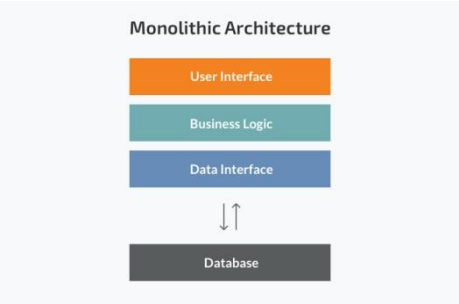
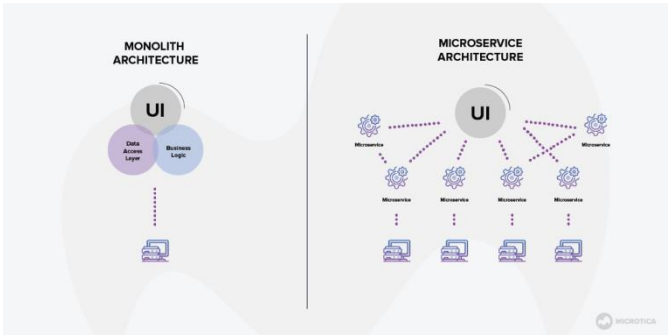
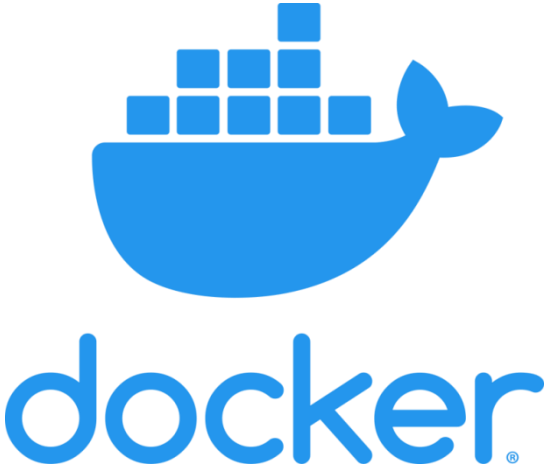
INTRODUCTION TO CLOUD SYSTEMS

Lecture 2 – Kubernetes

Previous lecture





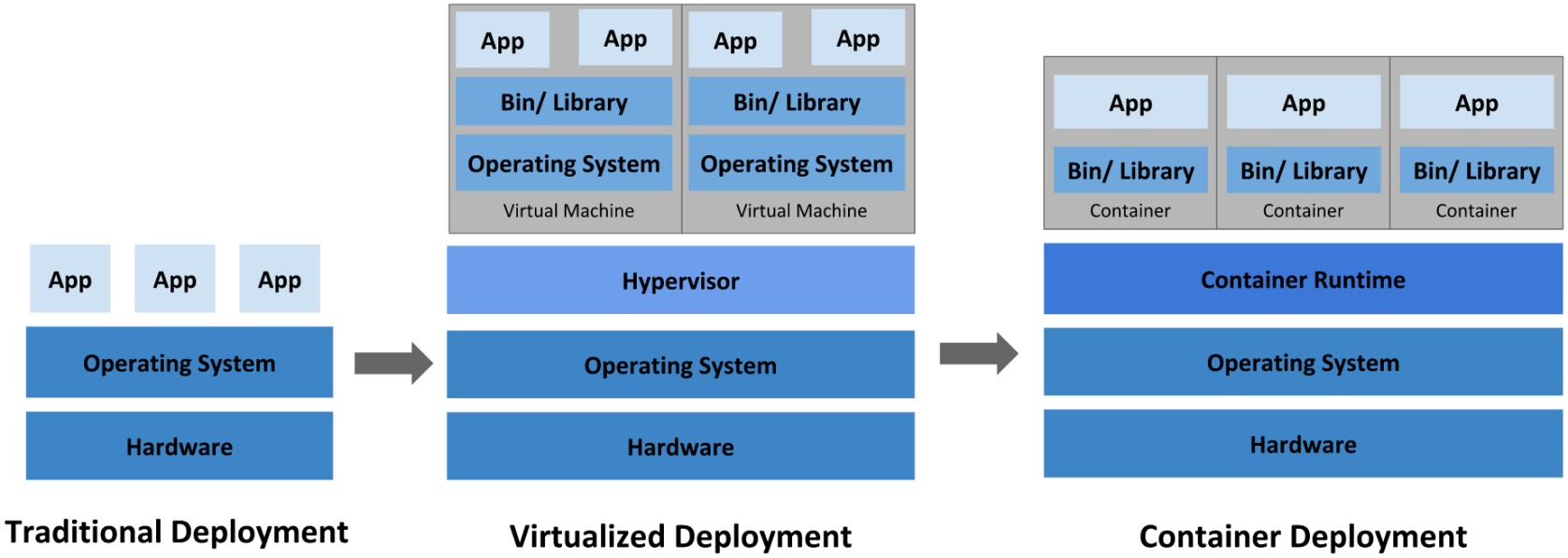


Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s". Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

Kubernetes


App
App
App
Operating System
Hardware
Traditional Deployment
App
App
App
App
Bin/ Library
Bin/ Library
Operating System
Operating System
Virtual Machine
Virtual Machine
Hypervisor
Operating System
Hardware
Virtualized Deployment
App
App
App
Bin/ Library
Bin/ Library
Bin/ Library
Container
Container
Container
Container Runtime
Operating System
Hardware
Container Deployment

Kubernetes

Containers have become popular because they provide extra benefits, such as:

- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability: not only surfaces OS-level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.

