Kubernetes-1

CLARUSWAY©

WAY TO REINVENT YOURSELF

Table of Contents

- Monolith vs Microservices
- Orchestration
- Declarative vs Imperative
- What is Kubernetes?
- Why you need Kubernetes?
- Kubernetes components
- kubectl

1

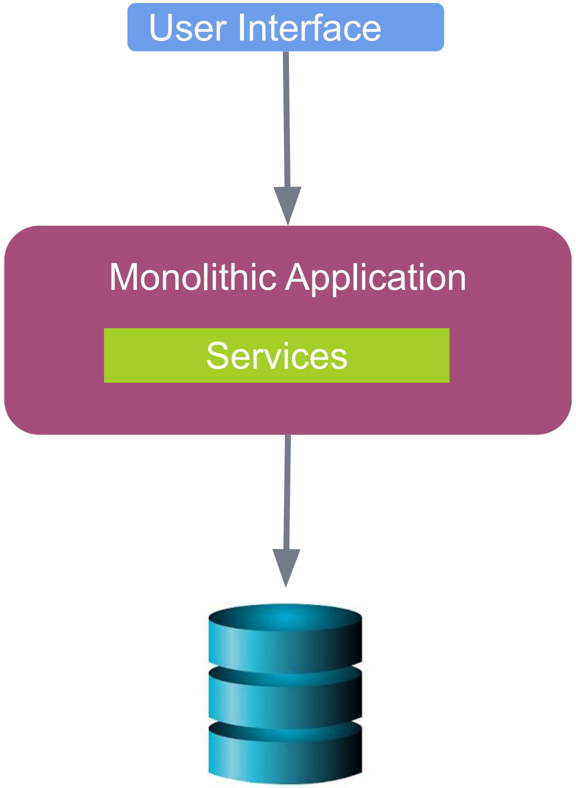
Monolith vs Microservices

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.



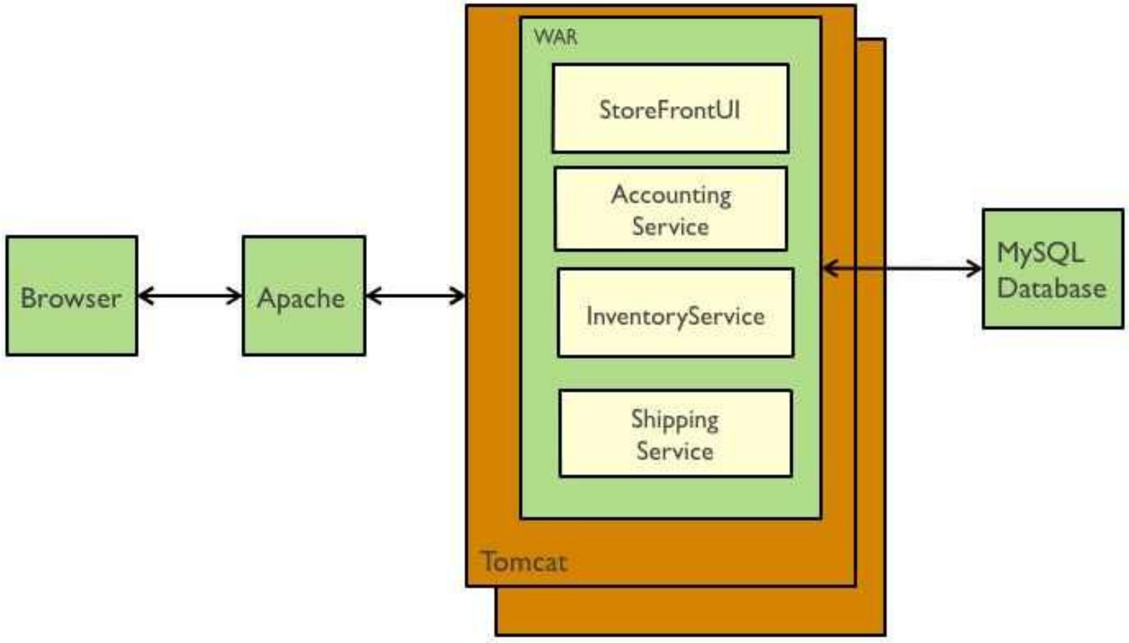
Monolith vs Microservices

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:

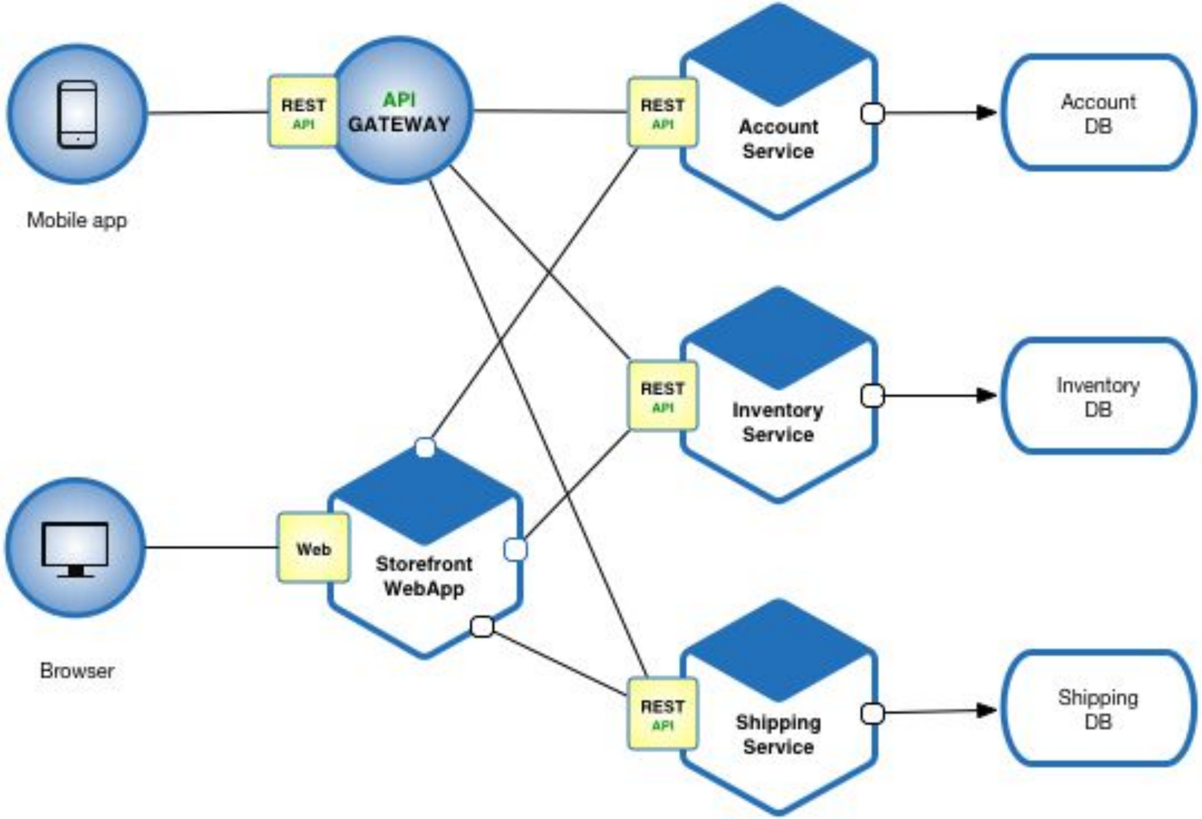
- **Easier to develop.** As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.
- **Easier to deploy.** You need to deploy your application only once instead of performing multiple 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

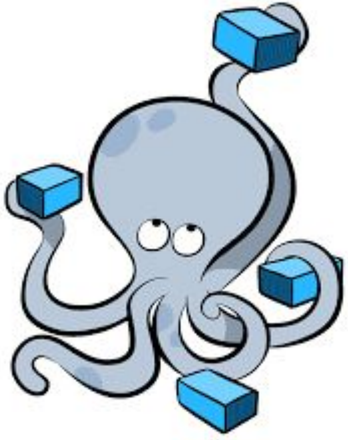
Cons of microservices:

- **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.

