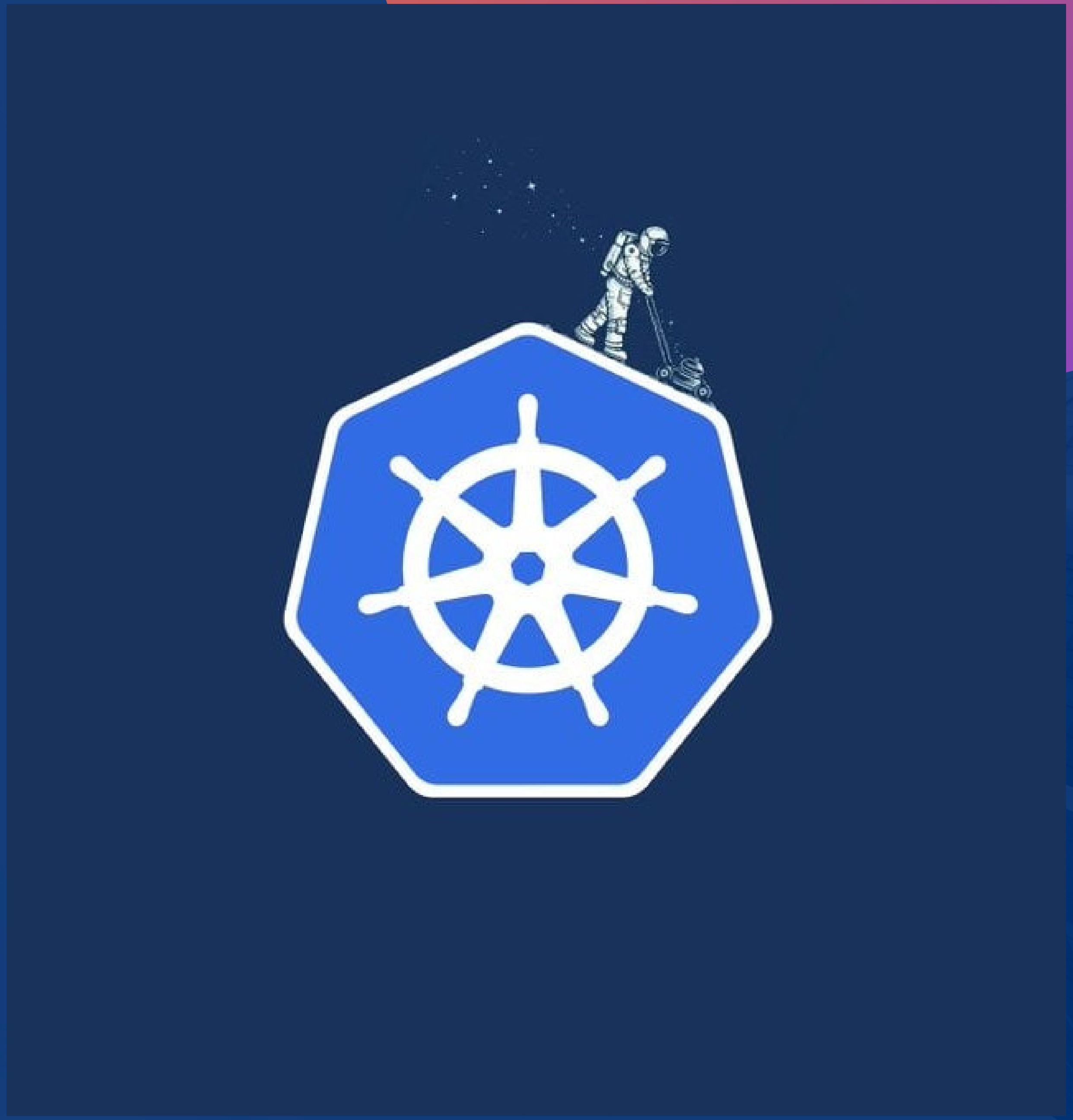
Kubernetes



AMOL SHETE

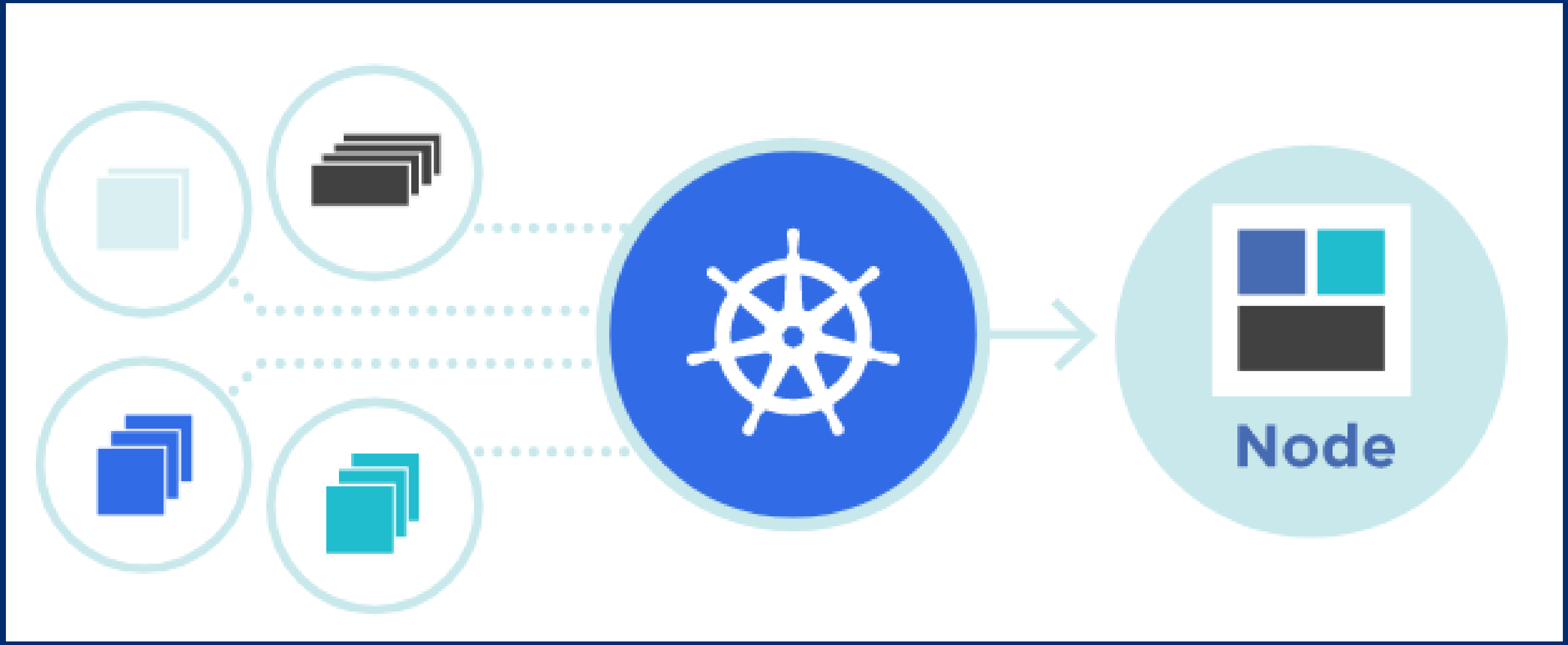
Little history about K8's

Kubernetes was originally developed by Google in 2014 as an open-source container orchestration tool. It was created to help manage and automate the deployment, scaling, and management of containerized applications.
Over the years, Kubernetes has evolved and improved through community contributions, with