Kubernetes - By P Nageswara Rao

Gamut Gurus Technologies

&

Wiculty Learning Solutions

Email: info@Gamutgurus.com, **info@wiculty.com**

Contact: +91- 97393 68768

What is Kubernetes?

Kubernetes is:

- Kubernetes is an Orchestrator for containerized Applications
- Kubernetes is a Data centre OS

Docker & Kubernetes Integration

What is Docker & Kubernetes?

Docker: You write your application code in your favourite language, Build it, Test it, then use Docker to package it and ship it. Yes… That's what Docker is.

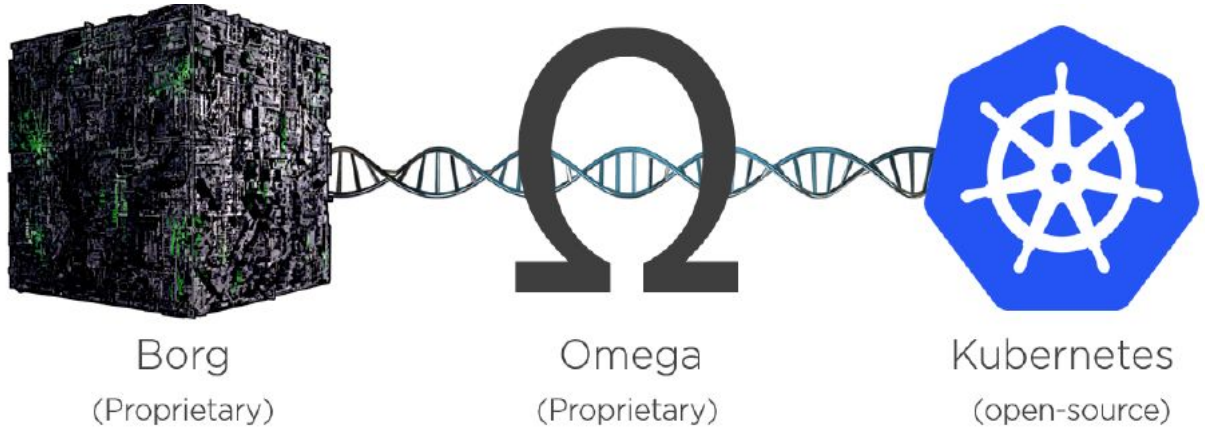
Then, who will handle the final step of running it on Test or Prod?

Hello Kubernetes 

Yes, It's Kubernetes! Kubernetes Orchestrates your application!!

Kubernetes - Background

- Kubernetes came out of Google! It was open-sourced on 2014 and Handed over to “Cloud Native Computing Foundation (CNCF).
- Google has been running many of it’s systems on containers for years even before Docker came along.
- Google goes through billions of containers per week for applications like Gmail, Google Search , GFS (Google File System) ..etc. Lots of Containers hun??
- Kubernetes is written in Go (Golang)
- Logo – Meaning “the person who steers the ship”
- Kubernetes short name K8S.



Container World Challenges

Application deployment & operational challenges:

- Fail over if one or more nodes experience an outage
- Scale-up & Scale-down: ability to add or remove containers based on application demand
- Zero downtime releases (ZDR)
- Application updates & rollbacks
- Health checks & Self healing systems
  - Networking after scaling and self healing, logs, alerting.. Etc
- Trafic routing and Load Balancing (LB), & More…

What is an Orchestrator?

What is an Orchestrator?

Orchestrator is a system that performs below without you having to **supervise**.

- Deploy the application
- Scale it up and down
- Performs rolling updates
- Rollbacks

Kubernetes Advantage:

- All these we can achieve with declarative configurations.
- Declare desired state of the system rather than executing a series of instruction.

Example:

- Declare how many containers, replicas are required for your application (replicas = 3)
- It’s not about just creating replicas, but it will continuously monitor to ensure the desired state.

Orchestrator - Simple Analogy

Foot ball team is made of individuals

- No two are same
- Each has different role to play in the team

- Some defend, some attack, some are great at passing
- Some are great at shooting
- LW/RW-wingers, LB/RB-backs, AM-attacking mid-fielder

- Coach gives everyone a position and organizes them into a team with purpose
- Coach makes sure that team maintain the formation and stick to the plan

KUBERNETES – By P Nageswara Rao – Gamut Gurus

- In the sports world we call this coaching.
- In the application world we call it orchestration.

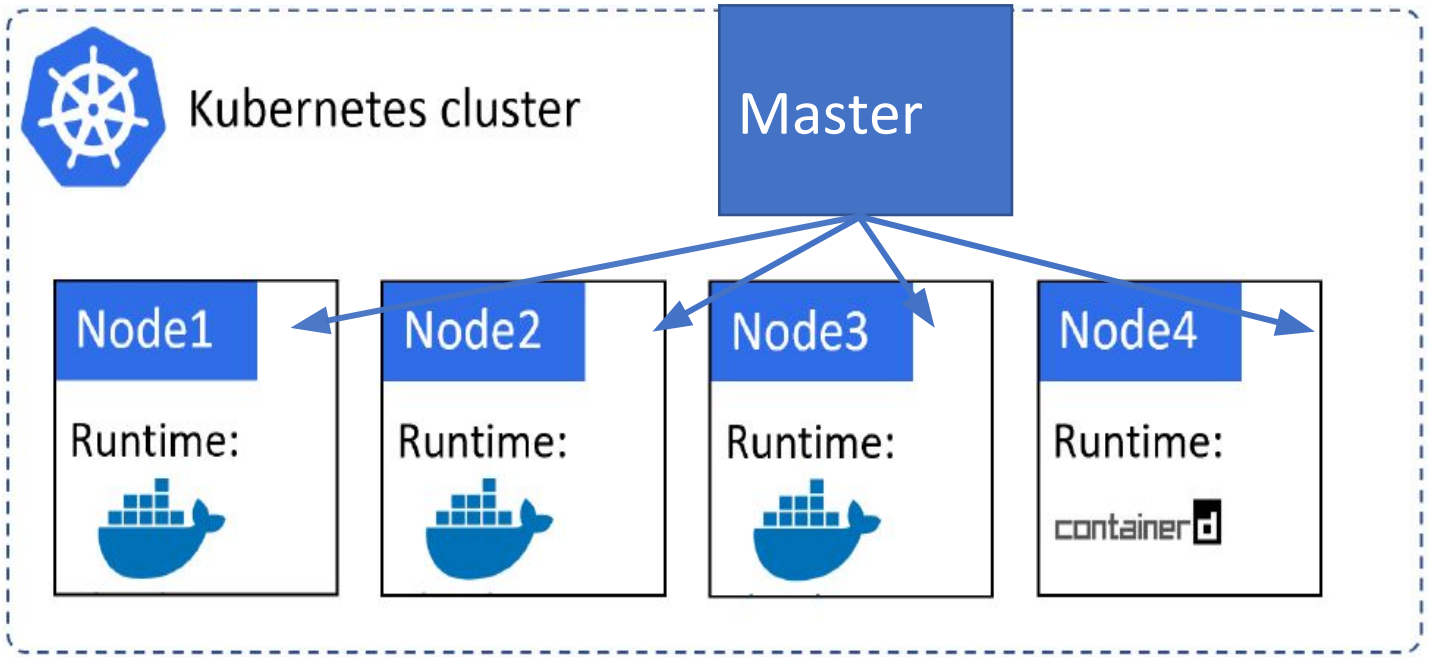
In Application World:

- Some serve Web pages
- Some do authentication
- Some do Searches
- Some do store data

- Kubernetes comes along a bit like the coach in the football analogy – and organizes everything into a useful app and keeps things running smoothly.

Kubernetes Cluster

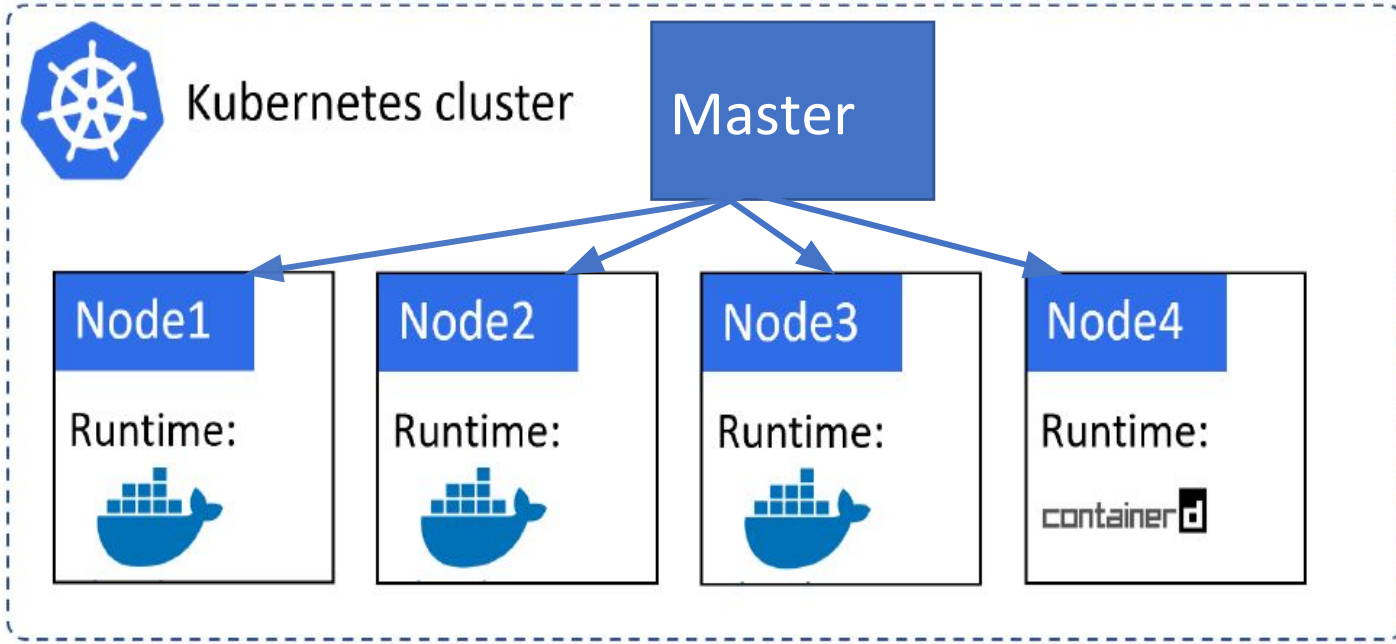
- The cluster is made up of one or more *Masters*, and a bunch of *Nodes*.



