Kubernetes Fundamentals

.NET

Kubernetes is a portable, extensible, open-source, cloud-native platform for managing containerized workloads and services.

HTTPS://KUBERNETES.IO/DOCS/CONCEPTS/OVERVIEW/WHAT-IS-KUBERNETES/

Cloud Native Applications

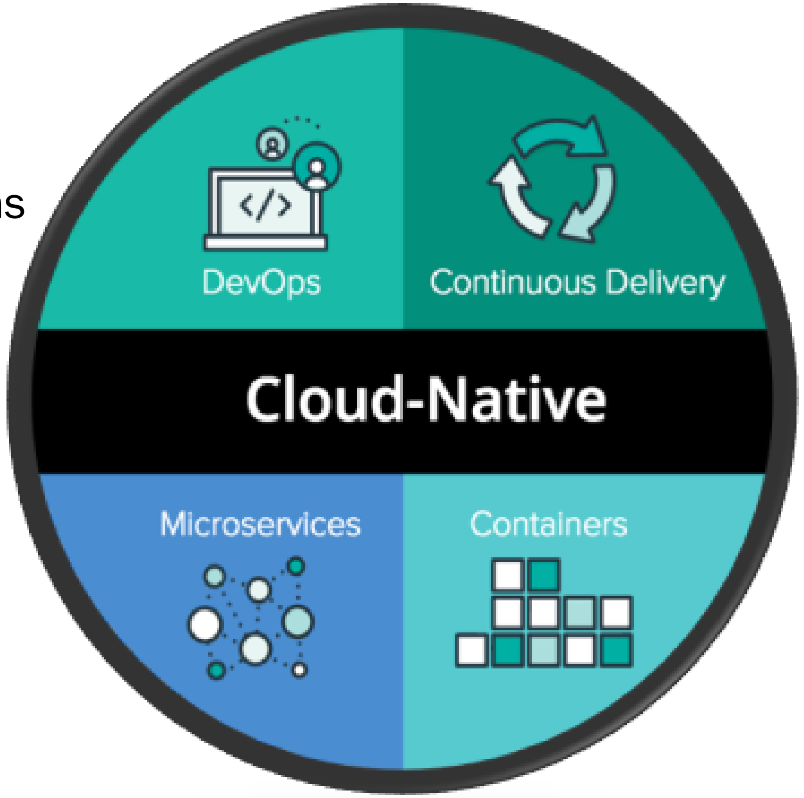
https://www.redhat.com/en/topics/cloud-native-apps
https://medium.com/velotio-perspectives/cloud-native-applications-the-why-the-what-the-how-9b2d31897496

Cloud-native applications are small, independent, loosely coupled services, that are designed to deliver business value. They rapidly incorporate user feedback, maintain higher up-time, and are optimized CI/CD.

Cloud-native app development is a way to speed up how applications are built, how they optimize existing applications, and how services are interconnected.

A "cloud-native," app is specifically designed to provide consistent development, automated management, and greater fault-tolerance experience across all cloud types (private, public, hybrid). Cloud computing increases the scalability and availability of apps through self-service and on-demand provisioning of resources, as well as automating the application life cycle from development to production.

Kubernetes is one of the first and most popular cloud native applications.



Cloud Native Computing Foundation

https://landscape.cncf.io/
https://www.cncf.io/blog/2018/03/08/introducing-the-cloud-native-landscape-2-0-interactive-edition/
https://github.com/cncf/foundation/blob/master/charter.md

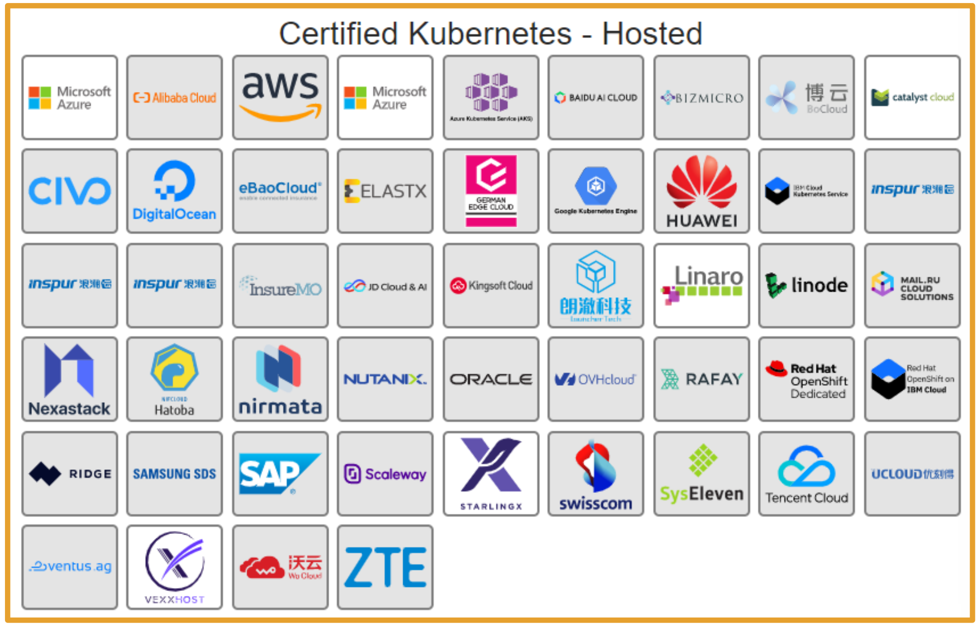
CLOUD NATIVE
COMPUTING FOUNDATION

Kubernetes is part of what is called the Cloud Native Interactive Landscape. This landscape is a category of "Cloud Native" services. These are loosely coupled services solely located in the cloud and whose intent is to enhance the ability of applications to be managed and scaled as needed.

The ***Cloud Native Computing Foundation (CNCF)*** hosts critical components of the global technology infrastructure.

The CNCF mission is to:

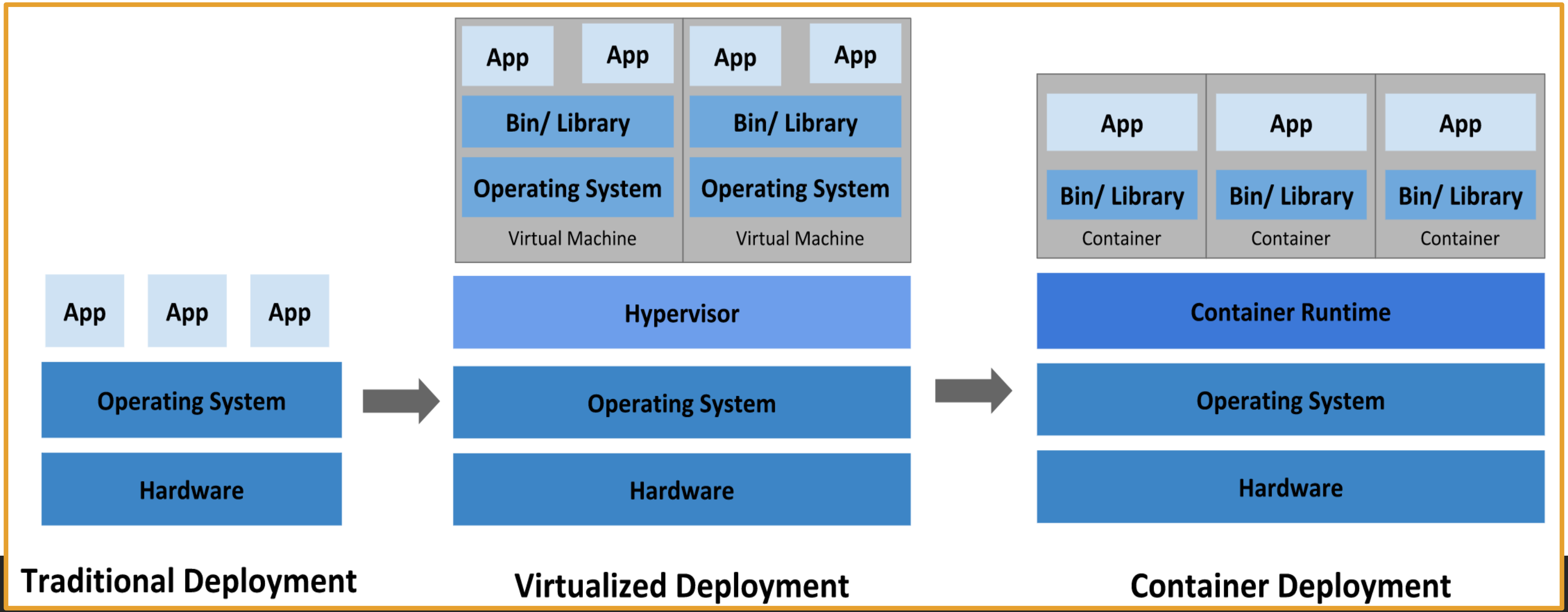
- make cloud native computing ubiquitous,
- steward new projects,
- foster the growth and evolution of the cloud ecosystem,
- promote underlying technologies, and
- make the technology accessible and reliable.



What is Kubernetes?

https://developer.ibm.com/technologies/microservices/articles/why-should-we-use-microservices-and-containers/
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

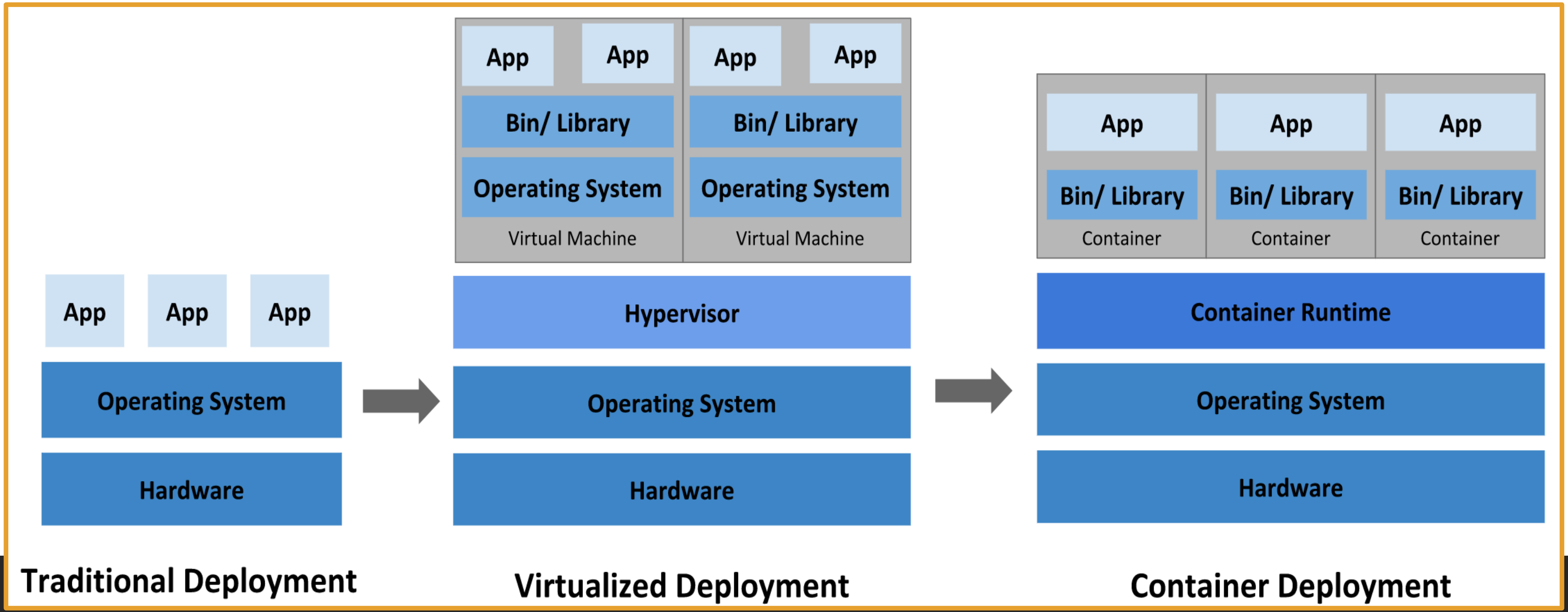
Kubernetes is a production-grade, open-source infrastructure for the deployment, scaling, management, and composition of application containers across ***clusters*** of hosts. It is inspired by previous work at Google ***Kubernetes project***. The name ***Kubernetes*** originates from Greek for helmsman or pilot.



What is Kubernetes?

https://developer.ibm.com/technologies/microservices/articles/why-should-we-use-microservices-and-containers/
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