Kubernetes

- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- Resource isolation: predictable application performance.
- Resource utilization: high efficiency and density.

Kubernetes

Why you need Kubernetes and what it can do

- **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

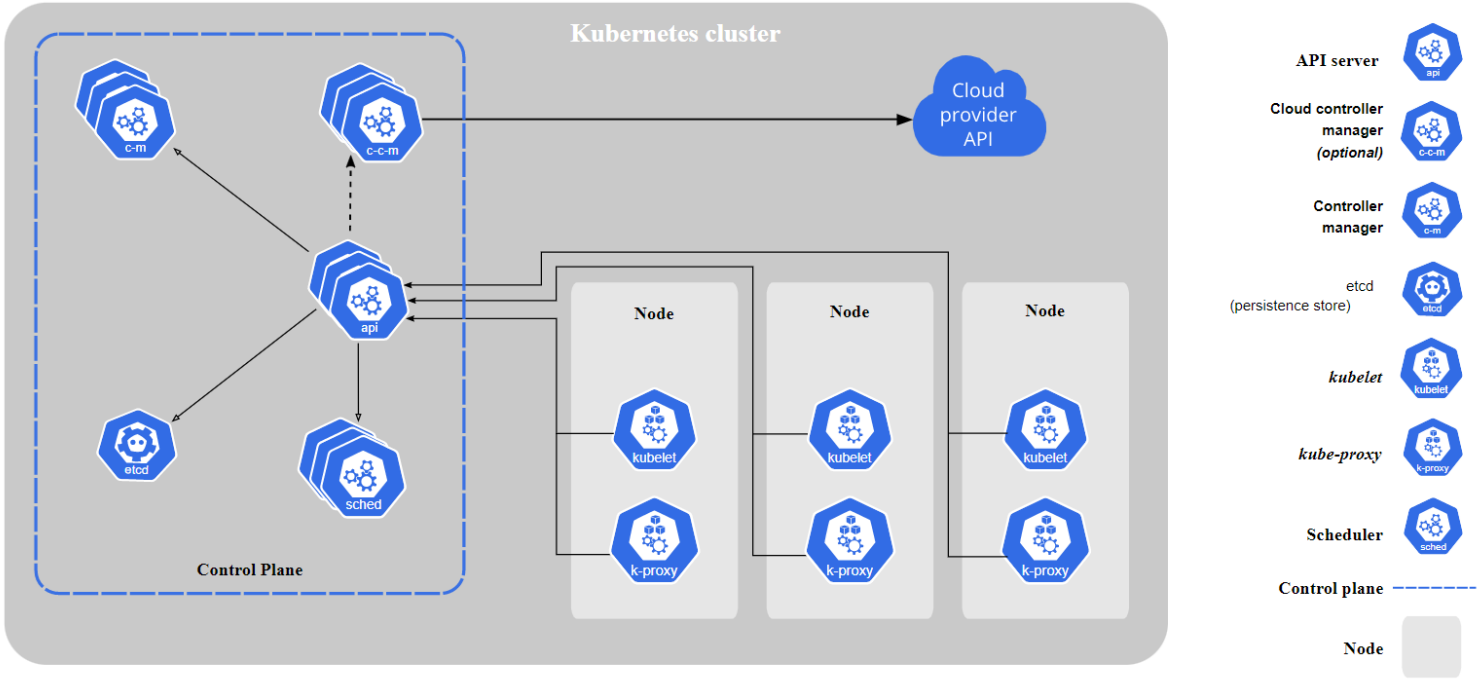
Kubernetes

Why you need Kubernetes and what it can do

- **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
- **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