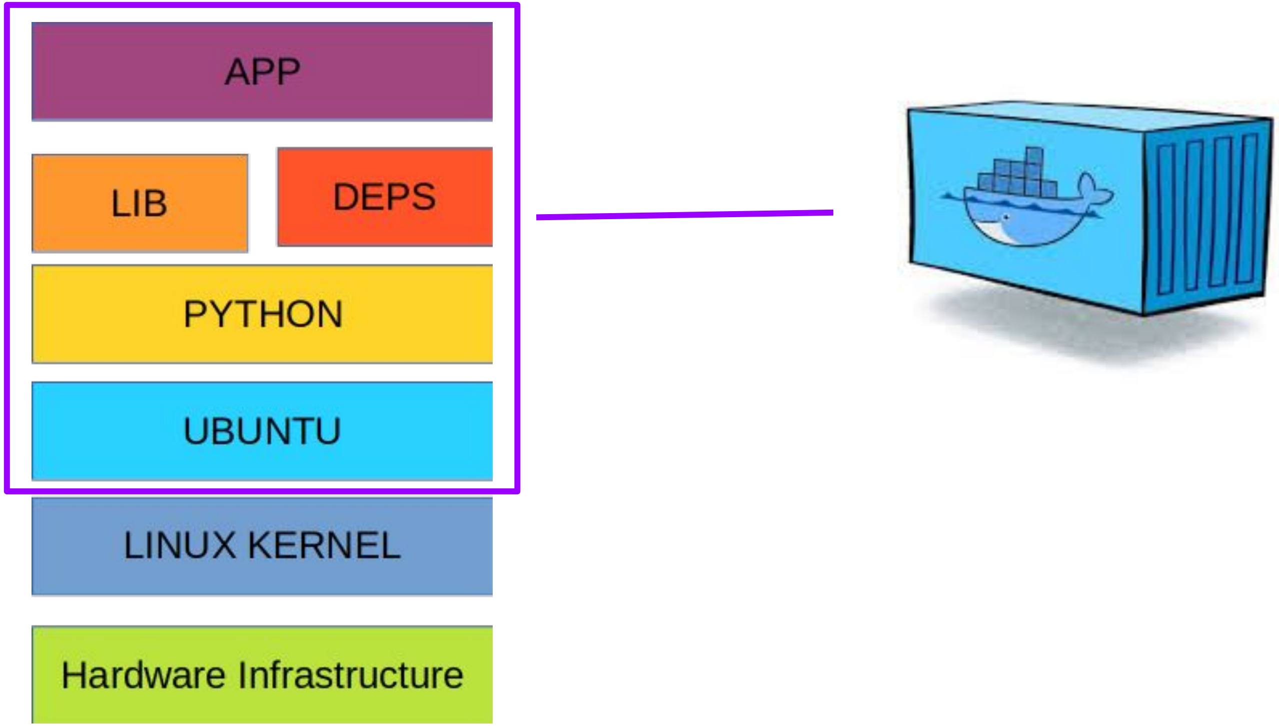
- **System distribution.** A microservices architecture is a complex system of multiple modules and databases so all the connections have to be handled carefully.

- **Testing.** A multitude of independently deployable components makes testing a microservices-based solution much harder.

