Introduction

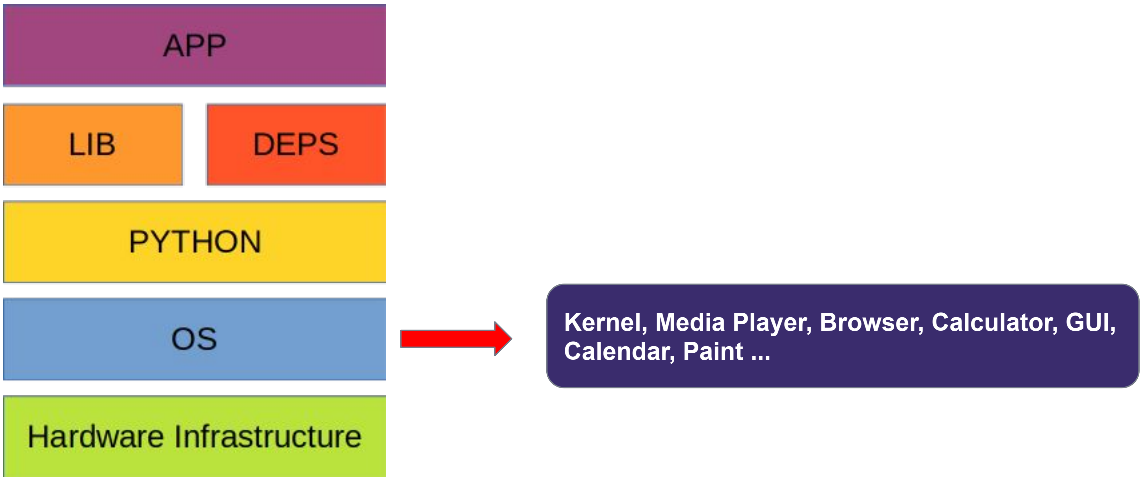
Table of Contents

- What is Container?
- Monolith vs Microservices
- Container Orchestration
- What is Kubernetes?
- Kubernetes components

1

What is Container?

What is Container?



APP

LIB

DEPS

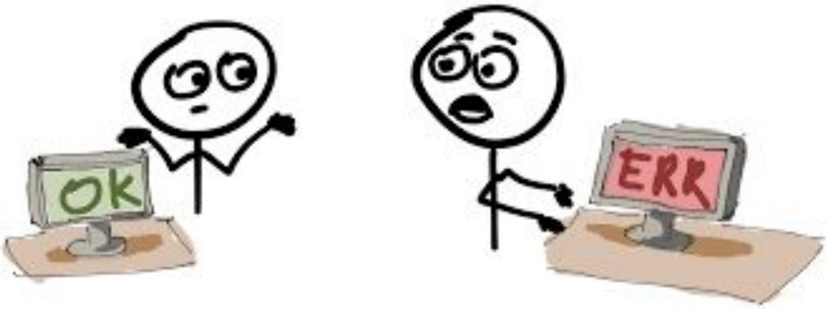
PYTHON

OS

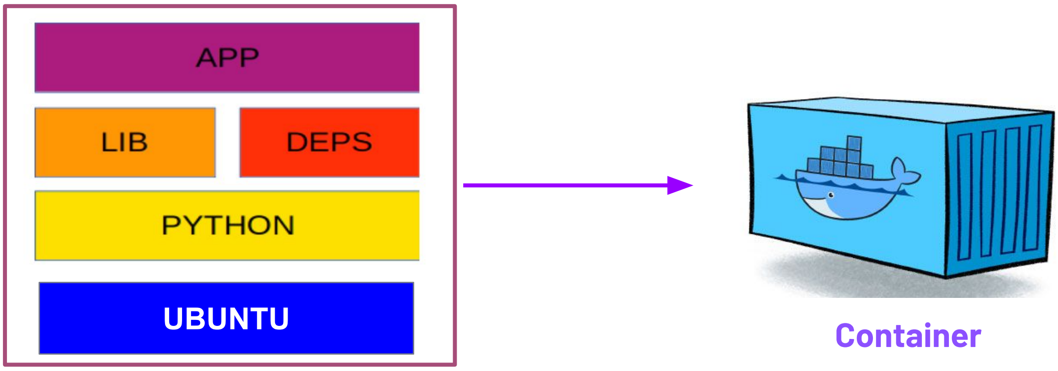
Hardware Infrastructure

Kernel, Media Player, Browser, Calculator, GUI, Calendar, Paint ...