Kubernetes

Components



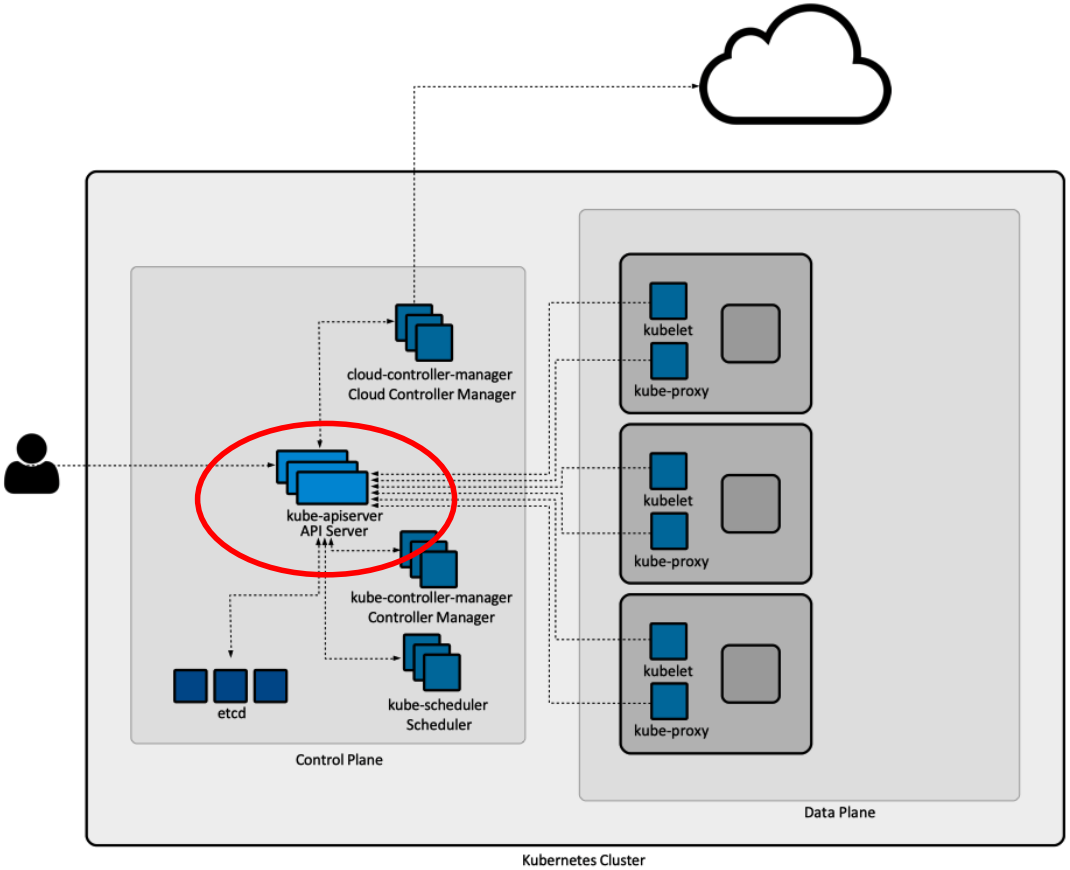
Kubernetes

Control Plane Components

kube-apiserver

The API server is a component of the Kubernetes control plane that 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-apiserver. kube-apiserver is designed to scale horizontally— it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

