<http://bit.ly/m/bibinwilson>

Kubernetes Architecture Explained [Comprehensive Guide]

1. Understand kubernetes architecture
2. Understand the workflow of Kubernetes core components

Then you’ll love this guide.

**TABLE OF CONTENTS**

1. [Kubernetes Architecture](https://devopscube.com/kubernetes-architecture-explained/#kubernetes-architecture)
   1. [Control Plane](https://devopscube.com/kubernetes-architecture-explained/#control-plane)
   2. [Worker Node](https://devopscube.com/kubernetes-architecture-explained/#worker-node)
2. [Kubernetes Control Plane Components](https://devopscube.com/kubernetes-architecture-explained/#kubernetes-control-plane-components)
   1. [1. kube-apiserver](https://devopscube.com/kubernetes-architecture-explained/#1-kube-apiserver)
   2. [2. etcd](https://devopscube.com/kubernetes-architecture-explained/#2-etcd)
   3. [3. kube-scheduler](https://devopscube.com/kubernetes-architecture-explained/#3-kube-scheduler)
   4. [4. Kube Controller Manager](https://devopscube.com/kubernetes-architecture-explained/#4-kube-controller-manager)
   5. [5. Cloud Controller Manager (CCM)](https://devopscube.com/kubernetes-architecture-explained/#5-cloud-controller-manager-ccm)
3. [Kubernetes Worker Node Components](https://devopscube.com/kubernetes-architecture-explained/#kubernetes-worker-node-components)
   1. [1. Kubelet](https://devopscube.com/kubernetes-architecture-explained/#1-kubelet)
   2. [2. Kube proxy](https://devopscube.com/kubernetes-architecture-explained/#2-kube-proxy)
   3. [3. Container Runtime](https://devopscube.com/kubernetes-architecture-explained/#3-container-runtime)
4. [Kubernetes Cluster Addon Components](https://devopscube.com/kubernetes-architecture-explained/#kubernetes-cluster-addon-components)
   1. [1. CNI Plugin](https://devopscube.com/kubernetes-architecture-explained/#1-cni-plugin)
5. [Kubernetes Architecture FAQs](https://devopscube.com/kubernetes-architecture-explained/#kubernetes-architecture-faqs)
   1. [What is the main purpose of the Kubernetes control plane?](https://devopscube.com/kubernetes-architecture-explained/#what-is-the-main-purpose-of-the-kubernetes-control-plane)
   2. [What is the purpose of the worker nodes in a Kubernetes cluster?](https://devopscube.com/kubernetes-architecture-explained/#what-is-the-purpose-of-the-worker-nodes-in-a-kubernetes-cluster)
   3. [How is communication between the control plane and worker nodes secured in Kubernetes?](https://devopscube.com/kubernetes-architecture-explained/#how-is-communication-between-the-control-plane-and-worker-nodes-secured-in-kubernetes)
   4. [What is the purpose of the etcd key-value store in Kubernetes?](https://devopscube.com/kubernetes-architecture-explained/#what-is-the-purpose-of-the-etcd-key-value-store-in-kubernetes)
   5. [What happens to Kubernetes applications if etcd goes down?](https://devopscube.com/kubernetes-architecture-explained/#what-happens-to-kubernetes-applications-if-etcd-goes-down)
6. [Conclusion](https://devopscube.com/kubernetes-architecture-explained/#conclusion)

**Note**: To understand Kubernetes architecture better, there are some prerequisites Please check the prerequisites in the [kubernetes learning guide](https://devopscube.com/learn-kubernetes-complete-roadmap/) to know more.

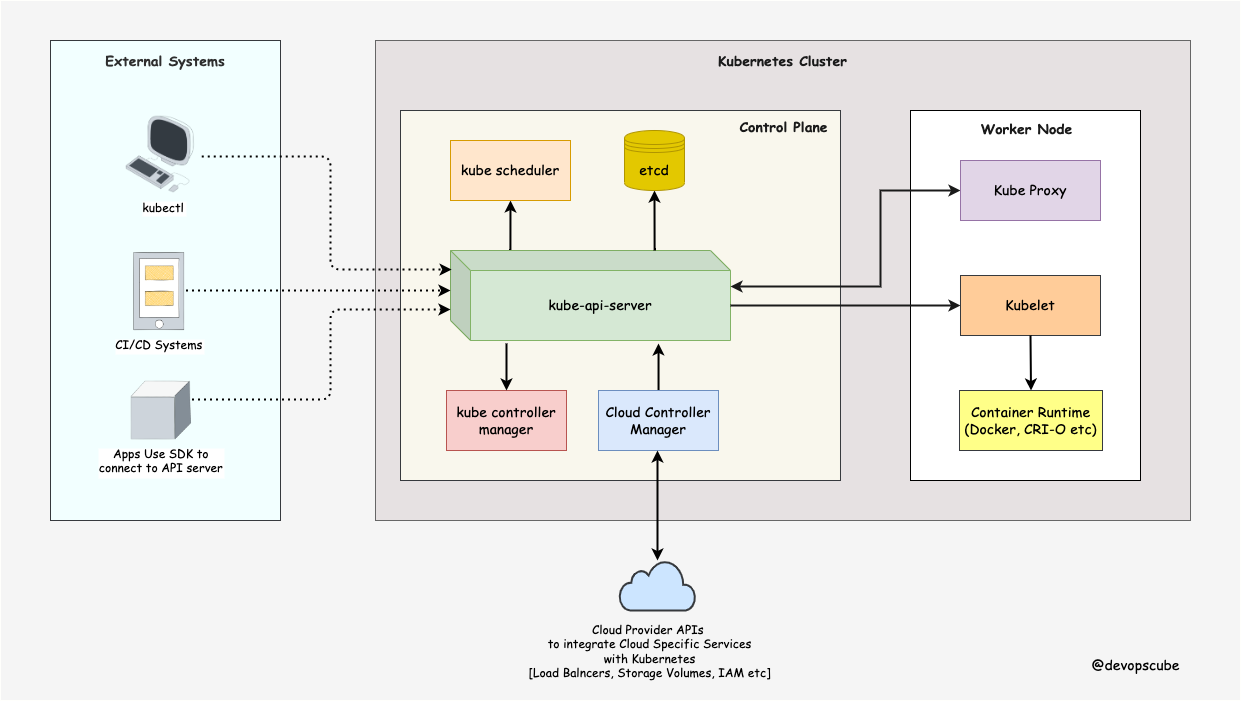
[**https://devopscube.com/learn-kubernetes-complete-roadmap/**](https://devopscube.com/learn-kubernetes-complete-roadmap/)

**TABLE OF CONTENTS**

1. [Prerequisites To Learn Kubernetes](https://devopscube.com/learn-kubernetes-complete-roadmap/#prerequisites-to-learn-kubernetes)
2. [Learn Kubernetes Architecture](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-kubernetes-architecture)
3. [$1000+ Free Cloud Credits to Deploy Clusters](https://devopscube.com/learn-kubernetes-complete-roadmap/#1000-free-cloud-credits-to-deploy-clusters)
4. [Kubernetes Cluster Setup](https://devopscube.com/learn-kubernetes-complete-roadmap/#kubernetes-cluster-setup)
5. [Understand Kubeconfig File](https://devopscube.com/learn-kubernetes-complete-roadmap/#understand-kubeconfig-file)
6. [Understand Kubernetes Objects And Resources](https://devopscube.com/learn-kubernetes-complete-roadmap/#understand-kubernetes-objects-and-resources)
7. [Learn About Pod & Associated Resources](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-about-pod-associated-resources)
8. [Learn Pod Dependent Objects](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-pod-dependent-objects)
9. [Learn Ingress & Ingress Controllers](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-ingress-ingress-controllers)
10. [Learn End to End Microservices Application Deployment on Kubernetes](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-end-to-end-microservices-application-deployment-on-kubernetes)
11. [Learn About Securing Kubernetes Cluster](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-about-securing-kubernetes-cluster)
12. [Learn About Kubernetes Configuration Management Tools](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-about-kubernetes-configuration-management-tools)
13. [Learn About Kubernetes Operator Pattern](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-about-kubernetes-operator-pattern)
14. [Learn Important Kubernetes Configurations](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-important-kubernetes-configurations)
15. [Learn Kubernetes Best Practices](https://devopscube.com/learn-kubernetes-complete-roadmap/#learn-kubernetes-best-practices)
16. [The Best Resources to Learn Kubernetes Online](https://devopscube.com/learn-kubernetes-complete-roadmap/#the-best-resources-to-learn-kubernetes-online)
17. [Kubernetes Learning GitHub Repository](https://devopscube.com/learn-kubernetes-complete-roadmap/#kubernetes-learning-github-repository)
18. [What is the Best Way to Learn Kubernetes?](https://devopscube.com/learn-kubernetes-complete-roadmap/#what-is-the-best-way-to-learn-kubernetes)
19. [What’s New in the Latest Kubernetes Release](https://devopscube.com/learn-kubernetes-complete-roadmap/#whats-new-in-the-latest-kubernetes-release)
20. [Conclusion](https://devopscube.com/learn-kubernetes-complete-roadmap/#conclusion)

**Kubernetes Architecture**

The following Kubernetes architecture diagram shows all the components of the Kubernetes cluster and how external systems connect to the Kubernetes cluster.

[](https://devopscube.com/wp-content/uploads/2022/12/k8s-architecture.drawio-1.png)

The first and foremost thing you should understand about Kubernetes is, it is a **distributed system**. Meaning, it has multiple components spread across different servers over a network. These servers could be Virtual machines or bare metal servers. We call it a Kubernetes cluster.

A Kubernetes cluster consists of control plane nodes and worker nodes.

**Control Plane**

The control plane is responsible for container orchestration and maintaining the desired state of the cluster. It has the following components.

1. kube-apiserver
2. etcd
3. kube-scheduler
4. kube-controller-manager
5. cloud-controller-manager

**Worker Node**

The Worker nodes are responsible for running containerized applications. The worker Node has the following components.

1. kubelet
2. kube-proxy
3. Container runtime

**Kubernetes Control Plane Components**

First, let’s take a look at each control plane component and the important concepts behind each component.

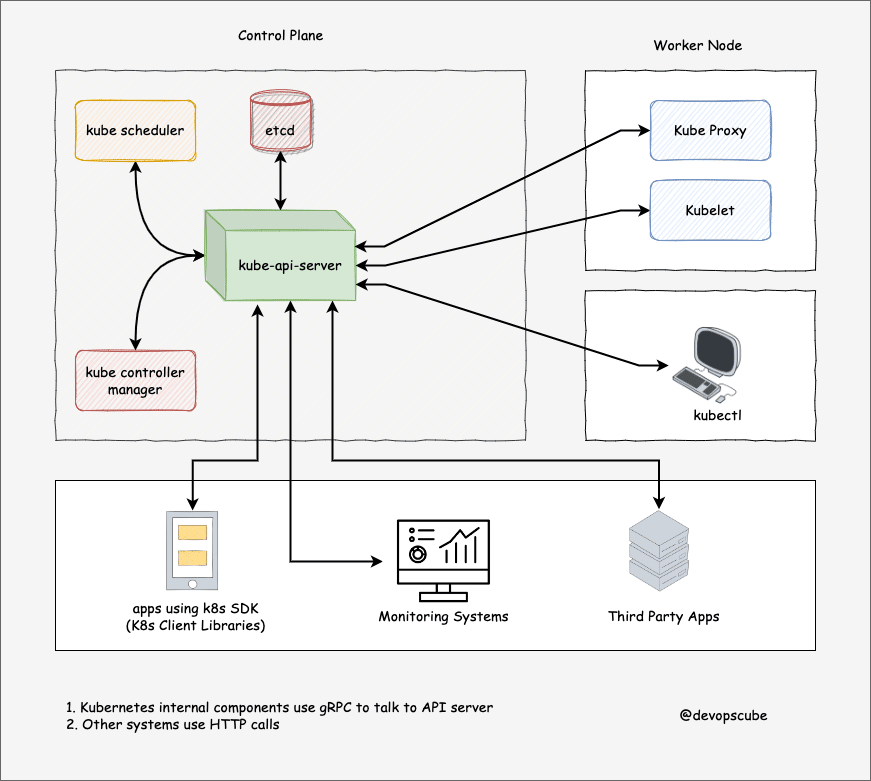
**1. kube-apiserver**

The **kube-api server** is the central hub of the Kubernetes cluster that exposes the Kubernetes API.

End users, and other cluster components, talk to the cluster via the API server. Very rarely monitoring systems and third-party services may talk to API servers to interact with the cluster.

So when you use kubectl to manage the cluster, at the backend you are actually communicating with the API server through **HTTP REST APIs**. However, the internal cluster components like the scheduler, controller, etc talk to the API server using [gRPC](https://grpc.io/docs/what-is-grpc/introduction/" \t "_blank).

The communication between the API server and other components in the cluster happens over TLS to prevent unauthorized access to the cluster.

[](https://devopscube.com/wp-content/uploads/2022/10/kube-api-server.drawio-1.png)

Kubernetes**api-server** is responsible for the following

1. **API management**: Exposes the cluster API endpoint and handles all API requests.
2. **Authentication** (Using client certificates, bearer tokens, and HTTP Basic Authentication) and **Authorization** (ABAC and RBAC evaluation)
3. Processing API requests and validating data for the API objects like pods, services, etc. (Validation and Mutation Admission controllers)
4. It is the only component that communicates with etcd.
5. api-server coordinates all the processes between the control plane and worker node components.

**Note**: To reduce the cluster attack surface, it is crucial to secure the API server. The Shadowserver Foundation has conducted an experiment that discovered [380 000 publicly accessible Kubernetes API servers](https://www.shadowserver.org/news/over-380-000-open-kubernetes-api-servers/).

**2. etcd**

Kubernetes is a distributed system and it needs an efficient distributed database like etcd that supports its distributed nature. It acts as both a backend service discovery and a database. You can call it the brain of the Kubernetes cluster.

[etcd](https://github.com/etcd-io/etcd) is an open-source strongly consistent, distributed key-value store. So what does it mean?

1. **Strongly consistent:** If an update is made to a node, strong consistency will ensure it gets updated to all the other nodes in the cluster immediately. Also if you look at [CAP theorem](https://en.wikipedia.org/wiki/CAP_theorem), achieving 100% availability with strong consistency and & Partition Tolerance is impossible.
2. **Distributed**: etcd is designed to run on multiple nodes as a cluster without sacrificing consistency.
3. **Key Value Store:** A nonrelational database that stores data as keys and values. It also exposes a key-value API. The datastore is built on top of [BboltDB](https://github.com/etcd-io/bbolt" \t "_blank) which is a fork of BoltDB.

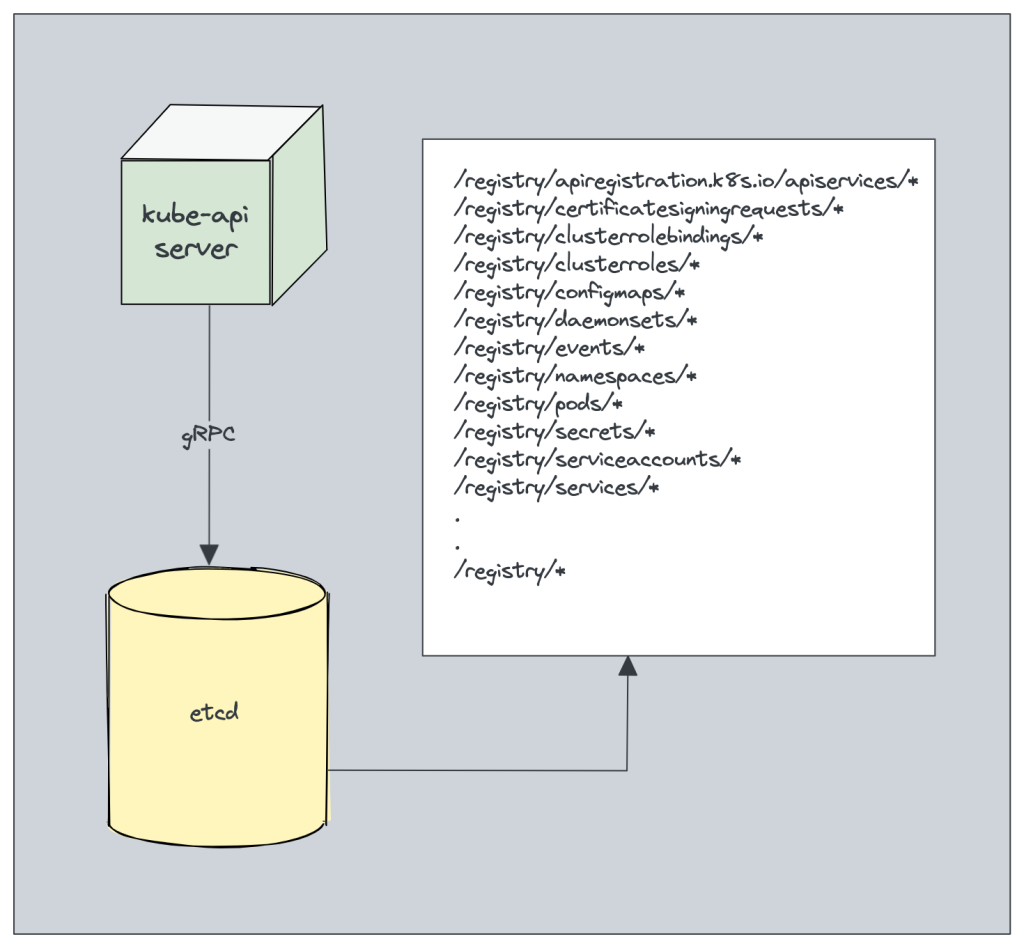
etcd uses [raft consensus algorithm](https://raft.github.io/)for strong consistency and availability. It works in a leader-member fashion for high availability and to withstand node failures.