2 Orchestration



Orchestration



APP

LIB

DEPS

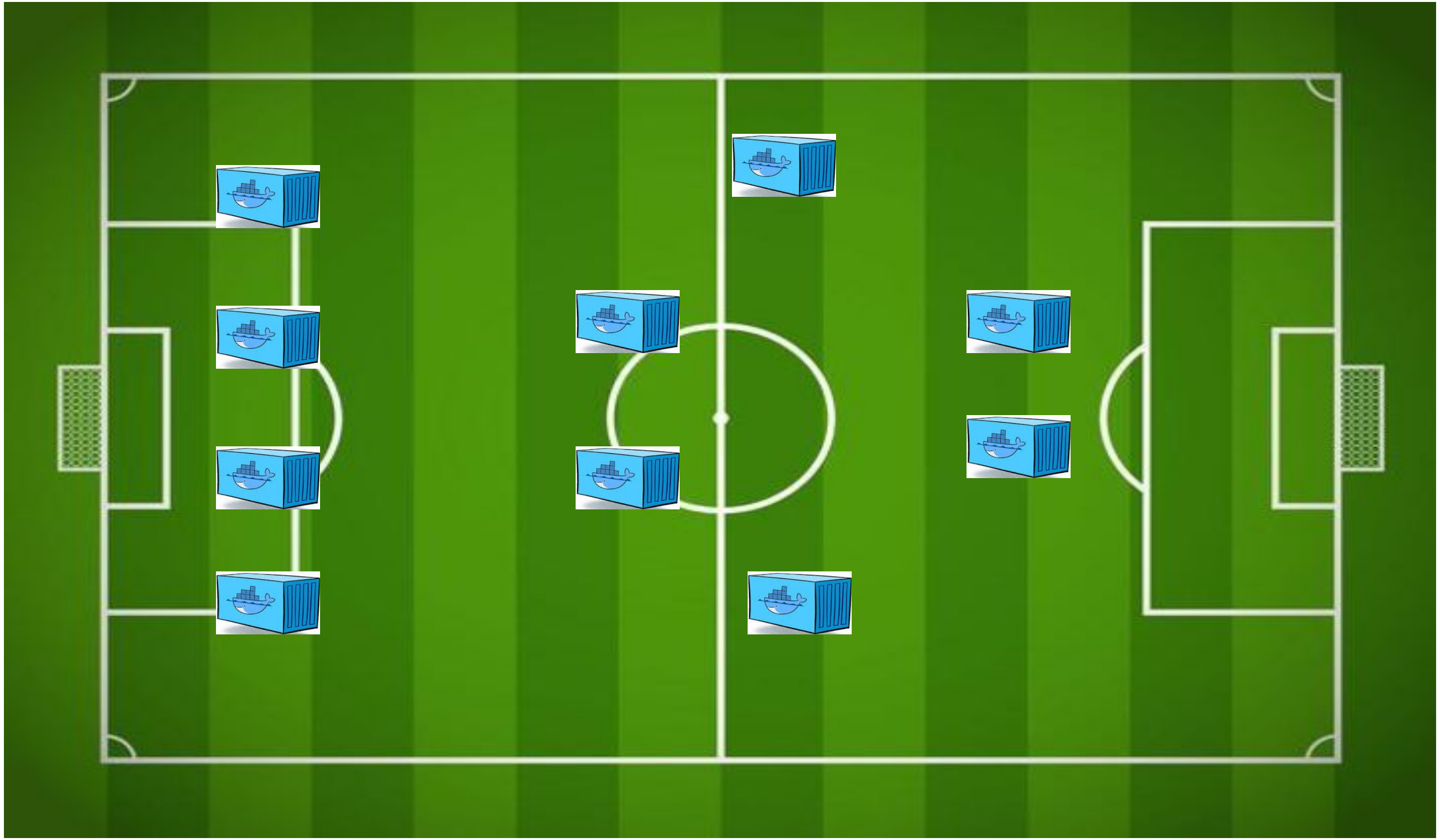
PYTHON

UBUNTU

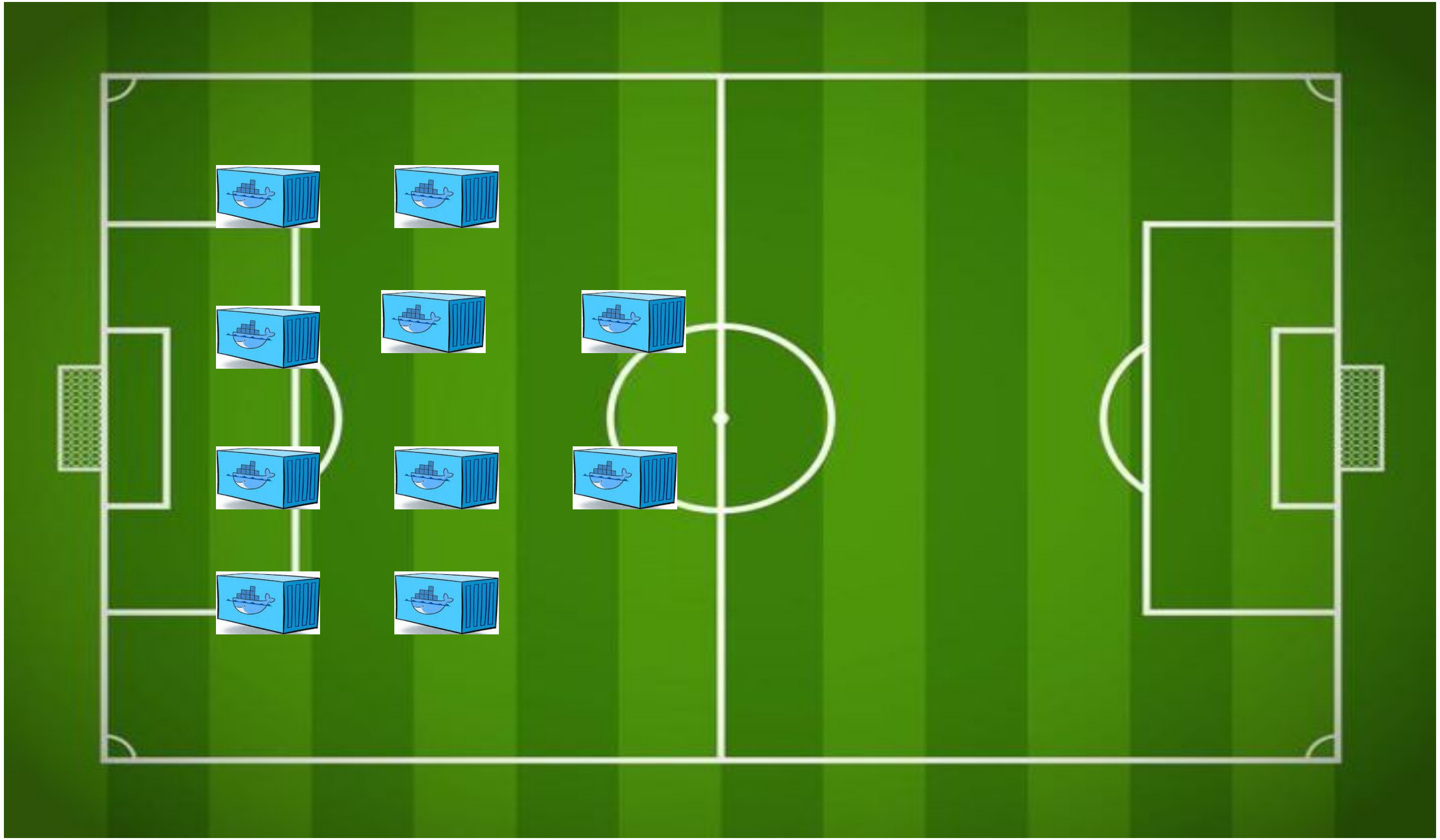
LINUX KERNEL

Hardware Infrastructure

Orchestration

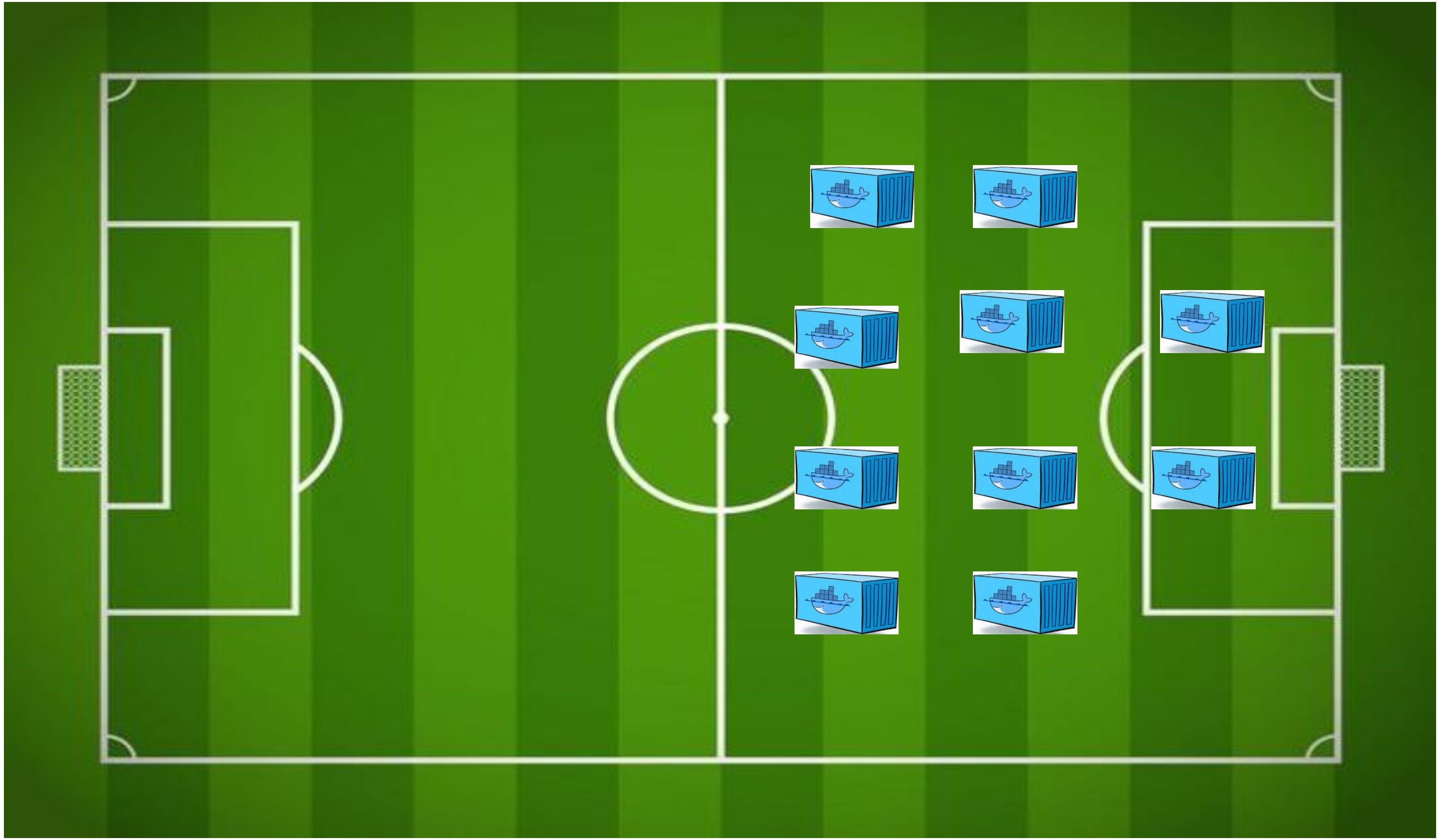


Orchestration

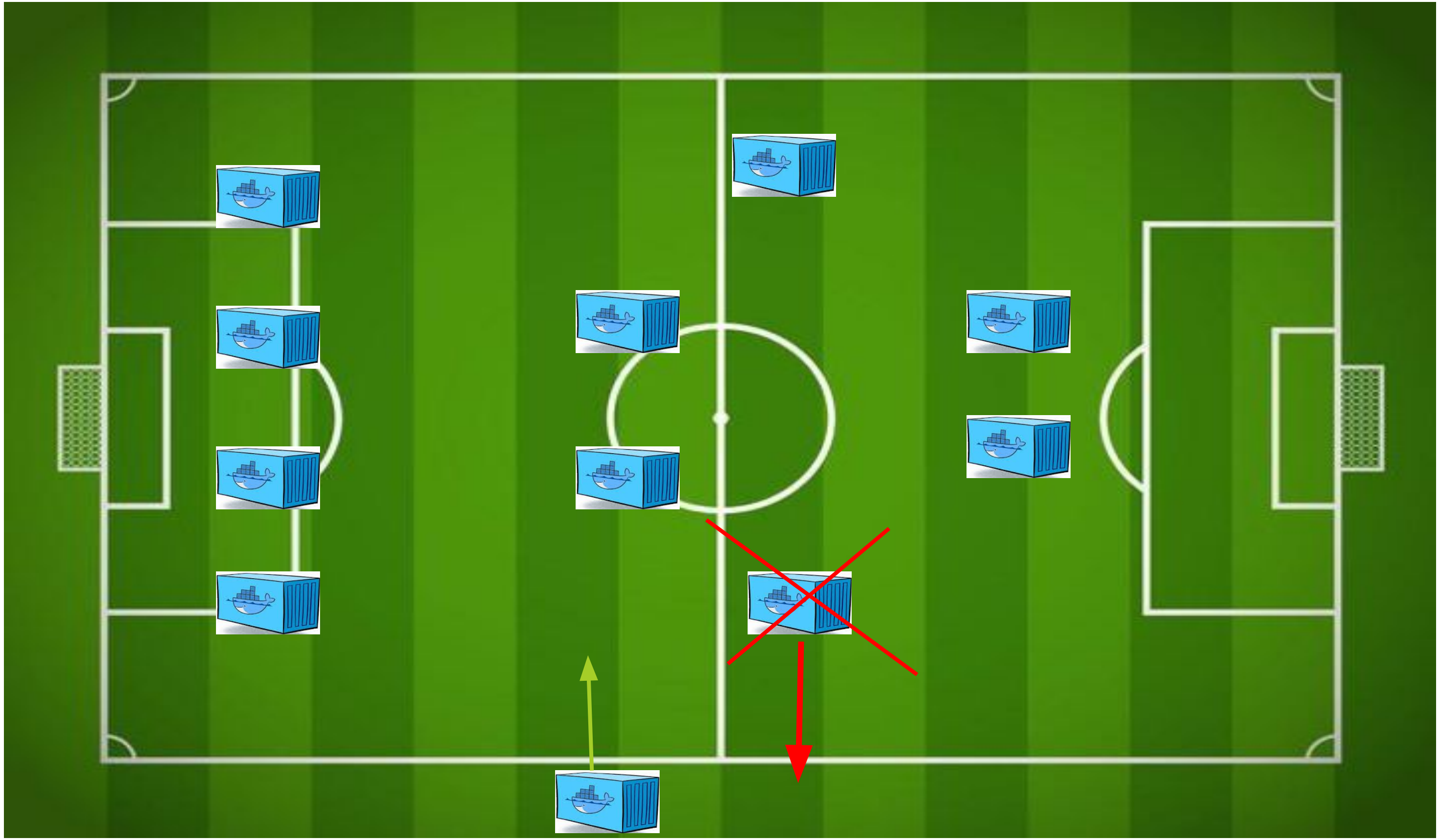


CLARUSWAY©
WAY TO REINVENT YOURSELF

Orchestration



Orchestration



Orchestration

Orchestration

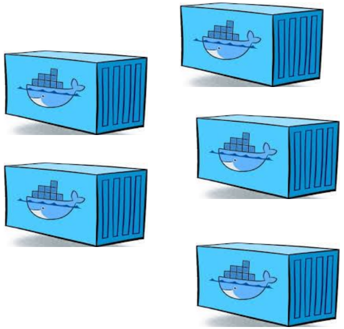
Orchestration

- 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**.

RANCHER
docker
SWARM
Nomad
MESOS
kubernetes

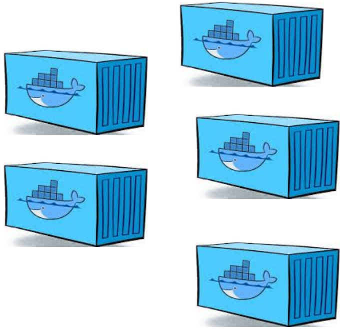
Orchestration



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