- You might have K8S cluster with 10 nodes to run your production application.
- Behind the scenes each node is running Docker as it's container runtime.
- This means, that Docker is the low-level technology that starts, stops containers etc.
- And.. K8S looks after big picture things like.. Deciding when to scale up or down, update, rollback ..etc.

Kubernetes Cluster

- Kubernetes abstracts the runtime
- Container runtime can be 'Docker' or 'Containerd'
- Containerd is overtaking Docker



- Containerd is stripped-down version of Docker.

- **Abstraction**:- Facilitated by Container Runtime Interface

using which you can integrate any 3rd party container runtime. Thanks CRI!

Kubernetes Vs Docker Swarm

- **Orchestrators**:
  - Docker swarm
  - Mesosphere's DCOS
  - Kubernetes

- During 2016 and 2017 Orchestrator wars, Kubernetes WON with active development and market-share.

Kubernetes - Data centre OS

What is an Data centre OS?

- Sometimes, we call Kubernetes as “Data Center OS”
- In modern Data Centre architectures, we are abandoning traditional view of Data Centre as collection of computers. **Instead..,**
- We are viewing it as a single large computer. **What does that mean?**

To understand that in better way.. Look at this example.

- A typical computer is a collection of CPU, RAM, Storage and Networking.
- Having OS, for example.. It’s rare for a developer to take care which CPU core or exact Memory address their application uses.
- We let the OS decide all of that! Means it abstracts away all of the CPU, Memory details.

Kubernetes - Data centre OS

- Now, apply this same abstraction to data center resources.

- A typical Data Center is collection of computers. In modern data center architecture, we view the data center as just a pool of compute, network and storage.

- This means we no longer need to care about which server or LUN our containers are running on - just leave this up to the data center OS.

- Gone are the days of taking your app and saying “Run this part of the app on this node, with this IP, on this specific LUN…“.

- In the cloud-native Kubernetes world, we’re more about saying “Hey Kubernetes, I’ve got this app and it consists of these parts… just run it for me please”.

- Kubernetes then goes off and does all the hard scheduling and orchestration work.

Data Centre OS - Simple Analogy

Kubernetes is Data Center OS

Think about the process of sending goods via courier services:

- You need to only package and put a label on it.
- Rest of the complex logistics like which plane, truck, drivers will be taken care by the courier service provider
- Kubernetes just does the same in Data Center

Conclusion

- You develop your application with Docker and then use Kubernetes to Run/Orchestrate it.

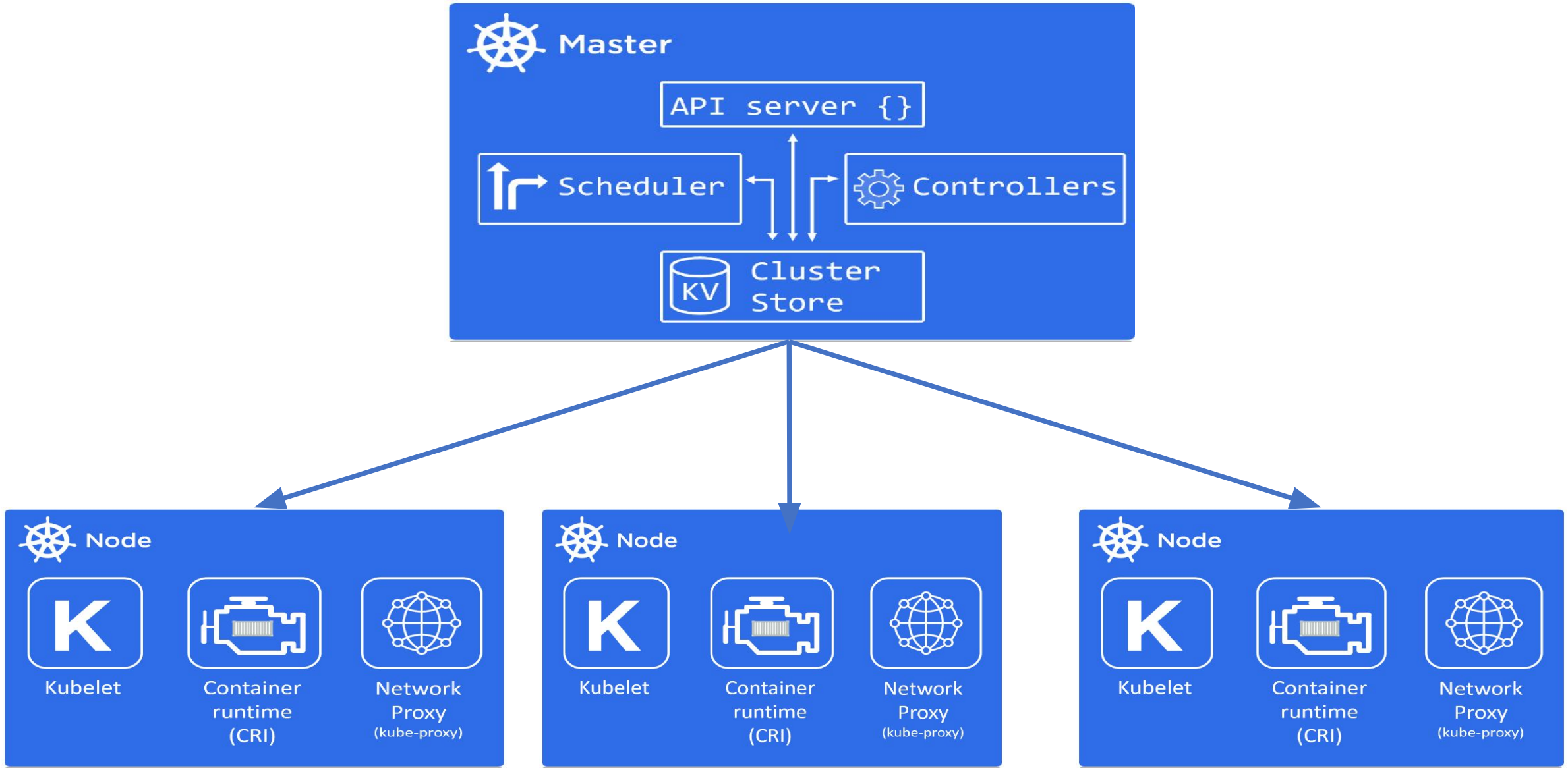
- Use Kubernetes to run your containerized Application in Test or Production with Orchestrator capabilities such as Deploy, Scale-up, Scale-down, Perform updates & rollback without much supervision.

- Kubernetes and Docker are Complimentary technologies.

- We integrate Docker and Kubernetes to Package the application code with dependencies and Run the application on Production or Test.

- Kubernetes is a Data centre OS (OS for Dev & Kubernetes for Data centre)

Kubernetes Architecture



Kubernetes Architecture

- A Kubernetes cluster is made of Masters and Nodes

- **Master/Nodes are Linux hosts that can be:**

  - VMs

  - Bare metal servers

  - Cloud instances.

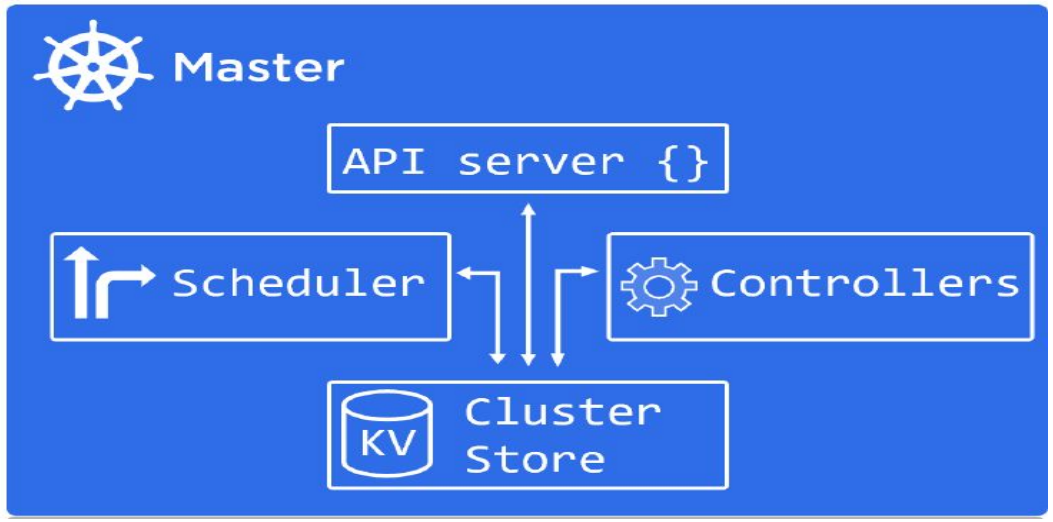
Kubernetes Architecture - Master

Master/Control Plane:

- The Master is in-charge of the cluster.

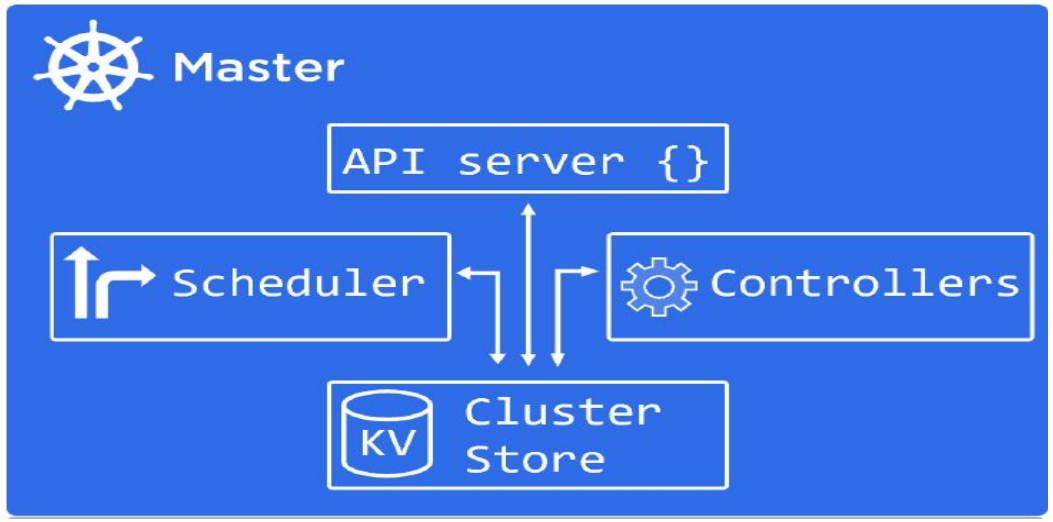
- Master:
  - Makes Scheduling decisions
  - Monitors the cluster
  - Implement the Changes
  - Respond to Events

- So, we call Master as Control Plane.



