Kubernetes

Containers

- A good way to bundle and run our applications.
- In a production environment
  - We need to manage the containers that
    - Run the applications and ensure that there is no downtime.
    - If a container goes down, another container needs to start.
- Kubernetes
  - Provides us with a framework to run distributed systems resiliently.
  - It takes care of scaling and failover for our application, provides deployment patterns, and more.
  - Eg: kubernetes can easily manage a canary deployment for our system.

Kubernetes

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

Kubernetes provides us with:

- 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 us to automatically mount a storage system of our choice, such as local storages, public cloud providers.

Kubernetes provides us with:

- Automated rollouts and rollbacks
  - We can describe the desired state for our deployed containers using Kubernetes
    - and it can change the actual state to the desired state at a controlled rate.
    - Eg: we can automate Kubernetes to create new containers for our deployment, remove existing containers and adopt all their resources to the new container.
- Automatic bin packing
  - We provide Kubernetes with a cluster of nodes that it can use to run containerized tasks.
  - We tell Kubernetes how much CPU and memory (RAM) each container needs.
  - Kubernetes can fit containers onto our nodes to make the best use of our resources.

Kubernetes provides us with:

- Self-healing
  - Kubernetes
    - restarts containers that fail,
    - replaces containers,
    - kills containers that don't respond to our user-defined health check, and
      doesn't advertise them to clients until they are ready to serve.

Kubernetes provides us with:

- Secret and configuration management
  - Kubernetes lets us store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys.
  - we can deploy and update secrets and application configuration without rebuilding our container images, and without exposing secrets in our stack configuration.
- Batch execution
  - In addition to services, Kubernetes can manage our batch and CI workloads, replacing containers that fail, if desired.

Kubernetes provides us with:

- Horizontal scaling
  - Scale our application up and down with a simple command, with a UI, or automatically based on CPU usage.
- IPv4/IPv6 dual-stack
  - Allocation of IPv4 and IPv6 addresses to Pods and Services
- Designed for extensibility
  - Add features to our Kubernetes cluster without changing upstream source code.

Kubernetes provides us with:

- Secret and configuration management
  - Kubernetes lets us store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys.
  - we can deploy and update secrets and application configuration without rebuilding our container images, and without exposing secrets in our stack configuration.
- Batch execution
  - In addition to services, Kubernetes can manage our batch and CI workloads, replacing containers that fail, if desired.

Kubernetes

- Not a traditional, all-inclusive PaaS system.
- operates at the container level rather than at the hardware level,
  - it provides some generally applicable features common to PaaS offerings, such as
    - deployment, scaling, load balancing, and
    - lets users integrate their logging, monitoring, and alerting solutions.
- It is not monolithic, and these default solutions are optional and pluggable.
- provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

Kubernetes:

- Does not limit the types of applications supported.
  - aims to support an extremely diverse variety of workloads,
    - including stateless, stateful, and data-processing workloads.
    - If an application can run in a container, it should run great on Kubernetes.
- Does not deploy source code and does not build our application.
  - CI/CD workflows are determined by organization cultures and preferences as well as technical requirements.

Kubernetes:

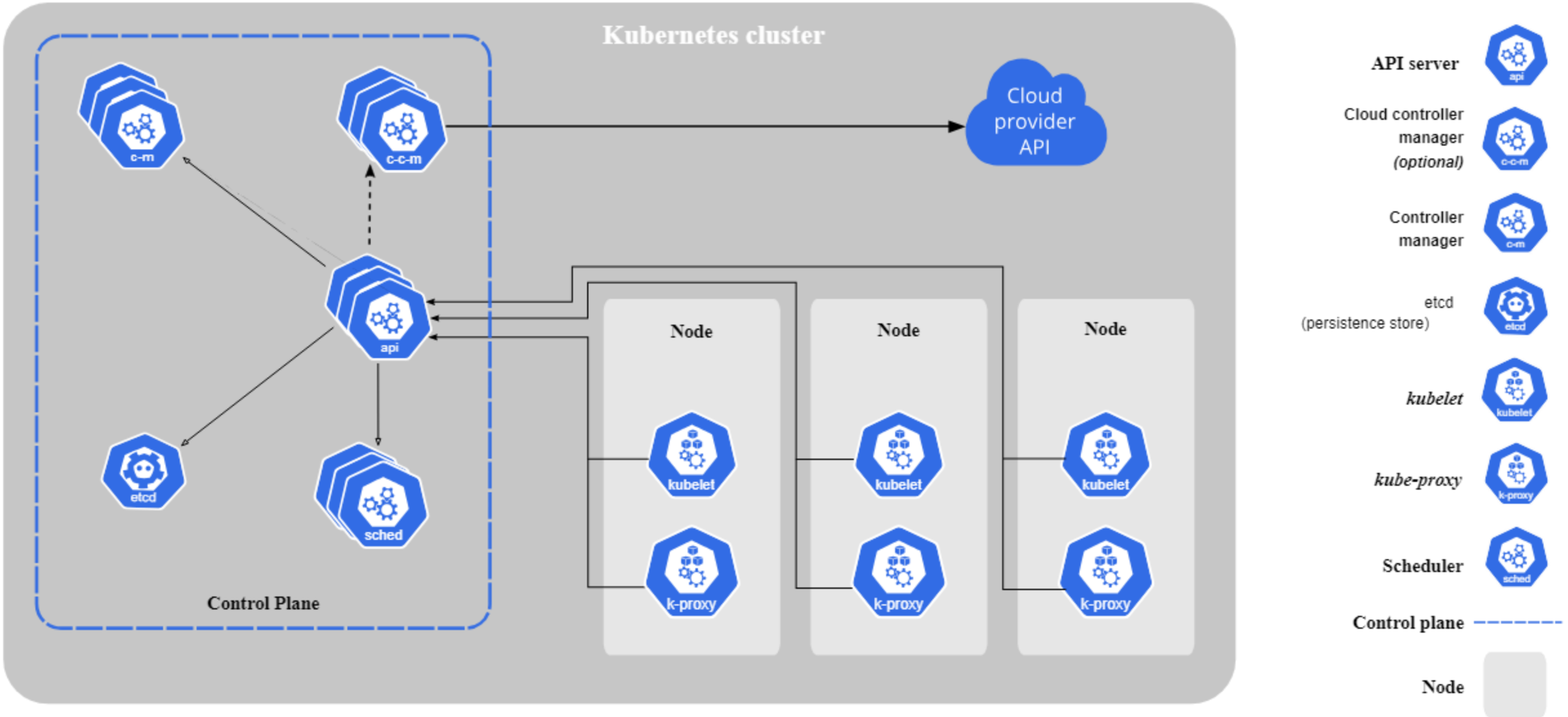
- Does not provide application-level services, such as
  - middleware (eg: message buses),
  - data-processing frameworks (eg: Spark),
  - databases (eg: MySQL),
  - caches, nor cluster storage systems (eg: Ceph) as built-in services.
- Such components
  - can run on Kubernetes and/or can be accessed by applications running on Kubernetes through portable mechanisms, such as the Open Service Broker.
- Does not dictate
  - logging, monitoring, or alerting solutions.
  - It provides some integrations as proof of concept, and mechanisms to collect and export metrics.
- Does not provide nor mandate a configuration language/system (eg: Jsonnet).
- It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.

Kubernetes:

- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.
- Kubernetes is not a mere orchestration system.
  - In fact eliminates the need for orchestration.
- orchestration
  - execution of a defined workflow: first do A, then B, then C.
- Kubernetes comprises a set of
  - independent, composable control processes that continuously drive the current state towards the provided desired state.
  - It shouldn't matter how us get from A to C.
  - Centralized control is also not required.
  - This results in a system that is easier to use and more powerful, robust, resilient, and extensible.

Kubernetes Components

- When we deploy Kubernetes, we get a cluster.
- A Kubernetes cluster consists
  - of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
- 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 environment
    - the control plane usually runs across multiple computers and
    - a cluster usually runs multiple nodes, providing fault-tolerance and high availability.