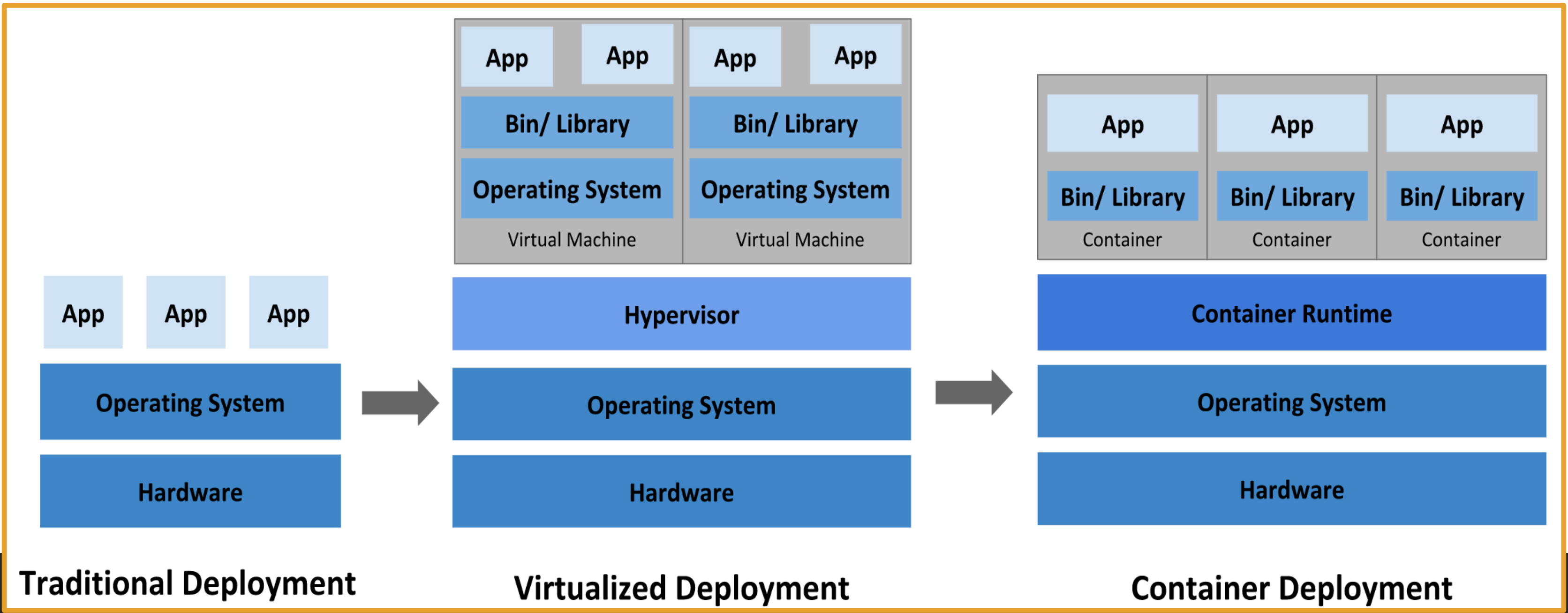
Kubernetes provides a framework to run distributed systems resiliently. It manages scaling and failover and provides deployment patterns. *Kubernetes* allows you to automate the deployment of containerized microservices. This makes it easier to manage the components and microservices in your application.



What is Kubernetes?

https://developer.ibm.com/technologies/microservices/articles/why-should-we-use-microservices-and-containers/
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kubernetes-design-and-architecture

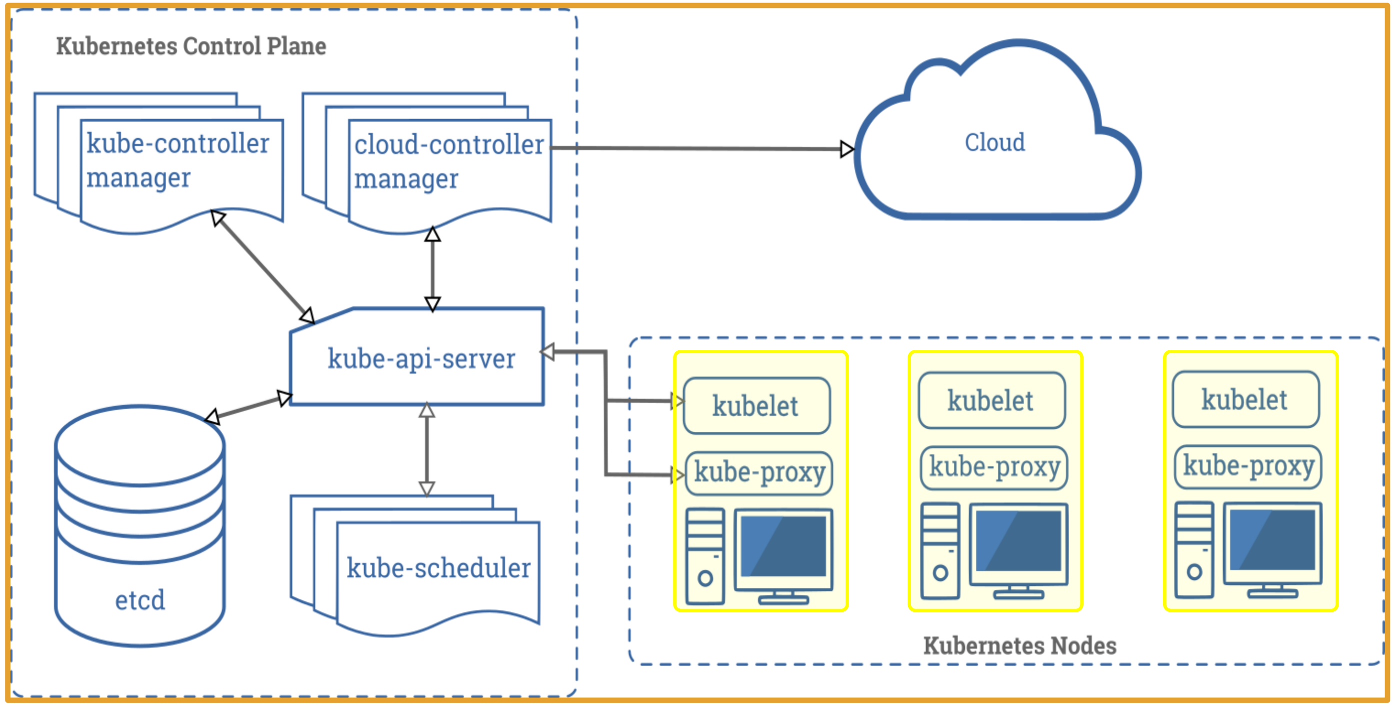
Kubernetes containers allow you to:			
Deploy images quickly	Maintain CI/CD	Enhance Separation of Concerns	Run applications anywhere
Have an elastic, scalable MSA	Isolate resources	Use resources effectively	Run an application on any platform



Kubernetes Architecture – Overview (1/2)

