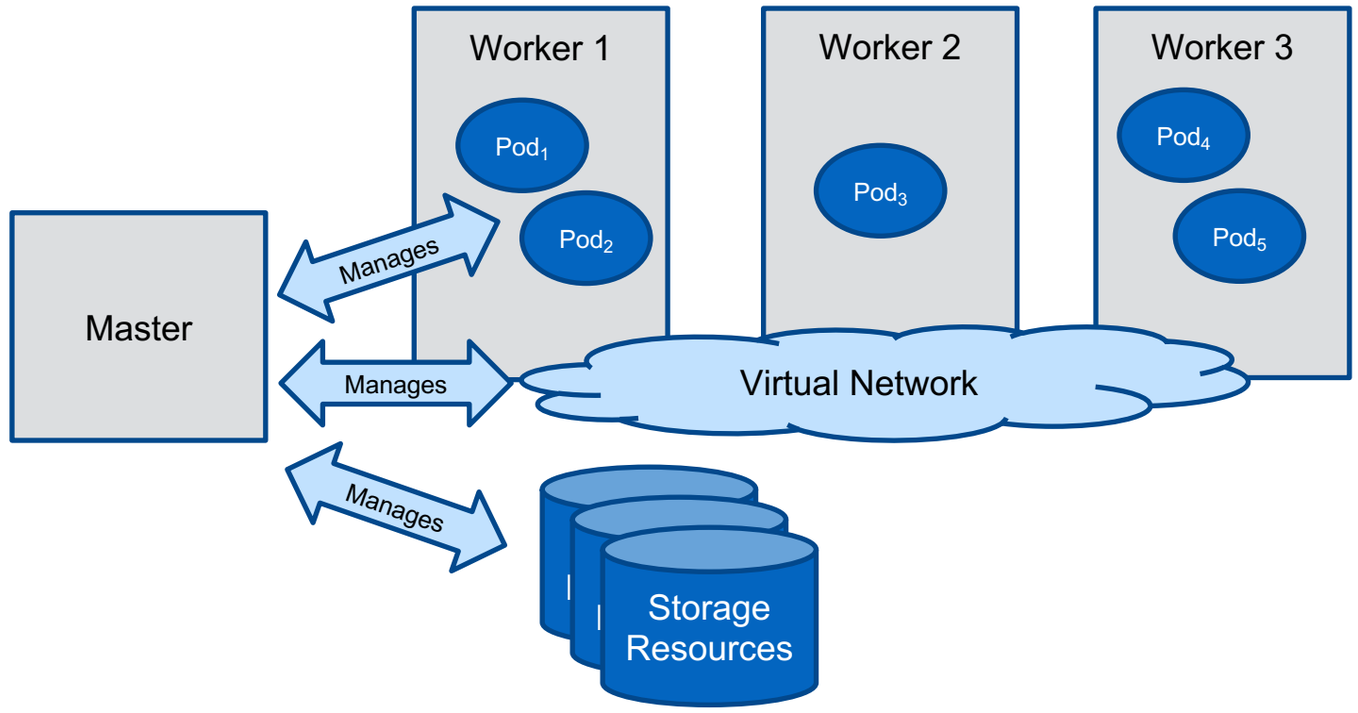
- **Pod -** computing unit composed by one or more application containers (e.g., Docker, containerd, CRI-O)
- **Deployment –** Specifies the runtime environment for pods (i.e., number of replicas)
- **Service –** Exposes a group of pods (application) as a network service