3

Declarative vs Imperative

Declarative vs Imperative

imperative focuses on **how** and declarative focuses on **what**.

Imperative approach:

1. Build the foundation
2. Put in the framework
3. Add the walls
4. Add the doors and windows

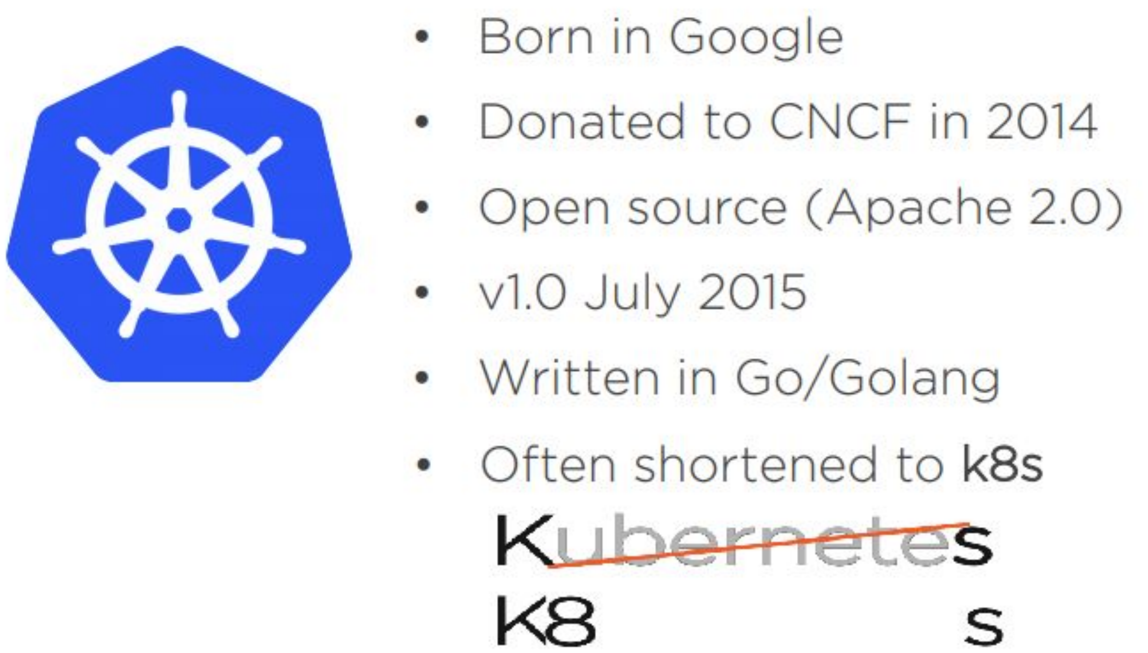
Declarative approach:

I want a tiny and cute house.

4

What is 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.

5

Why you need Kubernetes?

Why you need Kubernetes?

Containers are a perfect way to get the applications packaged and run. In a production environment, you should manage the containers that run the applications and ensure no downtime.

Why you need Kubernetes?