https://kubernetes.io/docs/concepts/overview/components/

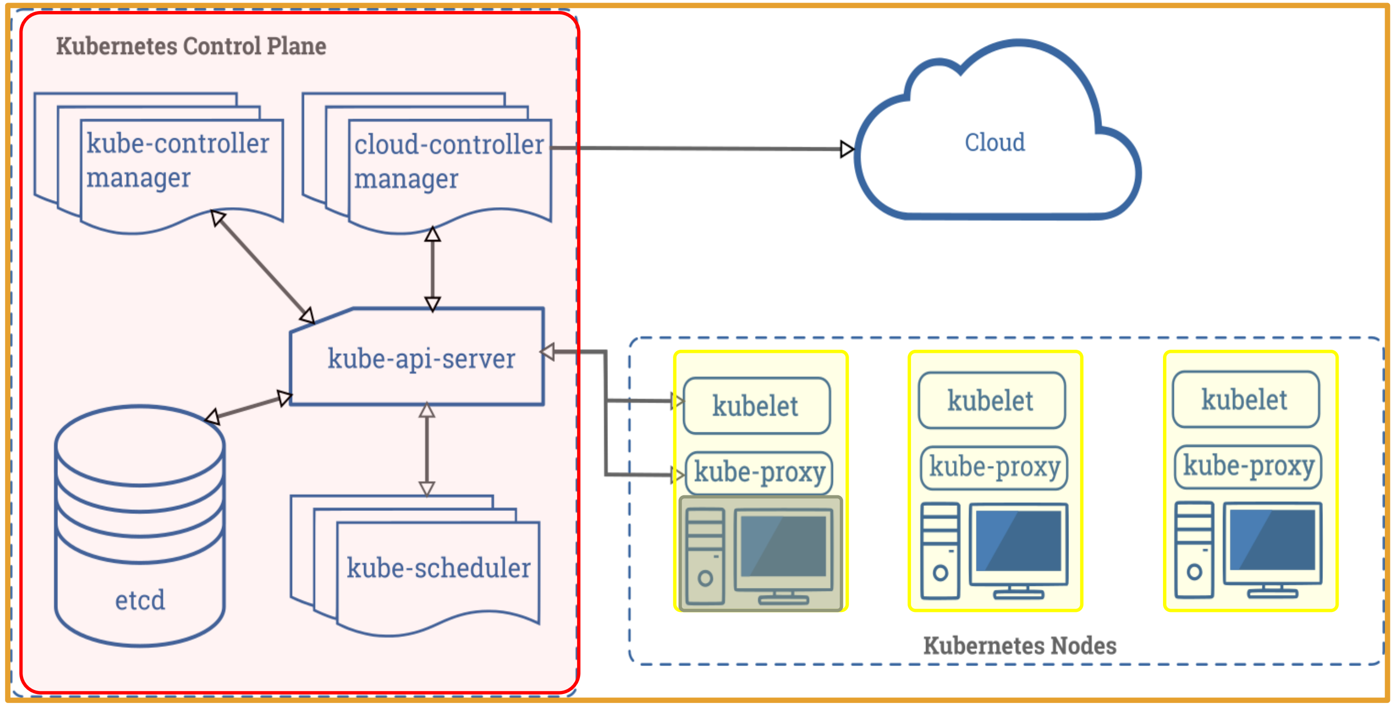
- *Kubernetes* is not a traditional, all-inclusive PaaS (Platform as a Service).
- *Kubernetes* operates at the container level rather than at the hardware level.
- When you deploy *Kubernetes*, you get a Cluster.
- A Cluster consists of worker machines (Nodes), that run *containerized* applications.



Kubernetes Architecture – Overview (2/2)

https://kubernetes.io/docs/concepts/overview/components/

- The worker node(s) host the Pods that are the components of the application workload.
- The control plane manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually operates across multiple servers and a cluster usually runs multiple nodes. This enhances fault-tolerance and provides high availability.

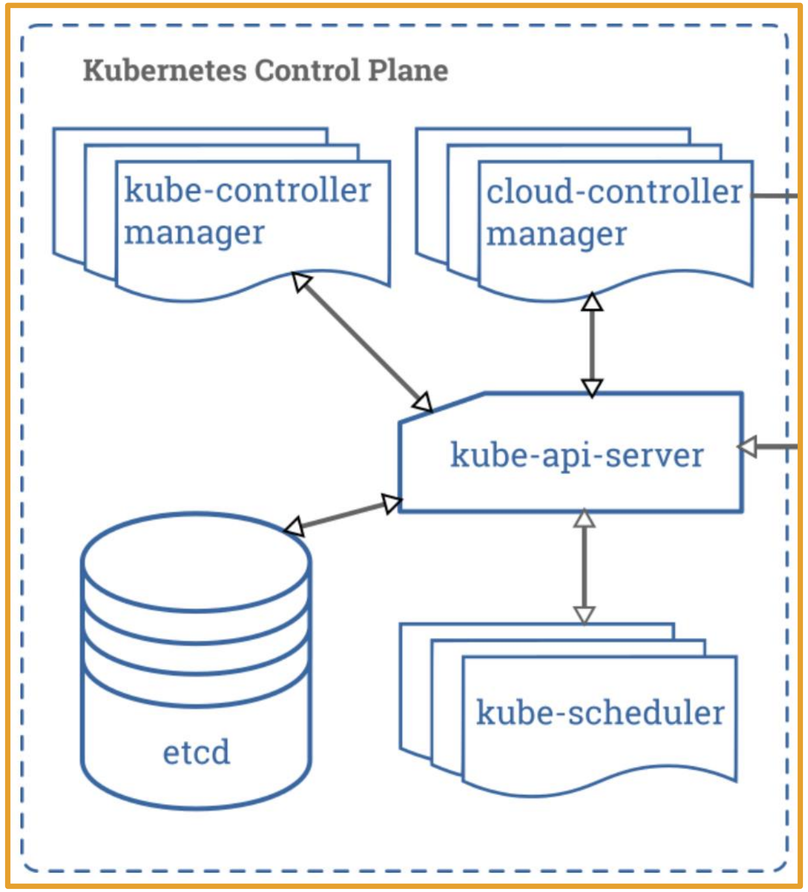


Kubernetes Control Plane (Master)

https://kubernetes.io/docs/concepts/overview/components/#control-plane-components

The ***control plane's*** components make global decisions about the cluster, as well as detecting and responding to ***cluster*** events, like starting up a new pod when a deployment .yml's 'replicas' field is unsatisfied.

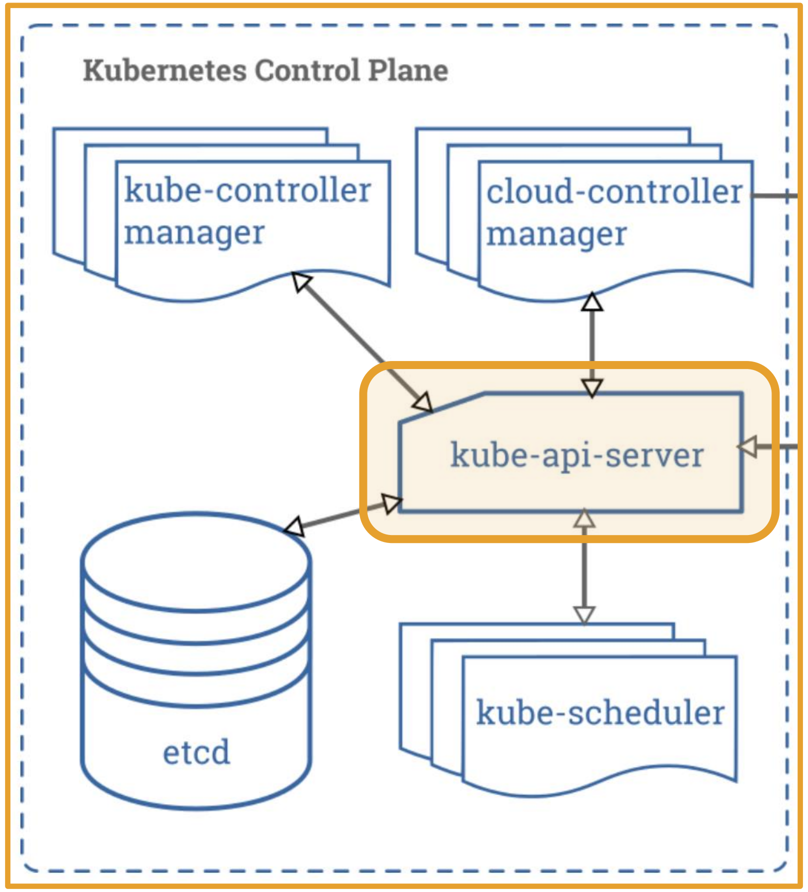
Control plane components can be run on any machine in the cluster, but typically set-up ***scripts*** start all ***control plane*** components on the same machine, and do not run user containers on that machine.