Kubernetes Architecture - Master

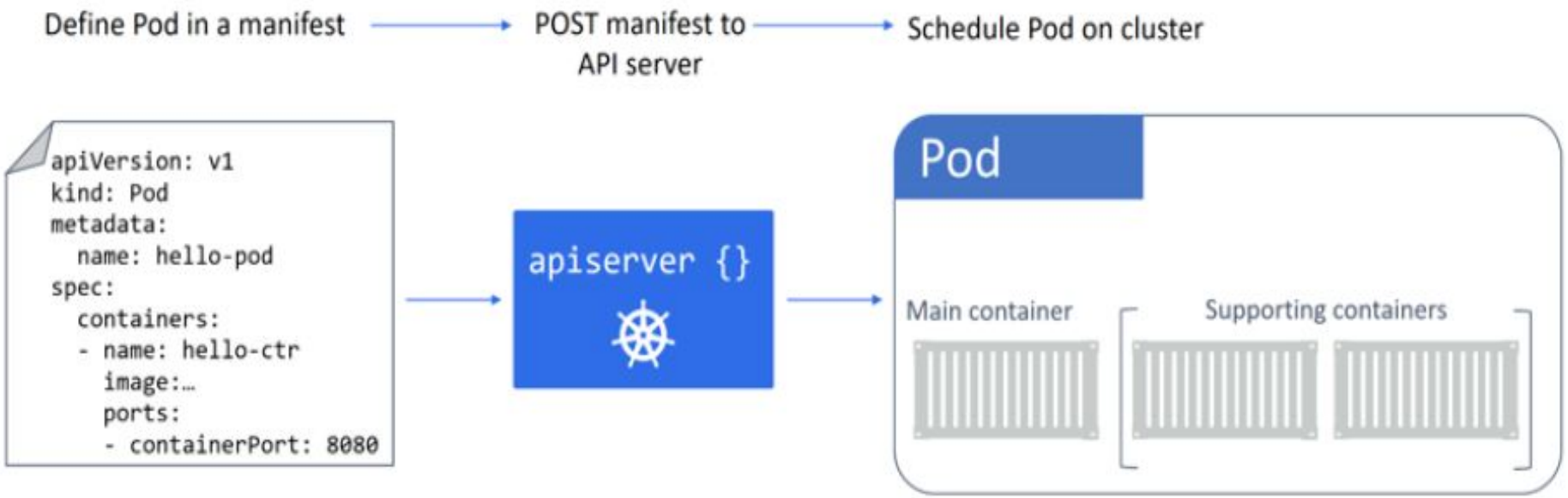
- **Control Plane Components:**
- API Server
- Cluster Store
- Control Manager
- Scheduler



API Server:

- Think of API server as the brains of the cluster.
- The API server is front door into Kubernetes.
- This is the only component in Control plane we interact with.

Kubernetes Architecture - Master



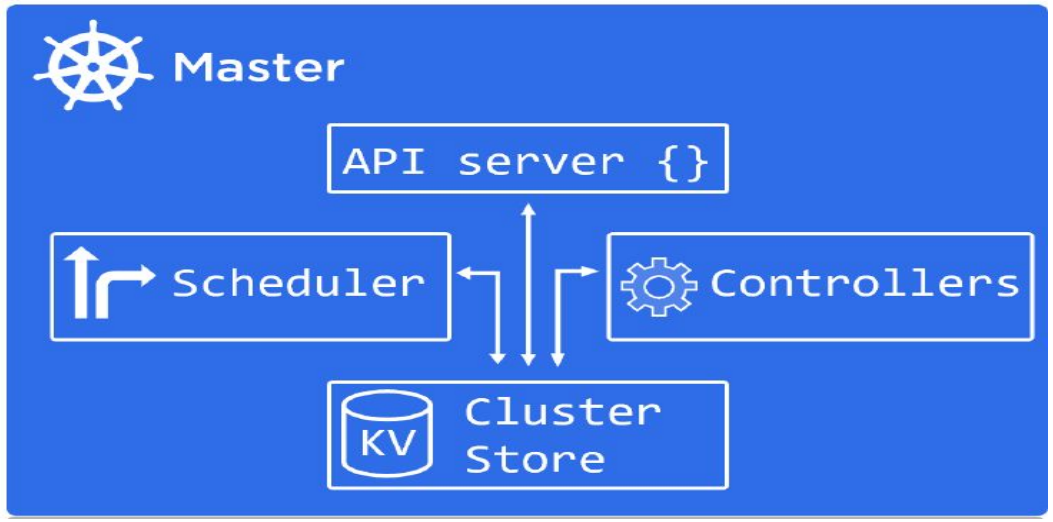
- We post YAML configuration files (manifests) to API server.
- This YAML file is validated, persisted to the Cluster store and deployed to the cluster.
- These YAML configuration files contains desired state of the application.

  Example: Which container image, Which ports to expose, How many Pod replicas .. Etc.

Kubernetes Architecture - Master

Cluster Store:

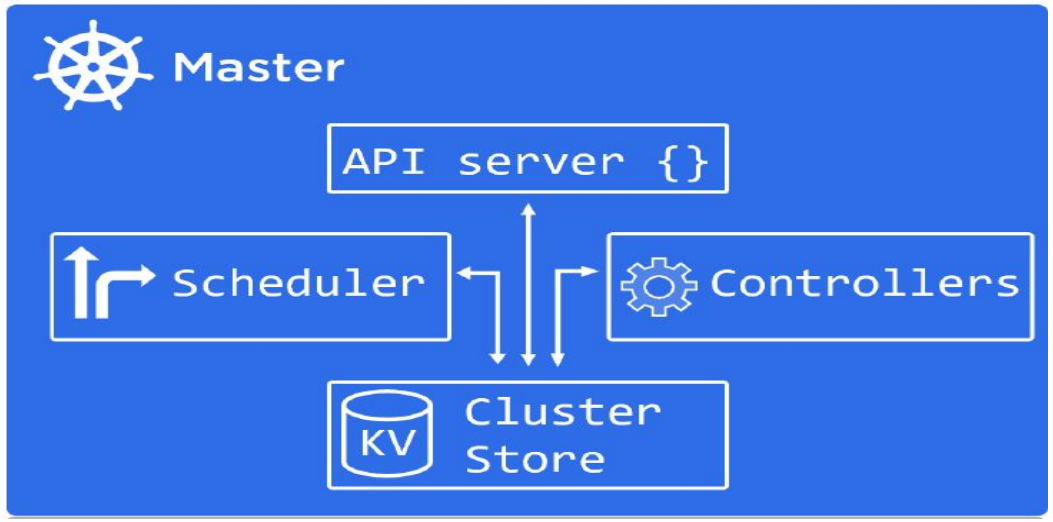
- The API server is brains of the cluster and ‘Cluster store’ is it’s memory.
- It stores the entire configuration and state of the cluster.
- No cluster store, No cluster!
- This cluster store is based on etcd, a popular distributed database.



Kubernetes Architecture - Master

Controller Manager:

- It implements several control loops that watch the cluster and respond to events.



- These Control loops run in background mode

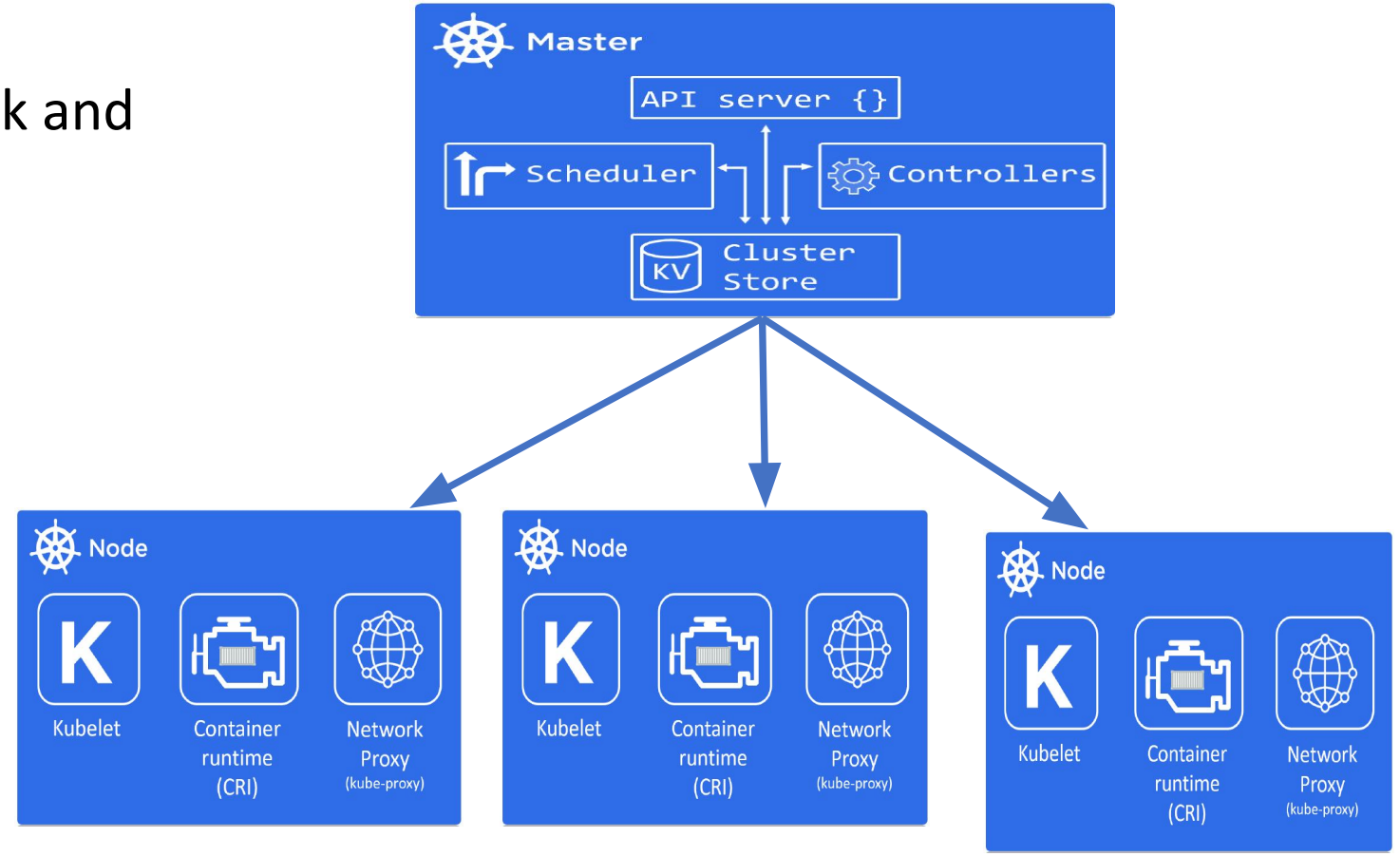
- They constantly watch API server for changes