So how etcd works with Kubernetes?

To put it simply, when you use kubectl to get kubernetes object details, you are getting it from etcd. Also, when you deploy an object like a pod, an entry gets created in etcd.

In a nutshell, here is what you need to know about etcd.

1. etcd stores all configurations, states, and metadata of Kubernetes objects (pods, secrets, daemonsets, [deployments](https://devopscube.com/kubernetes-deployment-tutorial/), configmaps, statefulsets, etc).
2. etcd allows a client to subscribe to events using Watch() API . Kubernetes api-server uses the etcd’s watch functionality to track the change in the state of an object.
3. etcd exposes key-value API [using gRPC](https://etcd.io/docs/v3.5/learning/api/). Also, the [gRPC gateway](https://etcd.io/docs/v3.3/dev-guide/api_grpc_gateway/" \t "_blank) is a RESTful proxy that translates all the HTTP API calls into gRPC messages. It makes it an ideal database for Kubernetes.
4. etcd stores all objects under the **/registry** directory key in key-value format. For example, information on a pod named Nginx in the default namespace can be found under **/registry/pods/default/nginx**

[](https://devopscube.com/wp-content/uploads/2023/01/image-5.png)

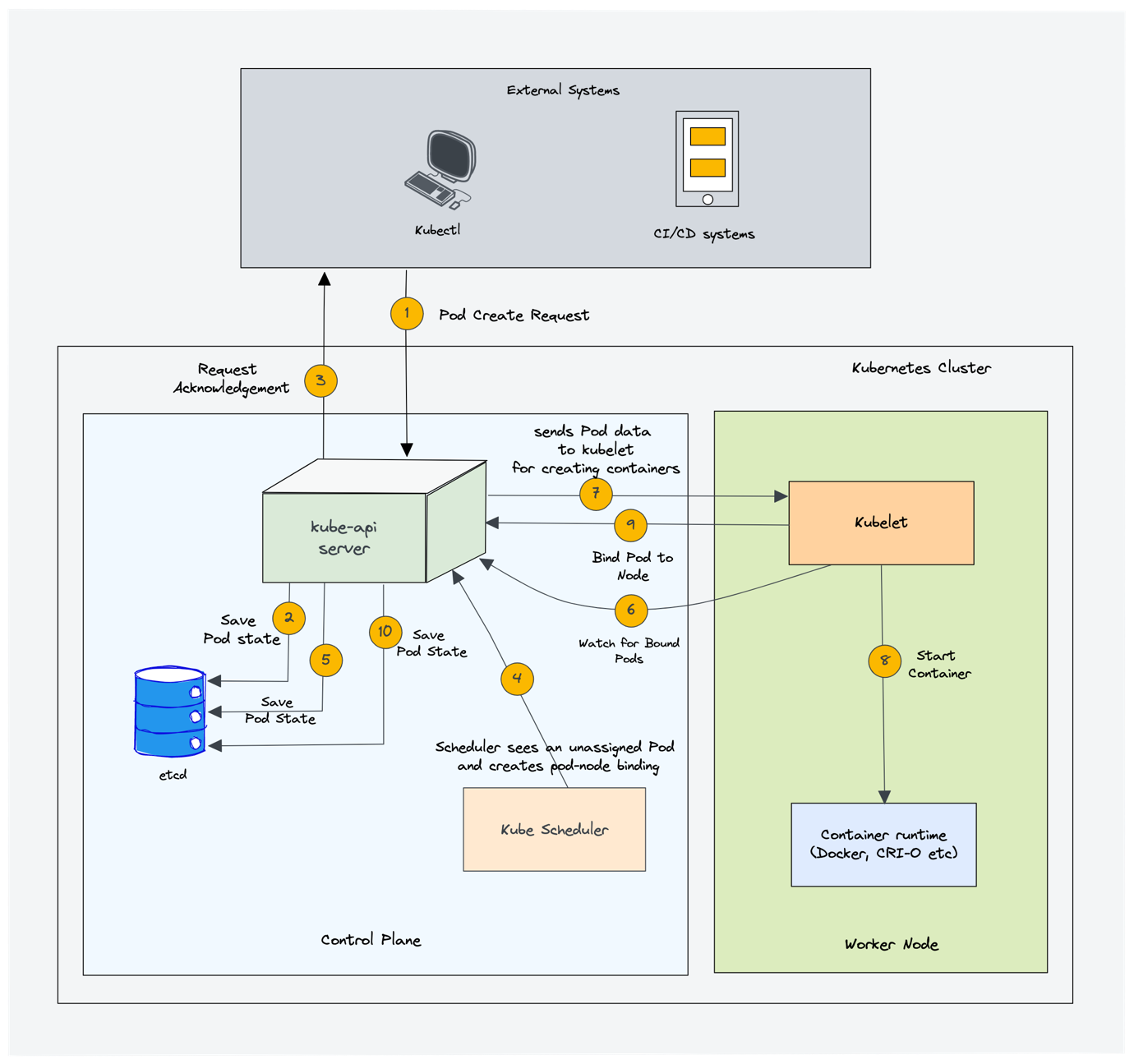
Also, etcd it is the only **Statefulset** component in the control plane.

**3. kube-scheduler**

The kube-scheduler is responsible for **scheduling pods on worker nodes**.

When you deploy a pod, you specify the pod requirements such as CPU, memory, affinity, taints or tolerations, priority, persistent volumes (PV),  etc. The scheduler’s primary task is to identify the create request and choose the best node for a pod that satisfies the requirements.

The following image shows a high-level overview of **how the scheduler works**.

[](https://devopscube.com/wp-content/uploads/2023/01/image-8.png)

In a Kubernetes cluster, there will be more than one worker node. So how does the scheduler select the node out of all worker nodes?

Here is how the scheduler works.

1. To choose the best node, the Kube-scheduler uses **filtering and scoring**operations.
2. In **filtering**, the scheduler finds the best-suited nodes where the pod can be scheduled. For example, if there are five worker nodes with resource availability to run the pod, it selects all five nodes. If there are no nodes, then the pod is unschedulable and moved to the scheduling queue. If It is a large cluster, let’s say 100 worker nodes, and the scheduler doesn’t iterate over all the nodes. There is a scheduler configuration parameter called **percentageOfNodesToScore**. The default value is typically **50%**. So it tries to iterate over 50% of nodes in a round-robin fashion. If the worker nodes are spread across multiple zones, then the scheduler iterates over nodes in different zones. For very large clusters the default **percentageOfNodesToScore** is 5%.
3. In the **scoring phase**, the scheduler ranks the nodes by assigning a score to the filtered worker nodes. The scheduler makes the scoring by calling multiple [scheduling plugins](https://kubernetes.io/docs/reference/scheduling/config/#scheduling-plugins). Finally, the worker node with the highest rank will be selected for scheduling the pod. If all the nodes have the same rank, a node will be selected at random.
4. Once the node is selected, the scheduler creates a binding event in the API server. Meaning an event to bind a pod and node.

Things should know about a scheduler.

1. It is a controller that listens to pod creation events in the API server.
2. The scheduler has two phases. **Scheduling cycle** and the **Binding cycle**. Together it is called the scheduling context.Thescheduling cycle selects a worker node and the binding cycle applies that change to the cluster.
3. The scheduler always places the high-priority pods ahead of the low-priority pods for scheduling. Also, in some cases, after the pod started running in the selected node, the pod might get evicted or moved to other nodes. If you want to understand more, read the [Kubernetes pod priority guide](https://devopscube.com/pod-priorityclass-preemption/)
4. You can create custom schedulers and run multiple schedulers in a cluster along with the native scheduler. When you deploy a pod you can specify the custom scheduler in the pod manifest. So the scheduling decisions will be taken based on the custom scheduler logic.
5. The scheduler has a [pluggable scheduling framework](https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/). Meaning, you can add your custom plugin to the scheduling workflow.

**4. Kube Controller Manager**

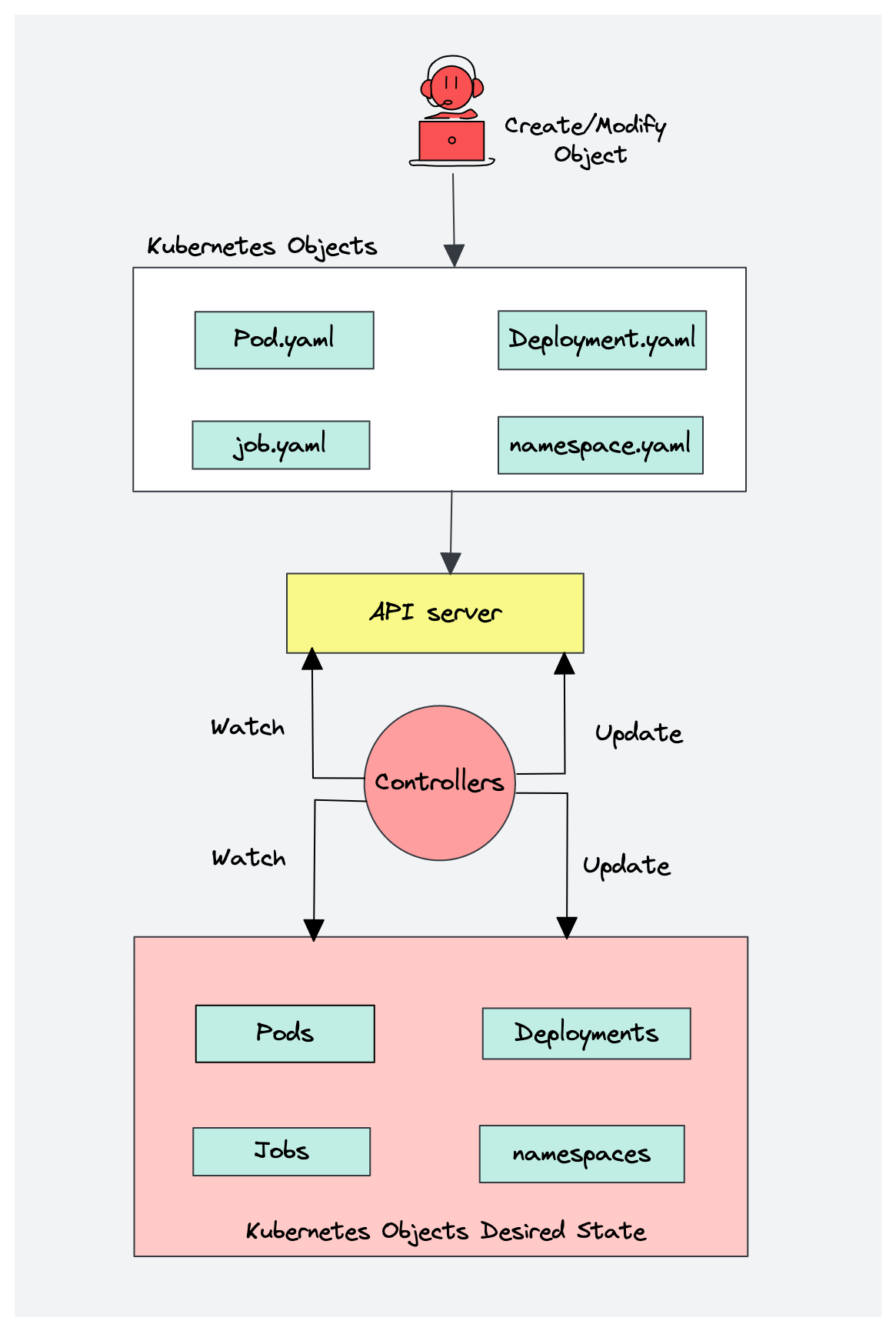
**What is a controller?**[Controllers](https://kubernetes.io/docs/concepts/architecture/controller/) are programs that run infinite control loops. Meaning it runs continuously and watches the actual and desired state of objects. If there is a difference in the actual and desired state, it ensures that the kubernetes resource/object is in the desired state.

As per the official documentation,

In Kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed. Each controller tries to move the current cluster state closer to the desired state.

Let’s say you want to create a deployment, you specify the desired state in the manifest YAML file (declarative approach). For example, 2 replicas, one volume mount, configmap, etc. The in-built deployment controller ensures that the deployment is in the desired state all the time. If a user updates the deployment with 5 replicas, the deployment controller recognizes it and ensures the desired state is 5 replicas.

**Kube controller manager** is a component that manages all the Kubernetes controllers. Kubernetes resources/objects like pods, namespaces, jobs, replicaset are managed by respective controllers. Also, the kube scheduler is also a controller managed by Kube controller manager.

[](https://devopscube.com/wp-content/uploads/2023/01/image-9.png)

Following is the list of important built-in Kubernetes controllers.

1. Deployment controller
2. Replicaset controller
3. DaemonSet controller
4. Job Controller ([Kubernetes Jobs](https://devopscube.com/create-kubernetes-jobs-cron-jobs/))
5. CronJob Controller
6. endpoints controller
7. namespace controller
8. [service accounts](https://devopscube.com/kubernetes-api-access-service-account/) controller.
9. Node controller

Here is what you should know about the Kube controller manager.

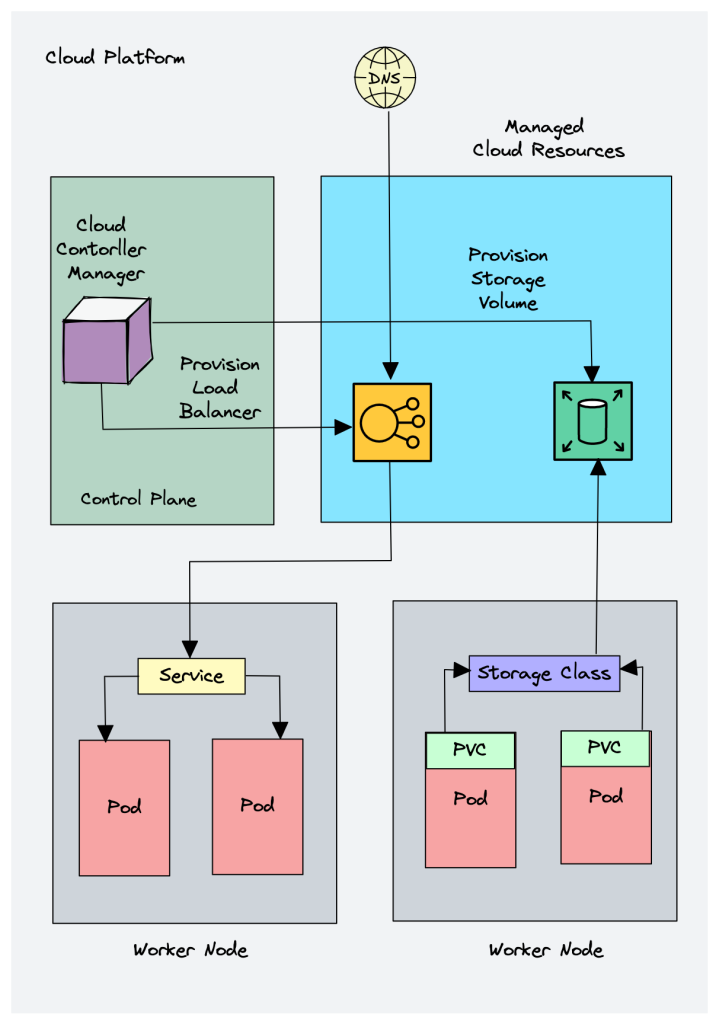
1. It manages all the controllers and the controllers try to keep the cluster in the desired state.
2. You can extend kubernetes with **custom controllers**associated with a custom resource definition.

**5. Cloud Controller Manager (CCM)**

When kubernetes is deployed in cloud environments, the cloud controller manager acts as a bridge between Cloud Platform APIs and the Kubernetes cluster.