Control Plane – kube-apiserver

https://kubernetes.io/docs/concepts/overview/components/#kube-apiserver

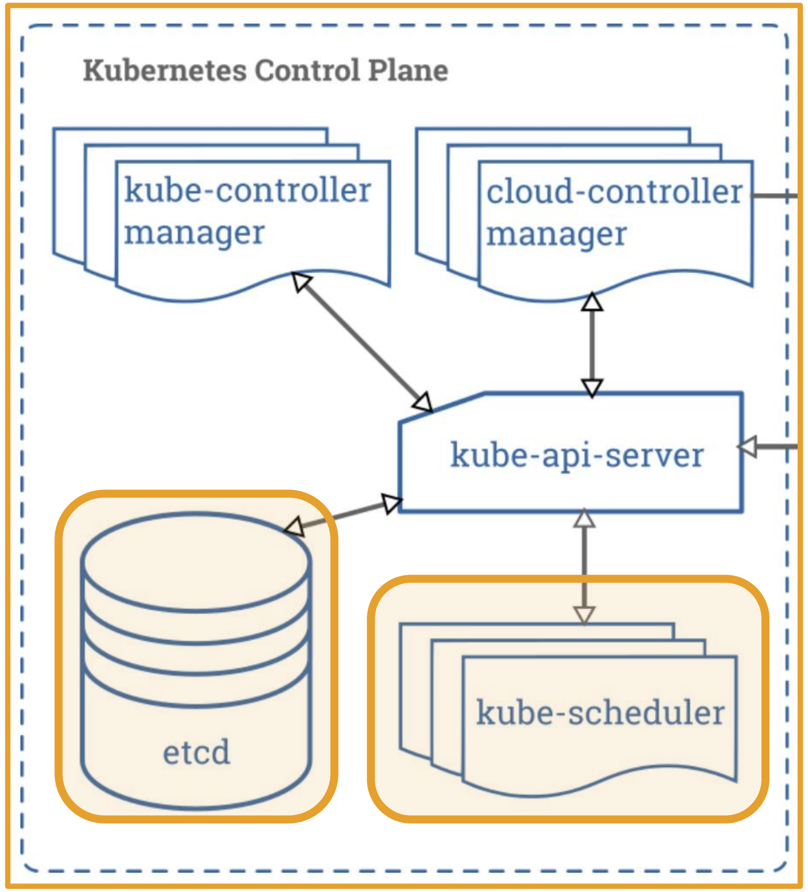
- The ***API server*** exposes the Kubernetes API. The ***API server*** is the front-end for the Kubernetes ***control plane***.
- The main implementation of a Kubernetes API server is ***kube-api-server***.
- ***kube-api-server*** is designed to scale horizontally (deploying more instances).
- You can run several instances of ***kube-api-server*** and a LoadBalancer balances traffic between the instances.



Control Plane – etcd and kube-scheduler

https://kubernetes.io/docs/concepts/overview/components/#etcd
https://kubernetes.io/docs/concepts/overview/components/#kube-scheduler
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#scheduler

- ***Etcd*** is a key-value store. It maintains the ***clusters'*** data.
- ***kube-scheduler*** watches for new ***Pods*** and assigns a ***node*** to them to run on based on predetermined requirements like:
  - hardware constraints,
  - affinity/anti-affinity specifications,
  - deadlines,
  - and many more.

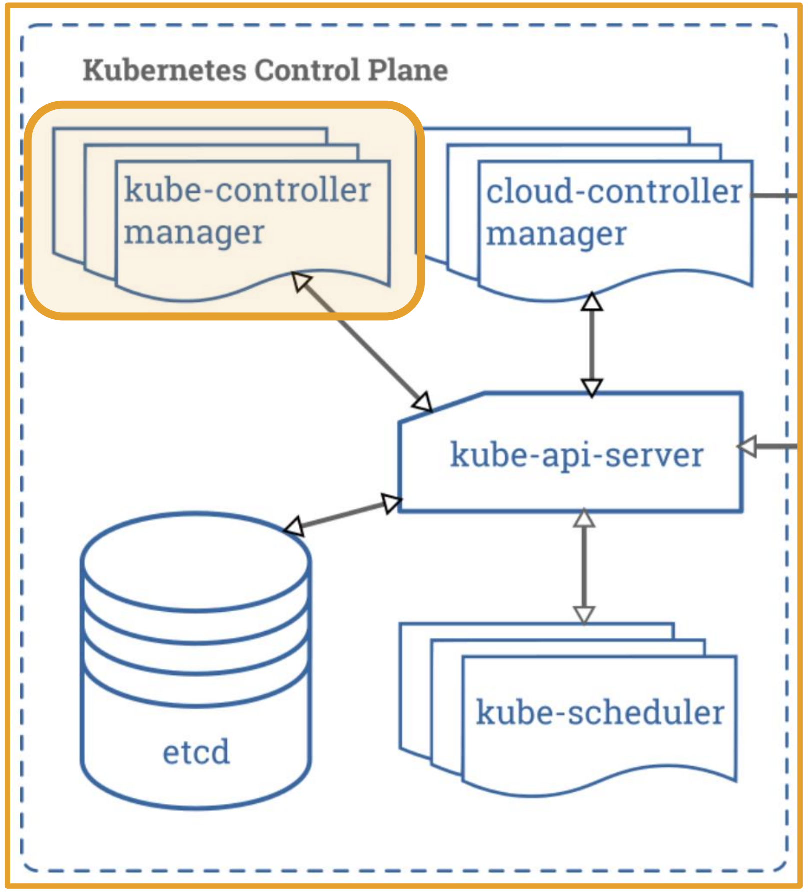


Control Plane – kube-controller manager

https://kubernetes.io/docs/concepts/overview/components/#kube-controller-manager

Kube-manager-controller runs the four ***controller processes***. There are 4 ***controller process*** types:

- Node controller: notices and responds when nodes go down.
- Replication controller: maintains the correct number of pods for every replication controller object in the system.
- Endpoints controller: Populates the Endpoints object (joins Services & Pods).
- Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.