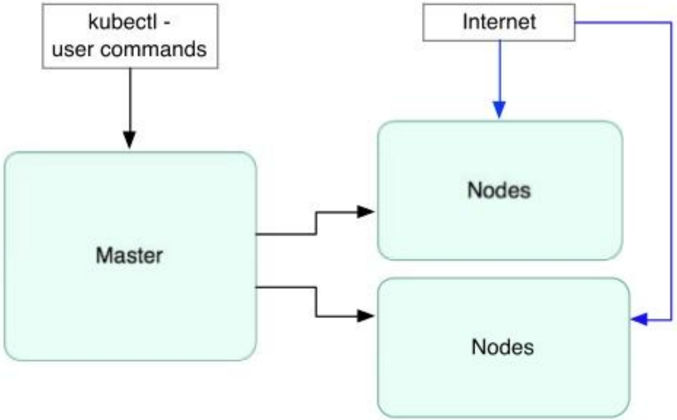
Kubernetes supplies you with:

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

6

Kubernetes Components

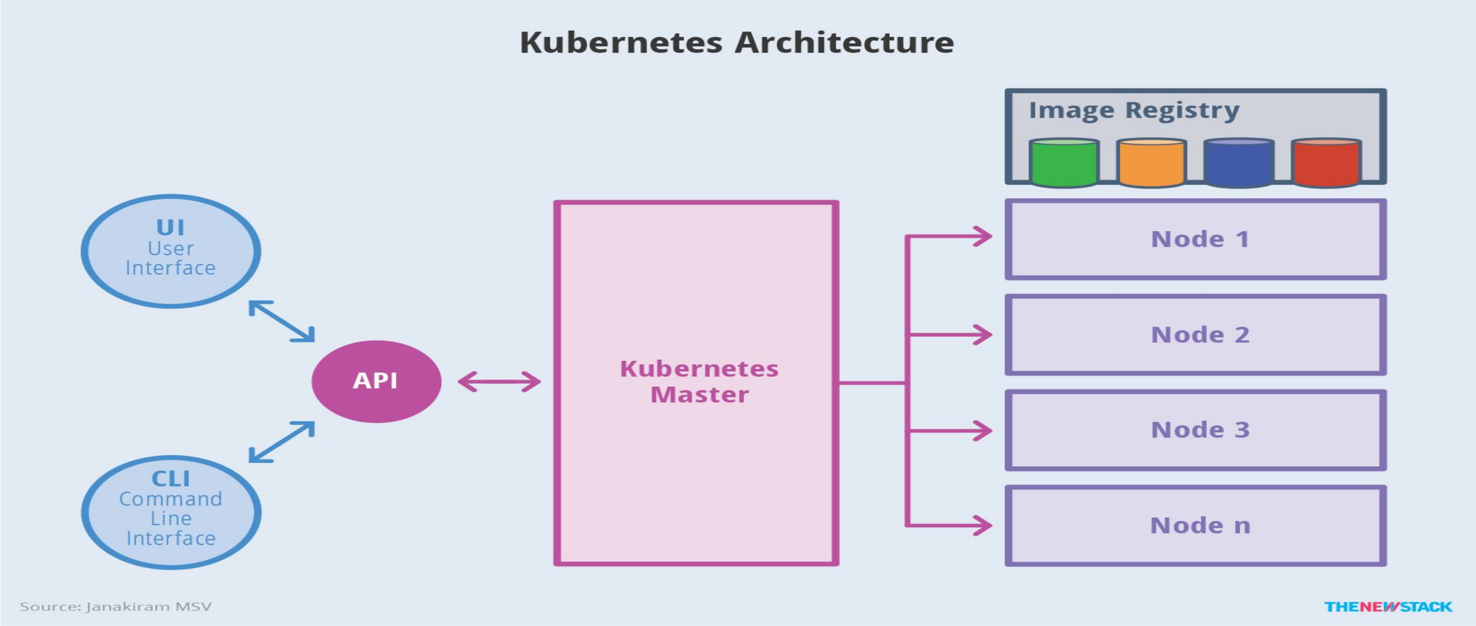
High Level Components



Kubernetes Components

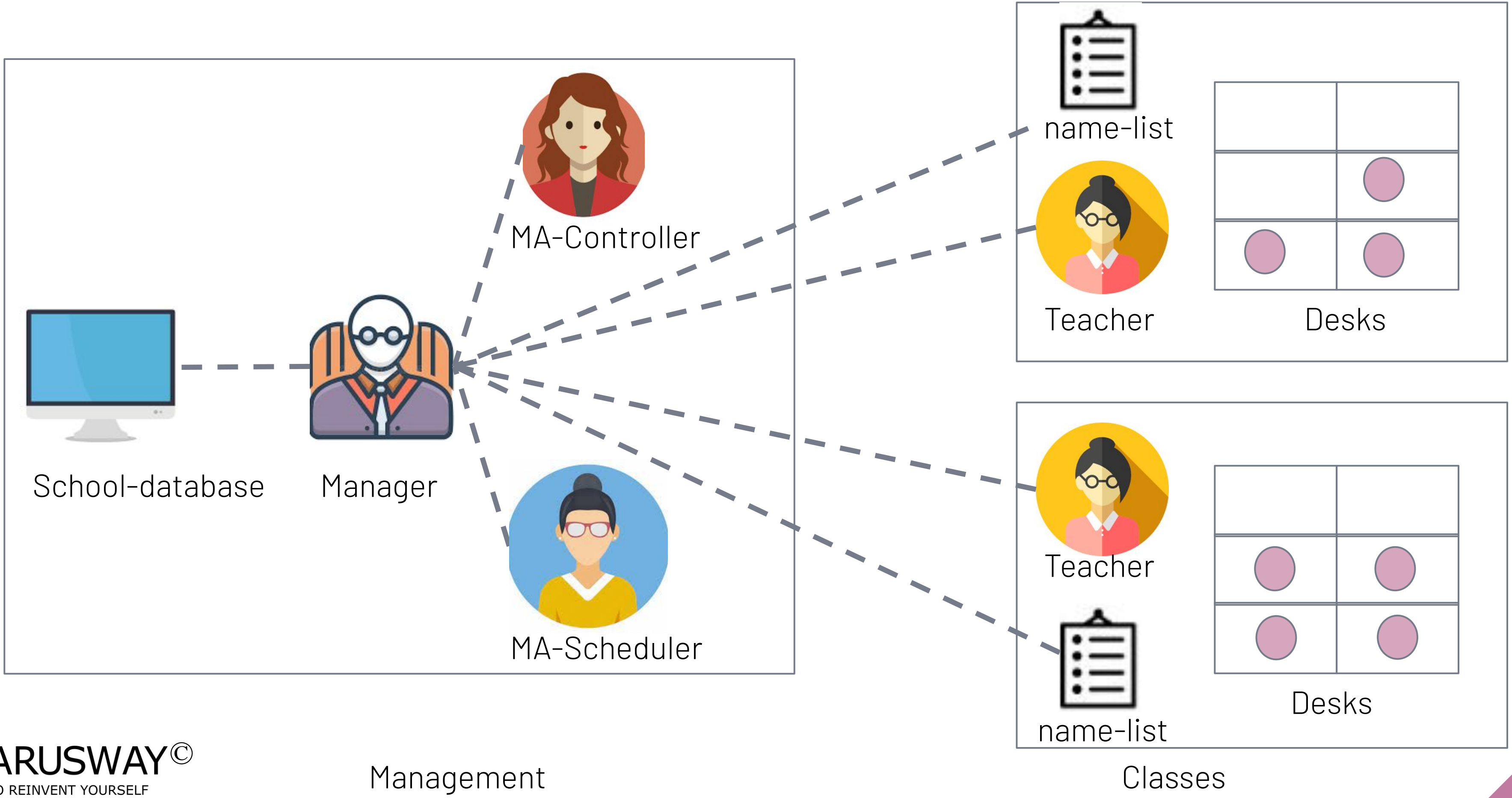
Kubernetes has the following main components:

- One or more master nodes
- One or more worker nodes.