Pod_1 – Web Server
Pod_2 - Database

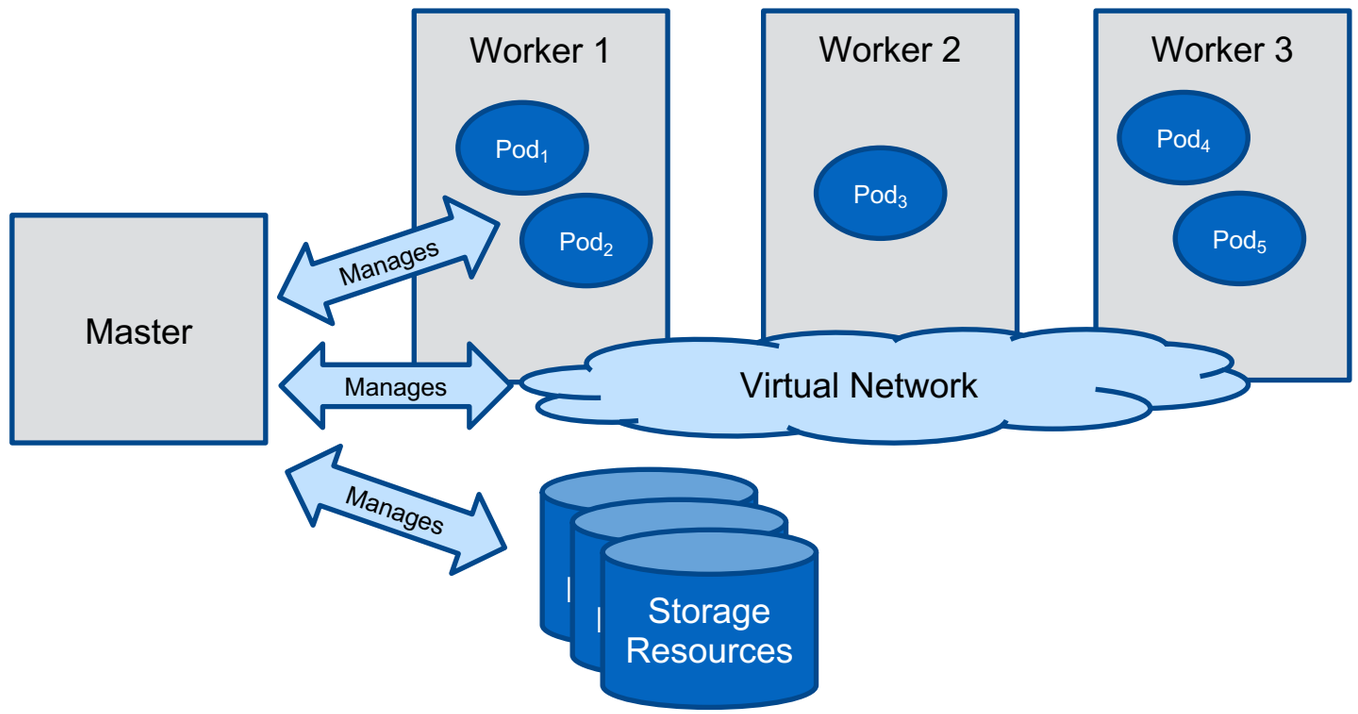
Network

- **Network –** each pod has a unique cluster-wide IP address
  - Networking across pods (even in distinct worker nodes) can be managed through networking overlays (e.g., Flannel, Calico)



Storage

- **Storage –** Pods can access storage volumes provided by different storage backends (e.g., iSCSI, NFS, AWS, GCP)
  - **Ephemeral** storage – available during the pod's lifetime
  - **Persistent** storage – available even if the pod is terminated



Summary
Containers vs VMs – Disclaimer!

Each technology is built with different goals and the best one depends on the targeted use case!

- VMs are useful when full server (OS) virtualization is needed
- Containers are useful for managing different libraries / applications

Advantages
Containers over VMs

- Faster testing/provisioning/migration
- Better resource utilization (lightweight) and performance
- Can be deployed on both physical and virtualized servers

Disadvantages
Containers over VMs

- Weaker isolation/security (OS/Kernel are shared)
- Less flexibility in running different OSs
  (e.g., Linux, Windows, BSD, ...)

Further reading

- S. Alapati. **Modern Linux Administration: How to Become a Cutting-edge Linux Administrator**. O'Reilly, 2016

- K. Morris. ***Infrastructure as Code: Managing Servers in the Cloud***. O'Reilly, 2016

- B. Burns. **Kubernetes Up & Running (Second Edition)**. O'Reilly, 2019.

Questions?