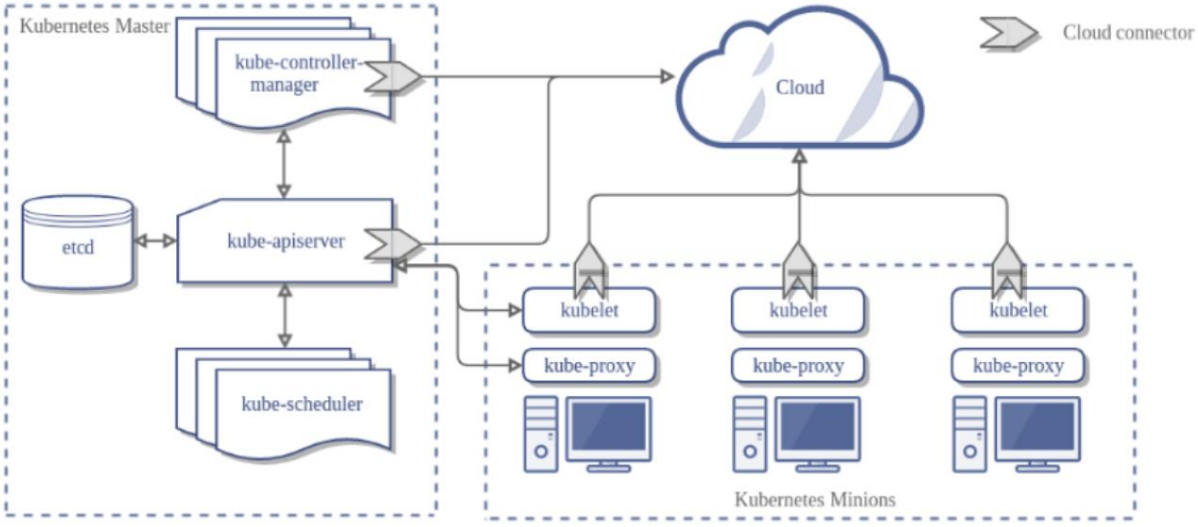
Lifecycle of K8S Request

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. It is designed by Google and now being maintained by CNCF.

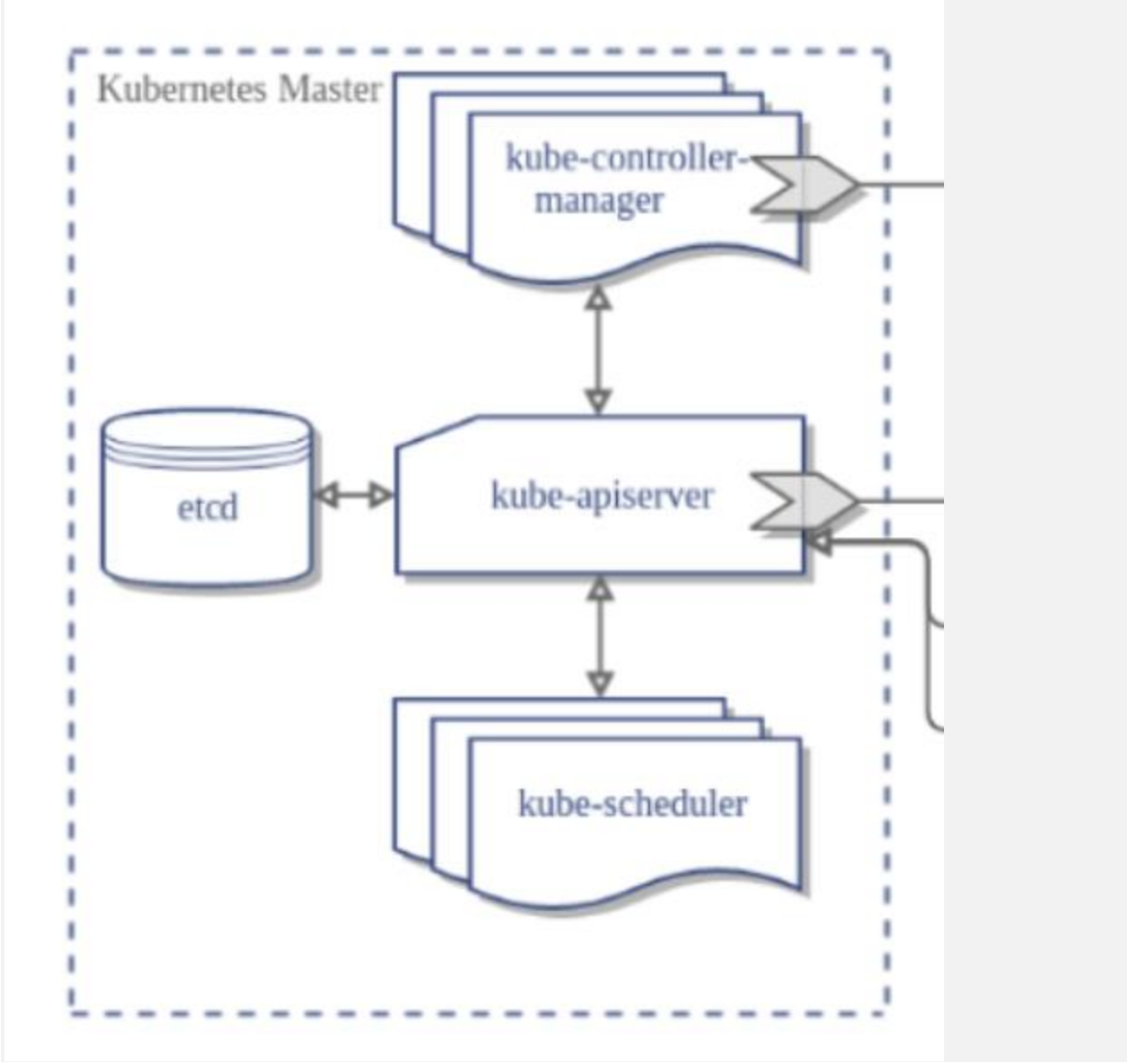
The term “Kubernetes” (κυβερνήτης) is originated from greek language, meaning “helmsman” or “pilot” or “governor”. Now taking this as a base meaning, Kubernetes in DevOps world is considered as an orchestration tool which actually manages/governs/dictates all container operations at server level.