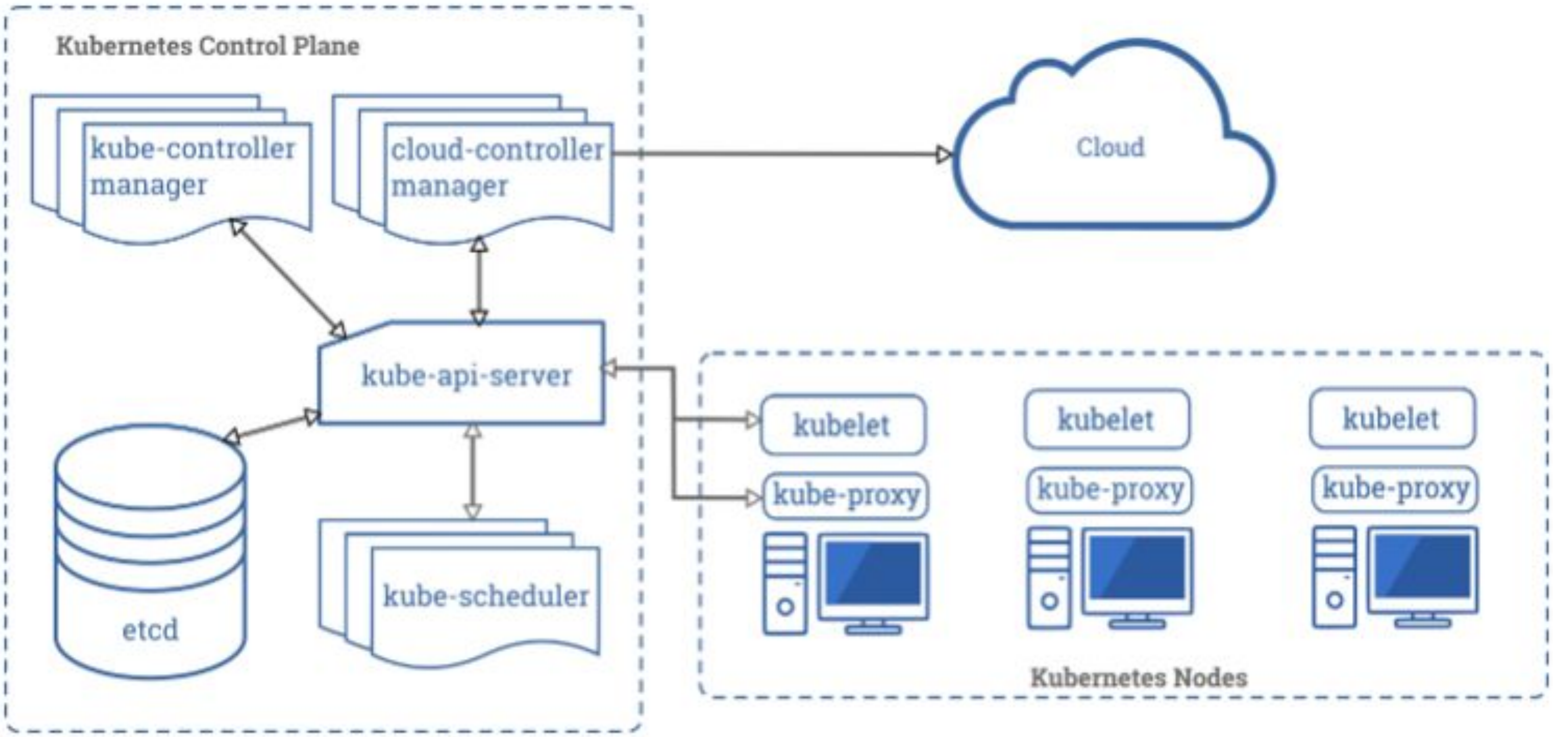


Source: Janakiram MSV

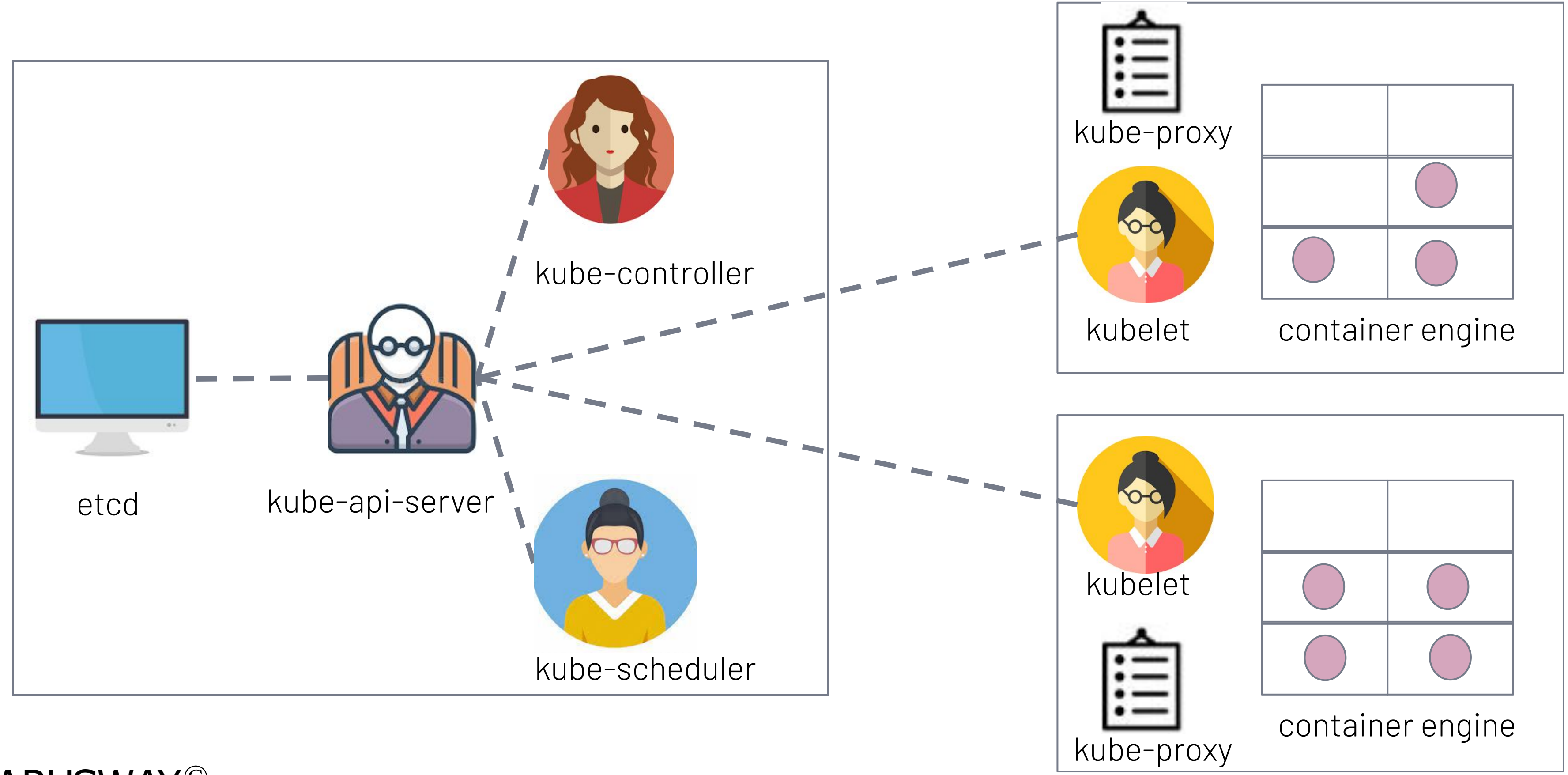
Control Plane Components



Control Plane Components



Control Plane Components



control plane

nodes

Control Plane Components

kube-apiserver:

- Provides a forward facing REST interface into the kubernetes control plane and datastore.
- 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, request validation, mutation, and admission control in addition to being the front-end to the backing datastore.