Kubernetes cluster
c-m
c-c-m
Cloud provider API
api
etcd
sched
Control Plane
Node
kubelet
k-proxy
Node
kubelet
k-proxy
Node
kubelet
k-proxy
API server
Cloud controller manager (optional)
Controller manager
etcd (persistence store)
kubelet
kube-proxy
Scheduler
Control plane
Node

Control Plane Components

- make global decisions about the cluster eg:
  - scheduling as well as
  - detecting and responding to cluster events (eg: starting up a new pod when a deployment's replicas field is unsatisfied).
- can be run on any machine in the cluster.
- To keep it simple, we can set up scripts to
  - start all control plane components on the same machine, and
  - do not run user containers on this machine.

Control Plane Components : kube-apiserver

- An API server that exposes the Kubernetes API.
- The API server is the front end for the Kubernetes control plane.
- kube-apiserver
  - Is the main implementation of a Kubernetes API server.
  - designed to scale horizontally—that is, it scales by deploying more instances.
  - We can run several instances of kube-apiserver and balance traffic between those instances.

Control Plane Components : etcd

- Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
- If our Kubernetes cluster uses etcd as its backing store, we need to have a back up plan for the data.

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.

Control Plane Components: kube-controller-manager

- Control plane component that runs controller processes.
- Logically, each controller is a separate process
  - to reduce complexity, they are all compiled into a single binary and run in a single process.
- Many different types of controllers:
  - 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.
  - EndpointSlice controller: Populates EndpointSlice objects (to provide a link between Services and Pods).
  - ServiceAccount controller: Create default ServiceAccounts for new namespaces.

Control Panel Componenet: cloud-controller-manager

- embeds cloud-specific control logic.
- lets us link our cluster into our cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with our cluster.
- only runs controllers that are specific to our cloud provider.
- If we are running Kubernetes on our own premises, or in a learning environment inside our own PC, the cluster does not have a cloud controller manager.
- Like the kube-controller-manager, it combines several logically independent control loops into a single binary that we run as a single process.
- We can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

Control Panel Component: cloud-controller-manager

- 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

- run on every node,
- maintaining running pods and
- providing the Kubernetes runtime environment.

Node Components: Kubelet

- An agent that runs on each node in the cluster.
- makes sure that containers are running in a Pod.
- takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.
- doesn't manage containers which were not created by Kubernetes.

Node Components : kube-proxy

- A network proxy that runs on each node in our cluster, implementing part of the kubernetes service concept.
- Maintains network rules on nodes.
  - These network rules allow network communication to our pods from network sessions inside or outside of your cluster.
- The operating system packet filtering layer if there is one and it's available.
  - Otherwise, forwards the traffic itself.

Node Components: Container runtime

- A fundamental component, empowers Kubernetes to run containers effectively.
- responsible for managing
  - the execution and
  - lifecycle of containers within the Kubernetes environment.
- Kubernetes supports container runtimes like
  - containerd,
  - CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

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.

Kubernetes Addon : DNS

- No addons are strictly required,
- But all Kubernetes clusters should have cluster DNS, as many examples rely on it.
- Cluster DNS
  - is a DNS server, in addition to the other DNS server(s) in our environment, which serves DNS records for Kubernetes services.
- Containers started by Kubernetes
  - automatically include this DNS server in their DNS searches.

Kubernetes Addon : Web UI (Dashboard)

- A general purpose, web-based UI for Kubernetes clusters.
- allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

Kubernetes Addon : Container Resource Monitoring

- records generic time-series metrics about containers in a central database,
- and provides a UI for browsing that data.