What is Container?



What is Container?



APP

LIB

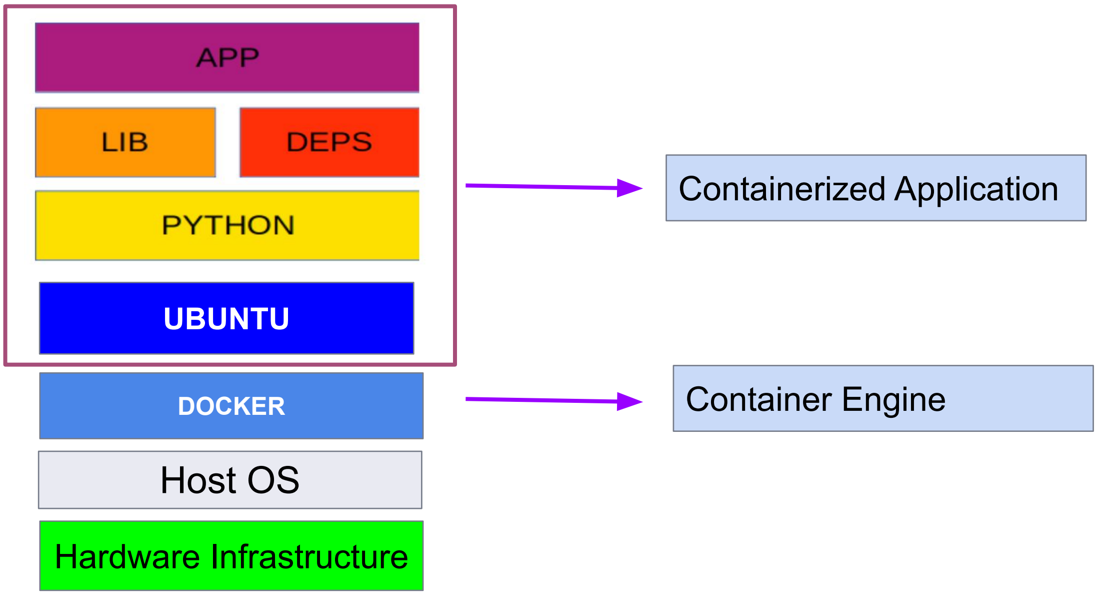
DEPS

PYTHON

UBUNTU

Container

What is Container?



APP

LIB

DEPS

PYTHON

UBUNTU

DOCKER

Host OS

Hardware Infrastructure

Containerized Application

Container Engine

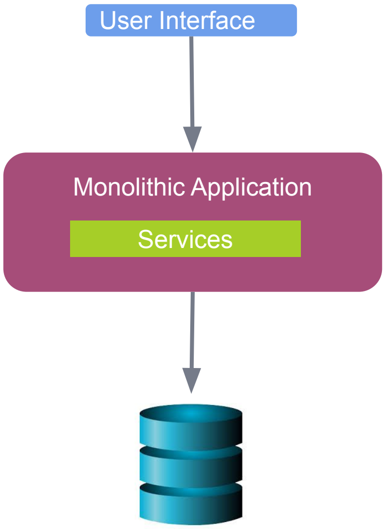
2 Monolith vs Microservices

Monolith vs Microservices

The word 'monolith' means 'one massive stone'. So we can describe monolithic as a large unified block.

Monolith vs Microservices

In software development, **monolithic architecture** is a traditional way to build an application as a single and indivisible unit.