Control Plane Components

etcd:

- etcd acts as the cluster datastore.
- Purpose in relation to Kubernetes is to provide a strong, consistent and highly available key-value store for persisting cluster state.
- Stores objects and config information.

Control Plane Components

kube-controller-manager:

- Serves as the primary daemon that manages all core component control loops.
- Monitors the cluster state via the apiserver and steers the cluster towards the desired state

Control Plane Components

kube-scheduler:

- Verbose policy-rich engine that evaluates workload requirements and attempts to place it on a matching resource.
- Default scheduler uses bin packing.
- Workload Requirements can include: general hardware requirements, affinity/anti-affinity, labels, and other various custom resource requirements.

Node Components

kubelet:

- Acts as the node agent responsible for managing the lifecycle of every pod on its host.
- Kubelet understands YAML container manifests that it can read from several sources:
  - file path
  - HTTP Endpoint
  - etcd watch acting on any changes
  - HTTP Server mode accepting container manifests over a simple API.

Node Components

kube-proxy:

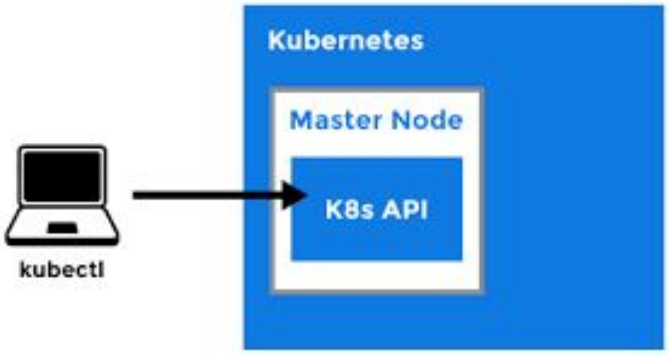
- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.
- Available Proxy Modes:
    - Userspace
    - iptables
    - ipvs (default if supported)

Node Components

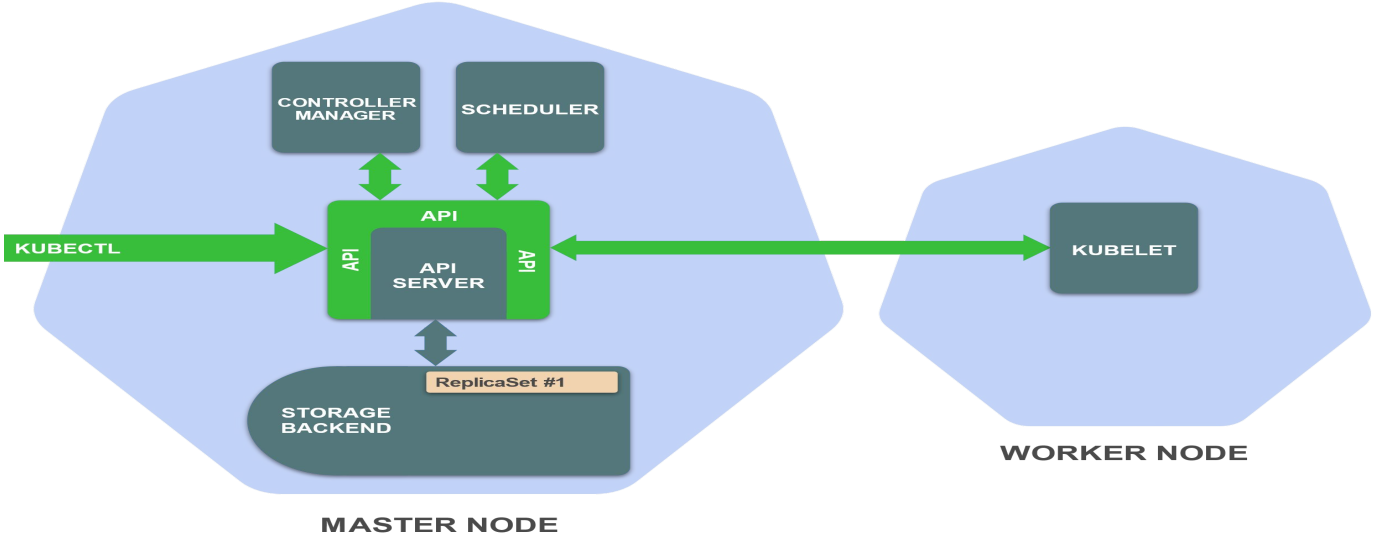
Container Runtime Engine:

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)
  - Cri-o
  - Rkt
  - Kata (formerly clear and hyper)
  - Virtlet (VM CRI compatible runtime)

7 kubectl



kubectl

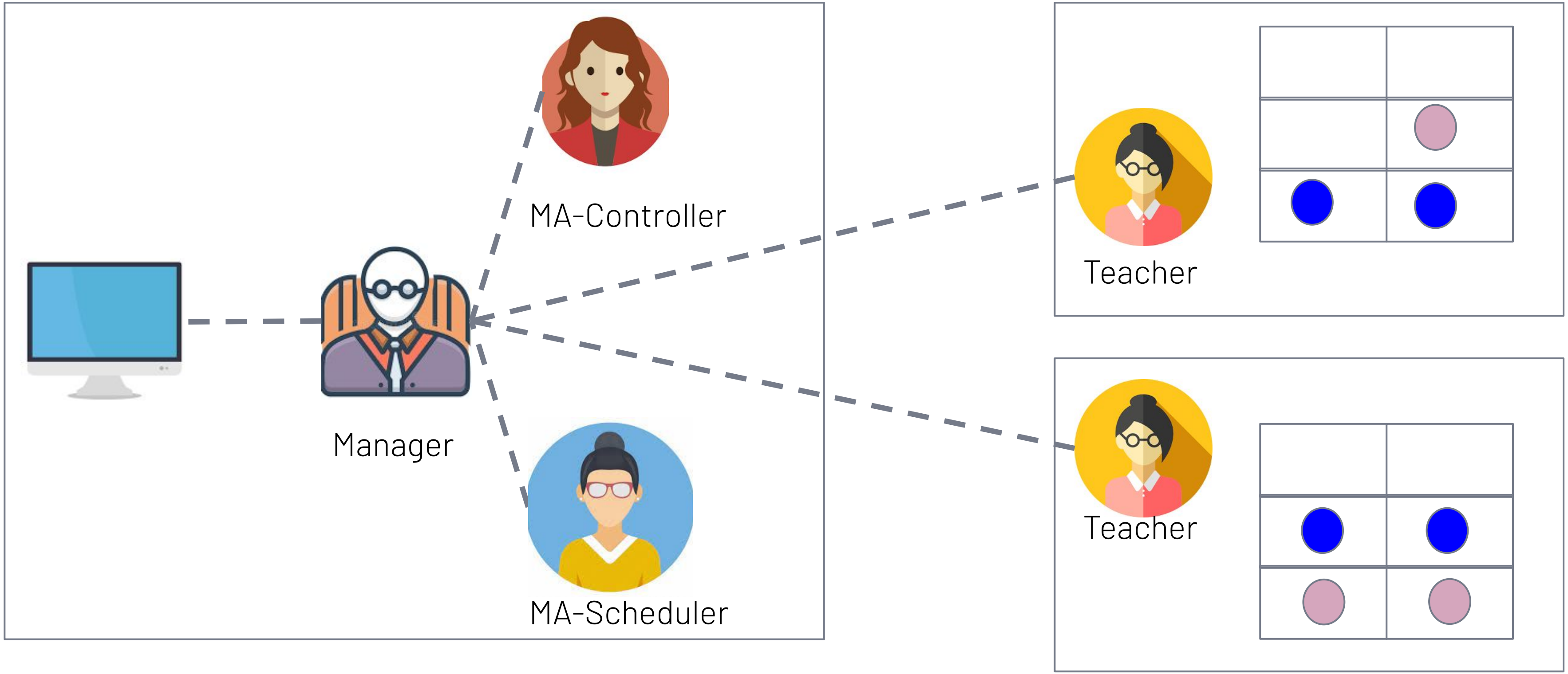


- **kubectl** is (almost) the only tool we'll need to talk to Kubernetes
- It is a rich CLI tool around the Kubernetes API
- Everything you can do with kubectl, you can do directly with the API
- kubectl can be pronounced "Cube C T L", "Cube cuttle", "Cube cuddle".