Controller Manager's Aim is to: Ensure the current state of the cluster matches the desired state

Kubernetes Architecture - Master

Scheduler:

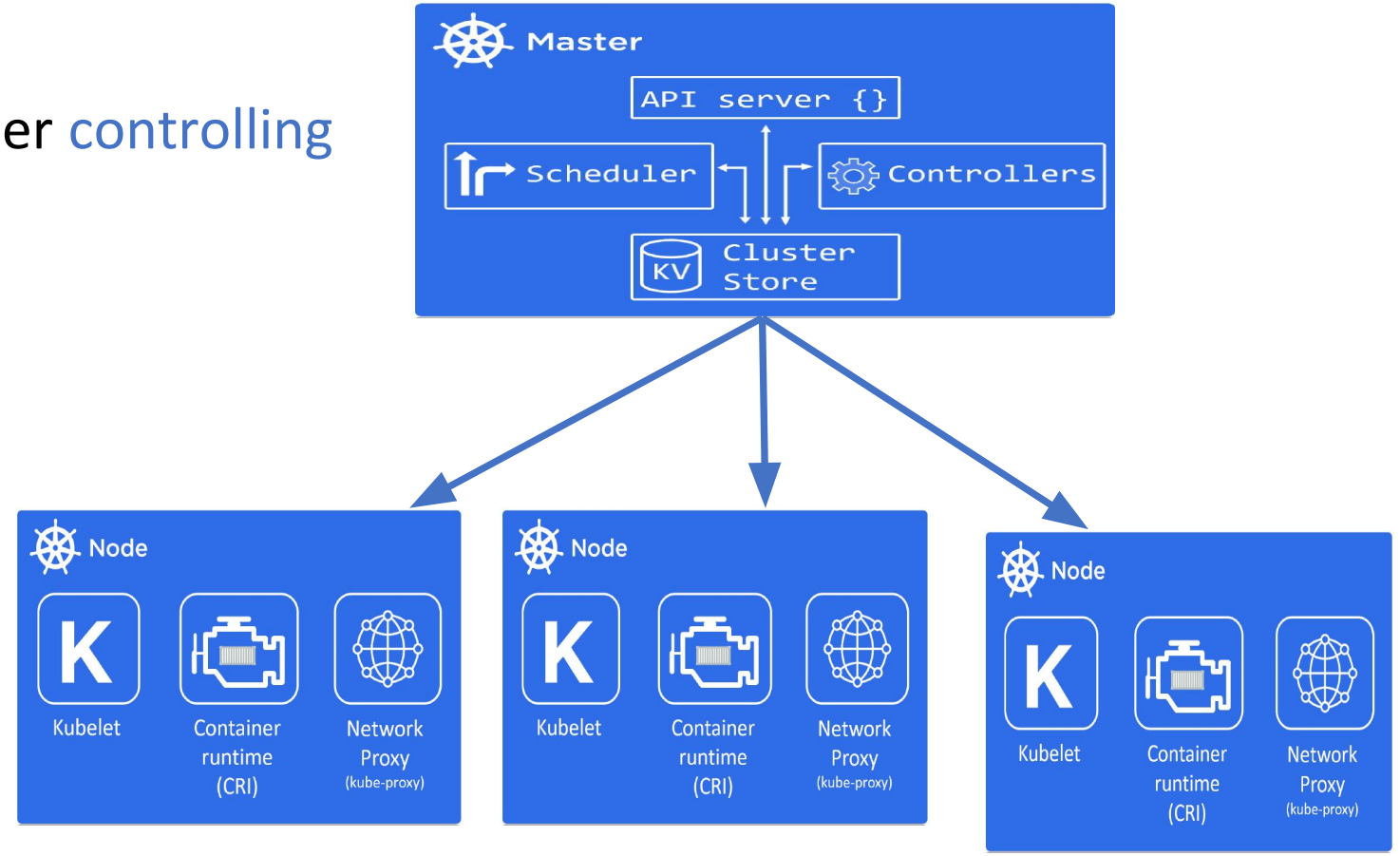
- The scheduler watches for new work and assigns it to nodes



Master/Control Plane - Summary

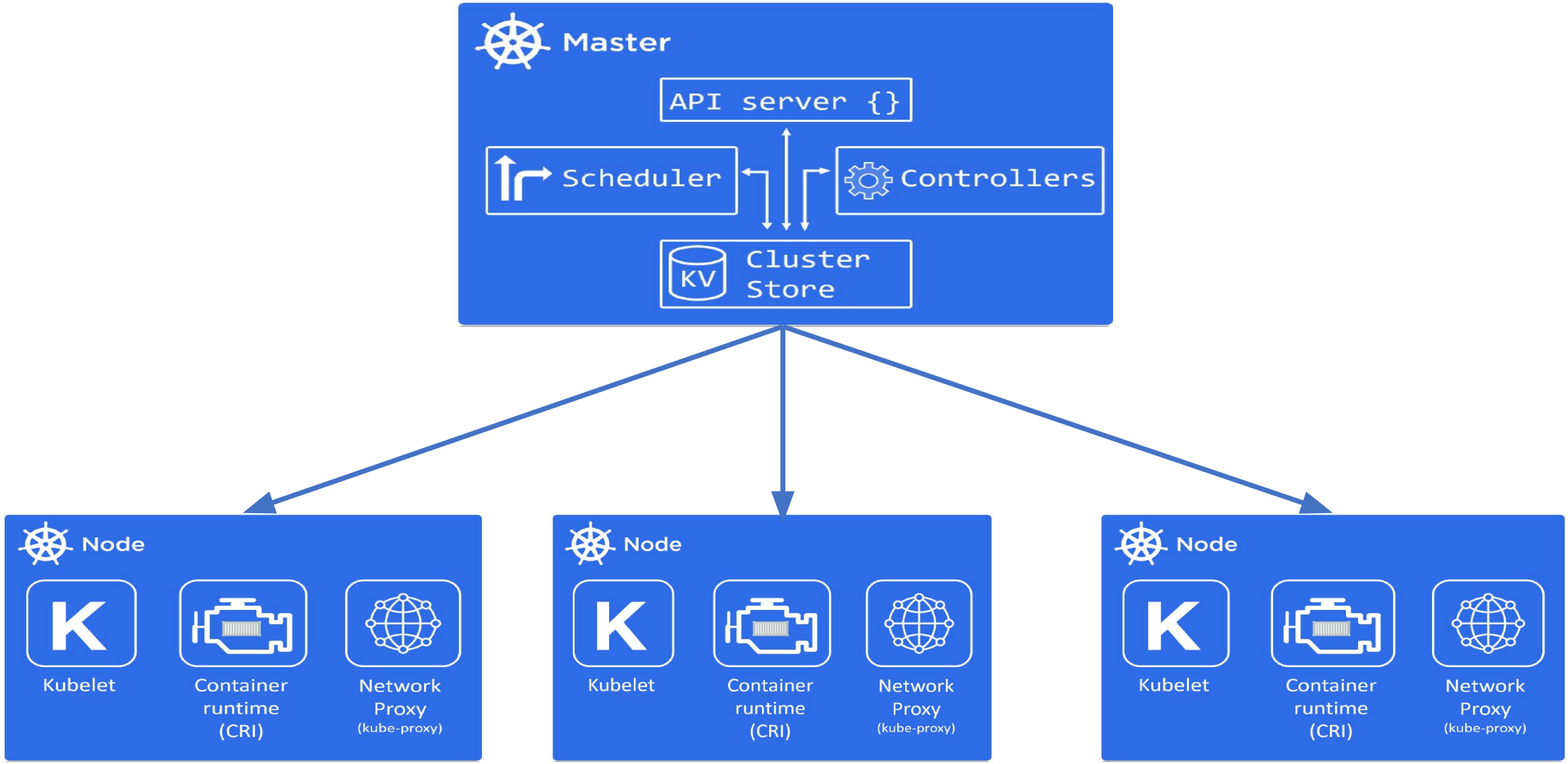
Master Server Summary:

- Master takes care of complete cluster controlling and scheduling.
- Master components are:
  - API server
  - Scheduler
  - Controllers
  - Cluster store

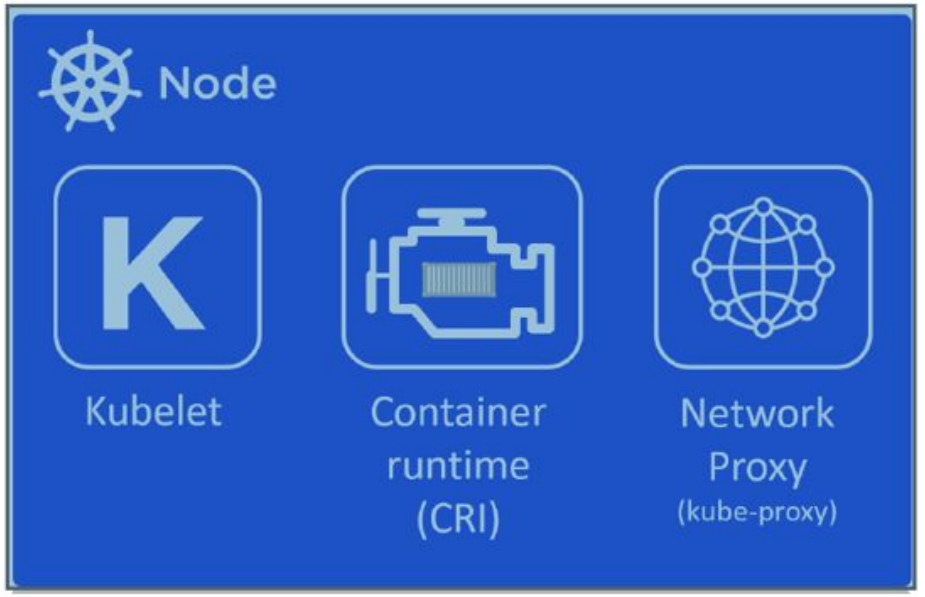


- It's also considered a good practice **not** to run application workloads on masters.
- This allows masters to concentrate entirely on managing the cluster.