This way the core kubernetes core components can work independently and allow the cloud providers to integrate with kubernetes using plugins. (For example, an interface between kubernetes cluster and AWS cloud API)

Cloud controller integration allows Kubernetes cluster to provision cloud resources like instances (for nodes), Load Balancers (for services), and Storage Volumes (for persistent volumes).

[](https://devopscube.com/wp-content/uploads/2023/01/Cloud-Controller-Manager.png)

Cloud Controller Manager contains a set of **cloud platform-specific controllers** that ensure the desired state of cloud-specific components (nodes, Loadbalancers, storage, etc). Following are the three main controllers that are part of the cloud controller manager.

1. **Node controller:** This controller updates node-related information by talking to the cloud provider API. For example, node labeling & annotation, getting hostname, CPU & memory availability, nodes health, etc.
2. **Route controller:** It is responsible for configuring networking routes on a cloud platform. So that pods in different nodes can talk to each other.
3. **Service controller**: It takes care of deploying load balancers for kubernetes services, assigning IP addresses, etc.

Following are some of the classic examples of cloud controller manager.

1. Deploying Kubernetes Service of type Load balancer. Here Kubernetes provisions a Cloud-specific Loadbalancer and integrates with Kubernetes Service.
2. Provisioning storage volumes (PV) for pods backed by cloud storage solutions.

Overall **Cloud Controller Manager manages the** lifecycle of cloud-specific resources used by kubernetes.

**Kubernetes Worker Node Components**

Now let’s look at each of the worker node components.

**1. Kubelet**

Kubelet is an agent component that runs on every node in the cluster. t does not run as a container instead runs as a daemon, managed by systemd.

It is responsible for registering worker nodes with the API server and working with the podSpec (Pod specification – YAML or JSON) primarily from the API server. podSpec defines the containers that should run inside the pod, their resources (e.g. CPU and memory limits), and other settings such as environment variables, volumes, and labels.

It then brings the podSpec to the desired state by creating containers.

To put it simply, kubelet is responsible for the following.

1. Creating, modifying, and deleting containers for the pod.
2. Responsible for handling liveliness, readiness, and startup probes.
3. Responsible for Mounting volumes by reading pod configuration and creating respective directories on the host for the volume mount.
4. Collecting and reporting Node and pod status via calls to the API server.

Kubelet is also a controller where it watches for pod changes and utilizes the node’s container runtime to pull images, run containers, etc.

Other than PodSpecs from the API server, kubelet can accept podSpec from a file, HTTP endpoint, and HTTP server. A good example of “podSpec from a file” is Kubernetes static pods.

Static pods are controlled by kubelet, not the API servers.

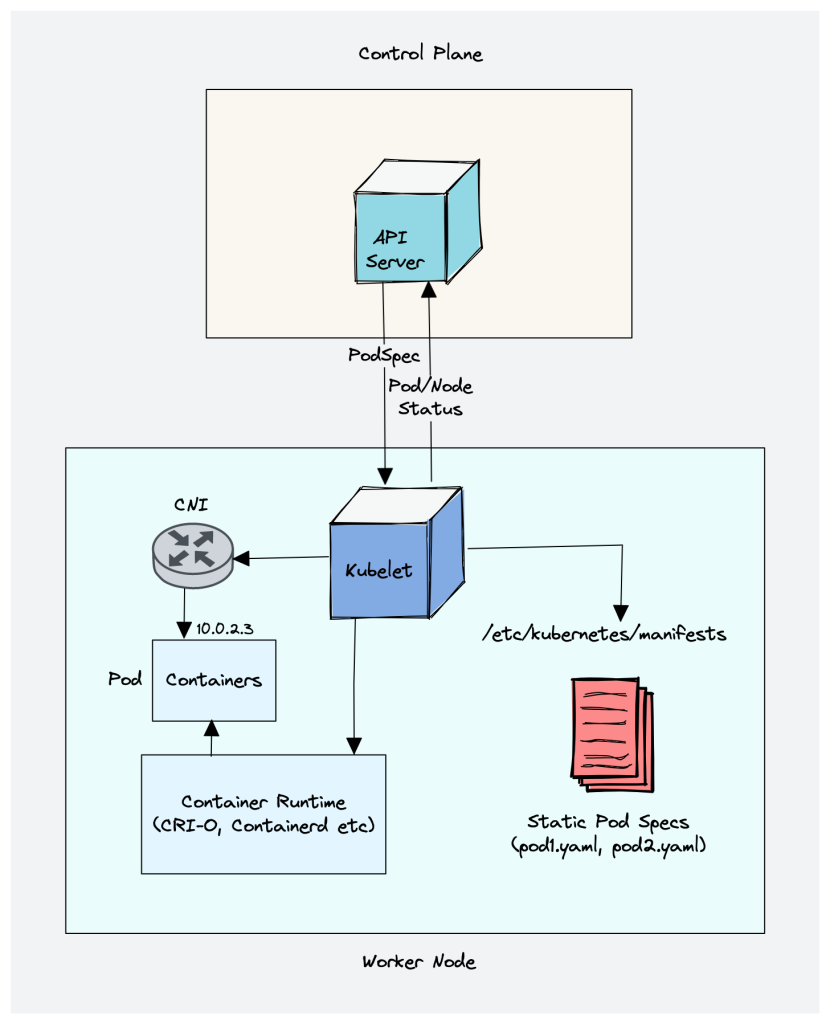
This means you can create pods by providing a pod YAML location to the Kubelet component. However, static pods created by Kubelet are not managed by the API server.

Here is a real-world example use case of the static pod.

While bootstrapping the control plane, kubelet starts the api-server, scheduler, and controller manager as static pods from podSpecs located at /etc/kubernetes/manifests

Following are some of the key things about kubelet.

1. Kubelet uses the CRI (container runtime interface) gRPC interface to talk to the container runtime.
2. It also exposes an HTTP endpoint to stream logs and provides exec sessions for clients.
3. Uses the CSI (container storage interface) gRPC to configure block volumes.
4. It uses the CNI plugin configured in the cluster to allocate the pod IP address and set up any necessary network routes and firewall rules for the pod.

[](https://devopscube.com/wp-content/uploads/2023/01/kubelet-architecture.png)

**2. Kube proxy**

To understand kube proxy, you need to have a basic knowledge of Kubernetes Service & endpoint objects.

Service in Kubernetes is a way to expose a set of pods internally or to external traffic. When you create the service object, it gets a virtual IP assigned to it. It is called clusterIP. It is only accessible within the Kubernetes cluster.

The Endpoint object contains all the IP addresses and ports of pod groups under a Service object. The endpoints controller is responsible for maintaining a list of pod IP addresses (endpoints). The service controller is responsible for configuring endpoints to a service.

You cannot ping the ClusterIP because it is only used for service discovery, unlike pod IPs which are pingable.

Now let’s understand Kube Proxy.

Kube-proxy is a daemon that runs on every node as a daemonset. It is a proxy component that implements the Kubernetes Services concept for pods. (single DNS for a set of pods with load balancing).

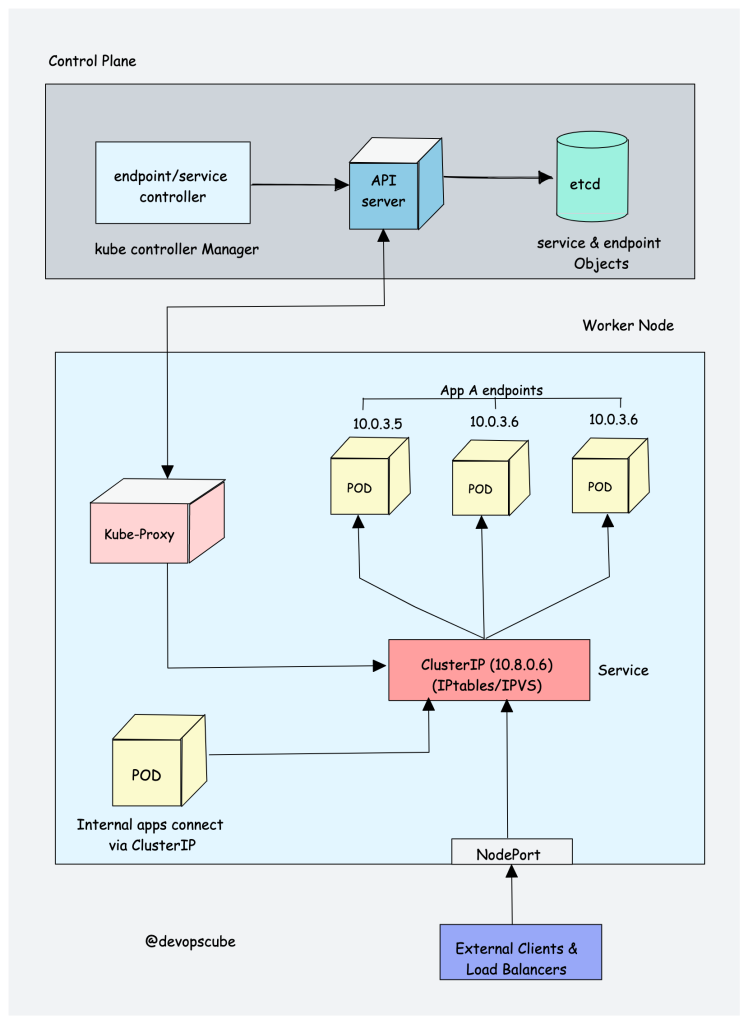
When you expose pods using a Service (ClusterIP), Kube-proxy creates network rules to send traffic to the backend pods (endpoints) grouped under the Service object. Meaning, all the load balancing, and service discovery are handled by the Kube proxy.

**So how does Kube-proxy work?**

Kube proxy talks to the API server to get the details about the Service (ClusterIP) and respective pod IPs & ports (endpoints). It also monitors for changes in service and endpoints.

Kube-proxy then uses any one of the following modes to create/update rules for routing traffic to pods behind a Service

1. [**IPTables**](https://wiki.centos.org/HowTos/Network/IPTables): It is the default mode. In IPTables mode, the traffic is handled by IPtable rules. In this mode, kube-proxy chooses the backend pod random for load balancing. Once the connection is established, the requests go to the same pod until the connection is terminated.
2. [**IPVS**](https://en.wikipedia.org/wiki/IP_Virtual_Server)**:** For clusters with services exceeding 1000, IPVS offers performance improvement. It supports the following load-balancing algorithms for the backend.
   1. rr: round-robin : It is the default mode.
   2. lc: least connection (smallest number of open connections)
   3. dh: destination hashing
   4. sh: source hashing
   5. sed: shortest expected delay
   6. nq: never queue
3. **Userspace** (legacy & not recommended)
4. **Kernelspace**: This mode is only for windows systems.

[](https://devopscube.com/wp-content/uploads/2023/01/image-14.png)

If you would like to understand the performance difference between kube-proxy IPtables and IPVS mode, [read this article](https://www.tigera.io/blog/comparing-kube-proxy-modes-iptables-or-ipvs/).

Also, you can run a Kubernetes cluster without kube-proxy by replacing it with [Cilium](https://docs.cilium.io/en/v1.9/gettingstarted/kubeproxy-free/).

**3. Container Runtime**

You probably know about [Java Runtime (JRE)](https://aws.amazon.com/what-is/java-runtime-environment/). It is the software required to run Java programs on a host. In the same way, container runtime is a software component that is required to run containers.

Container runtime runs on all the nodes in the Kubernetes cluster. It is responsible for pulling images from container registries, running containers, allocating and isolating resources for containers, and managing the entire lifecycle of a container on a host.

To understand this better, let’s take a look at two key concepts:

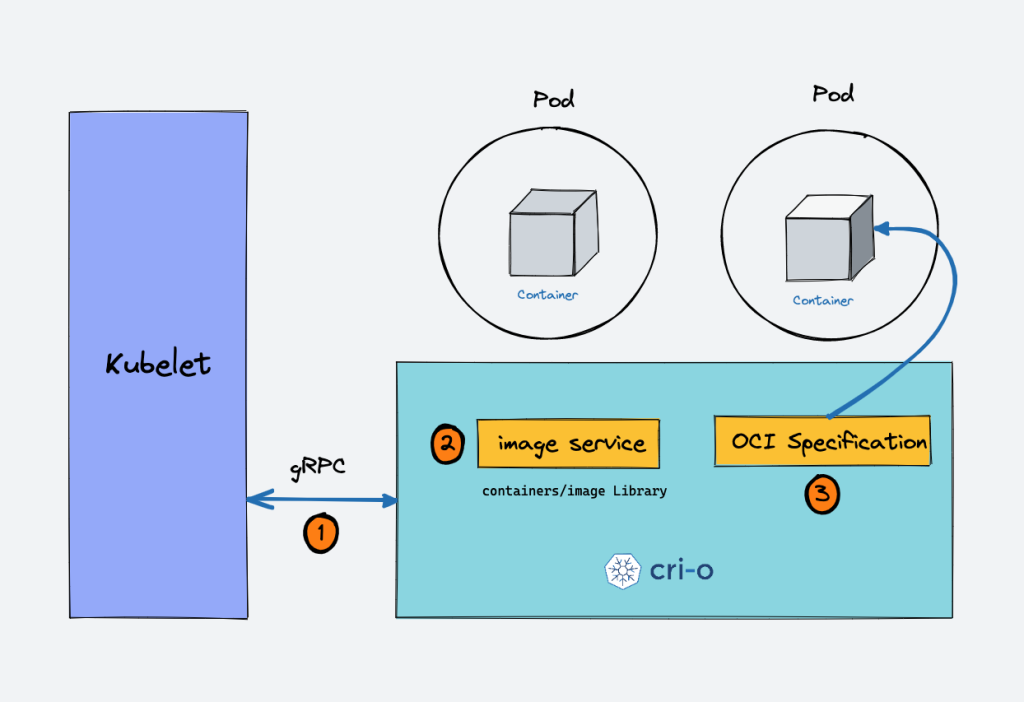
1. **Container Runtime Interface (CRI):** It is a set of APIs that allows Kubernetes to interact with different container runtimes. It allows different container runtimes to be used interchangeably with Kubernetes. The CRI defines the API for creating, starting, stopping, and deleting containers, as well as for managing images and container networks.
2. **Open Container Initiative (OCI):** It is a set of standards for container formats and runtimes

Kubernetes supports multiple [container runtimes](https://kubernetes.io/docs/setup/production-environment/container-runtimes/) (CRI-O, Docker Engine, containerd, etc) that are compliant with [**Container Runtime Interface (CRI)**](https://github.com/kubernetes/community/blob/master/contributors/devel/sig-node/container-runtime-interface.md). This means, all these container runtimes implement the CRI interface and expose gRPC [CRI APIs](https://kubernetes.io/docs/concepts/architecture/cri/) (runtime and image service endpoints).

So how does Kubernetes make use of the container runtime?

As we learned in the Kubelet section, the kubelet agent is responsible for interacting with the container runtime using CRI APIs to manage the lifecycle of a container. It also gets all the container information from the container runtime and provides it to the control plane.

Let’s take an example of [CRI-O](https://cri-o.io/) container runtime interface. Here is a high-level overview of how container runtime works with kubernetes.

[](https://devopscube.com/wp-content/uploads/2022/12/image-5.png)

1. When there is a new request for a pod from the API server, the kubelet talks to CRI-O daemon to launch the required containers via Kubernetes Container Runtime Interface.
2. CRI-O checks and pulls the required container image from the configured container registry using **containers/image** library.
3. CRI-O then generates OCI runtime specification (JSON) for a container.
4. CRI-O then launches an OCI-compatible runtime (runc) to start the container process as per the runtime specification.

**Kubernetes Cluster Addon Components**

Apart from the core components, the kubernetes cluster needs addon components to be fully operational. Choosing an addon depends on the project requirements and use cases.

Following are some of the popular addon components that you might need on a cluster.

1. **CNI Plugin (Container Network Interface)**
2. **CoreDNS (For DNS server):** CoreDNS acts as a DNS server within the Kubernetes cluster. By enabling this addon, you can enable DNS-based service discovery.
3. **Metrics Server (For Resource Metrics):** This addon helps you collect performance data and resource usage of Nodes and pods in the cluster.
4. **Web UI (Kubernetes Dashboard):** This addon enables the Kubernetes dashboard for managing the object via web UI.

**1. CNI Plugin**

First, you need to understand [**Container Networking Interface (CNI)**](https://www.cni.dev/)

It is a plugin-based architecture with vendor-neutral [specifications](https://github.com/containernetworking/cni/blob/spec-v1.0.0/SPEC.md) and libraries for creating network interfaces for Containers.