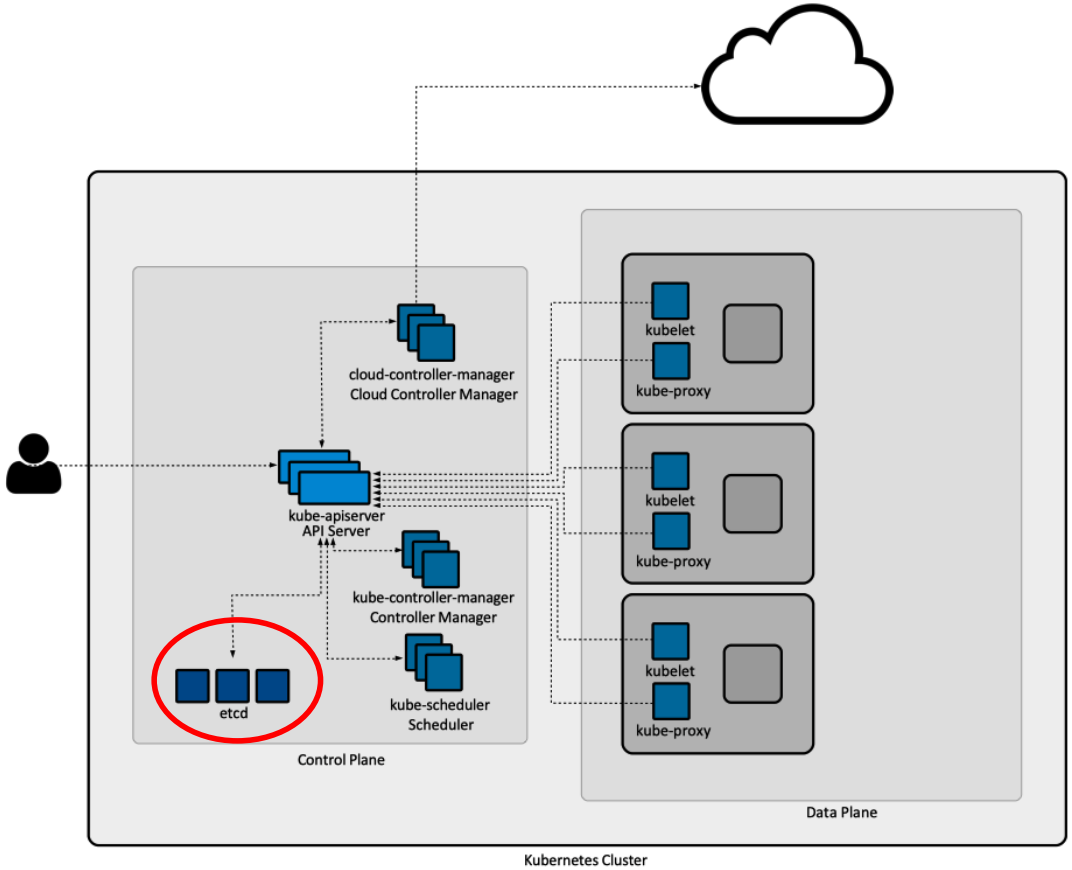


Kubernetes

Control Plane Components

etcd

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



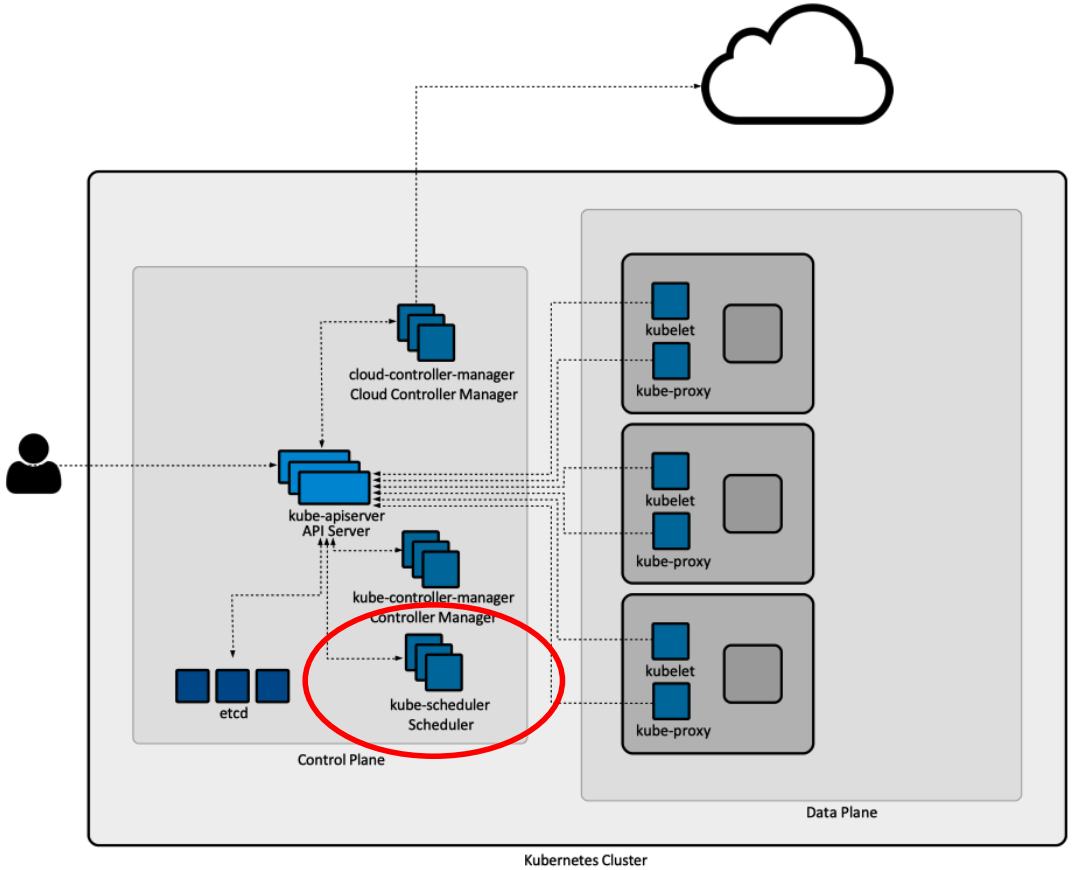
Kubernetes

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.

Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.



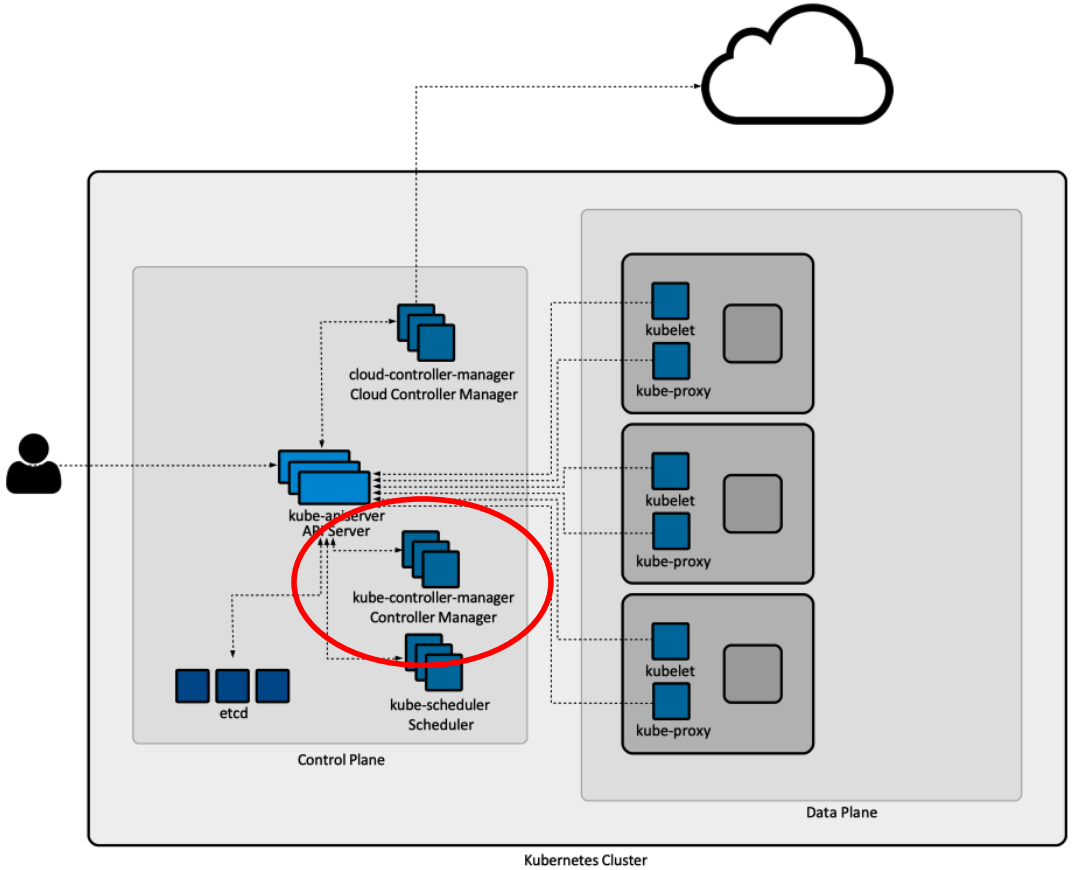
Kubernetes

Control Plane Components

kube-controller-manager

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.
Some types of these controllers 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.
- **Endpoints controller**: Populates the Endpoints object (that is, joins Services & Pods).
- **Service Account & Token controllers**: Create default accounts and API access tokens for new namespaces.