several major releases introducing new features and improvements.

Before Kubernetes, the project was known as the "Borg" system, which was developed by Google to manage their own large-scale containerized applications. Borg was not open-sourced, and its functionality was limited to internal use within Google.

What is Kubernetes?

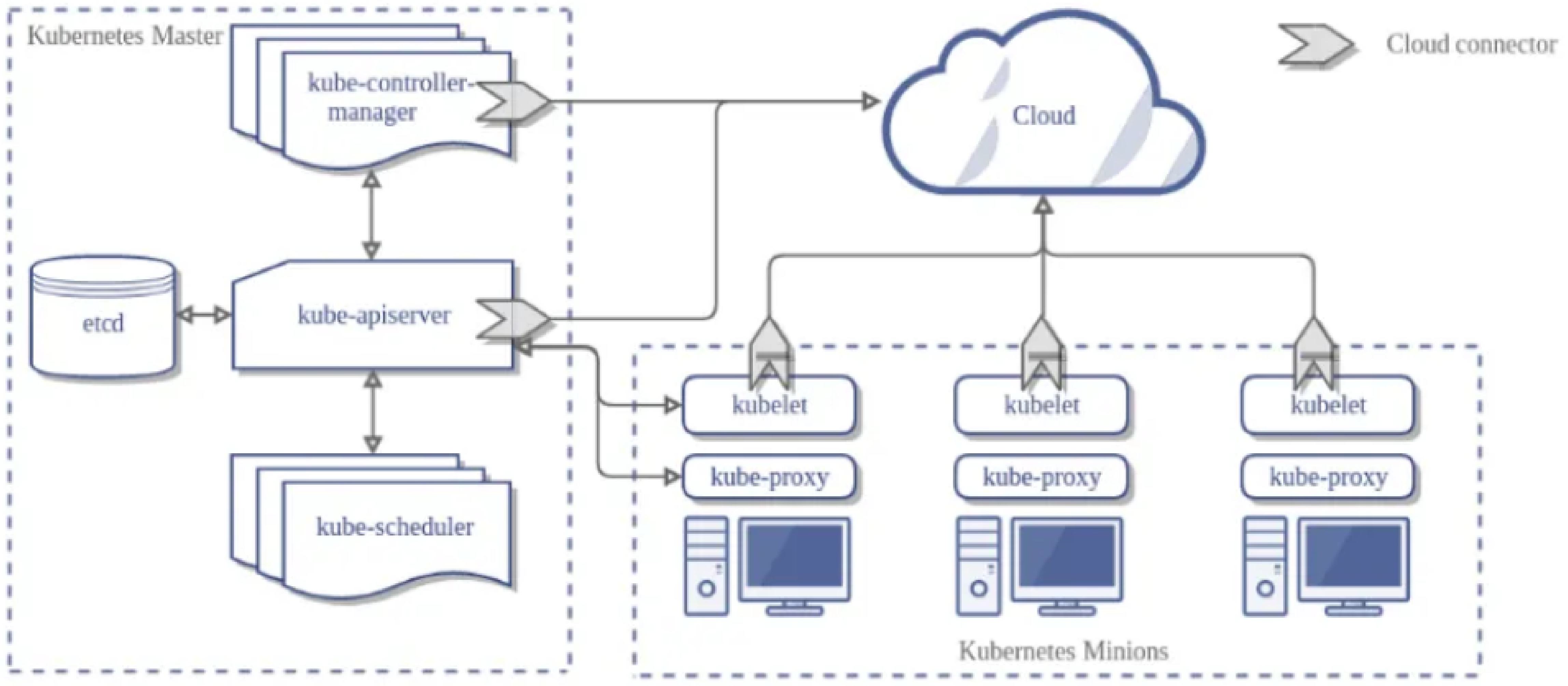
Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.