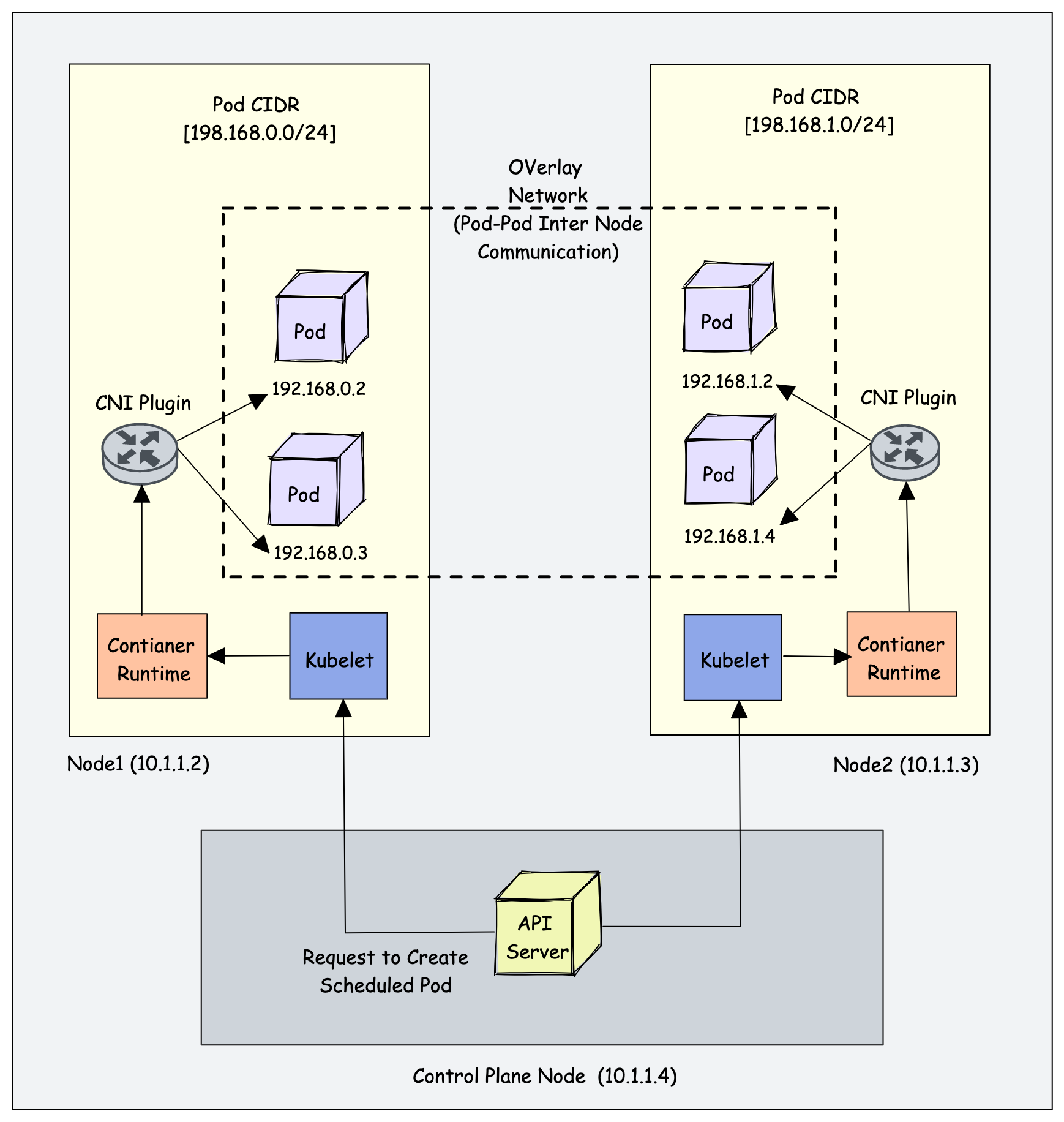
It is not specific to Kubernetes. With CNI container networking can be standardized across container orchestration tools like Kubernetes, Mesos, CloudFoundry, Podman, Docker, etc.

When it comes to container networking, companies might have different requirements such as network isolation, security, encryption, etc. As container technology advanced, many network providers created CNI-based solutions for containers with a wide range of networking capabilities. You can call it as CNI-Plugins

This allows users to choose a networking solution that best fits their needs from different providers.

How does CNI Plugin work with Kubernetes?

1. The Kube-controller-manager is responsible for assigning pod CIDR to each node. Each pod gets a unique IP address from the pod CIDR.
2. Kubelet interacts with container runtime to launch the scheduled pod. The CRI plugin which is part of the Container runtime interacts with the CNI plugin to configure the pod network.
3. CNI Plugin enables networking between pods spread across the same or different nodes using an overlay network.

[](https://devopscube.com/wp-content/uploads/2023/01/image-18.png)

Following are high-level functionalities provided by CNI plugins.

1. Pod Networking
2. Pod network security & isolation using Network Policies to control the traffic flow between pods and between namespaces.

Some popular CNI plugins include:

1. Calico
2. Flannel
3. Weave Net
4. Cilium (Uses eBPF)
5. Amazon VPC CNI (For AWS VPC)
6. Azure CNI (For Azure Virtual network)Kubernetes networking is a big topic and it differs based on the hosting platforms.

**Kubernetes Architecture FAQs**

**What is the main purpose of the Kubernetes control plane?**

The control plane is responsible for maintaining the desired state of the cluster and the applications running on it. It consists of components such as the API server, etcd, Scheduler, and controller manager.

**What is the purpose of the worker nodes in a Kubernetes cluster?**

Worker nodes are the servers (either bare-metal or virtual) that run the container in the cluster. They are managed by the control plane and receive instructions from it on how to run the containers that are part of pods.

**How is communication between the control plane and worker nodes secured in Kubernetes?**

Communication between the control plane and worker nodes is secured using PKI certificates and communication between different components happens over TLS. This way, only trusted components can communicate with each other.

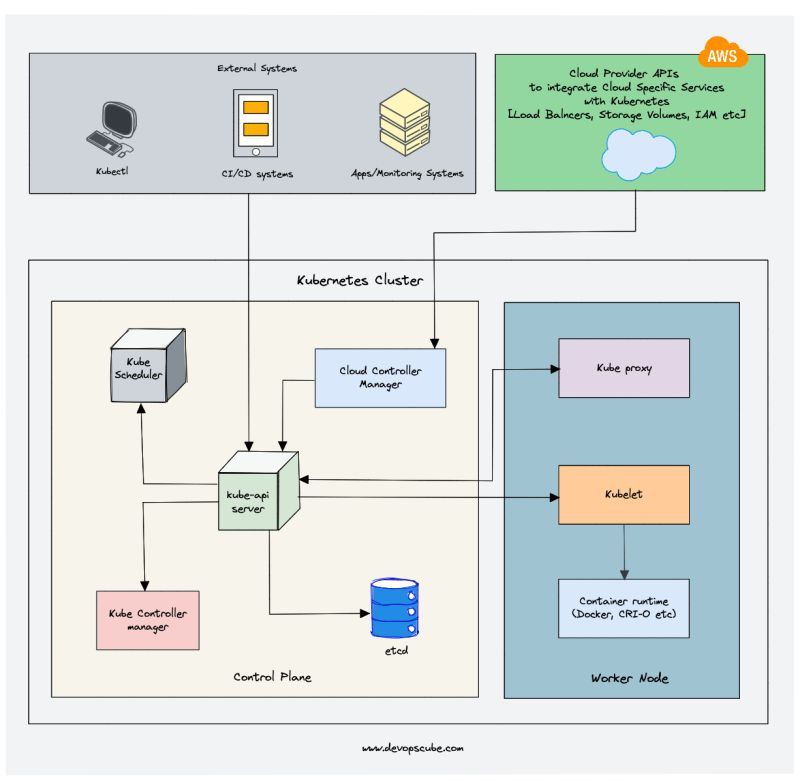
**What is the purpose of the etcd key-value store in Kubernetes?**

Etcd primarily stores the kubernetes objects, cluster information, node information, and configuration data of the cluster, such as the desired state of the applications running on the cluster.

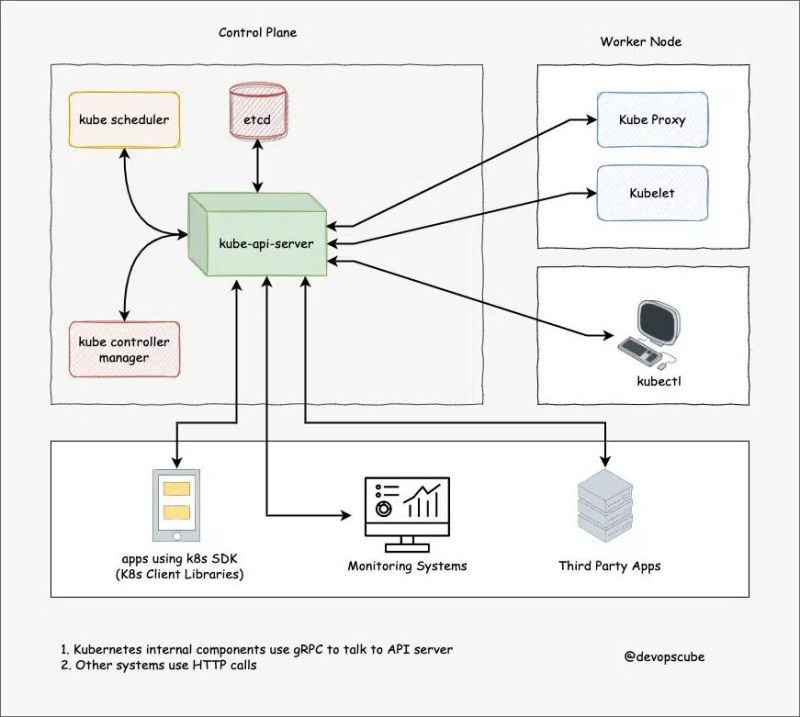
**What happens to Kubernetes applications if etcd goes down?**

While the running applications will not be affected if etcd experiences an outage, it will not be possible to create or update any objects without a functioning etcd

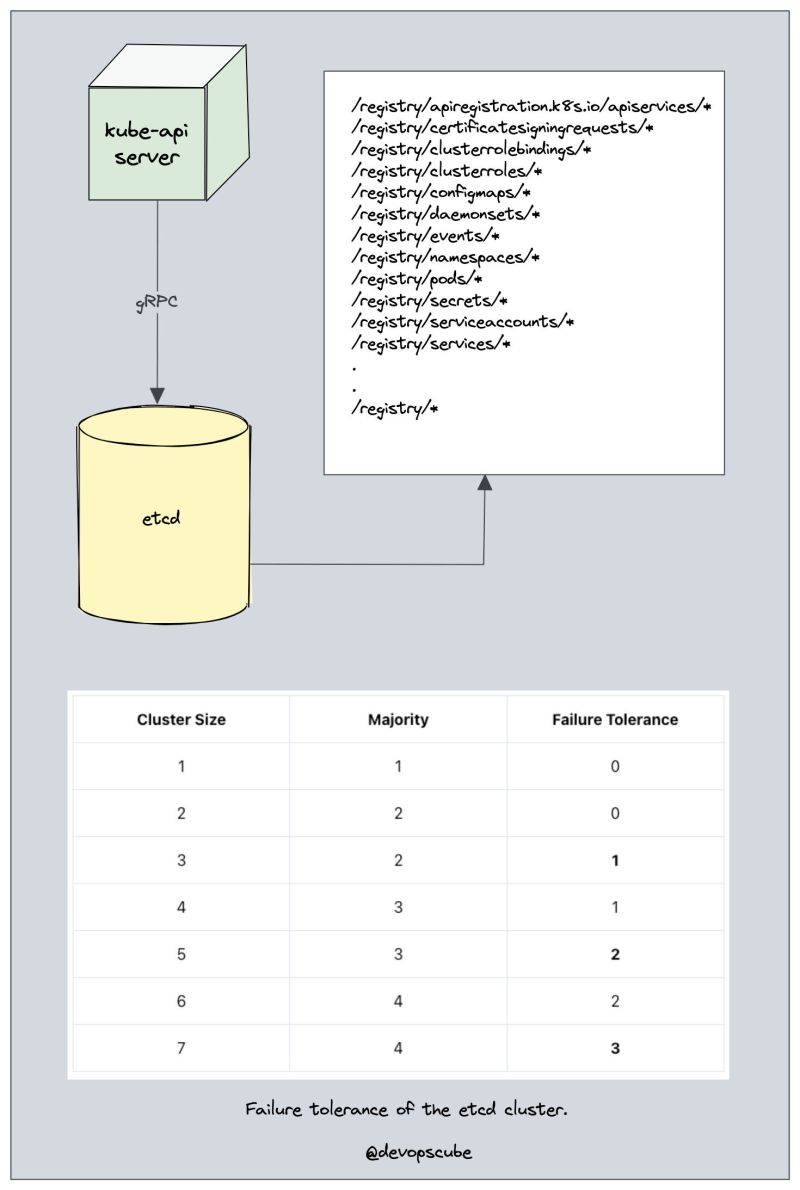
Kubernetes Learning [Day 02]  
  
Let's learn about Kubernetes Cluster Components 🚀  
  
It is essential to understand that Kubernetes is a distributed system.  
  
Meaning, it has multiple components spread across different servers over a network. These servers could be Virtual machines or bare metal servers.  
  
As a whole, we call it a Kubernetes cluster.  
  
A Kubernetes cluster consists of a control plane and worker nodes.  
  
✅ The control plane is responsible for maintaining the desired state of the cluster. It is also responsible for node/pod lifecycle management and exposing cluster API. It has the following components.  
  
1. kube-apiserver  
2. etcd  
3. kube-scheduler  
4. kube-controller-manager  
5. cloud-controller-manager  
  
✅ The Worker nodes are responsible for running containerized applications. The worker Node has the following components.  
  
1. kubelet  
2. kube-proxy  
3. Container runtime (CRI-O, Docker Engine, Containerd, etc)  
  
✅ Additionally there are add-on components that you can add to the cluster to extend its functionality and make the cluster fully functional for application deployments.  
  
Following are some of the common Kubernetes add-ons  
  
1. Web UI  
2. Core DNS  
3. Metrics Server  
4. CNI Plugins (Container Network Interface)  
  
In this Kubernetes distributed scenario, the connection happening through the network to connect with different Kubernetes components should be secure and all the components should be able to authenticate each other.  
  
Kubernetes uses PKI certificates for authentication over TLS.  
  
In the upcoming posts, I will cover each component and its significance in detail.  
  
Tomorrow we will learn about kube-apiserver in detail.  
  
Note: Open the image in a new tab for good resolution.  
  
Check out the comments for links to my Kubernetes learning resources.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018086516690210817) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018086516690210817) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018086516690210817) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018086516690210817)



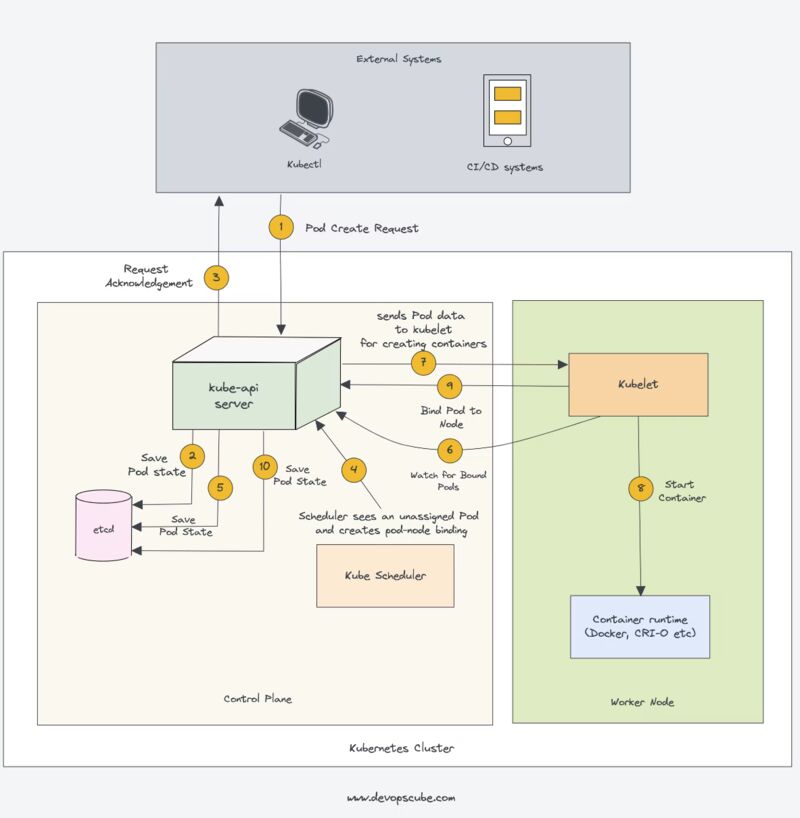
Learning Kubernetes [Day 03]  
  
Let's learn about 𝗸𝘂𝗯𝗲-𝗮𝗽𝗶𝘀𝗲𝗿𝘃𝗲𝗿 🚀  
  
The kube-api server is the central hub of the Kubernetes cluster that exposes the Kubernetes API.  
  