Kubernetes Architecture - Nodes



Kubernetes Architecture - Nodes



Nodes/Data Plane:

1. Nodes are the workers of a Kubernetes cluster
2. Nodes are where application services run.

Nodes do:

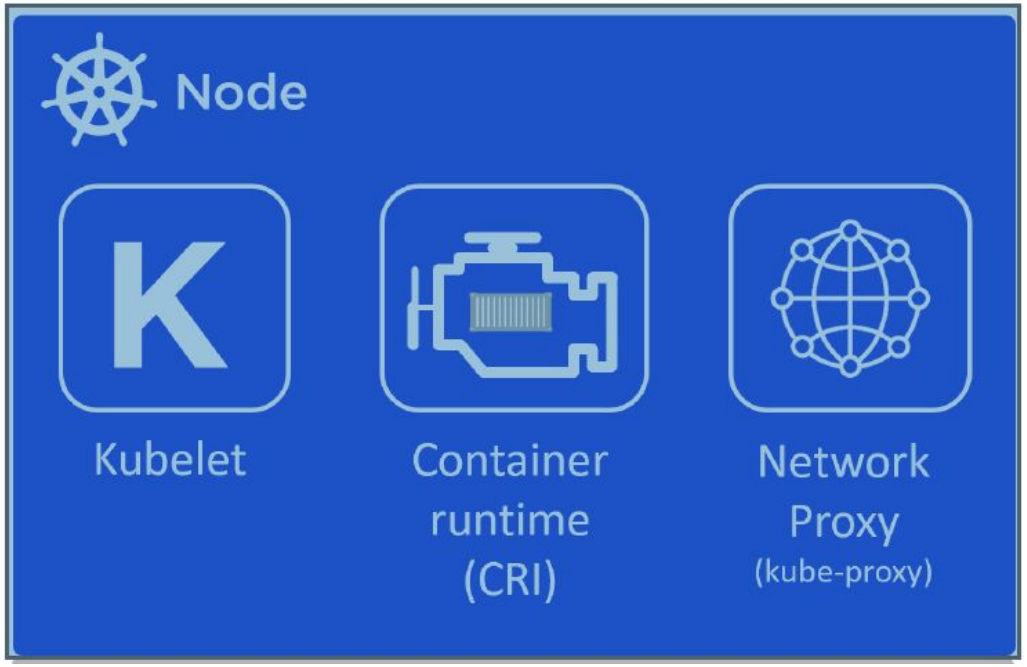
- Constantly watch the API Server for new work assignments
- Execute new work assignments
- Report back to the control plane

- We call Node sometimes as Data Plane.

Kubernetes Architecture - Nodes

Node Major Components:

- Kubelet
- Container Runtime
- Kube Proxy



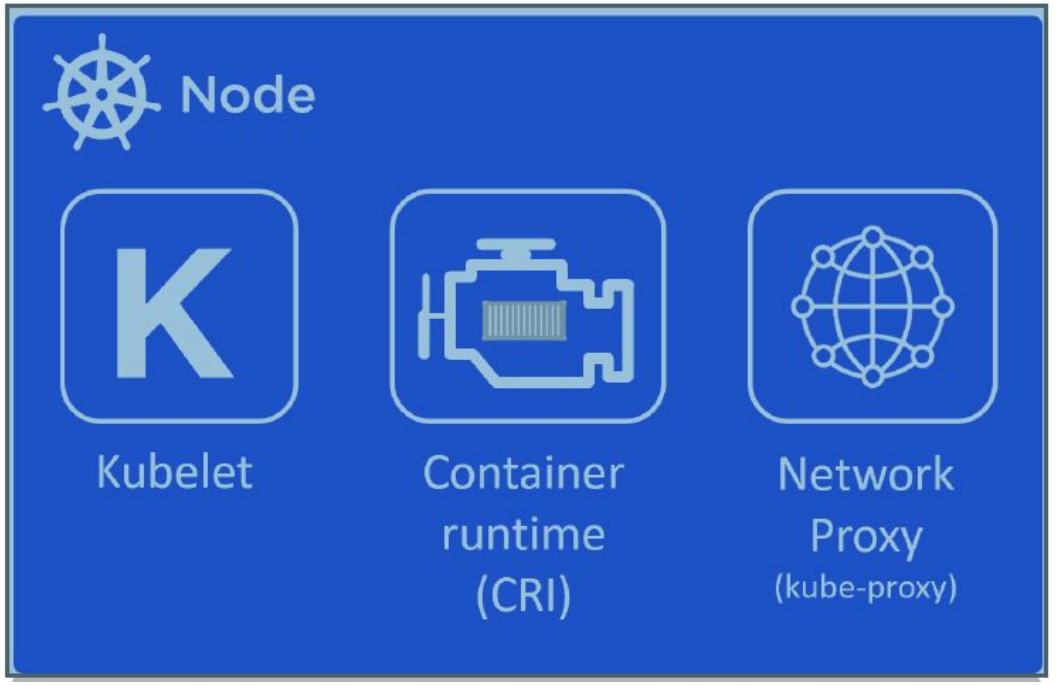
Kubelet:

- Kublet is Kubernetes agent that runs on all nodes In the cluster.
- Kublet is the component that watches the API server for new work assignments.
- If there is any work, it carries out the task and reports back to master.
- If can't execute any task also, it report back to master and master decides what next.
- Ex: If a Pod fails on a node it's NOT Kubelet's responsibility to find another. But simply reports to master and master takes care of it.

Kubernetes Architecture - Nodes

Container Runtime:

- Kubelet need Container runtime for:
  - Pulling the images
  - Starting & Stopping the containers
- Kubernetes supports below container runtimes:
  - Docker
  - Containerd



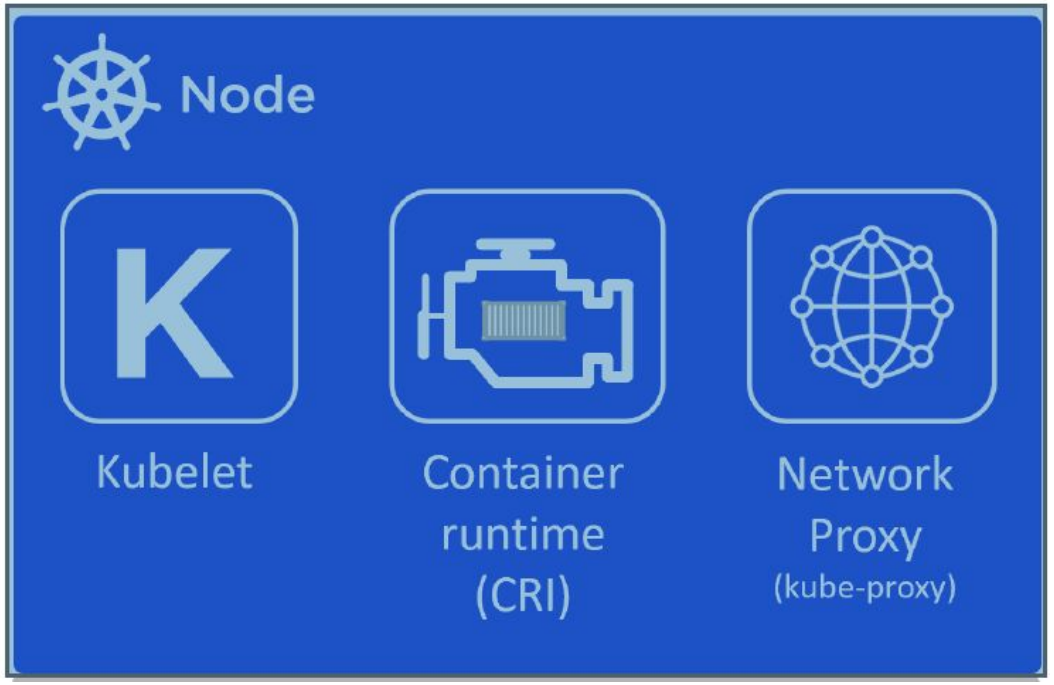
Kubernetes Architecture - Nodes

Kube-proxy:

- Runs on every node of the cluster and is responsible for local networking.

Example:

- It makes sure each node gets it's own unique IP
- Implements IPTables
- IPVS rules to handle Load Balancing



Applications Deployments

Applications Deployments

Microservices

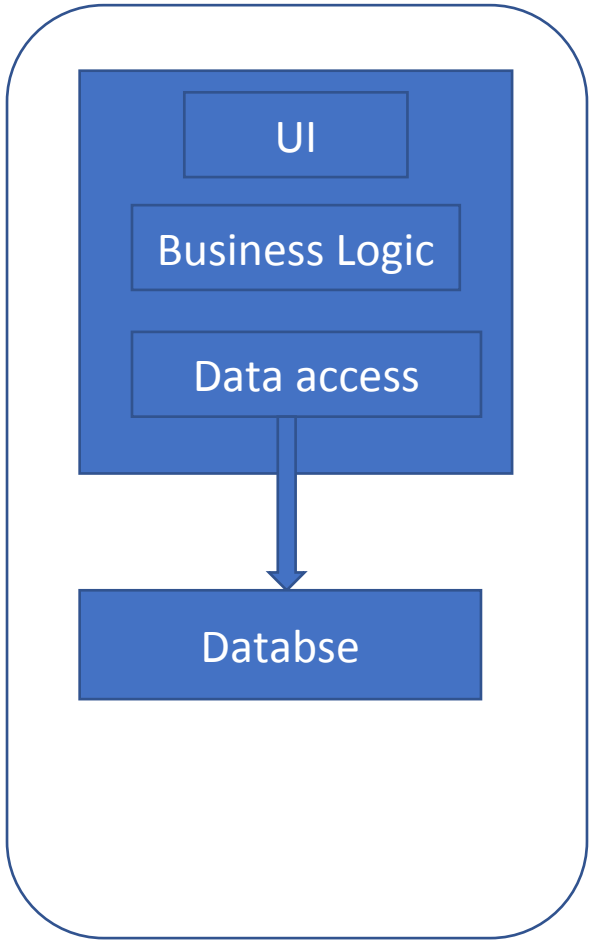
- Docker and Kubernetes are especially useful in Microservices world.