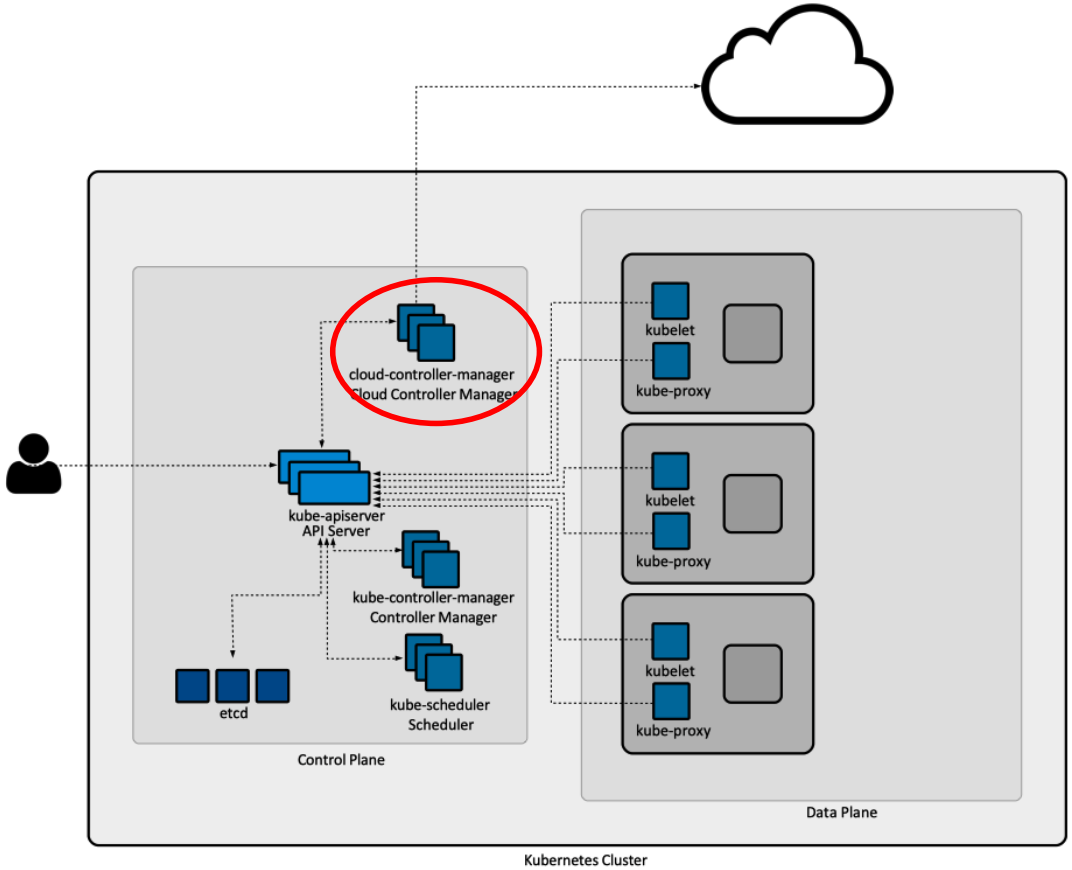


Kubernetes

Control Plane Components

cloud-controller-manager

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.
The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.