Kubernetes Addon : Cluster-level Logging

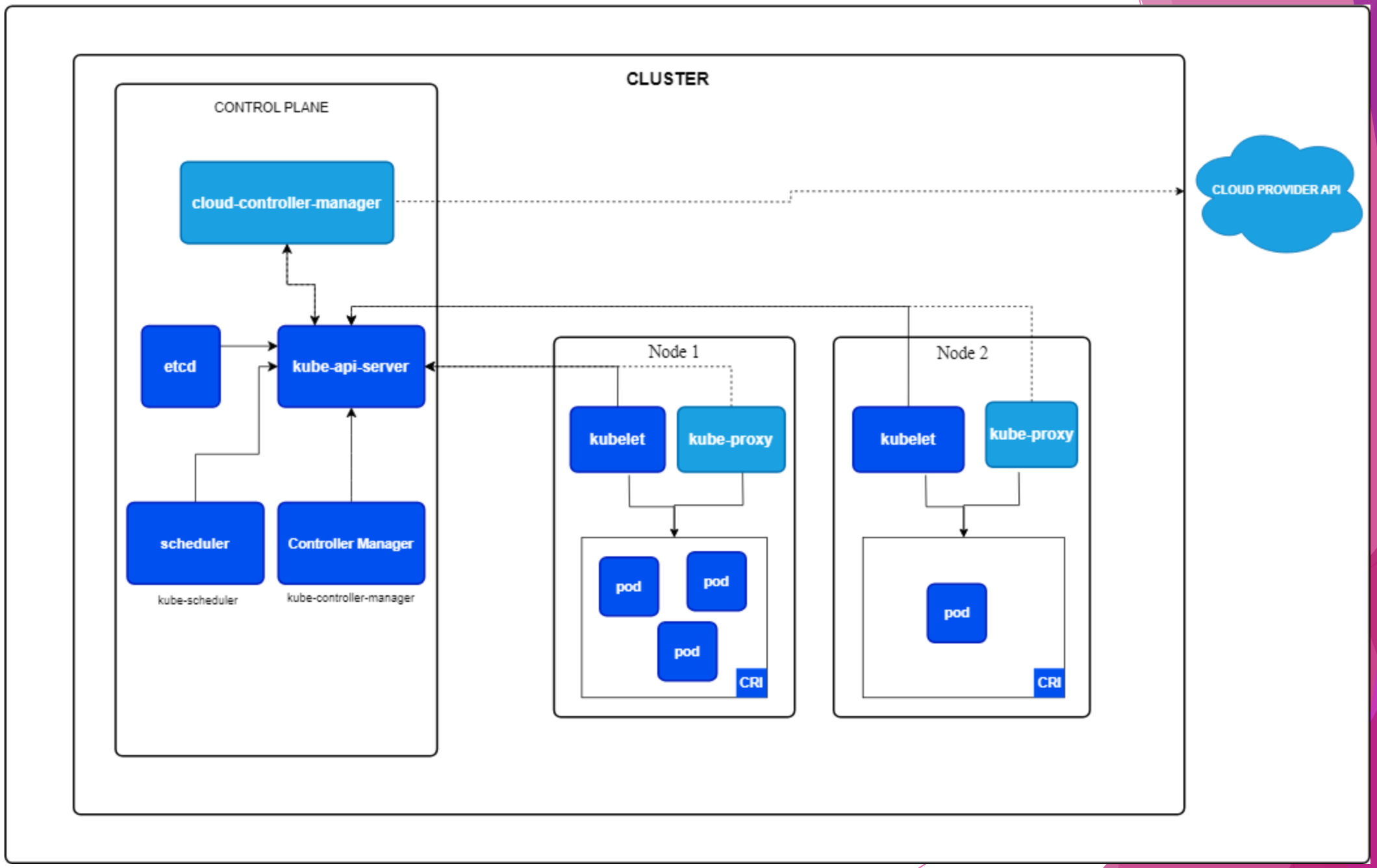
- A cluster-level logging mechanism
- responsible for saving container logs to a central log store with search/browsing interface.

Kubernetes Addon : Network Plugins

- software components that implement the container network interface (CNI) specification.
- responsible for allocating IP addresses to pods and enabling them to communicate with each other within the cluster.

Any Questions?

Kubernetes Cluster Architecture


CLUSTER
CONTROL PLANE
cloud-controller-manager
CLOUD PROVIDER API
etcd
kube-api-server
scheduler
Controller Manager
kube-scheduler
kube-controller-manager
Node 1
kubelet
kube-proxy
pod
pod
pod
CRI
Node 2
kubelet
kube-proxy
pod
CRI

Use Case

- Create a deployment to rollout a replica set
- Declare a new state for a pod
- Rollback to an earlier deployment revision
- Scale up the Deployment to facilitate more load
- Pause the rollout of the deployment
- Use the status of the deployment
- Clean up of replicasets not needed anymore