There are two architectural styles in application software development.

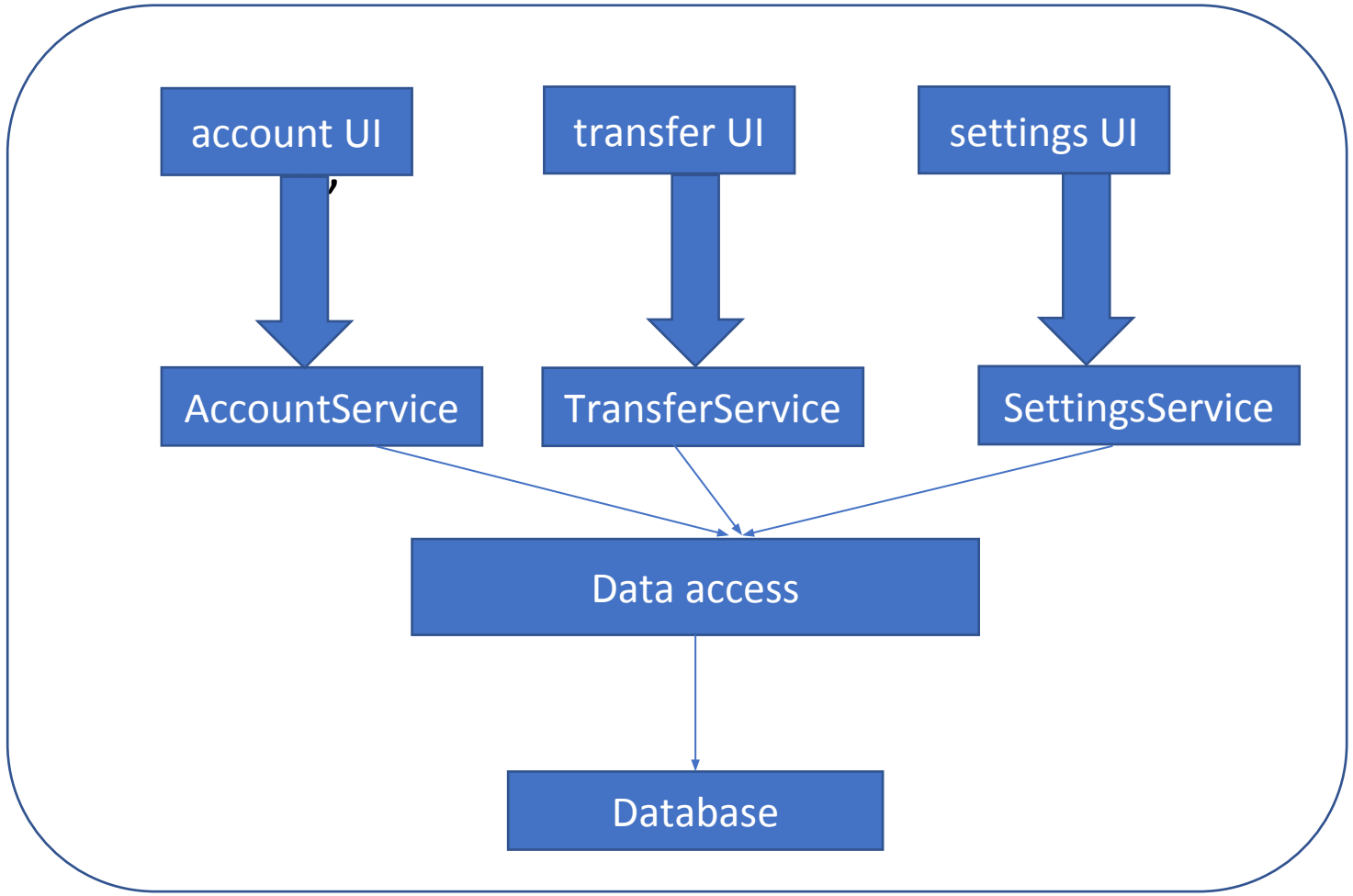
1. Monolithic application
2. Microservices application

- The best way to understand Microservices is to compare them with an old, traditional approach for building large applications, the monolithic design.

Microservices Vs Monolithic



Monolithic Application



Microservices Application

Microservices

Monolithic Design:

- The whole application is monolith, a single logical executable. Good for small projects.
- To make any change to the system, we must build and deploy whole application.
- Scaling up is very difficult. What if we want to scale only one service ex: DB, Login, Video stream ..etc.
- Whole server must be provisioned with enough memory and CPU for all. This can be expensive.
- Compile and build times become longer. Developers need to checkout whole project from SCM, loading in IDE takes more time.
- In case of deployment failure, whole system is unavailable.
- Not suitable for Agile development and Continuous integration processes as we need to release almost at once.

Microservices

Microservices Design:

- This architecture decomposes the application into smaller pieces.
- Microservice Architecture(MSA), is an architectural style and design pattern which says that an application should consist of a collection of loosely-coupled services.
- In other words, each of the services will have its own responsibilities or provide specific functionality, independent of others.
- Each service is deployed on a separate host.
- You don't need to stop whole system to upgrade a piece of functionality.
- When deployed, microservices improve the fault tolerance for entire application.

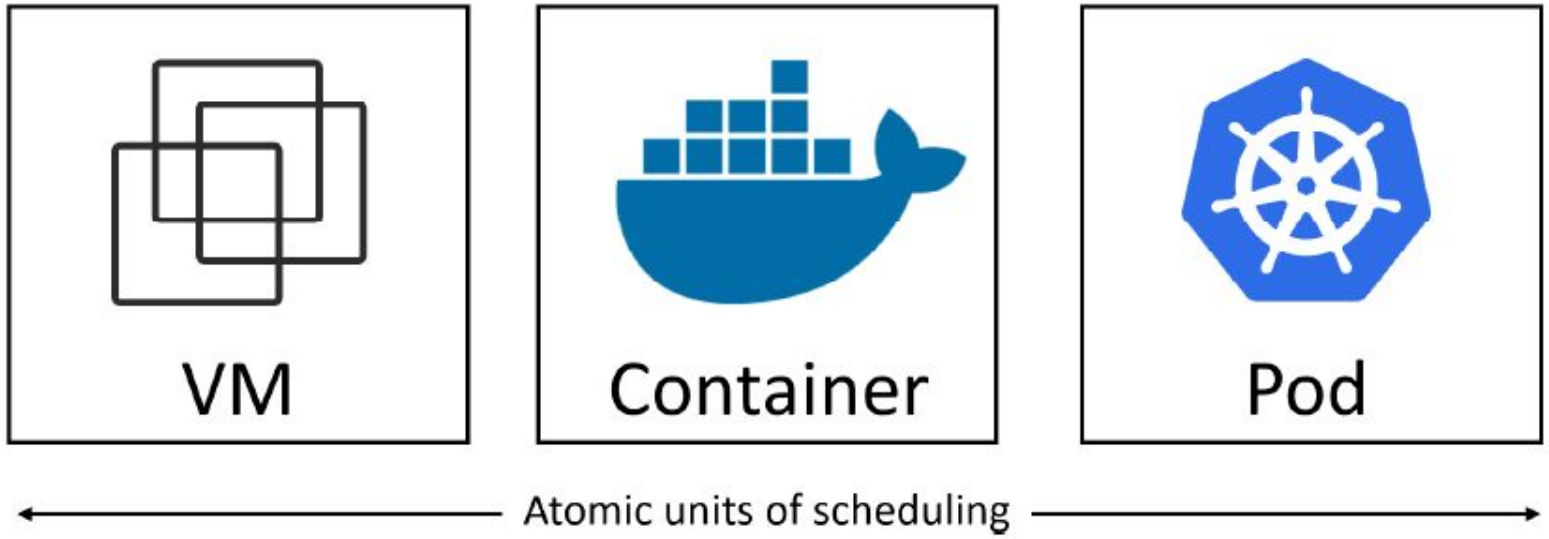
Ex: if there is any memory leak in once service or some other problem, only this service will be affected and can then be fixed and upgraded without interfering with the rest of the system.

- Each microservice is small, easier to understand by developer, easy to develop and deploy.
- Better for continuous delivery as small units are easier to manage, test and deploy.

Applications Deployments

Applications Deployments

Running Applications on VM, Container & K8S

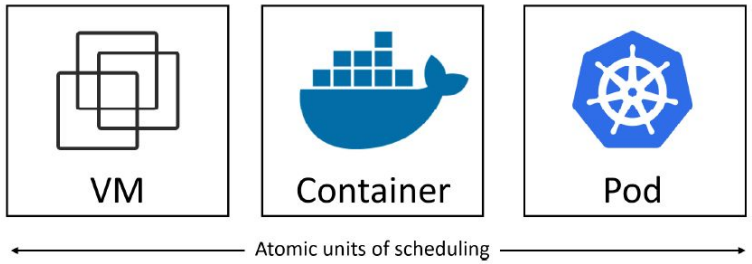


Virtualization – Virtual Machine

Docker – Container

Kubernetes – Pod (Pods are just a vehicle for deploying applications)

Pods



- **Virtualization - World**

  In the Virtualization world, the atomic unit of scheduling is the **Virtual Machine**.

  That means, We deploy the applications on VMs

- **Docker – World**

  In The Docker world, the atomic unit is the **Container**.

  This means , We deploy the applications inside of containers.