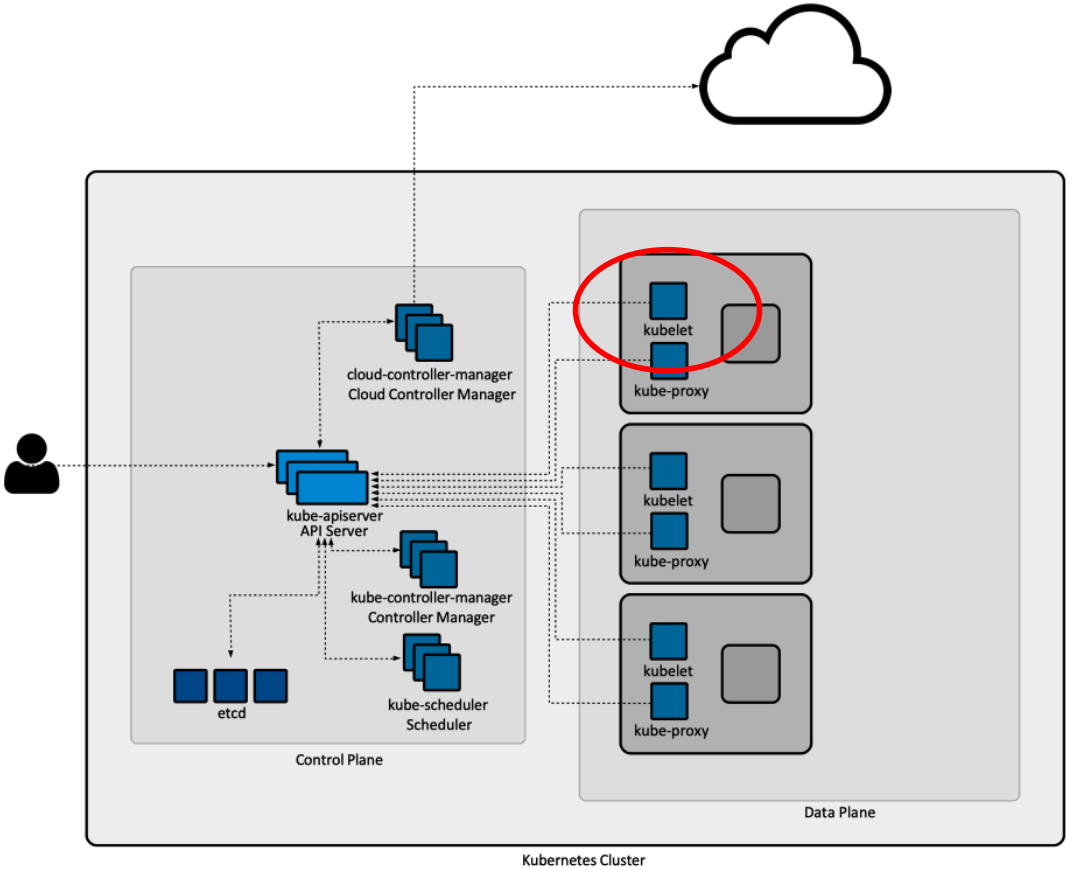
Kubernetes

Node Components

kubelet

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

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.



Kubernetes

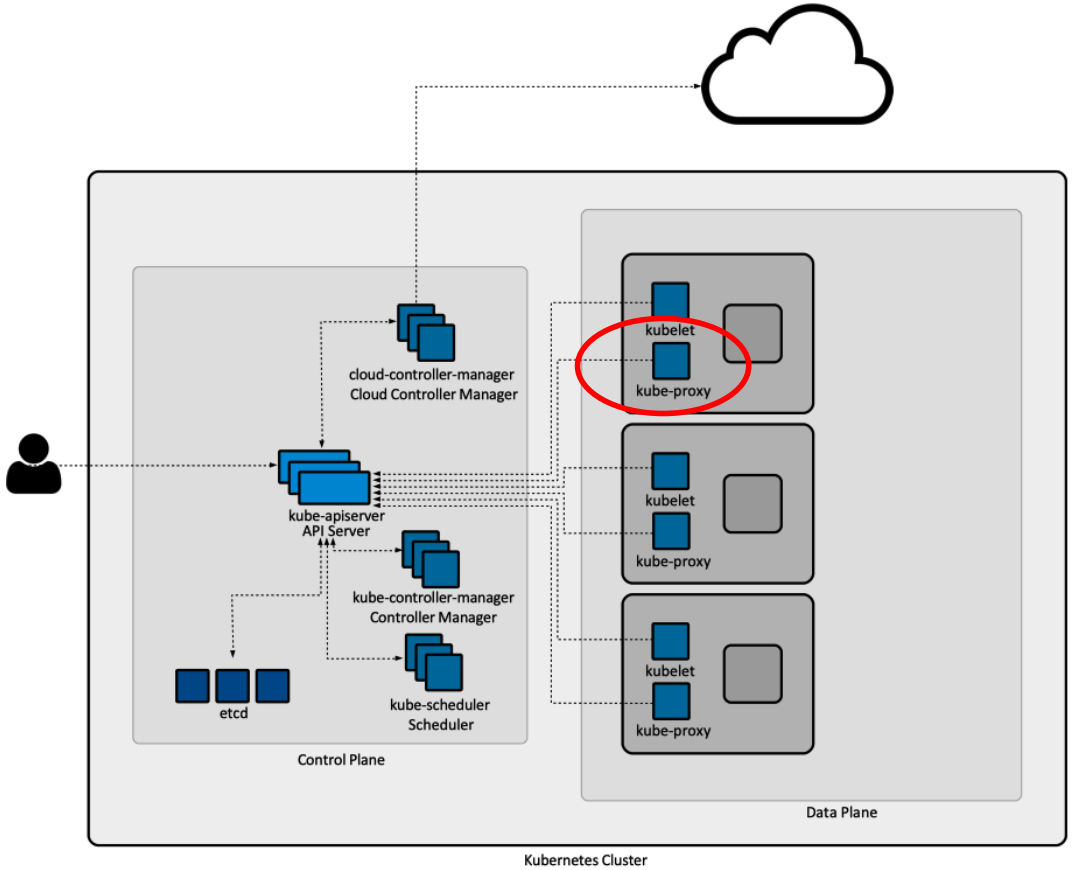
Node Components

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.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.