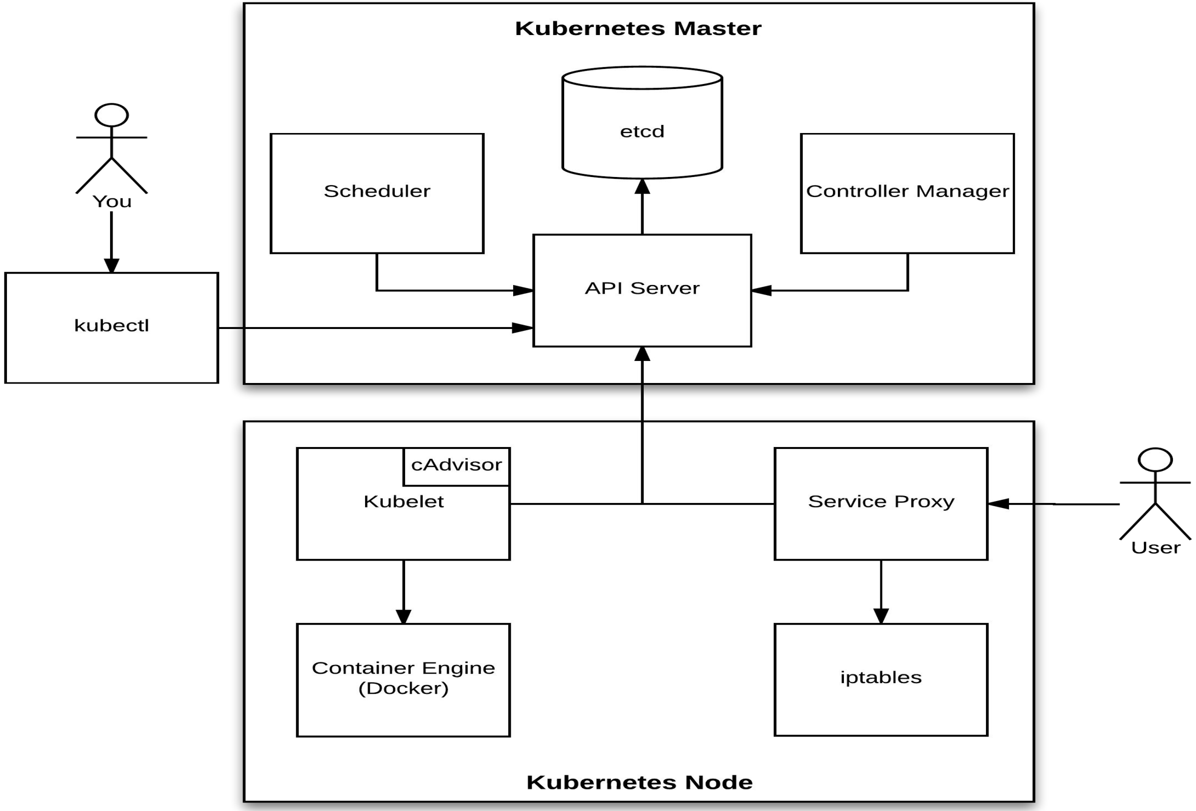
THANKS!

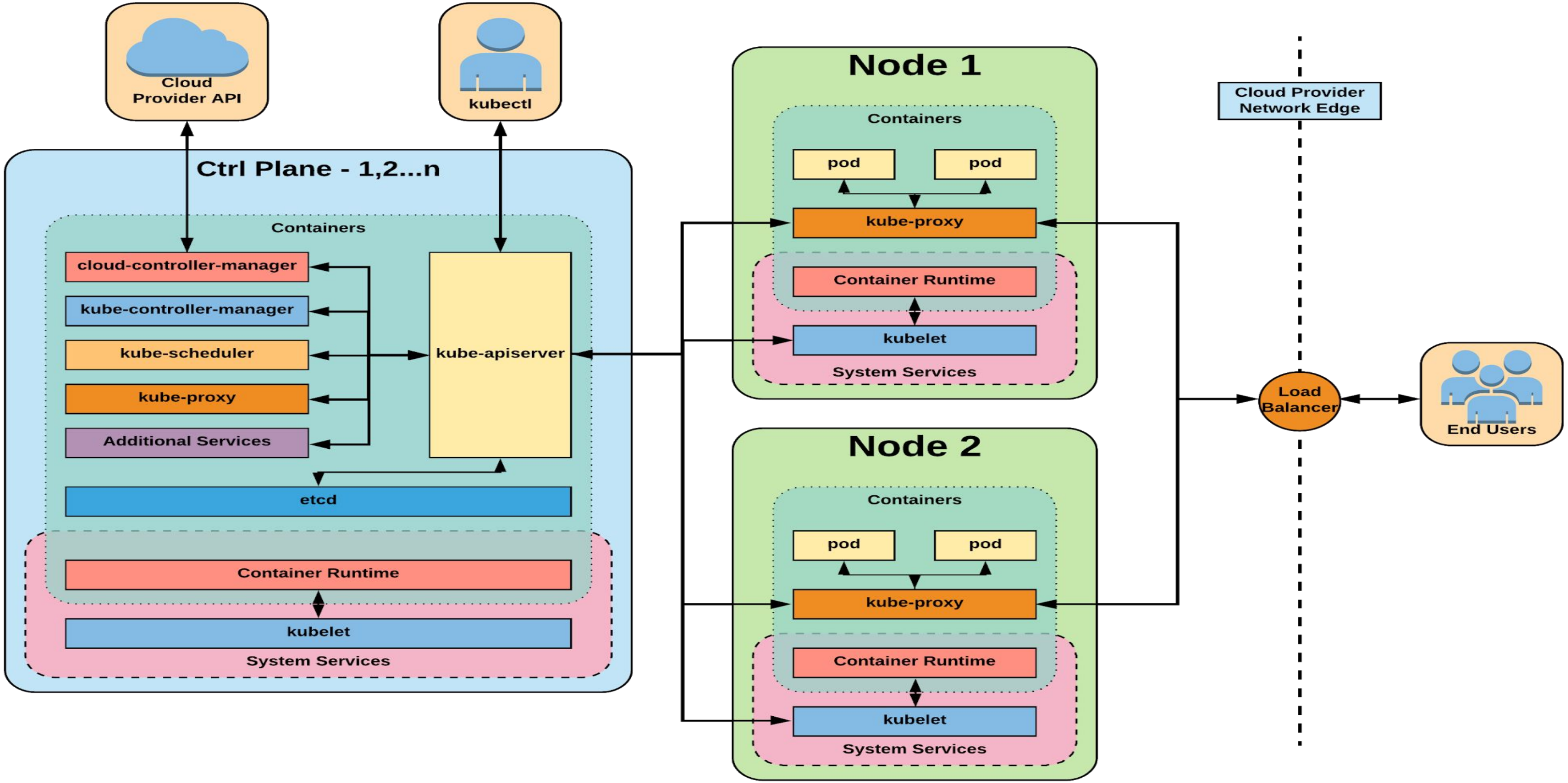
Any questions?

Control Plane Components



Node Components




Cloud Provider API
kubectl
Ctrl Plane - 1,2...n
Containers
cloud-controller-manager
kube-controller-manager
kube-scheduler
kube-proxy
Additional Services
kube-apiserver
etcd
Container Runtime
kubelet
System Services
Node 1
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Node 2
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users