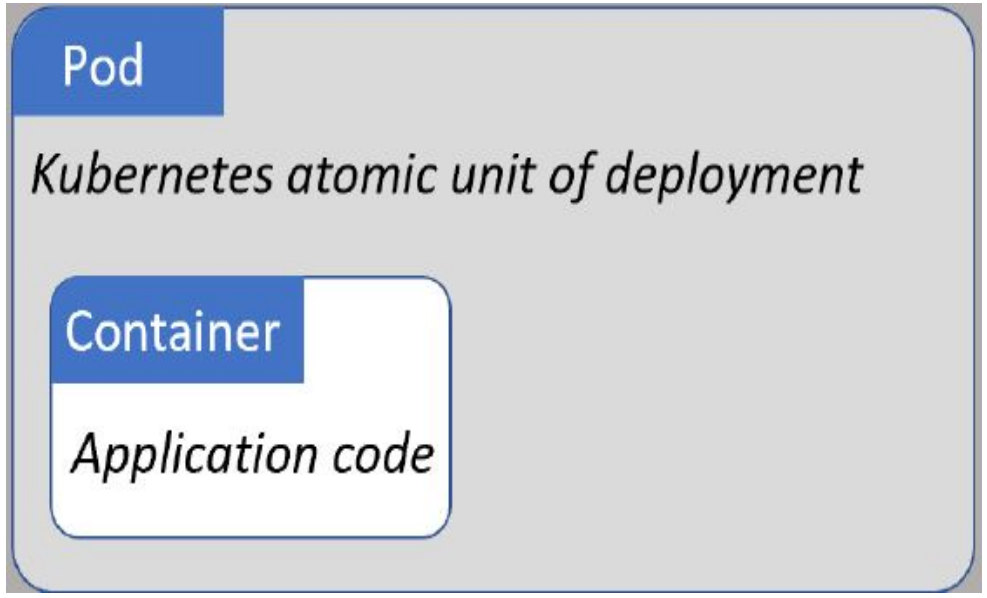
- **Kubernetes – World**

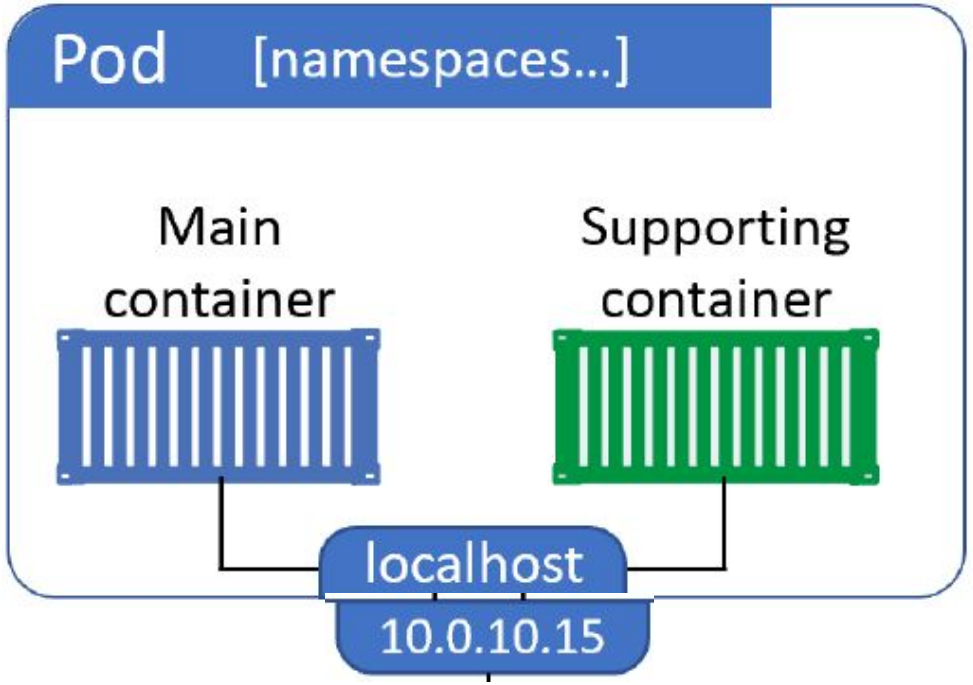
  In the Kubernetes world, the atomic unit is the **Pod**.

  This means, We deploy the applications in Pods

POD

- We can’t run a container directly on a Kubernetes cluster
- Containers must always run inside of Pods!
- Pod can have single or multiple containers. All container share same Pod resources.
- Pods connect to each other using Pod’s ‘localhost’ network interface.





Namespace & Cgroups

Docker is built based on below Linux Kernel features:

- **NameSpace**
- **Cgroups**

- ‘Namespace’ and ‘Cgroups’ isolates the processes from each other.

NameSpace: Isolates processes, networking, file system ..etc.

Cgroups: Limits how much CPU, Memory one of the process/container is using.

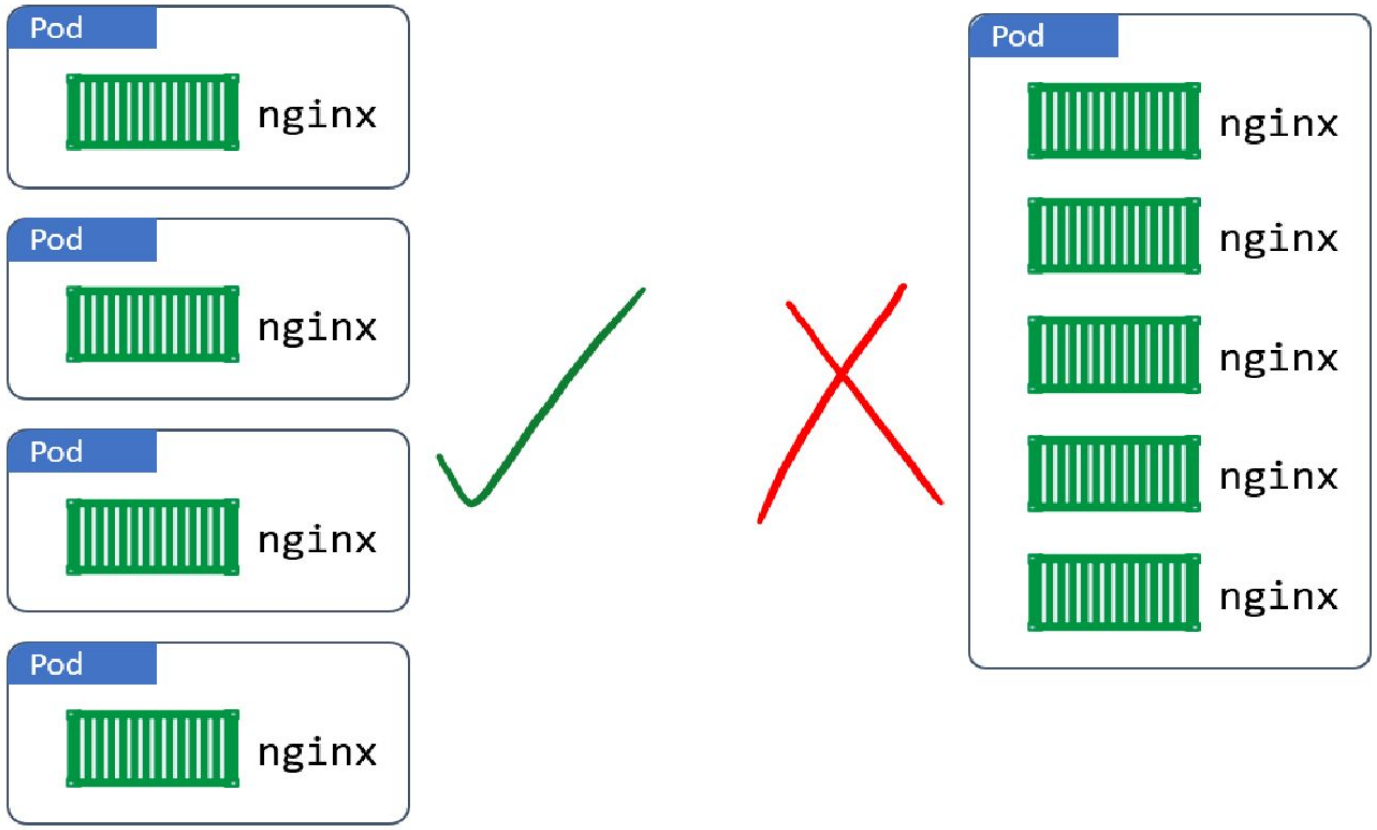
Namespace & Cgroups

What is POD:

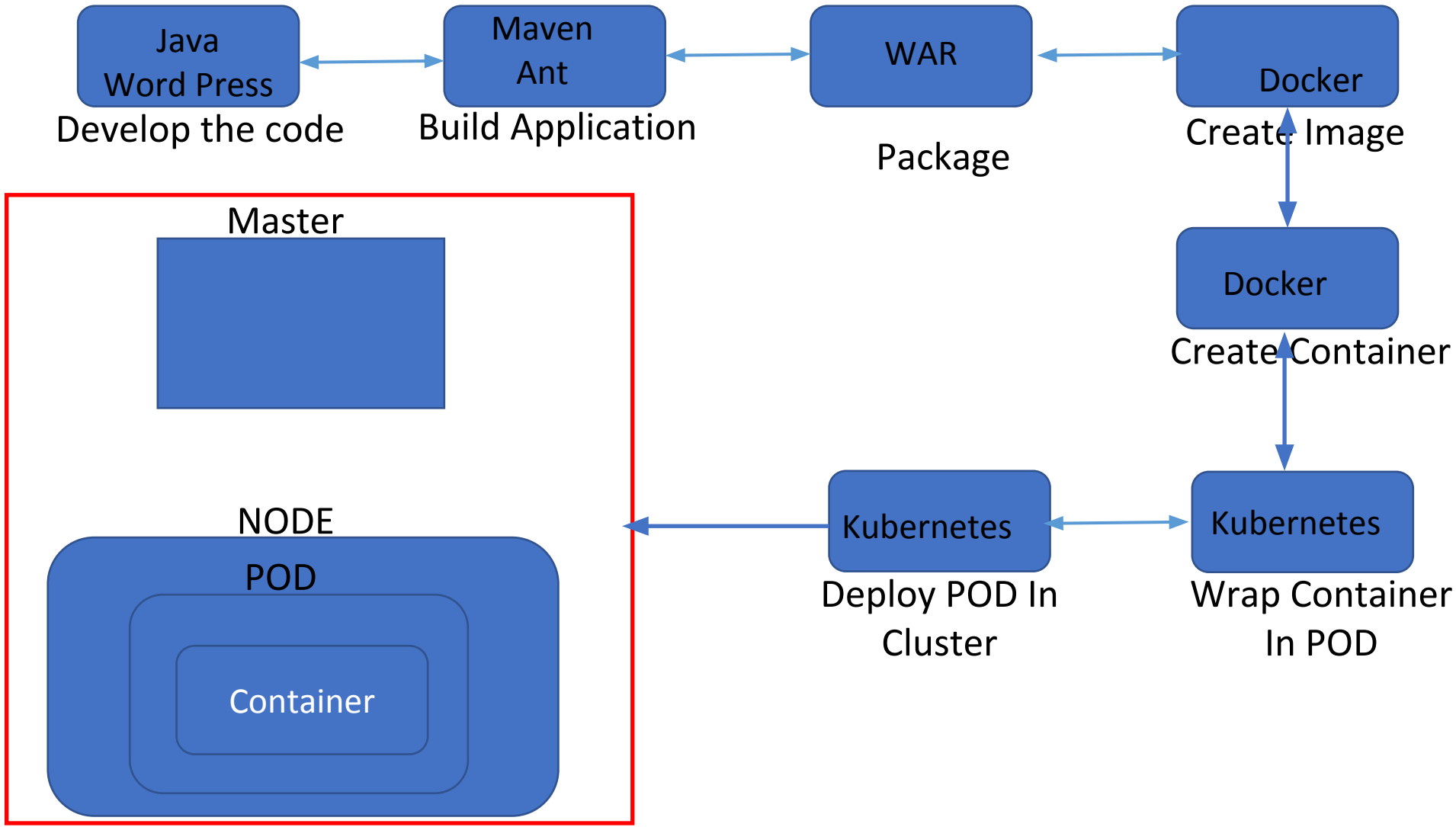
- A Pod is a shared execution environment for one or more containers.
- These include: IP address, Ports, Hostname, Memory, Volumes ..etc.