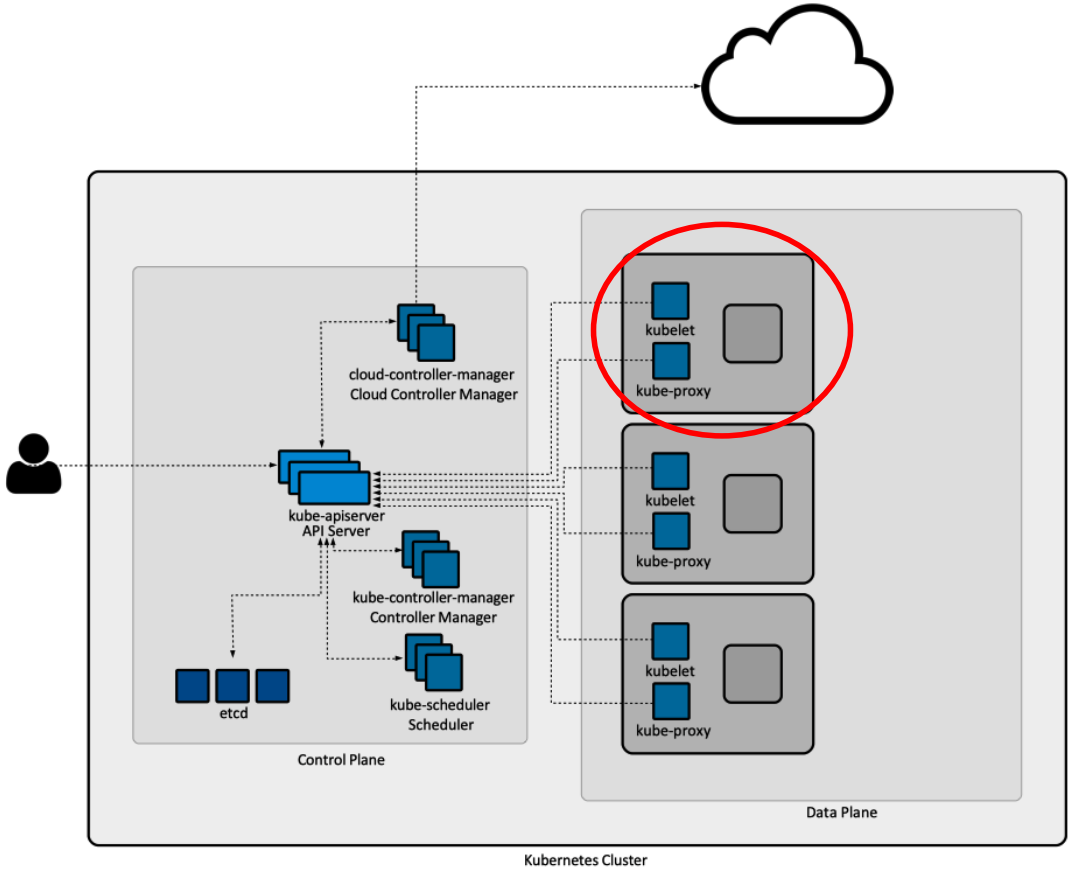
Kubernetes

Node Components

Container runtime

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

Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).



Kubernetes

Addons

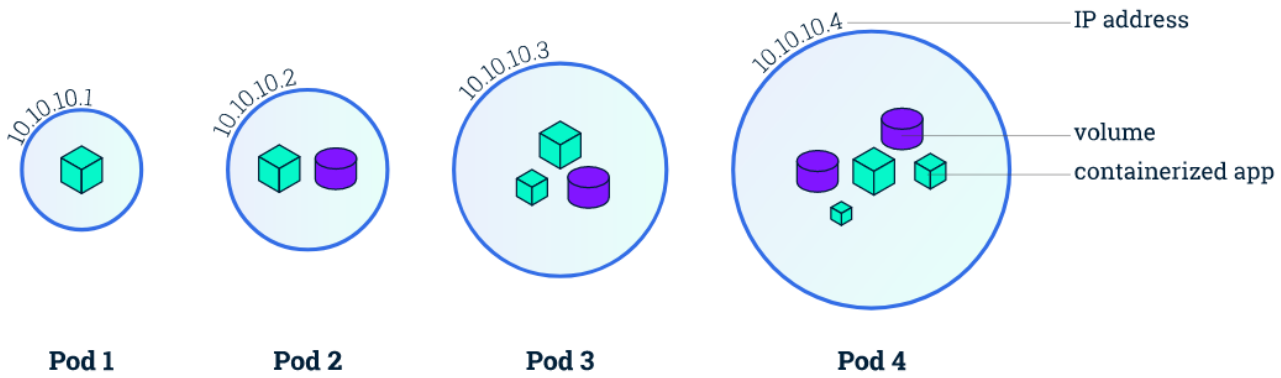
Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

- **Cluster DNS** - in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.
- **Web UI** (Dashboard) - general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.
- **Container Resource Monitoring** - records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.
- **Cluster-level Logging** - mechanism is responsible for saving container logs to a central log store with search/browsing interface.