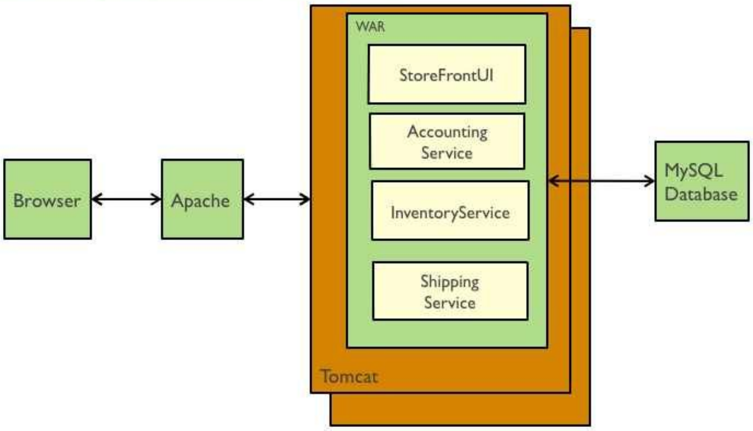
Monolith vs Microservices

Let's imagine that we are building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them.

The application consists of several components including the StoreFrontUI, which implements the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders.

Monolith vs Microservices

The application is deployed as a single monolithic application. For example, a Java web application consists of a single WAR file that runs on a web container such as Tomcat.



Monolith vs Microservices

Pros of monolithic architecture:

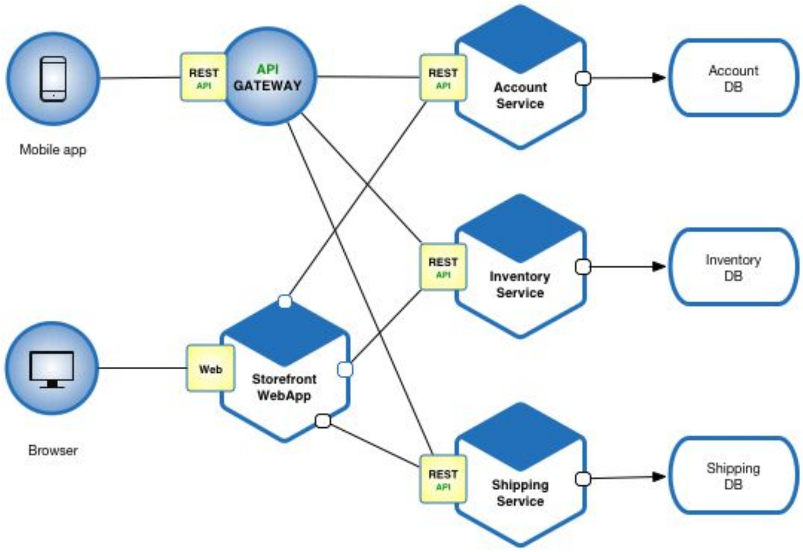
- **Easier to develop.** As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.
- **Easier to deploy.** You need to deploy your application only once instead of performing multiple deployments of different files.
- **Easier to test and debug.** Since a monolithic app is a single indivisible unit, you can run end-to-end testing much faster.

Monolith vs Microservices

Cons of monolithic architecture:

- **Understanding.** When a monolithic application scales up, it becomes too complicated to understand.
- **Making changes.** Any code change affects the whole system so it has to be thoroughly coordinated.
- **Scalability.** You cannot scale components independently, only the whole application.
- **New technology barriers.** It is extremely problematic to apply a new technology in a monolithic application because then the entire application has to be rewritten.

Monolith vs Microservices



Monolith vs Microservices

Pros of microservices:

- **Independent components.**
  - All the services can be deployed and updated independently, which gives more flexibility.
  - A bug in one microservice has an impact only on a particular service and does not influence the entire application.
  - It is much easier to add new features to a microservice application than a monolithic one.