Pods – Atomic Unit

- The deployment of Pod is an atomic operation.
- If we have to scale application, we add or remove Pods.
- We don’t scale by adding more containers to an existing Pod.



Applications Deployments In Kubernetes



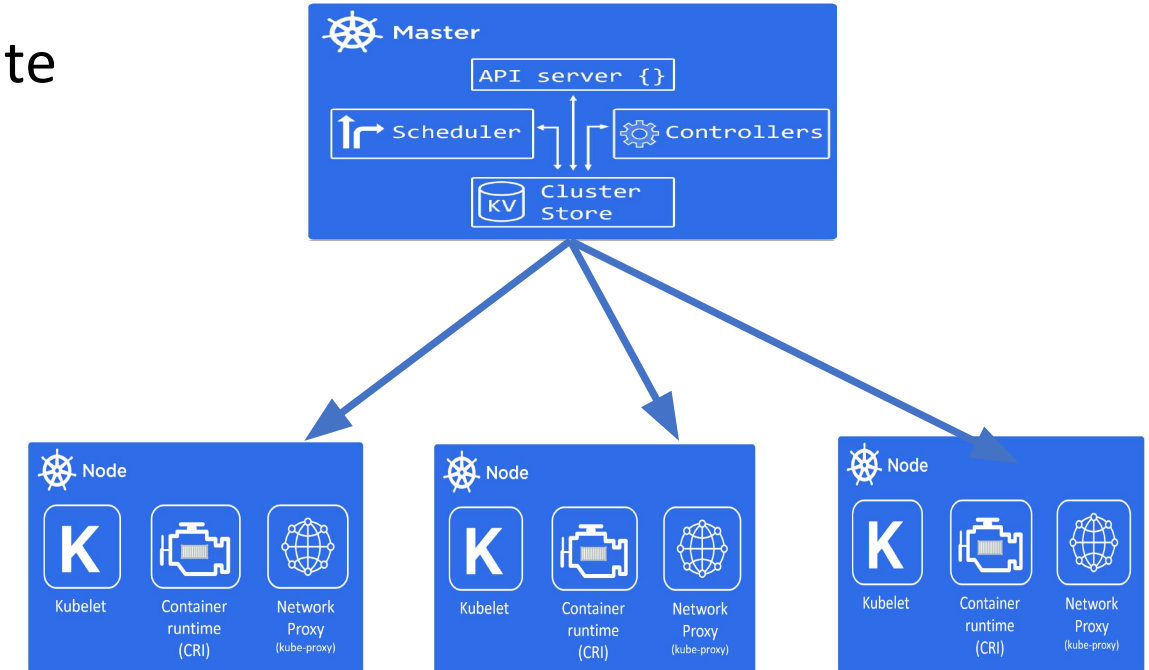
Running Applications on Kubernetes Cluster

Summary:

1. Write the application as small independent services
2. Package each service in it's own container
3. Wrap each container in it's own Pod
4. Deploy the Pods to the cluster

Running Applications on Kubernetes Cluster

1. We declare the desired state of an application in a manifest file
2. We POST it to the Kubernetes API server
3. Kubernetes stores this in the cluster store as the application's desired state
4. Kubernetes implements the desired state on the cluster
5. Kubernetes implements watch loops to make sure the current state of the application doesn't vary from desired state



What's Next?

Pods don't:

self-heal

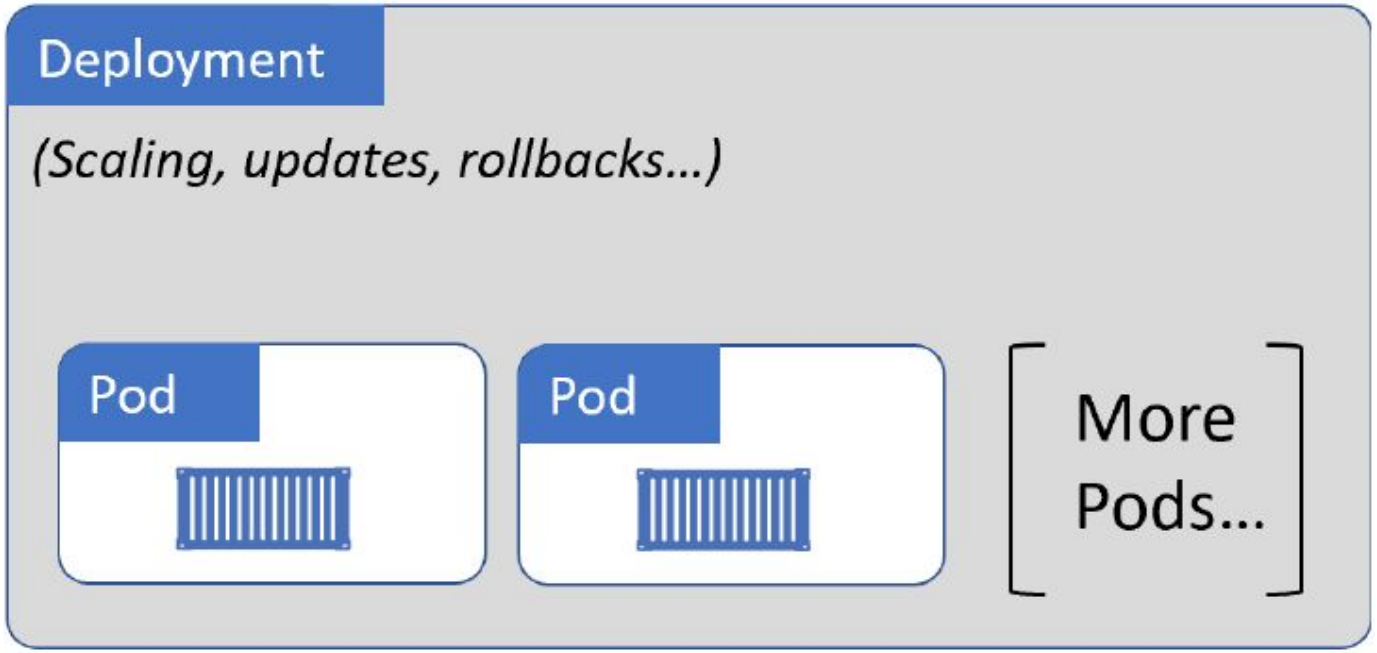
scale

Allow for easy updates

Deployments

- Pods don’t self-heal, they don’t scale, and they don’t allow for easy updates.

$ ENTER DEPLOYMENTS..

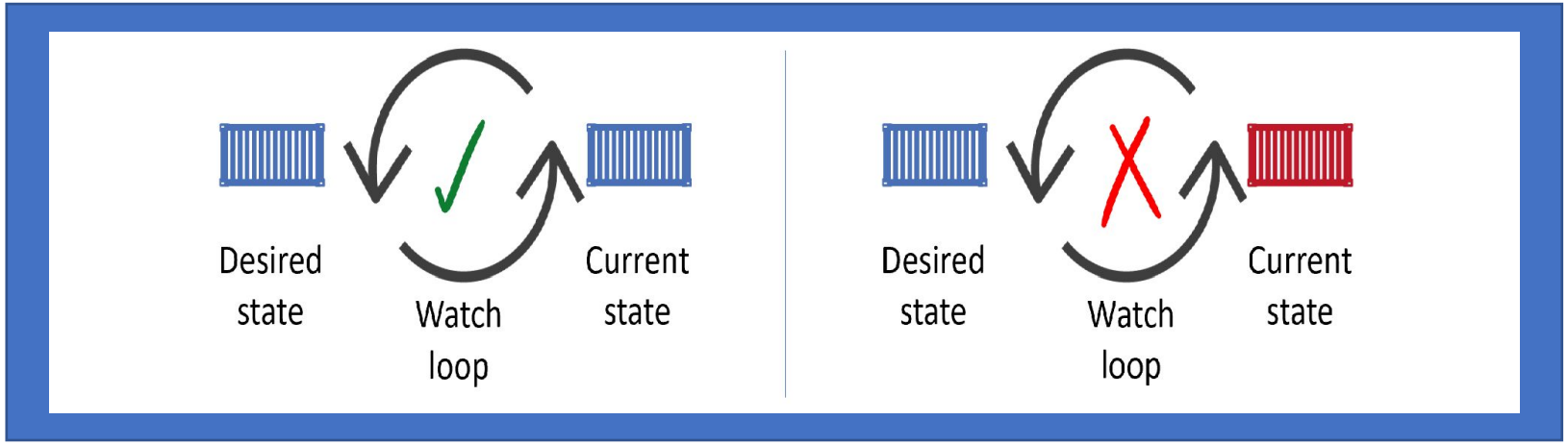


- Deployments do all things like
  - "scale" (scale-up/down)
  - "self-heal"
  - "rolling updates"
  - "roll backs“

- That's why we almost always deploy Pods via 'Deployments"

Desired State.. Current State.. Declarative Model

Just a Concept:



- *Desired State: is What we want*

- *Current State: is What you have*

- *Declarative Model: is a way of telling Kubernetes what our desired state is, without getting into the detail of how to implement it.*

Desired State.. Current Stage..Declarative Model

"Self-healing" with "Deployments"– Deep Dive:

If Pod being manged by Deployment fails, It will be replaced.

Example:

- You need 5 FE and 2 BE Pods.
- Just declare desired state. Sit back! K8S does all the hard work.

Not just that..

- Kubernetes also implements watch loops to make sure current state matches desired state.
- If 2 FE Pods go down, Kubernetes automatically detects the discrepancy and brings 2 Pods