Kubernetes

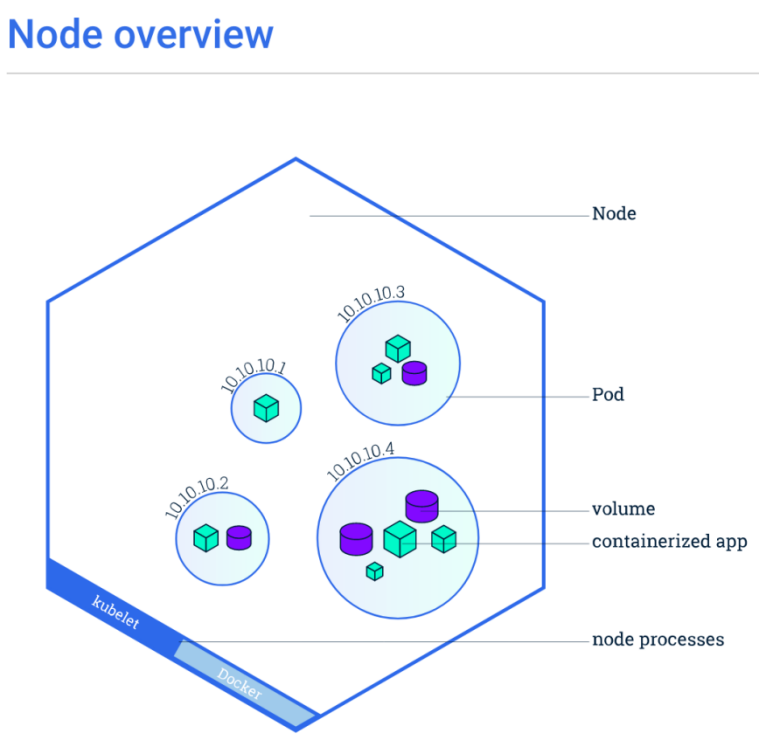
Nodes

Pods overview



Kubernetes

Nodes



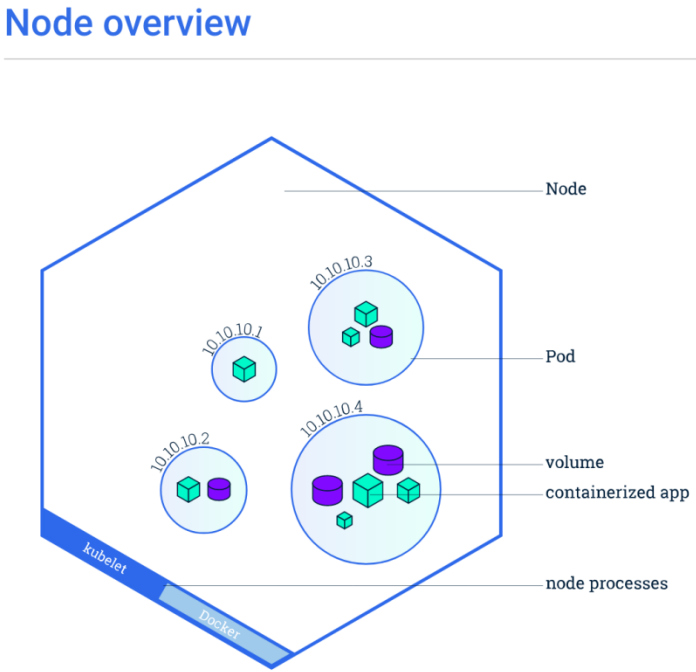
Kubernetes

Nodes

Kubernetes runs your workload by placing containers into Pods to run on Nodes. A node may be a virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods.

Typically you have several nodes in a cluster; in a learning or resource-limited environment, you might have only one node.

The components on a node include the kubelet, a container runtime, and the kube-proxy.



Kubernetes

Nodes - Management

There are two main ways to have Nodes added to the API server:

- The kubelet on a node self-registers to the control plane
- You (or another human user) manually add a Node object

After you create a Node object, or the kubelet on a node self-registers, the control plane checks whether the new Node object is valid. For example, if you try to create a Node from the following JSON manifest:

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "10.240.79.157",
    "labels": {
      "name": "my-first-k8s-node"
    }
  }
}
```

Kubernetes creates a Node object internally (the representation). Kubernetes checks that a kubelet has registered to the API server that matches the metadata.name field of the Node. If the node is healthy (i.e. all necessary services are running), then it is eligible to run a Pod. Otherwise, that node is ignored for any cluster activity until it becomes healthy.

Kubernetes

Nodes - Management

Key characteristics:

- Node name uniqueness - The name identifies a Node. Two Nodes cannot have the same name at the same time.
- Self-registration of Nodes - When the kubelet flag *--register-node* is true (the default), the kubelet will attempt to register itself with the API server. This is the preferred pattern, used by most distros.

Kubernetes

Nodes - Statuses

A Node's status contains the following information:

- Addresses
- Conditions
- Capacity and Allocatable
- Info

Kubernetes

Nodes - Statuses

A Node's status contains the following information:

- **Addresses**
- Conditions
- Capacity and Allocatable
- Info

The usage of these fields varies depending on your cloud provider or bare metal configuration.

- HostName: The hostname as reported by the node's kernel. Can be overridden via the kubelet --hostname-override parameter.
- ExternalIP: Typically the IP address of the node that is externally routable (available from outside the cluster).
- InternalIP: Typically the IP address of the node that is routable only within the cluster.

Kubernetes

Nodes - Statuses

A Node's status contains the following information:

- Addresses
- **Conditions**
- Capacity and Allocatable
- Info

Ready	True if the node is healthy and ready to accept pods, False if the node is not healthy and is not accepting pods, and Unknown if the node controller has not heard from the node in the last node-monitor-grace-period (default is 40 seconds)
DiskPressure	True if pressure exists on the disk size—that is, if the disk capacity is low; otherwise False
MemoryPressure	True if pressure exists on the node memory—that is, if the node memory is low; otherwise False
PIDPressure	True if pressure exists on the processes—that is, if there are too many processes on the node; otherwise False
NetworkUnavailable	True if the network for the node is not correctly configured, otherwise False

Kubernetes

Nodes - Statuses

A Node's status contains the following information:

- Addresses
- **Conditions**
- Capacity and Allocatable
- Info

```
"conditions": [
  {
    "type": "Ready",
    "status": "True",
    "reason": "KubeletReady",
    "message": "kubelet is posting ready status",
    "lastHeartbeatTime": "2019-06-05T18:38:35Z",
    "lastTransitionTime": "2019-06-05T11:41:27Z"
  }
]
```

Kubernetes

Nodes - Statuses

A Node's status contains the following information:

- Addresses
- Conditions
- **Capacity and Allocatable**
- Info

Describes the resources available on the node: CPU, memory, and the maximum number of pods that can be scheduled onto the node.

Kubernetes

Nodes - Statuses

A Node's status contains the following information:

- Addresses
- Conditions
- Capacity and Allocatable
- **Info**

Describes general information about the node, such as kernel version, Kubernetes version (kubelet and kube-proxy version), container runtime details, and which operating system the node uses. The kubelet gathers this information from the node and publishes it into the Kubernetes API.