Monolith vs Microservices

Pros of microservices:

- **Easier understanding.** Split up into smaller and simpler components, a microservice application is easier to understand and manage.
- **Better scalability.** Each element can be scaled independently. So the entire process is more cost- and time-effective than with monoliths when the whole application has to be scaled even if there is no need in it.

Monolith vs Microservices

Pros of microservices:

- **Flexibility in choosing the technology.** The engineering teams are not limited by the technology chosen from the start. They are free to apply various technologies and frameworks for each microservice.
- **The higher level of agility.** Any fault in a microservices application affects only a particular service and not the whole solution. So all the changes and experiments are implemented with lower risks and fewer errors.

Monolith vs Microservices

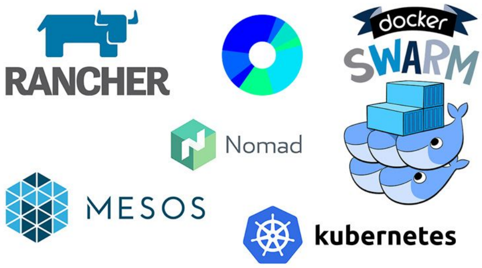
Cons of microservices:

- **Extra complexity.** Since a microservices architecture is a distributed system, you have to choose and set up the connections between all the modules and databases.
- **System distribution.** A microservices architecture is a complex system of multiple modules and databases so all the connections have to be handled carefully.
- **Testing.** A multitude of independently deployable components makes testing a microservices-based solution much harder.

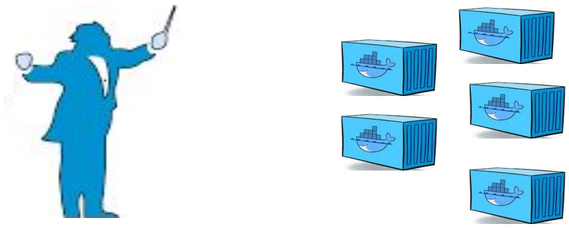
3 Container Orchestration

Orchestration

- Containers are great, but when you get lots of them running, at some point, you need them all working together in harmony to solve business problems.
- Tools to manage, scale, and maintain containerized applications are called orchestrators, and the most common example of this is **Kubernetes**.



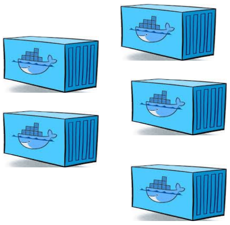
Orchestration



Container orchestration is used to automate the following tasks at scale:

- Provisioning and deployments of containers
- Availability of containers
- Load balancing, traffic routing and service discovery of containers
- Health monitoring of containers

Orchestration



- Securing the interactions between containers.
- Configuring and scheduling of containers
- Allocation of resources between containers

4 What is Kubernetes?

kubernetes

What is Kubernetes?

- Born in Google
- Donated to CNCF in 2014
- Open source (Apache 2.0)
- v1.0 July 2015
- Written in Go/Golang
- Often shortened to k8s

K~~ubernete~~s
K8 s

CNCF: Cloud Native Computing Foundation

What is Kubernetes?

- Kubernetes is Open Source Orchestration system for Containerized Applications.
- Kubernetes is a platform that eliminates the manual processes involved in deploying containerized applications.
- Kubernetes used to manage the State of Containers.
- Start Containers on Specific Nodes.
- Restart Containers when gets Killed.
- Move containers from one Node to Another.