K8s Architecture

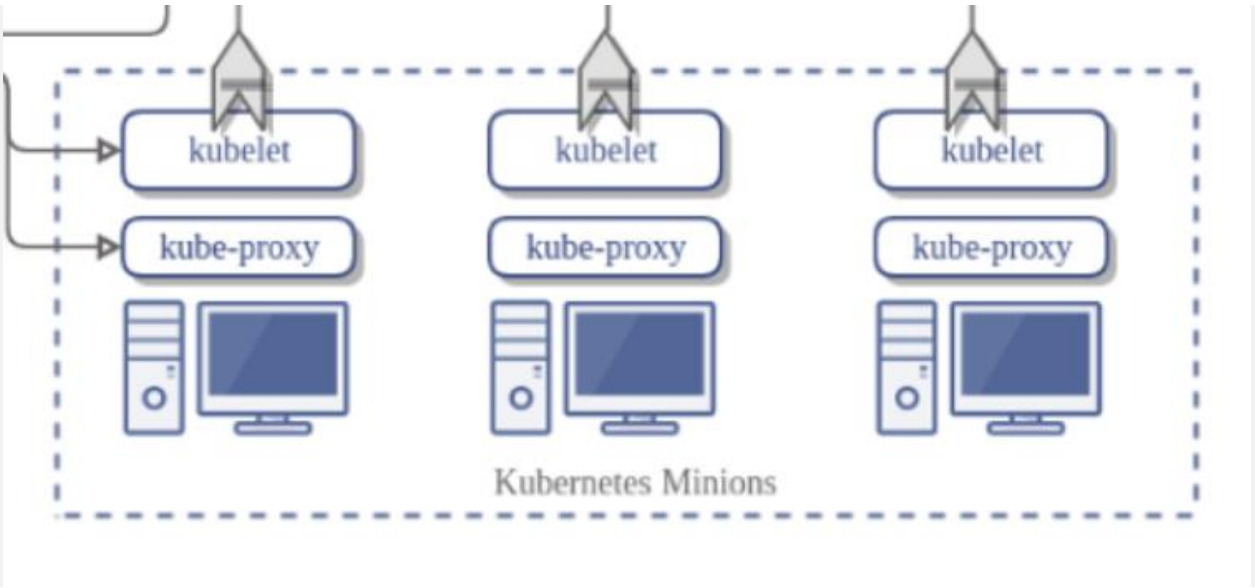
Kubernetes Architecture can be divided into two components.