It is the main entry point to the cluster.  
  
End users, and other cluster components, talk to the cluster via the API server.  
  
Also, Monitoring systems and third-party services may talk to API servers to interact with the cluster.  
  
All these systems communicate with the API server through HTTP REST APIs.  
  
However, the internal cluster components like the scheduler, controller, etc talk to the API server using gRPC.  
  
Kubernetes api-server is responsible for the following  
  
✅ 𝗔𝗣𝗜 𝗺𝗮𝗻𝗮𝗴𝗲𝗺𝗲𝗻𝘁: Exposes the cluster API endpoint and handles all API requests.  
  
✅ 𝗔𝘂𝘁𝗵𝗲𝗻𝘁𝗶𝗰𝗮𝘁𝗶𝗼𝗻: Using client certificates, bearer tokens, and HTTP Basic Authentication  
  
✅ 𝗔𝘂𝘁𝗵𝗼𝗿𝗶𝘇𝗮𝘁𝗶𝗼𝗻: ABAC and RBAC evaluation  
  
✅ 𝗩𝗮𝗹𝗶𝗱𝗮𝘁𝗶𝗼𝗻: Processing API requests and validating data for the API objects like pods, services, etc. (Validation and Mutation Admission controllers)  
  
✅ It is the only component that communicates with etcd.  
  
✅ API-server coordinates all the container orchestration processes between the control plane and worker node components.  
  
The communication between the API server and other components in the cluster happens over TLS to prevent unauthorized access to the cluster.  
  
From a security standpoint, it is important to reduce the cluster attack surface by securing the API server.  
  
⚠️ The Shadowserver Foundation has conducted an experiment that discovered 𝟯𝟴𝟬𝟬𝟬𝟬 𝗽𝘂𝗯𝗹𝗶𝗰𝗹𝘆 𝗮𝗰𝗰𝗲𝘀𝘀𝗶𝗯𝗹𝗲 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗔𝗣𝗜 𝘀𝗲𝗿𝘃𝗲𝗿𝘀.  
  
𝗡𝗼𝘁𝗲: Admission controller is an important concept you should know. I will cover it in a future post with examples.  
  
Tomorrow we will learn about the etcd component in detail.  
  
To access my previous Kubernetes posts & resources check out the link in the comment.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018441551676665856) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018441551676665856) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018441551676665856) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018441551676665856)



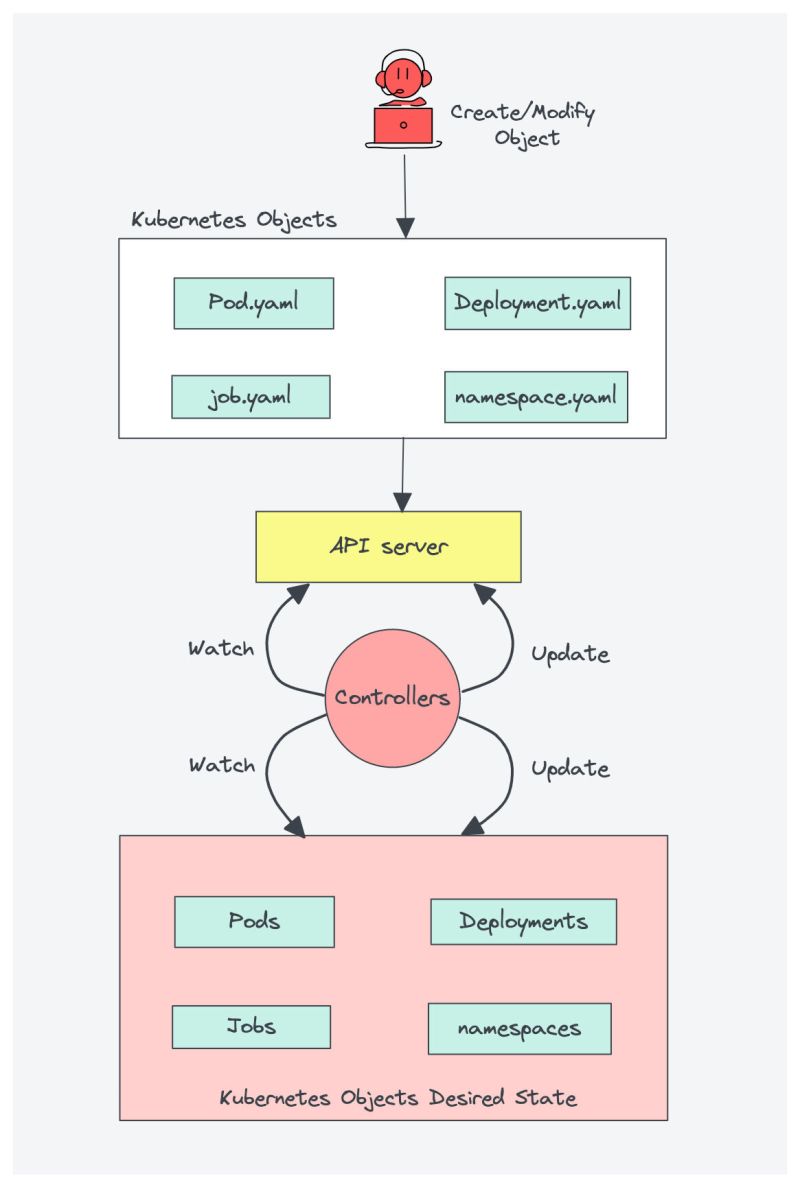
𝗟𝗲𝗮𝗿𝗻𝗶𝗻𝗴 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 [𝗗𝗮𝘆 𝟬𝟰]  
  
Let's learn about 𝗲𝘁𝗰𝗱 and how Kubernetes uses it. 🚀  
  
Kubernetes is a distributed system and it needs an efficient distributed database like etcd.  
  
etcd acts as both a backend service discovery and a database.  
  
You can call it the brain of the Kubernetes cluster.  
  
etcd is an open-source strongly consistent, distributed key-value store. So what does it mean?  
  
✅ 𝗦𝘁𝗿𝗼𝗻𝗴𝗹𝘆 𝗰𝗼𝗻𝘀𝗶𝘀𝘁𝗲𝗻𝘁: If an update is made to a node, strong consistency will ensure it gets updated to all the other nodes in the cluster immediately. Also if you look at the CAP theorem, achieving 100% availability with strong consistency & Partition Tolerance is impossible.  
  
✅ 𝗗𝗶𝘀𝘁𝗿𝗶𝗯𝘂𝘁𝗲𝗱: etcd is designed to run on multiple nodes as a cluster without sacrificing consistency.  
  
✅ 𝗞𝗲𝘆 𝗩𝗮𝗹𝘂𝗲 𝗦𝘁𝗼𝗿𝗲: A nonrelational database that stores data as keys and values. It also exposes a key-value API. The datastore is built on top of BboltDB which is a fork of BoltDB.  
  
etcd uses 𝗿𝗮𝗳𝘁 𝗰𝗼𝗻𝘀𝗲𝗻𝘀𝘂𝘀 𝗮𝗹𝗴𝗼𝗿𝗶𝘁𝗵𝗺 for strong consistency and availability. It works in a leader-member fashion for high availability and to withstand node failures.  
  
You should have a minimum of 3 etcd nodes to withstand 1 failure. To withstand 2 node failures, you should have 5 nodes, and so on.  
  
𝗦𝗼 𝗵𝗼𝘄 𝗲𝘁𝗰𝗱 𝘄𝗼𝗿𝗸𝘀 𝘄𝗶𝘁𝗵 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀?  
  
To put it simply, when you use kubectl to get Kubernetes object details, you are getting it from etcd.  
  
Also, when you deploy an object like a pod, an entry gets created in etcd.  
  
In a nutshell, here is what you need to know about etcd.  
  
✅ etcd stores all configurations, states, and metadata of Kubernetes objects (pods, secrets, daemonsets, deployments, configmaps, statefulsets, etc).  
  
✅ Kubernetes uses the etcd’s watch functionality to track the change in the state of an object (Watch() API)  
  
✅ etcd exposes key-value API using gRPC. Also, the gRPC gateway which is a RESTful proxy that translates all the HTTP API calls into gRPC messages. It makes it an ideal database for Kubernetes.  
  
✅ etcd stores all objects under the /registry directory key in key-value format. For example, information on a pod named Nginx in the default namespace can be found under /registry/pods/default/nginx  
  
Also, etcd it is the only 𝗦𝘁𝗮𝘁𝗲𝗳𝘂𝗹𝘀𝗲𝘁 component in the control plane.  
  
When it comes to etcd HA architecture, there are two modes.  
  
➡️ 𝗦𝘁𝗮𝗰𝗸𝗲𝗱 𝗲𝘁𝗰𝗱: etcd deployed along with control plane nodes  
  
➡️ 𝗘𝘅𝘁𝗲𝗿𝗻𝗮𝗹 𝗲𝘁𝗰𝗱 𝗰𝗹𝘂𝘀𝘁𝗲𝗿: Dedicated etcd cluster.  
  
Tomorrow we will learn about Kube scheduler component in detail.  
  
My previous Kubernetes posts <https://lnkd.in/gyvQrrAs>  
  
𝗡𝗼𝘁𝗲: If you have trouble understanding the terms used, please learn the concepts I mentioned in the [𝗗𝗮𝘆 𝟬𝟭] 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗽𝗿𝗲𝗿𝗲𝗾𝘂𝗶𝘀𝗶𝘁𝗲𝘀 𝗽𝗼𝘀𝘁.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018803939282206721) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018803939282206721) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018803939282206721) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7018803939282206721)



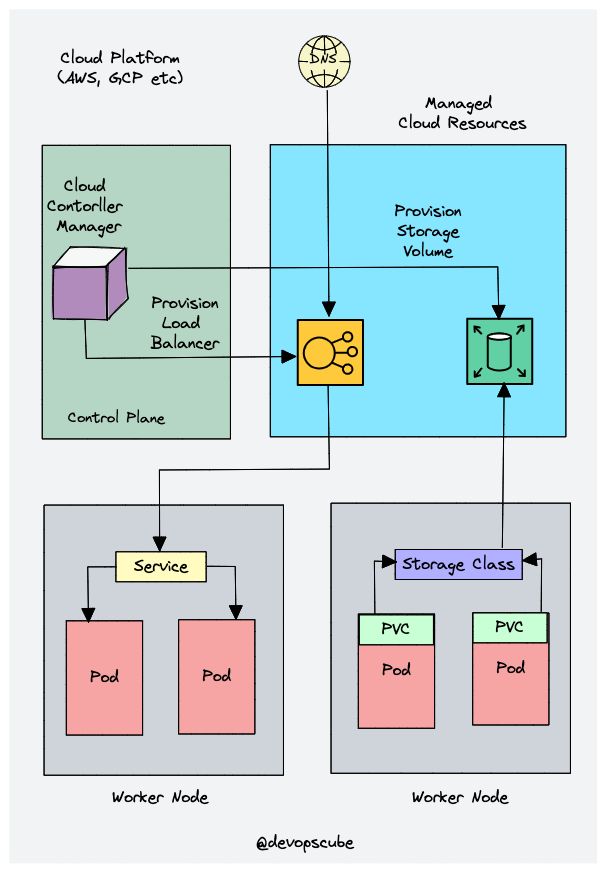
Learning Kubernetes [Day 05]  
  
Let's learn about the 𝗞𝘂𝗯𝗲 𝗦𝗰𝗵𝗲𝗱𝘂𝗹𝗲𝗿 component 🚀  
  
The kube-scheduler is responsible for scheduling pods on worker nodes.  
  
When you deploy a pod, you specify the pod requirements such as CPU, memory, Volume mounts, etc.  
  
The scheduler’s primary task is to identify the create request and choose the best node for a pod that satisfies the requirements.  
  
So how does the scheduler select the node out of all worker nodes?  
  
Here is how 👇  
  
➡️ To choose the best node, the Kube-scheduler uses 𝗳𝗶𝗹𝘁𝗲𝗿𝗶𝗻𝗴 𝗮𝗻𝗱 𝘀𝗰𝗼𝗿𝗶𝗻𝗴 operations.  
  
➡️ In 𝗳𝗶𝗹𝘁𝗲𝗿𝗶𝗻𝗴, the scheduler finds the best-suited nodes where the pod can be scheduled. For example, if there are five worker nodes with resource availability to run the pod, it selects all five nodes.  
  
➡️ If there are no nodes, then the pod is unschedulable and moved to the scheduling queue. If It is a large cluster, let’s say 100 worker nodes, and the scheduler doesn’t iterate over all the nodes.  
  
➡️ There is a scheduler configuration parameter called 𝗽𝗲𝗿𝗰𝗲𝗻𝘁𝗮𝗴𝗲𝗢𝗳𝗡𝗼𝗱𝗲𝘀𝗧𝗼𝗦𝗰𝗼𝗿𝗲. The default value is typically 𝟱𝟬%. So it tries to iterate over 50% of nodes in a round-robin fashion. If the worker nodes are spread across multiple zones, then the scheduler iterates over nodes in different zones. For very large clusters the default 𝗽𝗲𝗿𝗰𝗲𝗻𝘁𝗮𝗴𝗲𝗢𝗳𝗡𝗼𝗱𝗲𝘀𝗧𝗼𝗦𝗰𝗼𝗿𝗲 is 5%.  
  
➡️ In the 𝘀𝗰𝗼𝗿𝗶𝗻𝗴 𝗽𝗵𝗮𝘀𝗲, the scheduler ranks the nodes by assigning a score to the filtered worker nodes. The scheduler makes the scoring by calling multiple scheduling plugins.  
  
➡️ Finally, the worker node with the highest rank will be selected for scheduling the pod. If all the nodes have the same rank, a node will be selected at random.  
  
➡️ Once the node is selected, the scheduler creates a binding event in the API server. Meaning an event to bind a pod and node.  
  
Here is what you should know about a scheduler.  
  
✅ It is a controller that listens to pod creation events in the API server.  
  
✅ The scheduler has two phases. 𝗦𝗰𝗵𝗲𝗱𝘂𝗹𝗶𝗻𝗴 𝗰𝘆𝗰𝗹𝗲 and the 𝗕𝗶𝗻𝗱𝗶𝗻𝗴 𝗰𝘆𝗰𝗹𝗲. Together it is called the scheduling context. The scheduling cycle selects a worker node and the binding cycle applies that change to the cluster.  
  
✅ The scheduler always places the high-priority pods ahead of the low-priority pods for scheduling. Also, in some cases, after the pod started running in the selected node, the pod might get evicted or moved to other nodes.  
  
✅ You can create custom schedulers and run multiple schedulers in a cluster along with the native scheduler. When you deploy a pod you can specify the custom scheduler in the pod manifest. So the scheduling decisions will be taken based on the custom scheduler logic.  
  
Tomorrow we will learn about 𝗞𝘂𝗯𝗲 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿 component in detail.  
  
Check the comment for links to my previous posts and resources.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019173792644030465) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019173792644030465) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019173792644030465) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019173792644030465)



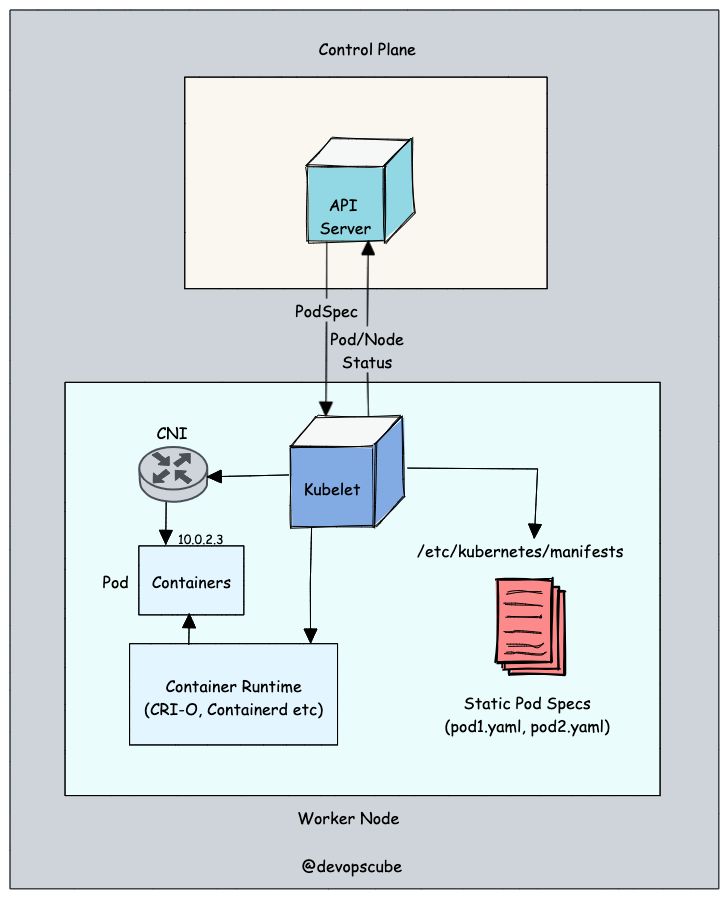
Learning Kubernetes [Day 06]  
  
Let's learn about 𝗞𝘂𝗯𝗲 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿 🚀  
  
So what is a controller?  
  
𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿𝘀 are programs that run infinite control loops.  
  
Meaning it runs continuously and watches the actual and desired state of objects via the API server.  
  
Controllers use the Kubernetes API to watch for changes in the state of the cluster and take appropriate action when a change is detected.  
  
If there is a difference in the actual and desired state, it ensures that the kubernetes resource/object is in the desired state.  
  
📌 Let's look at an example.  
  
If you want to create a deployment, you specify the desired state in the manifest YAML file (declarative approach).  
  
For example, 2 replicas, one volume mount, configmap, etc. The in-built deployment controller ensures that the deployment is in the desired state all the time.  
  
If a user updates the deployment with 5 replicas, the deployment controller recognizes it and ensures the desired state is 5 replicas.  
  
𝗞𝘂𝗯𝗲 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿 is a component that manages all the Kubernetes controllers. Kubernetes resources/objects like pods, namespaces, jobs, replicaset are managed by respective controllers.  
  
Following are some examples of core built-in Kubernetes controllers.  
  
➡️ Deployment controller  
  
➡️ Replicaset controller  
  
➡️ DaemonSet controller  
  
➡️ Job Controller  
  
➡️ Node controller  
  
Here is what you should know about the Kube controller manager.  
  
✅ It manages all the controllers and the controllers try to keep the cluster in the desired state.  
  
✅ You can extend kubernetes with custom controllers associated with a custom resource definition.  
  
✅ Controller uses the Kubernetes API to get the current state of the cluster and to update the state of the cluster by creating, updating, or deleting resources.  
  
✅ If there are multiple instances of Kube controller manager (Control Plane HA), it runs in a leader election mode. One instance is elected as a leader for actively making changes to the cluster at a time. This prevents conflicts and ensures that the cluster remains in a consistent state.  
  
In the next post, we will learn about the 𝗖𝗹𝗼𝘂𝗱 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿 component in detail.  
  
👉 My previous Kubernetes posts are listed here: <https://lnkd.in/gyvQrrAs>  
  
To get a notification on my next k8s post hit the 🔔 on my profile.  
  
𝗡𝗼𝘁𝗲: If you have trouble understanding the terms used, please learn the concepts I mentioned in the [𝗗𝗮𝘆 𝟬𝟭] 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗽𝗿𝗲𝗿𝗲𝗾𝘂𝗶𝘀𝗶𝘁𝗲𝘀 𝗽𝗼𝘀𝘁.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019537485491830784) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019537485491830784) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019537485491830784) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7019537485491830784)



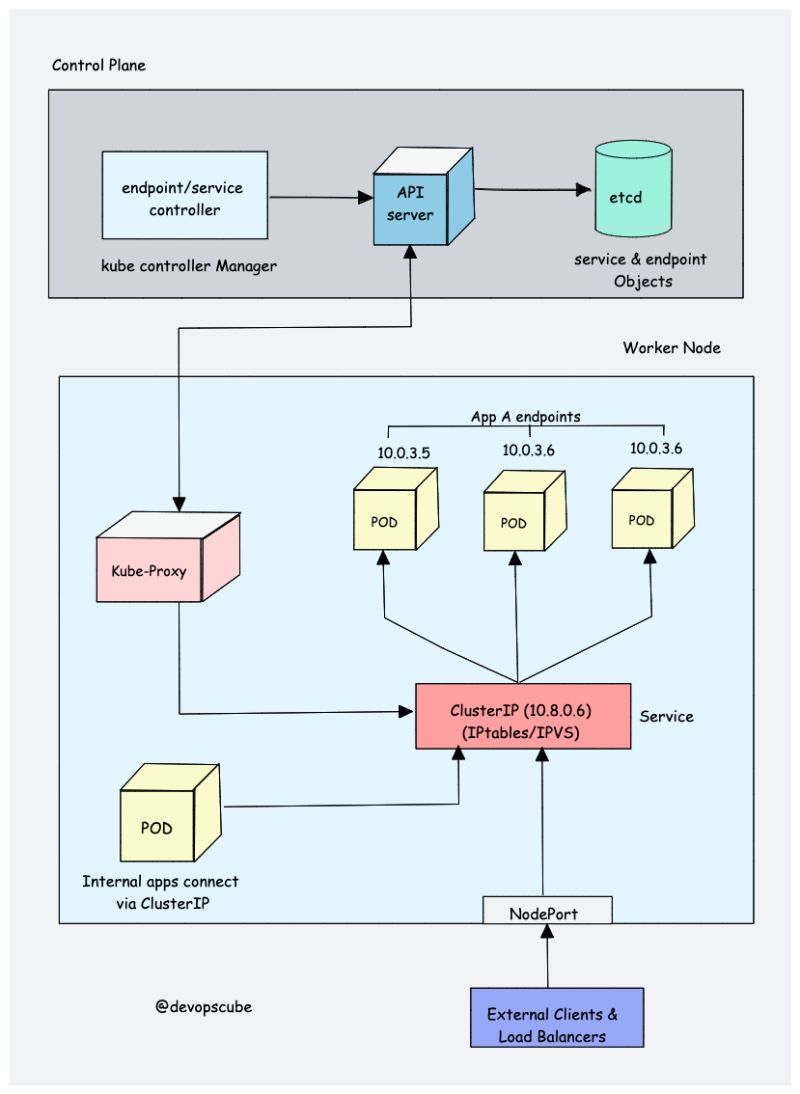
Learning Kubernetes [Day 07]  
  
Let's learn about 𝗖𝗹𝗼𝘂𝗱 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿 🚀  
  
When Kubernetes is deployed in cloud environments (AWS, GCP, Azure, etc),  
  
The cloud controller manager acts as a bridge between Cloud Platform APIs and the Kubernetes cluster.  
  
This way the core Kubernetes core components can work independently and allows the cloud providers to integrate with Kubernetes using plugins. (For example, an interface between Kubernetes cluster and AWS cloud API)  
  
Cloud controller integration allows Kubernetes cluster to manage cloud resources like servers, Load Balancers (for services), and Storage Volumes (for persistent volumes).  
  
Cloud Controller Manager contains a set of 𝗰𝗹𝗼𝘂𝗱 𝗽𝗹𝗮𝘁𝗳𝗼𝗿𝗺-𝘀𝗽𝗲𝗰𝗶𝗳𝗶𝗰 𝗰𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿𝘀 that ensure the desired state of cloud-specific components (nodes, Loadbalancers, storage, etc).  
  
Controllers are programs that run infinite control loops. Meaning it runs continuously and watches the actual and desired state of objects.  
  
There are three main controllers that are part of the cloud controller manager.  
  
➡️ 𝗡𝗼𝗱𝗲 𝗰𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿: This controller updates node-related information by talking to the cloud provider API. For example, node labeling & annotation, getting hostname, CPU & memory availability, nodes health, etc.  
  
➡️ 𝗥𝗼𝘂𝘁𝗲 𝗰𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿: It is responsible for configuring networking routes on a cloud platform. So that pods in different nodes can talk to each other.  
  
➡️ 𝗦𝗲𝗿𝘃𝗶𝗰𝗲 𝗰𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿: It takes care of deploying Load balancers for Kubernetes services, assigning IP addresses, etc.  
  
Following are some of the classic examples of cloud controller manager integrations.  
  
✅ Deploying Kubernetes Service of type Load balancer. Here Kubernetes provisions a cloud-specific Loadbalancer and integrates with Kubernetes Service.  
  
✅ Provisioning storage volumes (PV) for pods backed by cloud storage solutions.  
  
Overall Cloud Controller Manager manages the lifecycle of cloud-specific resources used by Kubernetes.  
  
Tomorrow we will learn about Kubelet component in detail.  
  
My previous Kubernetes posts & resources: [bit.ly/m/bibinwilson](http://bit.ly/m/bibinwilson)  
  
To get a notification on my next k8s post hit the 🔔 on my profile.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020615744346365952) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020615744346365952) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020615744346365952) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020615744346365952)



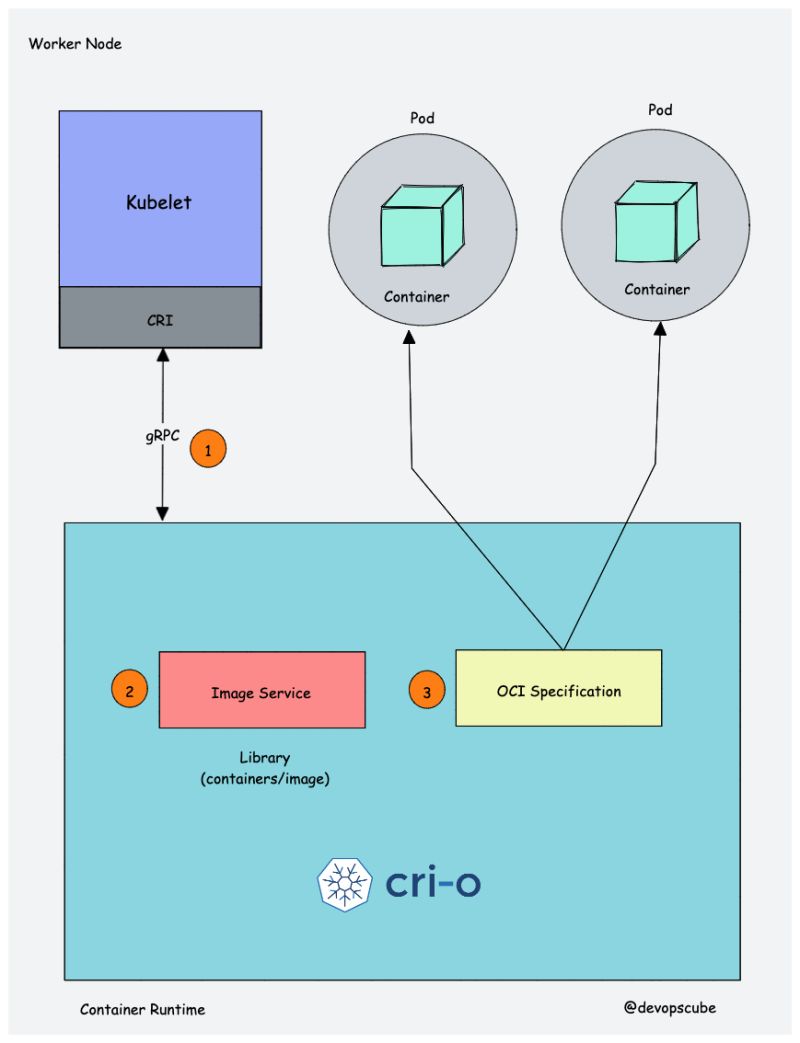
Learning Kubernetes [Day 08]  
  
Let's learn the important concepts of 𝗞𝘂𝗯𝗲𝗹𝗲𝘁 🚀  
  
Kubelet is an agent component that runs on every node in the cluster.  
  
It does not run as a container instead runs as a daemon, managed by systemd.  
  
It is responsible for registering worker nodes with the API server and working with the podSpec (Pod specification – YAML or JSON) it receives from the API server.  
  
podSpec defines the containers that should run inside the pod, their resources (e.g. CPU and memory limits), and other settings such as environment variables, volumes, and labels.  
  
It then brings the podSpec to the desired state by creating containers interacting with container runtime.  
  
To put it simply, kubelet is responsible for the following.  
  
✅ Creating, modifying, and deleting containers for the pod.  
  
✅ Runs the startup, liveness, and readiness probes  
  
✅ Mounting volumes  
  
✅ Collecting and reporting Node and pod status via calls to the API server.  
  
Kubelet is also a controller where it watches for pod changes and utilizes the node’s container runtime to pull images, run containers, etc.  
  
Other than PodSepcs from the API server, Kubelet can accept podSpec from a file, HTTP endpoint, and HTTP server.  
  
A good example of “podSpec from a file” is 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝘀𝘁𝗮𝘁𝗶𝗰 𝗽𝗼𝗱𝘀.  
  
Static pods are pods controlled by Kubelet on its nodes, not the API servers.  
  
This means you can create pods by providing a pod YAML location to the Kubelet component. However, static pods created by Kubelet are not managed by the API server.  
  
Here is a real-world example use case of the static pod.  
  
While bootstrapping the control plane, kubelet starts the api-server, scheduler, and controller manager as static pods from podSpecs located at /𝗲𝘁𝗰/𝗸𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀/𝗺𝗮𝗻𝗶𝗳𝗲𝘀𝘁𝘀.  
  
Following are some of the key things you should know about Kubelet.  
  
➡️ Kubelet uses the CRI (container runtime interface) gRPC interface to talk to the container runtime for managing the lifecycle of the workloads.  
  
➡️ It also exposes an HTTP endpoint to stream logs and provides exec sessions for clients.  
  
➡️ Uses the CSI (container storage interface) gRPC to configure block volumes.  
  
➡️ It uses the CNI plugin configured in the cluster to allocate the pod IP address and set up any necessary network routes and firewall rules for the pod.  
  
Tomorrow we will learn about the Kube Proxy component in detail.  
  
My previous Kubernetes posts & resources: [bit.ly/m/bibinwilson](http://bit.ly/m/bibinwilson)  
  
To get a notification on my next k8s post hit the 🔔 on my profile.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020999629072723968) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020999629072723968) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020999629072723968) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7020999629072723968)



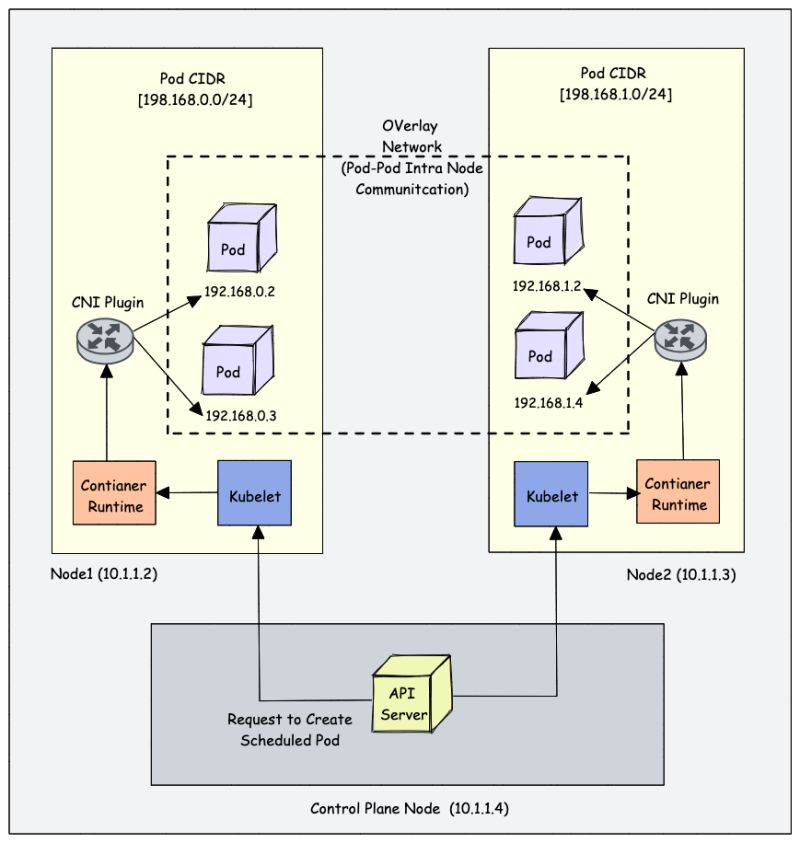
Learning Kubernetes [Day 09]  
  
Let's learn about 𝗞𝘂𝗯𝗲 𝗣𝗿𝗼𝘅𝘆 Component 🚀  
  
To understand kube proxy, you need to have a basic knowledge of Kubernetes Service & endpoint objects.  
  
✅ Service in Kubernetes is a way to expose a set of pods internally or to external traffic. When you create the service object, it gets a 𝘃𝗶𝗿𝘁𝘂𝗮𝗹 𝗜𝗣 assigned to it, called a 𝗖𝗹𝘂𝘀𝘁𝗲𝗿𝗜𝗣. It is only accessible within the Kubernetes cluster.  
  
✅ The Endpoint object contains all the IP addresses and ports of pods grouped under a Service object.  
  
✅ The endpoints controller is responsible for maintaining the list of pod IP addresses (endpoints). The service controller is responsible for configuring endpoints to a service.  
  