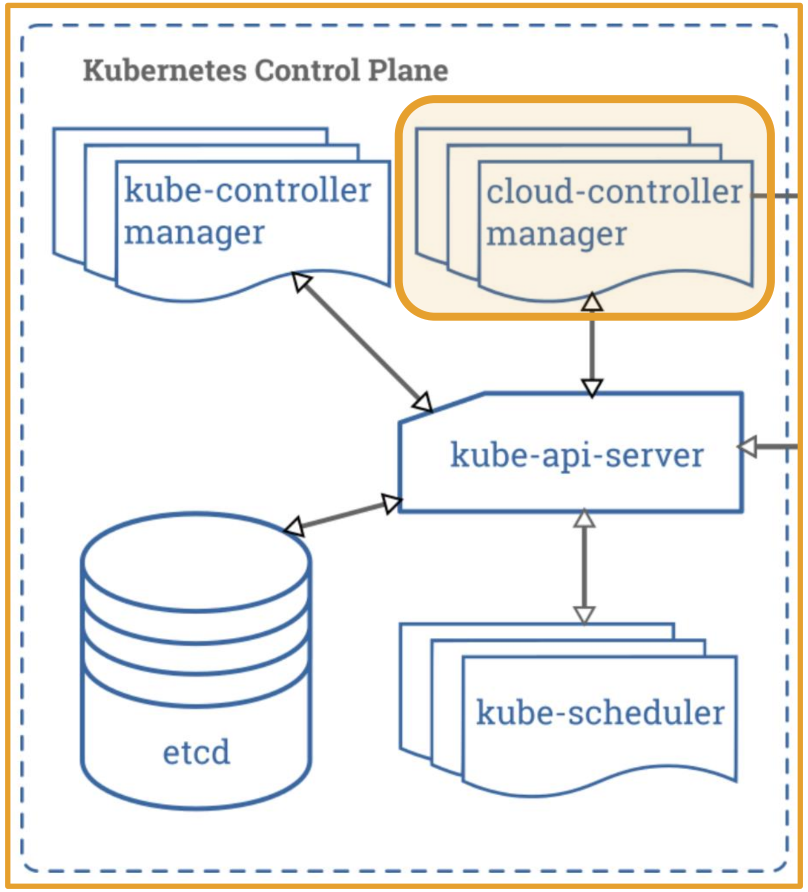


Control Plane – cloud-controller manager (1/2)

https://kubernetes.io/docs/concepts/overview/components/#cloud-controller-manager

The ***cloud-controller-manager*** allows linking a cluster into the cloud providers API. It will separate the components that interact with the cloud platform from components that only interact with the cluster.

cloud-controller-manager combines several logically independent control loops into a single binary that is run as a single process. Horizontal scaling (running more instances) allows for improved performance or help with failure tolerance.

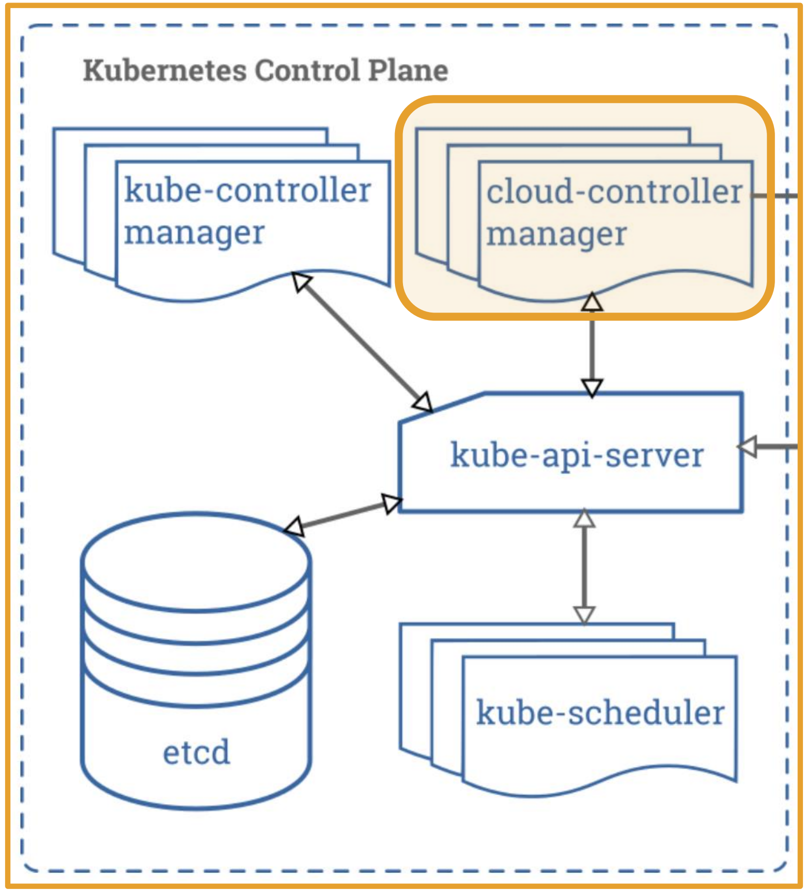


Control Plane – cloud-controller manager (2/2)

https://kubernetes.io/docs/concepts/overview/components/#cloud-controller-manager

These three ***controllers*** can have cloud provider dependencies:

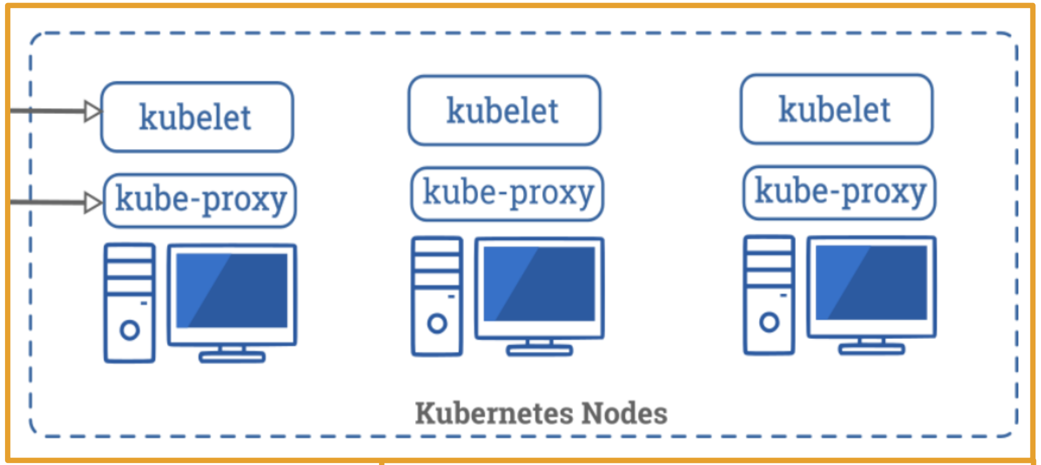
- Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.
- Route controller: For setting up routes in the underlying cloud infrastructure.
- Service controller: For creating, updating and deleting cloud provider load balancers.



Node Components - Kubelet

https://kubernetes.io/docs/concepts/overview/components/#node-components
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kubelet
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kube-proxy

A ***Kubelet*** agent runs on each ***node*** in the cluster. It is the primary implementer of the *Pod* and Node APIs that drive the container execution layer. The ***Kubelet*** uses ***PodSpecs*** to verify that containers described in those ***PodSpecs*** are running in the ***Pods***. The ***kubelet*** doesn't manage containers which were not created by ***Kubernetes***.