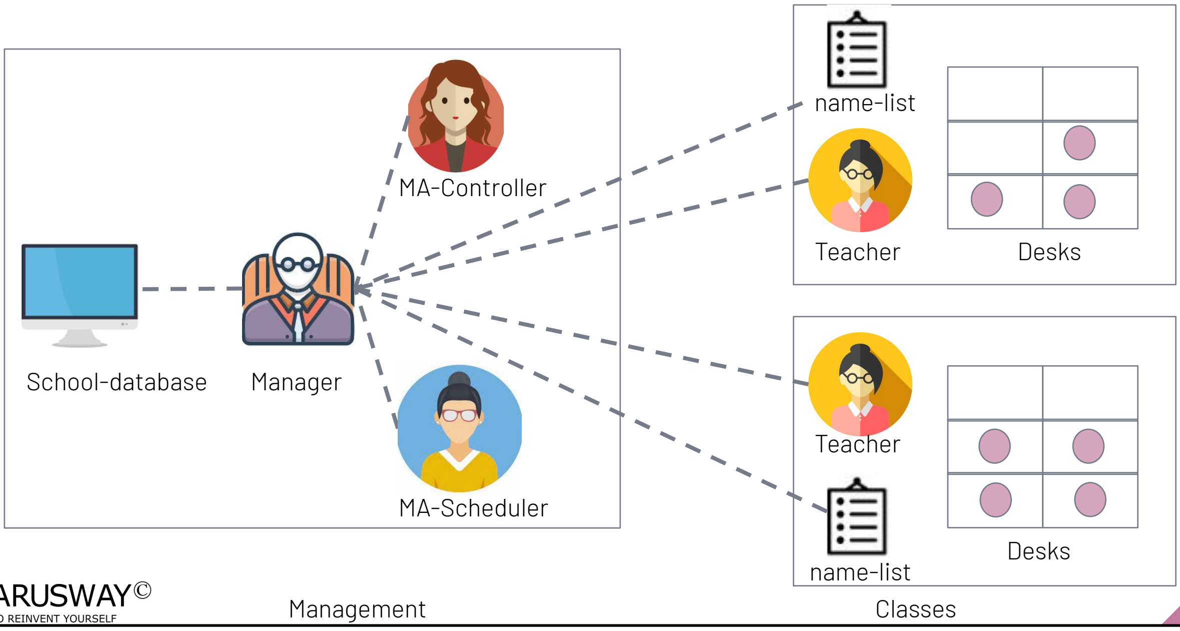
What is Kubernetes?

Kubernetes supplies you with:

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

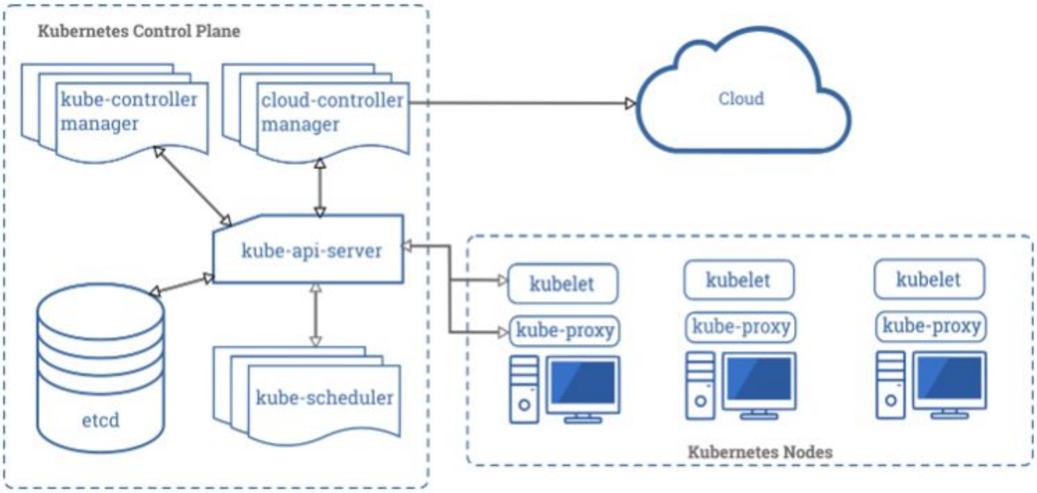
5 Kubernetes Components

Control Plane Components



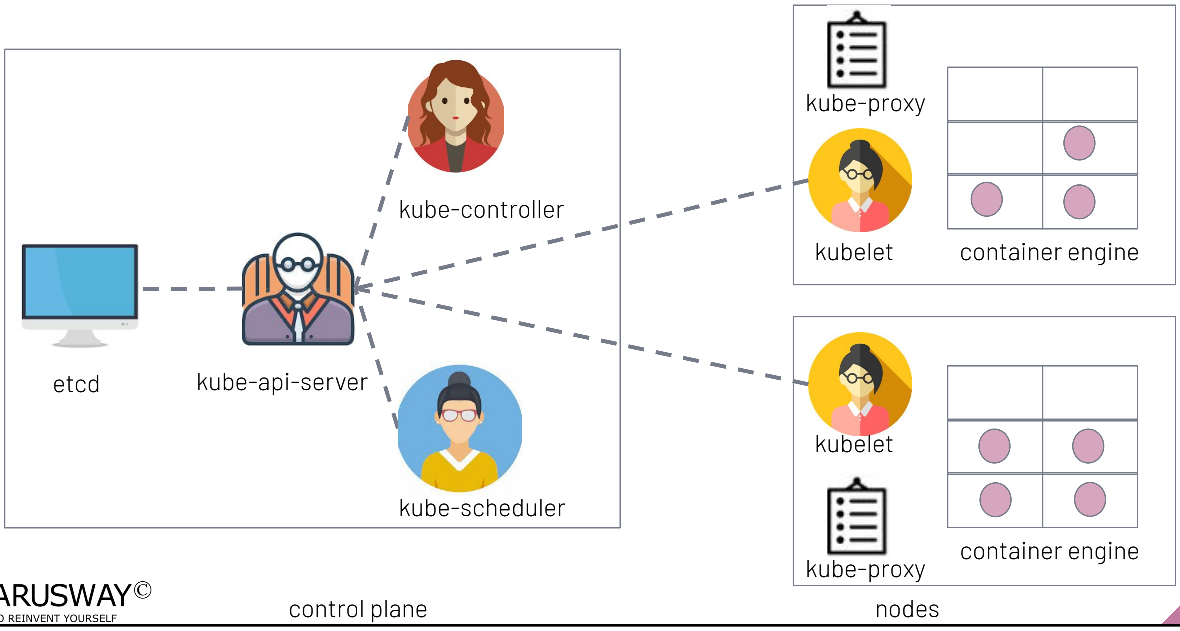
CLARUSWAY©
WAY TO REINVENT YOURSELF

Control Plane Components



CLARUSWAY©
WAY TO REINVENT YOURSELF

Control Plane Components



Control Plane Components

kube-apiserver:

- The API server exposes the Kubernetes API and it is the front end for the Kubernetes control plane.
- All clients and other applications interact with kubernetes strictly through the API Server.
- Acts as the gatekeeper to the cluster by handling authentication and authorization.
- It is the front-end to the backing datastore.

Control Plane Components

etcd:

- Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
- Stores objects and config information.

Control Plane Components

kube-controller-manager:

- Monitors the cluster state via the apiserver and steers the cluster towards the desired state.
- Control plane component that runs controller processes.
- Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Control Plane Components

kube-controller-manager:

- There are many different types of controllers. Some examples of them are:
  - Node controller: Responsible for noticing and responding when nodes go down.
  - Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
  - ServiceAccount controller: Create default ServiceAccounts for new namespaces.

Control Plane Components

kube-scheduler:

- Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Control Plane Components

cloud-controller-manager:

- The cloud controller manager lets you link your cluster into your cloud provider's API.
- The cloud-controller-manager only runs controllers that are specific to your cloud provider.

Node Components

kubelet:

- An agent that runs on each node in the cluster.
- It makes sure that containers are running in a Pod.

Node Components

kube-proxy:

- kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
- kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.
- Performs connection forwarding or load balancing for Kubernetes cluster services.

Node Components

Container Runtime Engine:

- A fundamental component that empowers Kubernetes to run containers effectively.
- It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

THANKS!

Any questions?

CLARUSWAY©
WAY TO REINVENT YOURSELF