✅ You cannot ping the clusterIP because it is only used for service discovery, unlike pod IPs which are pingable.  
  
Now, let's look into Kube Proxy.  
  
Kube-proxy is a daemon that runs on every node as a daemonset.  
  
It is a proxy component that implements the Kubernetes Services concept for pods (single DNS for a set of pods with load balancing).  
  
When you expose pods using a Service (ClusterIP), Kube-proxy creates network rules to send traffic to the backend pods (endpoints) grouped under the Service object.  
  
Meaning, all the load balancing, and service discovery are handled by the Kube proxy.  
  
𝗦𝗼, 𝗵𝗼𝘄 𝗱𝗼𝗲𝘀 𝗞𝘂𝗯𝗲-𝗽𝗿𝗼𝘅𝘆 𝘄𝗼𝗿𝗸?  
  
Kube proxy talks to the API server to get the details about the Service (ClusterIP) and respective pod IPs & ports (endpoints). It also monitors for changes in service and endpoints.  
  
Kube-proxy then uses any one of the following modes to create/update rules for routing traffic to pods behind a Service:  
  
➡️ 𝗜𝗣𝗧𝗮𝗯𝗹𝗲𝘀: It is the default mode. In IPTables mode, the traffic is handled by IPtable rules. In this mode, kube-proxy chooses the backend pod randomly for load balancing. Once the connection is established, the requests go to the same pod until the connection is terminated.  
  
➡️ 𝗜𝗣𝗩𝗦: For clusters with services exceeding 1000, IPVS offers performance improvement. It supports the following load-balancing algorithms for the backend:  
  
- rr: round-robin : It is the default mode.  
- lc: least connection (smallest number of open connections)  
- dh: destination hashing  
- sh: source hashing  
- sed: shortest expected delay  
- nq: never queue  
  
➡️ 𝗨𝘀𝗲𝗿𝘀𝗽𝗮𝗰𝗲: (legacy & not recommended)  
  
➡️ 𝗞𝗲𝗿𝗻𝗲𝗹𝘀𝗽𝗮𝗰𝗲: This mode is only for Windows systems.  
  
If you would like to understand the performance difference between kube-proxy IPtables and IPVS mode, check out the link in the comment.  
  
Also, you can run a Kubernetes cluster without kube-proxy by replacing it with Cilium.  
  
𝗡𝗼𝘁𝗲: There are service types that do not have a clusterIP. We will look at service objects and networking in detail in future posts.  
  
Tomorrow we will learn about 𝗖𝗼𝗻𝘁𝗮𝗶𝗻𝗲𝗿 𝗥𝘂𝗻𝘁𝗶𝗺𝗲 in detail.  
  
My previous Kubernetes posts & resources: [bit.ly/m/bibinwilson](http://bit.ly/m/bibinwilson)  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7021355680745869313) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7021355680745869313)



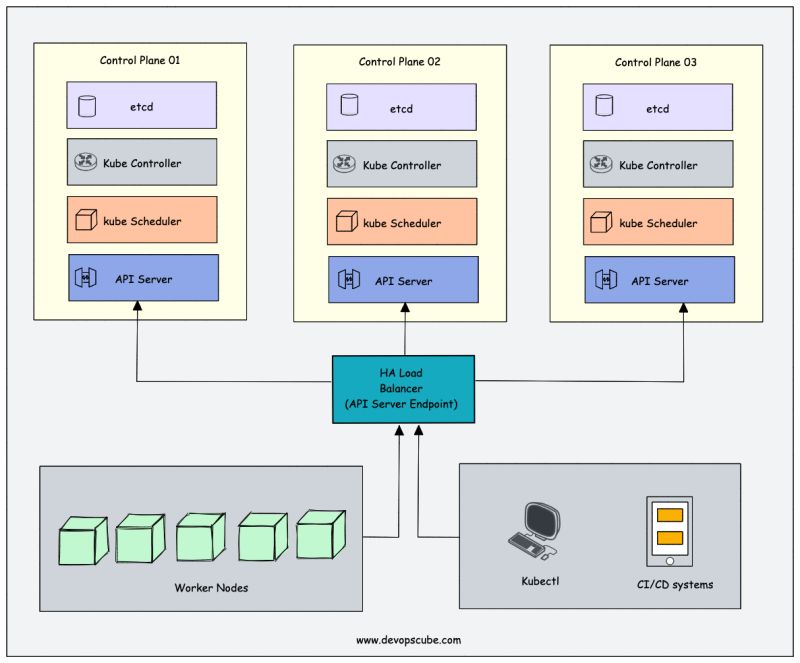
Learning Kubernetes [Day 10]  
  
Let's learn about 𝗖𝗼𝗻𝘁𝗮𝗶𝗻𝗲𝗿 𝗥𝘂𝗻𝘁𝗶𝗺𝗲 🚀  
  
A container runtime is a software component that runs on all the nodes in a Kubernetes cluster.  
  
It is responsible for pulling images from container registries, running containers, allocating and isolating resources for containers, and managing the entire lifecycle of a container on a host.  
  
To understand this better, let's take a look at two key concepts:  
  
✅ 𝗖𝗼𝗻𝘁𝗮𝗶𝗻𝗲𝗿 𝗥𝘂𝗻𝘁𝗶𝗺𝗲 𝗜𝗻𝘁𝗲𝗿𝗳𝗮𝗰𝗲 (𝗖𝗥𝗜): It is a set of APIs that allows Kubernetes to interact with different container runtimes. It allows different container runtimes to be used interchangeably with Kubernetes. The CRI defines the API for creating, starting, stopping, and deleting containers, as well as for managing images and container networks.  
  
✅ 𝗢𝗽𝗲𝗻 𝗖𝗼𝗻𝘁𝗮𝗶𝗻𝗲𝗿 𝗜𝗻𝗶𝘁𝗶𝗮𝘁𝗶𝘃𝗲 (𝗢𝗖𝗜): It is a set of standards for container formats and runtimes  
  
Kubernetes supports multiple container runtimes (such as CRI-O, Docker Engine, and containerd) that are compliant with the CRI.  
  
This means that all these container runtimes implement the CRI interface and expose gRPC CRI APIs (runtime and image service endpoints).  
  
So how does Kubernetes make use of the container runtime?  
  
As we learned in the Kubelet section, the kubelet agent is responsible for interacting with the container runtime using CRI APIs to manage the lifecycle of a container. It also gets all the container information from the container runtime and provides it to the control plane.  
  
Let’s take an example of CRI-O container runtime interface.  
  
Here is a high-level overview of CRI-O it works with Kubernetes.  
  
➡️ When there is a new request for a pod from the API server, the kubelet talks to CRI-O daemon to launch the required containers via Kubernetes Container Runtime Interface.  
  
➡️ CRI-O then checks and pulls the required container image from the configured container registry using containers/image library.  
  
➡️ It then generates OCI runtime specification (JSON) for a container.  
  
➡️ CRI-O then launches an OCI-compatible runtime (runc) to start the container process as per the runtime specification.  
  
Just a quick note on some popular container runtimes:  
  
👉 𝗰𝗼𝗻𝘁𝗮𝗶𝗻𝗲𝗿𝗱: - It is a CNCF project, originally created by Docker.  
  
👉 𝗖𝗥𝗜-𝗢: It is a CNCF Incubating project created by Redhat  
  
👉 𝗿𝘂𝗻𝗰: It is a low-level container runtime that was originally developed by Docker.  
  
Tomorrow we will learn about the Addon component 𝗖𝗡𝗜 𝗽𝗹𝘂𝗴𝗶𝗻𝘀 in detail.  
  
My previous Kubernetes posts <https://lnkd.in/gyvQrrAs>  
  
To get a notification on my next k8s post hit the 🔔 on my profile.  
  
𝗡𝗼𝘁𝗲: If you have trouble understanding the terms used, please learn the concepts I mentioned in the [𝗗𝗮𝘆 𝟬𝟭] 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗽𝗿𝗲𝗿𝗲𝗾𝘂𝗶𝘀𝗶𝘁𝗲𝘀 𝗽𝗼𝘀𝘁.



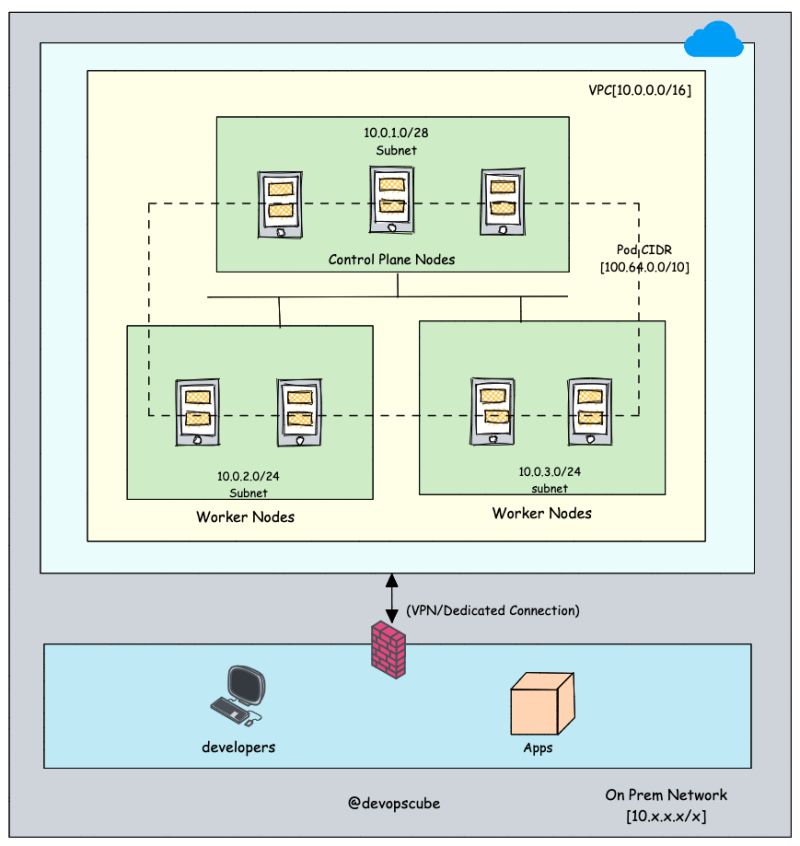
Learning Kubernetes [Day 11]  
  
Let's learn about 𝗖𝗡𝗜 𝗽𝗹𝘂𝗴𝗶𝗻𝘀 and why we need them.  
  
First, you need to understand the 𝗖𝗼𝗻𝘁𝗮𝗶𝗻𝗲𝗿 𝗡𝗲𝘁𝘄𝗼𝗿𝗸𝗶𝗻𝗴 𝗜𝗻𝘁𝗲𝗿𝗳𝗮𝗰𝗲 (𝗖𝗡𝗜).  
  
It is a plugin-based architecture with vendor-neutral specifications and libraries for creating network interfaces for containers.  
  
It is not specific to Kubernetes. With CNI, container networking is standardized across container orchestration tools such as Kubernetes, Mesos, CloudFoundry, Podman, and Docker.  
  
When it comes to container networking, companies might have different requirements such as network isolation, security, and encryption.  
  
As container technology advanced, many network providers created CNI-based solutions for containers with a wide range of networking capabilities.  
  
These solutions can be referred to as CNI plugins. This allows users to choose a networking solution that best fits their needs from different providers.  
  
How does the CNI plugin work with Kubernetes?  
  
➡️ The kube-controller-manager is responsible for assigning a pod CIDR to each node. Each pod gets a unique IP address from the pod CIDR.  
  
➡️ The kubelet interacts with the container runtime to launch the scheduled pod.  
  
➡️ The CRI (Container Runtime Interface) plugin, which is part of the container runtime, interacts with the CNI plugin to configure the pod network.  
  
➡️The CNI plugin then selects an available IP address from the configured pod CIDR and assigns it to the pod.  
  
➡️ Once the container is started, the pod will be able to connect to pods spread across the same or different nodes using an overlay network.  
  
The following are high-level functionalities provided by CNI plugins:  
  
✅ Pod networking  
  
✅ Pod network security and isolation using Network Policies to control the traffic flow between pods and between namespaces.  
  
Some popular CNI plugins include:  
  
👉 Calico  
👉 Flannel  
👉 Weave Net  
👉 Cilium (Uses eBPF)  
👉 Amazon VPC CNI (For AWS VPC)  
👉 Azure CNI (For Azure Virtual network)  
  
Kubernetes networking is a big topic and it differs based on underlying network infrastructure and requirements.  
  
Cloud platforms offer cloud-specific networking functionalities and options to use other Network Provider CNI plugins. We will look more into networking in hands-on tutorials.  
  
My previous Kubernetes posts: <https://lnkd.in/gyvQrrAs>  
  
To get notifications on my next k8s post, hit the 🔔 on my profile.  
  
𝗡𝗼𝘁𝗲: If you have trouble understanding the terms used, please learn the concepts I mentioned in the [Day 1] Kubernetes prerequisites post.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7022153392781201408) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7022153392781201408) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7022153392781201408) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7022153392781201408)



Kubernetes [Day 12]  
  
Let's learn about 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗖𝗹𝘂𝘀𝘁𝗲𝗿 𝗛𝗶𝗴𝗵 𝗔𝘃𝗮𝗶𝗹𝗮𝗯𝗶𝗹𝗶𝘁𝘆 🚀  
  
Kubernetes is a distributed system and it is subject to multiple faults.  
  
By implementing HA in Kubernetes, the risk of downtime is reduced, applications and services that run on the cluster remain available and accessible to users and the system can quickly recover from failures without human intervention.  
  
At a high level, this can be achieved by deploying multiple replicas of control plane components with a network topology spanning across multiple availability zones or regions.  
  
Running a single-node control plane could lead to a single point of failure of all the control plane components.  
  
To have a highly available Kubernetes control plane, you should have a minimum of three quorum control plane nodes with control plane components replicated across all three nodes.  
  
It's very important for you to understand the nature of each control plane component when deployed as multiple copies across nodes.  
  
➡️ 𝗲𝘁𝗰𝗱: When it comes to etcd HA architecture, there are two modes.  
  
1. Stacked etcd:- etcd deployed along with control plane nodes  
2. External etcd cluster:- Etcd cluster running externally on dedicated nodes.  
  
To have fault tolerance you should have a minimum of three node etcd cluster.  
  
➡️ 𝗔𝗣𝗜 𝗦𝗲𝗿𝘃𝗲𝗿: The API server is a stateless application that primarily interacts with the etcd cluster to store and retrieve data. This means that multiple instances of the API server can be run across different control plane nodes.  
  
To ensure that the cluster API is always available, a Load Balancer should be placed in front of the API server replicas. This Load Balancer endpoint is used by worker nodes, end-users, and external systems to interact with the cluster.  
  
➡️ 𝗞𝘂𝗯𝗲 𝗦𝗰𝗵𝗲𝗱𝘂𝗹𝗲𝗿, 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿 𝗮𝗻𝗱 𝗖𝗹𝗼𝘂𝗱 𝗖𝗼𝗻𝘁𝗿𝗼𝗹𝗹𝗲𝗿 𝗠𝗮𝗻𝗮𝗴𝗲𝗿:  
  
All these components run in a leader-follower fashion.  
  
So when you run multiple replicas of these components one instance will be elected as a leader and others will be marked as followers.  
  
This ensures that there is always a single instance of making decisions and avoiding conflicts and inconsistencies.  
  
In the event, that a leader fails, a follower will be elected a leader and takes over all the operations.  
  
𝗪𝗵𝗮𝘁 𝗛𝗮𝗽𝗽𝗲𝗻𝘀 𝗗𝘂𝗿𝗶𝗻𝗴 𝗖𝗼𝗻𝘁𝗿𝗼𝗹 𝗣𝗹𝗮𝗻𝗲 𝗙𝗮𝗶𝗹𝘂𝗿𝗲?  
  
Control Plane Failure leads to a loss of communication between the control plane components, such as the API server, and the worker nodes.  
  
As a result, new deployments, scaling, and updates to existing resources may not be possible until the control plane is restored. However, the applications that are already running on worker nodes should continue to function, as they are not directly affected by control plane failures.  
  
𝗡𝗼𝘁𝗲: I have covered only high-level concepts due to word limitation.  
  