1. Control Plane or Kubernetes Master



Control Plane (or) Kubernetes Master

2. Data Plane or Kubernetes Minions



Data Plane (or) Kubernetes Minions

Kubernetes Control Plane

Kubernetes Control Plane can be considered as nervous system of entire architecture that maintains a record of all K8s objects that continuously manages object states, responds to changes in the cluster which actually maintains the actual state of system objects and works to match the desired state.

It consists of 4 components:
1. kube-apiserver
2. etcd (Distributed database)
3. kube-controller-manager
4. kube-scheduler

kube-apiserver:

kube-apiserver is nucleus of entire control plane and can be considered as manager in office who actually receives/responds to

any primary request. In similar way, kube-apiserver also acts as the primary level end-point component that actually controls the functionality of any object request that the cluster receives.
The API server is the front end for the Kubernetes control plane and it is designed to scale horizontally — that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

Note: *All components in cluster are connected to kube-apiserver*.

etcd (Distributed Database):
etcd is a distributed database which can be considered as CEO of the company who actually maintains the entire data of company and kube-apiserver which is referred as manager in office is the only component that is actually connected to etcd database which updates any change in kubernetes cluster etcd database.

Why only etcd when there are many databases?
Kubernetes uses etcd like any normal application uses a database and etcd is built on the Raft consensus algorithm to ensure data store consistency across all nodes in a cluster-table stakes for a fault-tolerant distributed system. It stores configuration data, state, and metadata in etcd. As Kubernetes is a distributed system that runs on ideally one *master* and multiple *worker* nodes at basic level it needs a distributed database too.

One of the etcd feature that's really useful to Kubernetes is it makes easy to *watch* for changes to data stored inside it as many components of Kubernetes are implemented as a control loop. Kubernetes stores the *ideal* state and the *actual* state of objects and the control loop sees that these two states have diverged, it'll make changes to reconcile them. The *watch* feature makes it easier to find out when the states have changed.

kube-controller-manager:
kube-controller-manager is a combination of multiple smaller components *(Node controller, Job controller, End Points controller, Service Account & Token controllers)* which are packaged and installed as single binary which actually manages all the operational side of objects in cluster.

- Node Controller component helps to maintain the life cycle of nodes.
- Job Controller component helps to execute the tasks or jobs created for objects like creating, updating pods etc.
- End Points Controller component helps to populate the end point objects like services, pods, deployments etc.
- Service Account & Token Controllers component helps to create/maintian default accounts.

kube-scheduler:
kube-scheduler is considered as election commission body who schedules the elections at specific place as per the requirement, In similar way this component ensures each pod is assigned to a node to run on. All pods in kubernetes uses shared storage and network resources and it is kube-scheduler's responsibility to look for any new scheduled request.

Let's say a cluster has single master and multiple worker nodes with 2 GB RAM (Memory) and 4 CPU. When there is a request to create application pod, the pods are created on a node that has enough available resources. Therefore, the scheduler continues to run forever, watching to see if there are pods that need to be scheduled.

Kubernetes Data Plane
Data Plane consists of worker instances or nodes which run with set of components by default and these are responsible for maintaining running pods and providing the Kubernetes runtime environment.