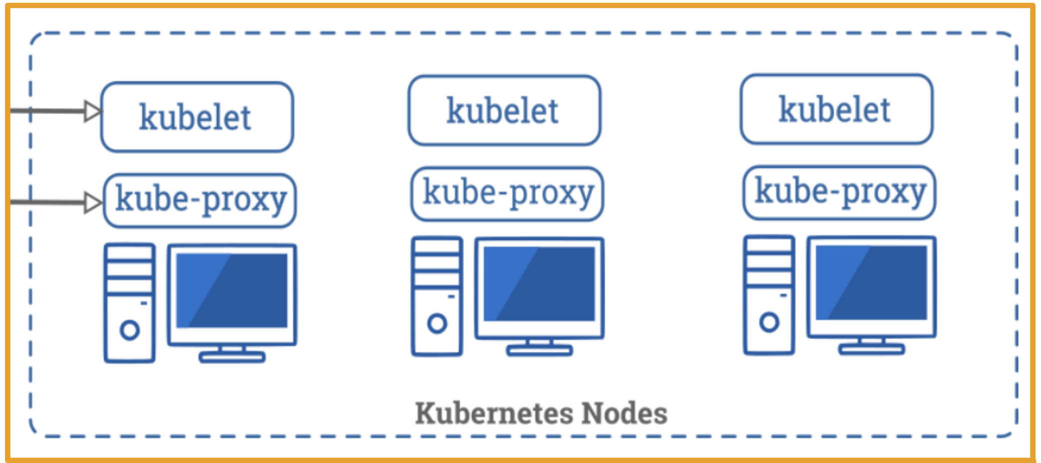


Node components run on every ***node***, maintain running ***pods***, and providing the Kubernetes runtime environment.

Node Components – kube-proxy

https://kubernetes.io/docs/concepts/overview/components/#node-components
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kubelet
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kube-proxy

A ***kube-proxy*** is a network proxy that runs on each ***node*** in your cluster. ***kube-proxy*** provides a way to group pods under a common access policy (e.g., ***load-balanced***).
This creates a virtual IP that clients can access which is transparently proxied (forwarded) to the ***pods*** in a Service.
Every ***node*** runs a ***kube-proxy*** process.
Kube-proxy programs IpTables rules to trap access to service IPs and redirect them to the correct backend.



Node components run on every ***node***, maintain running ***pods***, and provide the Kubernetes runtime environment.

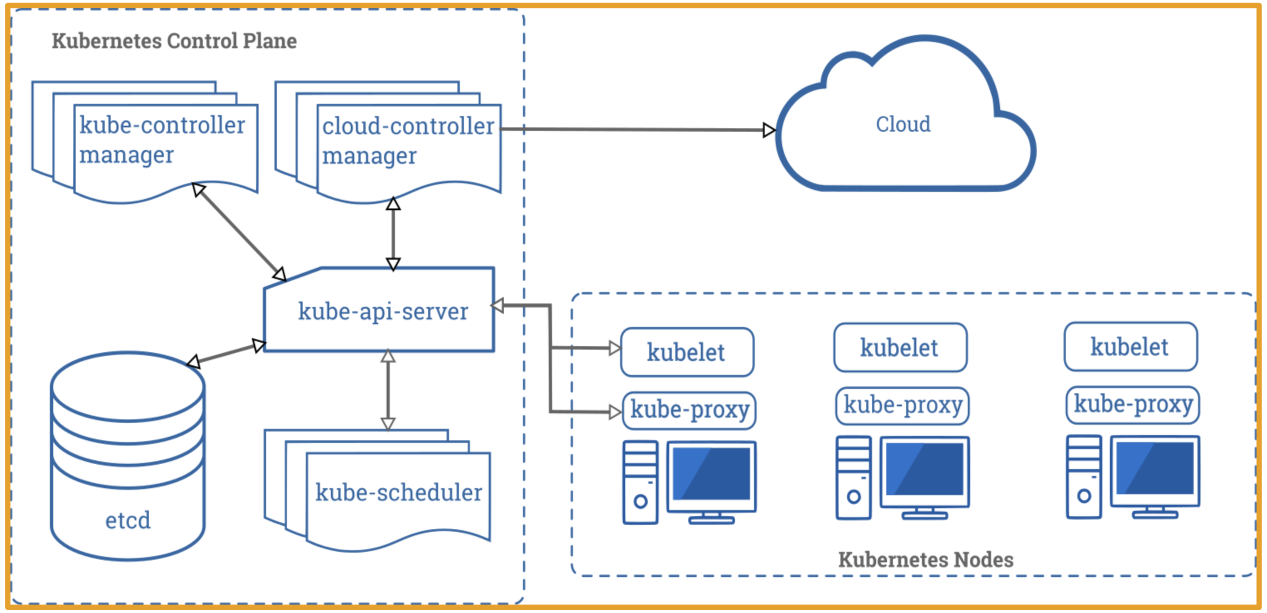
Node – Components

https://kubernetes.io/docs/concepts/overview/components/#node-components
https://kubernetes.io/docs/concepts/architecture/nodes/#management

The ***container runtime*** is the software that is responsible for running containers.

Kubernetes supports several container runtimes.

- Kubernetes Container Runtime Interface (CRI
- Docker
- containerd
- CRI-O

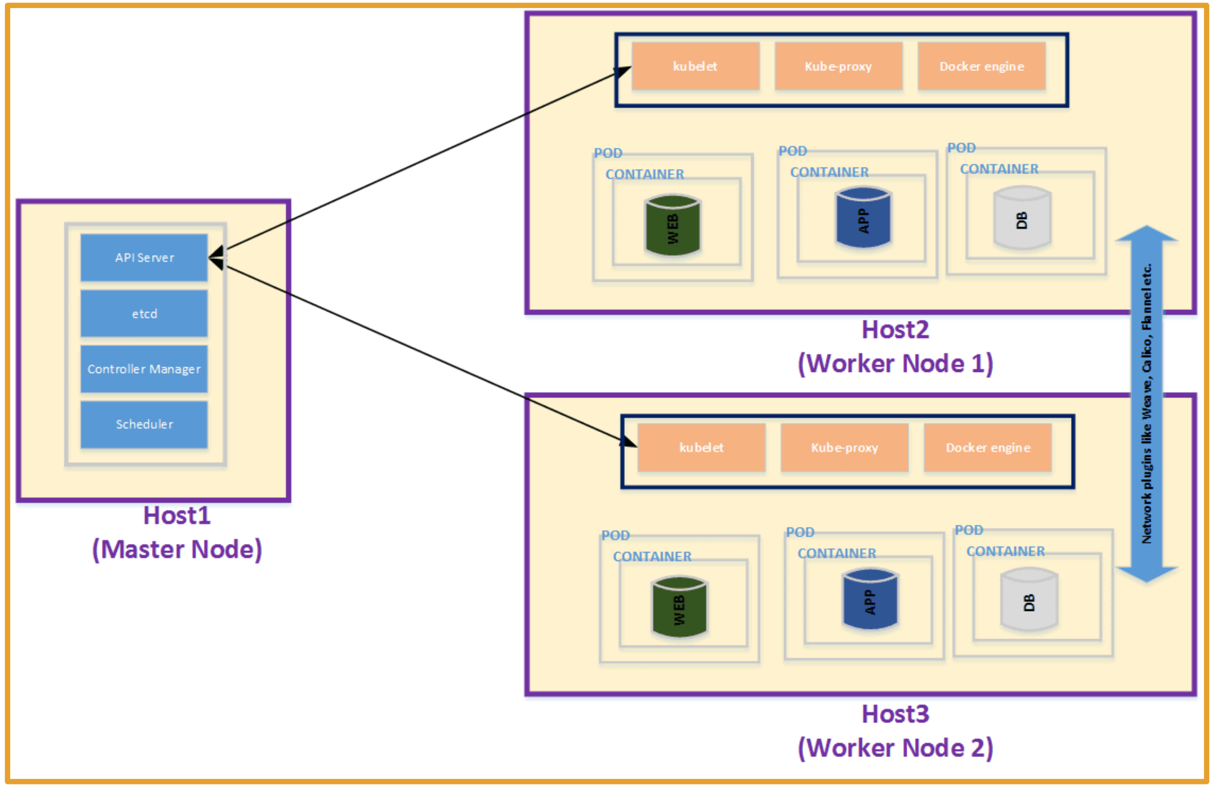


Node Structure

https://kubernetes.io/docs/concepts/architecture/nodes/

Each ***node*** contains the services necessary to run the ***Pods*** on it, which are managed by the ***control plane***.