Previous K8s posts: <https://lnkd.in/gyvQrrAs>  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7023220417746137088)

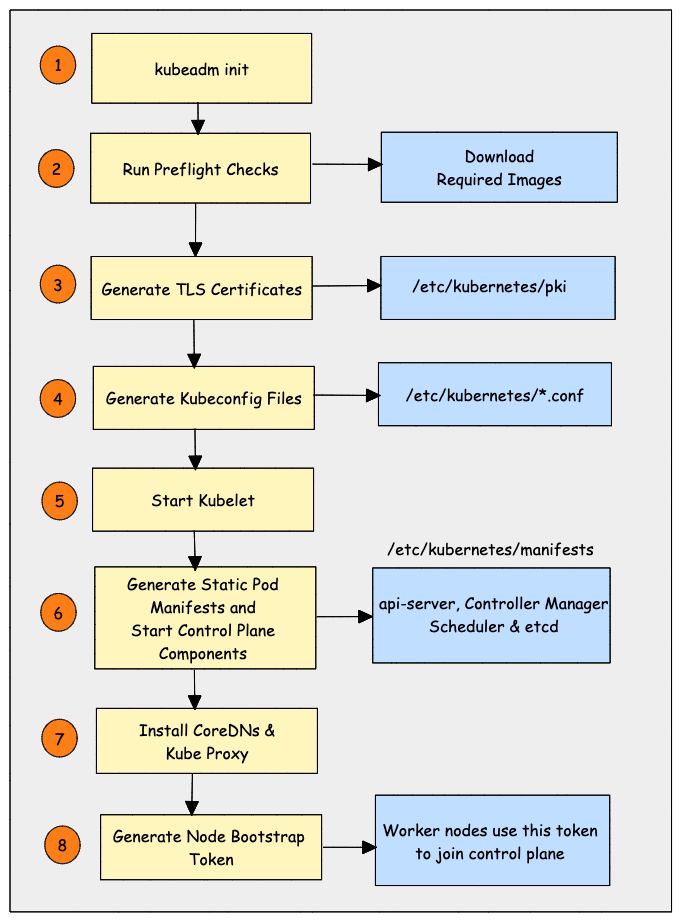


Kubernetes Learning [Day 13]  
  
Let's learn about 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗡𝗲𝘁𝘄𝗼𝗿𝗸 𝗗𝗲𝘀𝗶𝗴𝗻 🚀  
  
Creating a Kubernetes cluster in an open cloud network is pretty easy.  
  
However, it is not easy when it comes to cluster implementation in corporate networks.  
  
While creating a Kubernetes cluster, you need the following set of IP address ranges.  
  
➡️ 𝗡𝗼𝗱𝗲 𝗖𝗜𝗗𝗥: IP range for Cluster Nodes  
➡️ 𝗣𝗼𝗱 𝗖𝗜𝗗𝗥: IP range for Pods  
➡️ 𝗦𝗲𝗿𝘃𝗶𝗰𝗲 𝗖𝗜𝗗𝗥 (Specific cases like GKE)  
  
Each node would have /24 pod network (255 IPs) assigned with recommended pod limit of 110  
  
When it comes to managed Kubernetes services like GKE, EKS, etc ..the control plane will be part of a different network managed by the cloud provider.  
  
When deploying Kubernetes in a hybrid corporate network, try to avoid using the corporate network range for pods. It could eat up a lot of IP addresses.  
  
Most cloud providers offer Secondary IP ranges in VPC that can be used for POD CIDR ranges. These ranges are normally different from the corporate network range.  
  
For example, if the corporate network range is 𝟭𝟬.𝘅.𝘅.𝘅, secondary ranges could be 𝟭𝟳𝟮.𝘅.𝘅, 𝟭𝟵𝟴.𝘅.𝘅 𝗼𝗿 𝟭𝟬𝟬.𝟲𝟰.𝘅𝘅 ranges.  
  
The secondary ranges are routable within the VPC network or peered networks.  
  
However, if there is hybrid connectivity to on-prem data centers, in order for the pod to make requests to on-prem services, the second range should be part of corporate network routes.  
  
But if you don’t want to expose your pod IPs, you might need to use something like an IP masquerading agent in your cluster so that the outgoing traffic will always have the Node IP as the source identity. I have tried this in GKE.  
  
Kubernetes supports 𝗜𝗣𝘃𝟰/𝗜𝗣𝘃𝟲 𝗱𝘂𝗮𝗹-𝘀𝘁𝗮𝗰𝗸. Most managed Kubernetes services like GKE, EKS, etc support IPv6 networking.  
  
The most important thing is, the network range for Kubernetes should not be overlapping with the corporate network.  
  
✅ Always discuss with the network team and carve out an IP range from the available IP address space.  
  
❌ Don't allocate big ranges just because it is available. Calculate total node/pod/service requirements and choose a range based on that.  
  
🛡️ Also, if you plan to host PCI/PII-compliant apps, you need to discuss with both network/security teams about the compliance requirements.  
  
Networking is a complex topic and learnings come after project implementation.  
  
✅  So be a good devops engineer and collaborate with the right teams to accommodate future networking requirements and avoid potential issues.  
  
Please feel free to add your experiences and suggestions.  
  
Tomorrow we will look at using free cloud credits for launching clusters.  
  
To get a notification on my next k8s post hit the 🔔 on my profile.  
  
𝗡𝗼𝘁𝗲: If you have trouble understanding the terms used, please learn the concepts I mentioned in the [𝗗𝗮𝘆 𝟬𝟭] 𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝗽𝗿𝗲𝗿𝗲𝗾𝘂𝗶𝘀𝗶𝘁𝗲𝘀 𝗽𝗼𝘀𝘁.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7023889476804337664) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7023889476804337664) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7023889476804337664) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7023889476804337664)



Kubernetes Learning [Day 14]  
  
What is the best way to learn about Kubernetes Cluster configurations?  
  
By 𝘀𝗲𝘁𝘁𝗶𝗻𝗴 𝘂𝗽 𝘁𝗵𝗲 𝗰𝗹𝘂𝘀𝘁𝗲𝗿 𝘁𝗵𝗲 𝗵𝗮𝗿𝗱 𝘄𝗮𝘆 on a cloud platform.  
  
This way, you will learn most of the concepts we discussed in a hands-on manner.  
  
Meaning, you need to do the following:  
  
✅ Create a VPC network for the cluster  
✅ Configure firewall rules and network routes  
✅ Provision servers for control plane and worker nodes  
✅Provision CA certificates to generate SSL for cluster components  
✅ Install each and every Kubernetes component and set up authentication  
✅ Set up Pod Network routes  
✅ Test the cluster by deploying a test application.  
  
There is a popular repo named “𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 𝘁𝗵𝗲 𝗛𝗮𝗿𝗱 𝗪𝗮𝘆” which contains step-by-step instructions to perform all of the above on Google cloud with $300 free credits.  
  
Spend some time and finish the lab, trying to understand everything with further research.  
  
Github Repo: <https://lnkd.in/g3y_6waD>  
  
If you face any problems during the cluster setup, you are lucky. The best learning comes when fixing systems.

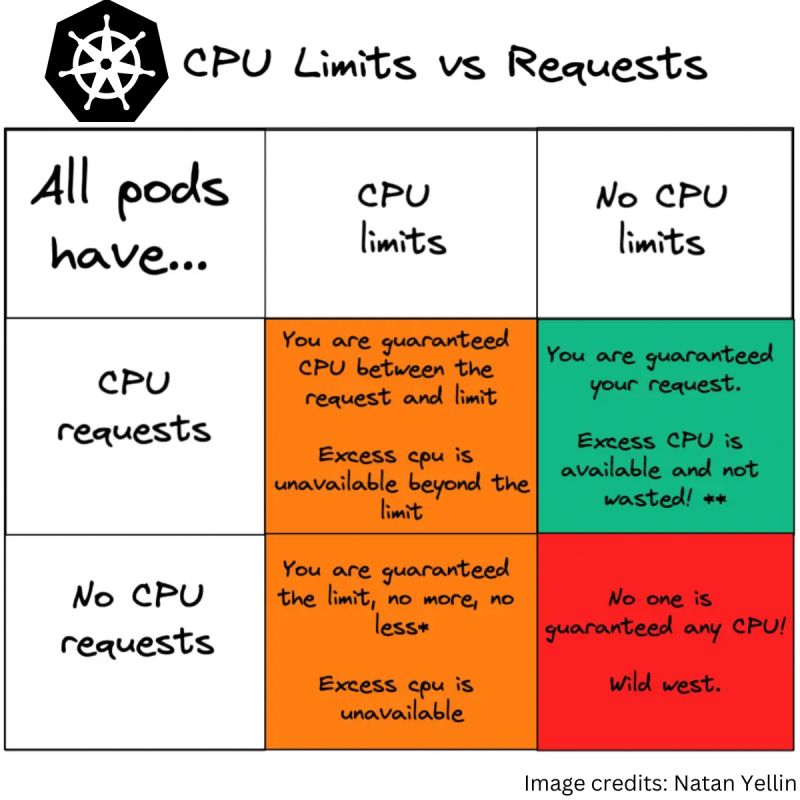
Learning Kubernetes [Day 14]  
  
Let's learn about 𝗞𝘂𝗯𝗲𝗮𝗱𝗺 utility 🚀  
  
For the CKA exam, it is essential to learn about Kubeadm, and also every k8s beginner should try creating clusters with Kubeadm.  
  
Kubeadm is a popular tool for setting up and managing Kubernetes clusters, and it's developed and maintained by the Kubernetes community.  
  
By using Kubeadm, you can create a minimum viable cluster easily and with minimum configuration, making it a great option for development and testing. The tool also includes prechecks to ensure that the server has all the necessary components and configurations to run Kubernetes.  
  
You can create a 𝗽𝗿𝗼𝗱𝘂𝗰𝘁𝗶𝗼𝗻-𝗹𝗶𝗸𝗲 𝗰𝗹𝘂𝘀𝘁𝗲𝗿 locally on a workstation for development and testing purposes.  
  
If you want to play around with the cluster configurations, components, or test utilities that are part of cluster administration, Kubeadm is the best option.  
  
I have written a 𝘀𝘁𝗲𝗽-𝗯𝘆-𝘀𝘁𝗲𝗽 𝗯𝗹𝗼𝗴 on getting started with a Kubeadm cluster (𝗩𝗲𝗿𝘀𝗶𝗼𝗻 𝟭.𝟮𝟲).  
  
You can use Vagrant local machines or cloud VMS for the setup.  
  
I have also created a repo with Vagrantfile, shell scripts, and the required manifest to set up the cluster quickly.  
  
𝗗𝗲𝘁𝗮𝗶𝗹𝗲𝗱 𝗛𝗮𝗻𝗱𝘀-𝗼𝗻 𝗕𝗹𝗼𝗴: <https://lnkd.in/gr3Cs3F>  
  
𝗞𝘂𝗯𝗲𝗮𝗱𝗺 𝗦𝗰𝗿𝗶𝗽𝘁𝘀 𝗥𝗲𝗽𝗼: <https://lnkd.in/gZiE_J3n>  
  
Tomorrow we will learn about the 𝗞𝘂𝗯𝗲𝗰𝗼𝗻𝗳𝗶𝗴 file in detail.  
  
My previous Kubernetes posts: [bit.ly/m/bibinwilson](http://bit.ly/m/bibinwilson)  
  
To get notifications on my next k8s post, hit the 🔔 on my profile.  
  
[#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7025791679035035648) [#learnkubernetes](https://www.linkedin.com/feed/hashtag/?keywords=learnkubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7025791679035035648) [#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7025791679035035648) [#devopsengineer](https://www.linkedin.com/feed/hashtag/?keywords=devopsengineer&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7025791679035035648)



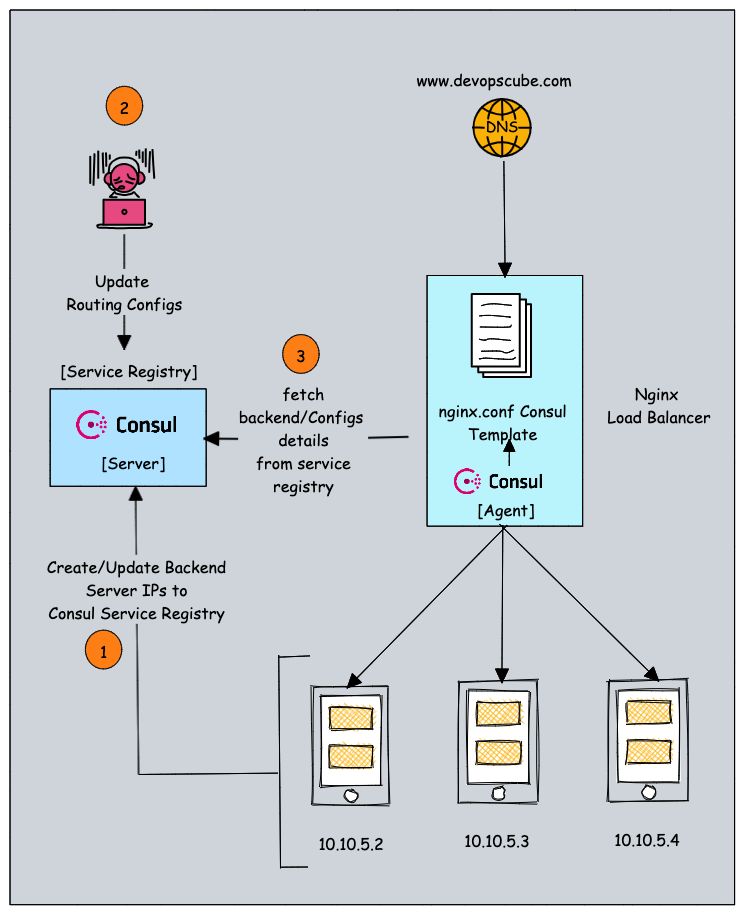
Activate to view larger image

[4 hours ago](https://www.linkedin.com/in/pavan-belagatti-growthmarketer?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAAAriex8BZ3Sl8QTrVC0Izjw-PnEn71SyX2Y" \t "_self)

Why configure [#Kubernetes](https://www.linkedin.com/feed/hashtag/?keywords=kubernetes&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7025798243863183360) memory limits?  
  
Kubernetes memory limits are set to ensure the cluster's resources are used optimally, prevent overuse of memory, ensure stable performance, and avoid crashing of nodes due to memory exhaustion.  
  
Many people think you need limits to prevent one pod from interfering with another pod. This is not true! You can remove Kubernetes CPU limits and still prevent a CPU hungry pod from causing CPU starvation! The trick is to just define CPU requests.  
  
The important part is highlighted. Pods always get the CPU requested by their CPU request! (They can take advantage of excess CPU too if they have no limit.)  
  
Disable your CPU limits! If you give all your K8s pods accurate CPU requests, then no-one can throttle them because CPU is reserved for them if they need it. This has nothing to do with limits.  
  
He summarises some best practices for CPU limits and requests on Kubernetes  
- Use CPU requests for everything  
- Make sure they are accurate  
- Do not use CPU limits.  
  
Read the original article by Natan Yellin - <https://lnkd.in/g_KjJrgt>



Practical DevOps: Learn 𝗦𝗲𝗿𝘃𝗶𝗰𝗲 𝗗𝗶𝘀𝗰𝗼𝘃𝗲𝗿𝘆 🚀  
  
Service discovery is an important concept every DevOps Engineer should learn.  
  
It is the concept of discovering service information in real-time.  
  
It involves a centralized service registry that maintains information about available services  
  
Service discovery is of two types  
  
✅ 𝗖𝗹𝗶𝗲𝗻𝘁 𝗦𝗶𝗱𝗲 𝗦𝗲𝗿𝘃𝗶𝗰𝗲 𝗗𝗶𝘀𝗰𝗼𝘃𝗲𝗿𝘆: In this model, the request goes to a service registry to get the information available for backend services. For example, an application connects to a service registry to get information about the database endpoint.  
  
✅ 𝗦𝗲𝗿𝘃𝗲𝗿 𝗦𝗶𝗱𝗲 𝗦𝗲𝗿𝘃𝗶𝗰𝗲 𝗗𝗶𝘀𝗰𝗼𝘃𝗲𝗿𝘆: In this model request goes to a load balancer and the load balancer uses the service registry to get the information of the backend servers. The example is shown in the image.  
  
To understand service discovery practically, here is what you can do.  
  
➡️ Setup Hashicorp Consul (Service registry)  
  
➡️ Setup Nginx load balancer with consul agent to query the consul server (Service registry)  
  
➡️ Configure the nginx.conf as a consul template (Go template) to retrieve backend IPs/Configs in real-time from the service registry.  
  
➡️ Set up backend servers for the load balancer and update the IPs to the service registry.  
  
➡️ Validate the setup by checking if the load balancer is able to serve the traffic by reading backend information from the service registry.  
  
➡️ If you change/update any backend info on the service registry, the consul agent running on the load balancer identifies it and updates the info in the nginx.conf file.  
  
𝗡𝗼𝘁𝗲: This is just a short and practical workflow to understand service discovery. In real projects, more automation is involved in the registry update and retrieval process.  
  
If you want some clarification, please drop a comment.  
  
[#devops](https://www.linkedin.com/feed/hashtag/?keywords=devops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7027600877934829568) [#practicaldevops](https://www.linkedin.com/feed/hashtag/?keywords=practicaldevops&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7027600877934829568)



Activate to view larger image