Every node is Data Plane consists of two components:
1. kubelet
2. kube-proxy

kubelet:
An agent that runs on each node in the cluster. It makes sure

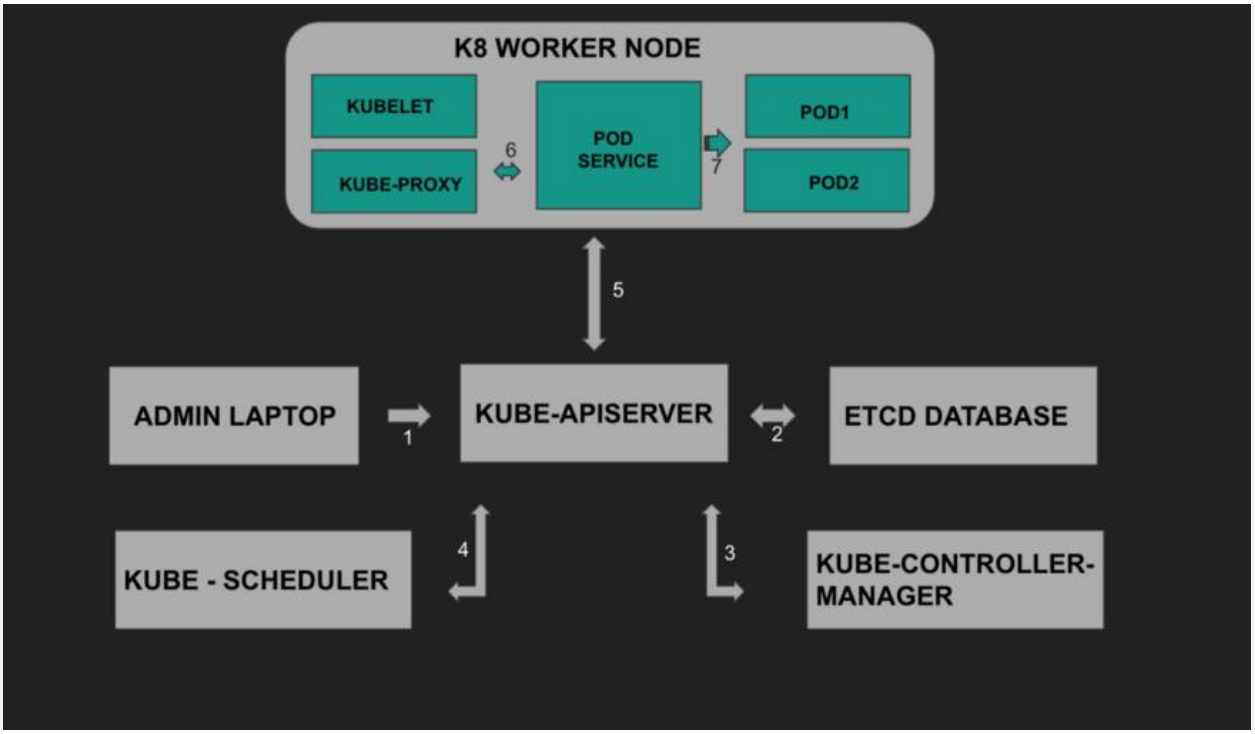
that containers are running in a Pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

Note: *Kubelet is the only component in entire K8 architecture that runs as software, rest of the components runs as pods when you initialize the cluster (kubeadm init). It's the task of kubernetes engineer to install kubelet, ideally installed using* ***apt-get install kubelet*** *on Ubuntu machines.*

kube-proxy:

kube-proxy is a network-proxy, daemon that runs on each node. It basically reflects the services defined in the cluster and manages the rules to load-balance requests to a service's backend pods.

Life Cycle of a request to K8's Cluster



Life Cycle of Pod Object Creation in K8 Cluster

Consider a scenario where Tejaswini wanted to create a create a pod **kubectl create -f pod.yaml** which says 4 replicas in pod configuration and with a service already created on Worker Node.

Here is the flow of request:

Step 1: User (Tejaswini) who has K8 Cluster access wants to create a nginx pod. She creates the pod by running **kubectl create -f nginx.yaml**. Now this request will be sent to kube-apiserver at step 1. kube-apiserver compiles the request at this step and routes to etcd database.

Step 2: The request will be routed from kube-apiserver to etcd database and at this step, etcd database saves the pod configuration you have requested and responds to kube-apiserver saying it saved all the data requested by server.

Step 3: Now kube-apiserver will send request to kube-controller-manager to create pods. In this case as we are creating pods, job controller manager component will create a task or job to create pod and responds back to kube-apiserver.

Step 4: Now kube-apiserver will route request to kube-scheduler and kube-scheduler will pick the task created by kube-controller-manager and scheduler will schedule the task to create pod as per pod specifications and route the request to kube-apiserver.

Step 5: Now kube-apiserver will route request to worker node(s) which has different objects that are already created like services etc.

Step 6: Inside worker node, when a request reaches the service virtual IP, the request is forwarded to one of the underlying pods.

Step 6: This is achieved by using the rules that kube-proxy have created. kube-proxy usually runs in different modes like iptables, ipvs to route the request to pods.