A ***node*** may be a virtual or physical machine.



Node Structure

https://kubernetes.io/docs/concepts/architecture/nodes/

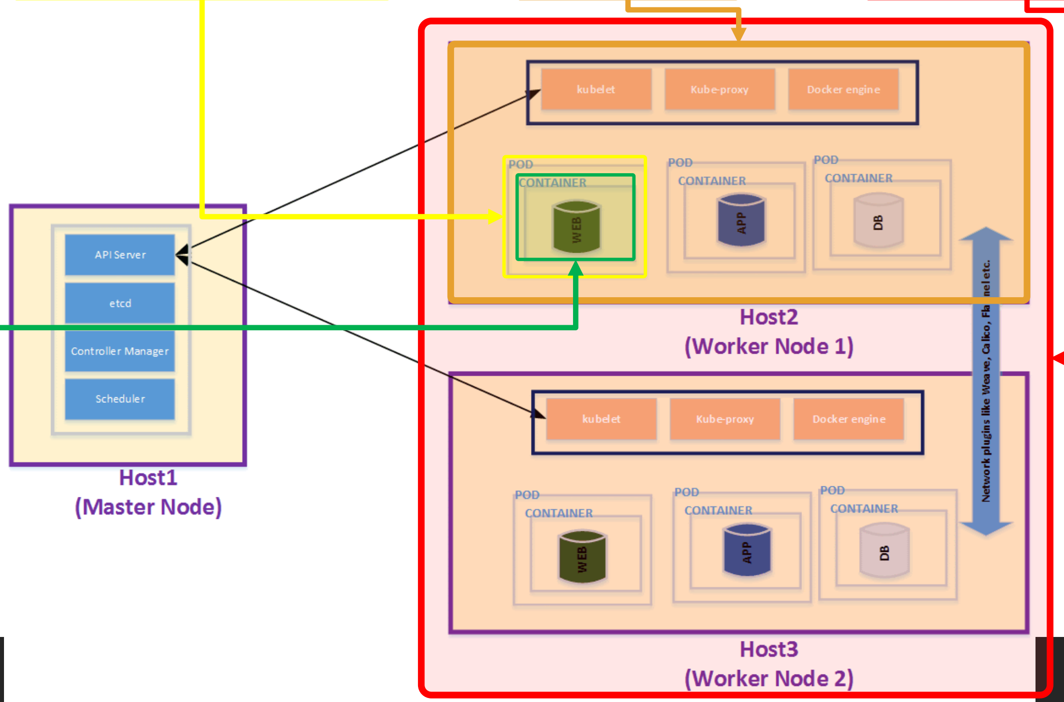
A ***container*** represents a containerized application

The ***container*** is placed into a ***Pod***.

A ***Pod*** runs on a ***Node***.

One or more ***Nodes*** make up a ***Cluster***.

The ***Cluster*** is the ***Kubernetes Workload***.

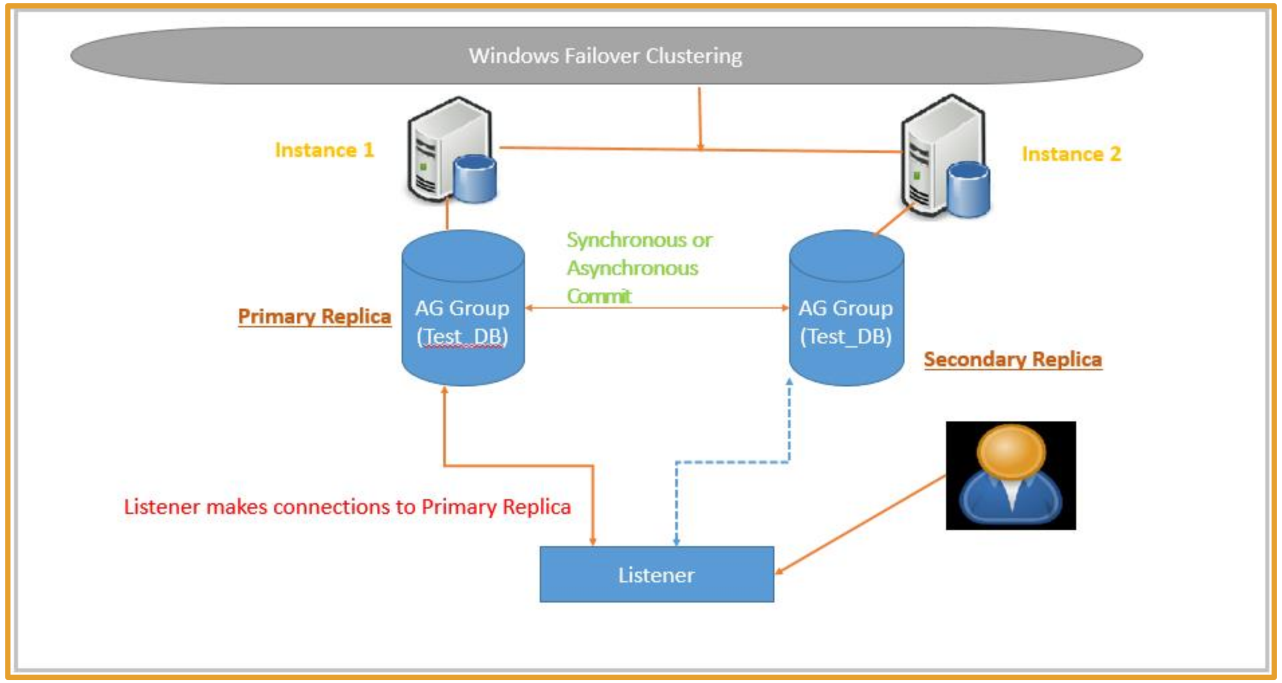


failover explained

https://en.wikipedia.org/wiki/Failover
https://devopsprodigy.com/blog/failover-in-kubernetes/

In computing and related technologies such as networking, ***failover*** is switching to a redundant or standby server, system, hardware component, or network upon the failure or abnormal termination of the previously active application, server, system, hardware component, or network.

In ***Kubernetes***, ***failover*** is avoided by creating self-balanced, self-scaled, and self-restored pods and containers. A major issues can create circumstances where an entire region or service goes down and there needs to be a backup cluster ready to handle such instances.



AzureCLI

https://docs.microsoft.com/en-us/cli/azure/install-azure-cli

az --version