Kubernetes Features

- Automated rollouts and rollback
- Service discovery and load balancing
- Storage orchestration
- Self-healing
- Secret and configuration management
- Horizontal scaling
- Designed for extensibility

Kubernetes Components



K8s Architecture

Master Node Components

Kube-APIServer:
APIs allow applications to communicate with one another. There is a component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.

Kube-Scheduler:
It is a component on the master node that watches newly created pods that have no node assigned 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.

Master Node Components

Kube-Controller-manager:

This is a component on the master that runs controllers.
Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.
These controllers include:

- Node Controller: Responsible for noticing and responding when nodes go down.
- Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- Endpoints Controller: Populates the Endpoints object (that is, it joins Services and Pods).
- Service Account and Token Controllers: Create default accounts and API access tokens for new namespaces.

Master Node Components

Cloud-Controller-Manager:
Cloud-controller-manager runs controllers that interact with the underlying cloud providers.
The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.
etcd :
It stores the configuration information which can be used by each of the nodes in the cluster. It is a high availability key-value store that can be distributed among multiple nodes. It is accessible only by Kubernetes API server as it may have some sensitive information. It is a distributed key-value store which is accessible to all.

Worker Node Components

kubelet :
An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.
The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.

kube-proxy :
kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

Worker Node Components

Container runtime :
The container runtime is the software that is responsible for running containers. Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

Follow the below Steps:

Create Google cloud platform account:

Create K8s cluster on GCP

Kubectl tool installation

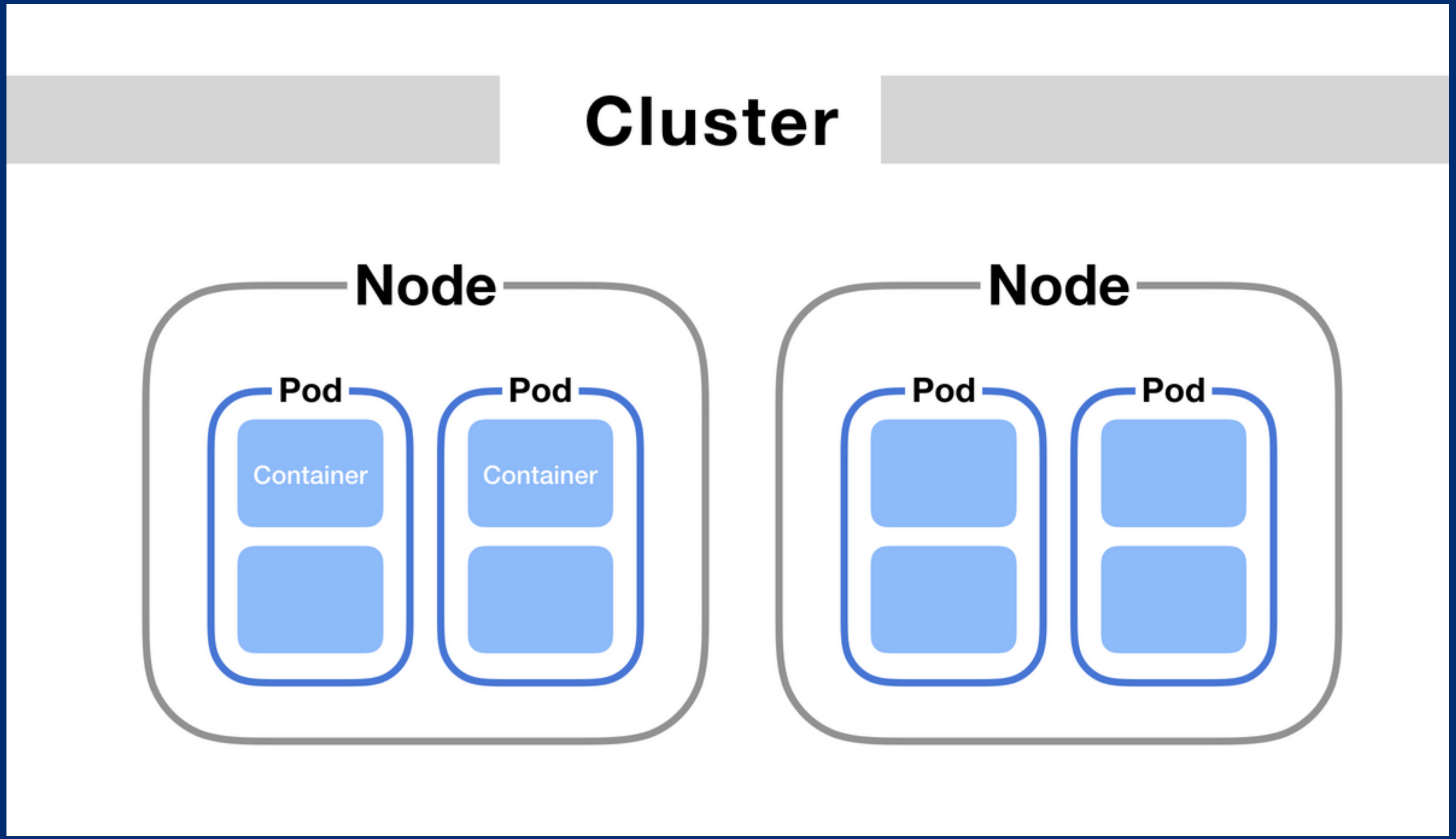
Do install Google SDK and setup with gcloud init

Generate credentials for k8s cluster for kubect:

https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectll

Pod:

In Kubernetes, a Pod is the smallest and simplest unit in the Kubernetes object model, and it represents a single instance of a running process in a cluster.



A Pod is a logical host for one or more containers, and it provides a shared environment for those containers to run in. Containers within a Pod share the same network namespace and can communicate with each other using localhost.

Few Command to create pod:

To create a Pod in Kubernetes using the kubectl command-line tool, you can use the kubectl run command followed by the name of the Pod and the image you want to use.

Eg:
```
kubectl run my-nginx --image=nginx
```

Pod Creation using Yaml manifest file

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    ports:
    - containerPort: 80
```

Here's a brief explanation of the fields in this manifest file:

- apiVersion: The version of the Kubernetes API being used, which in this case is "v1".
- kind: The type of Kubernetes object being created, which in this case is "Pod".
- metadata: Information about the Pod, including its name.
- spec: The specification for the Pod, including its containers.
- containers: An array of containers running within the Pod.
- name: The name of the container.
- image: The Docker image to use for the container.
- ports: An array of ports to expose for the container.
- containerPort: The port number to expose for the container.

Explain Services

In Kubernetes, you have a bunch of little programs (called "pods") that work together to make a big program. Each pod might be doing something different, like storing data or handling user requests.
But just like with the building blocks, you need a way to make sure each pod knows where the others are and how to talk to them. That's where services come in!
Services are like a special kind of block that sits in front of your pods. When a pod wants to talk to another pod, it sends a message to the service instead. The service then figures out which pod to send the message to and makes sure it gets there.
So services are really important in Kubernetes because they help all the different parts of your program talk to each other and work together as a team!

Different Types of services:

1. ClusterIP: This is the default type of service. It provides a stable IP address that other pods inside the cluster can use to connect to the service. This type of service is only accessible from inside the cluster, meaning it can't be accessed from outside the cluster.
2. NodePort: This type of service makes the service accessible from outside the cluster by mapping a port on the cluster's nodes to a port on the service. For example, if you have a NodePort service running on port 30001, you can access it from outside the cluster by connecting to any of the nodes in the cluster on port 30001.

Different Types of services:

3. LoadBalancer: This type of service creates a load balancer in the cloud provider's network that routes traffic to the service. This type of service is useful when you need to expose the service to the public internet or to a specific set of IP addresses.

Exposing services in K8s:

https://cloud.google.com/kubernetes-engine/docs/how-to/exposing-apps

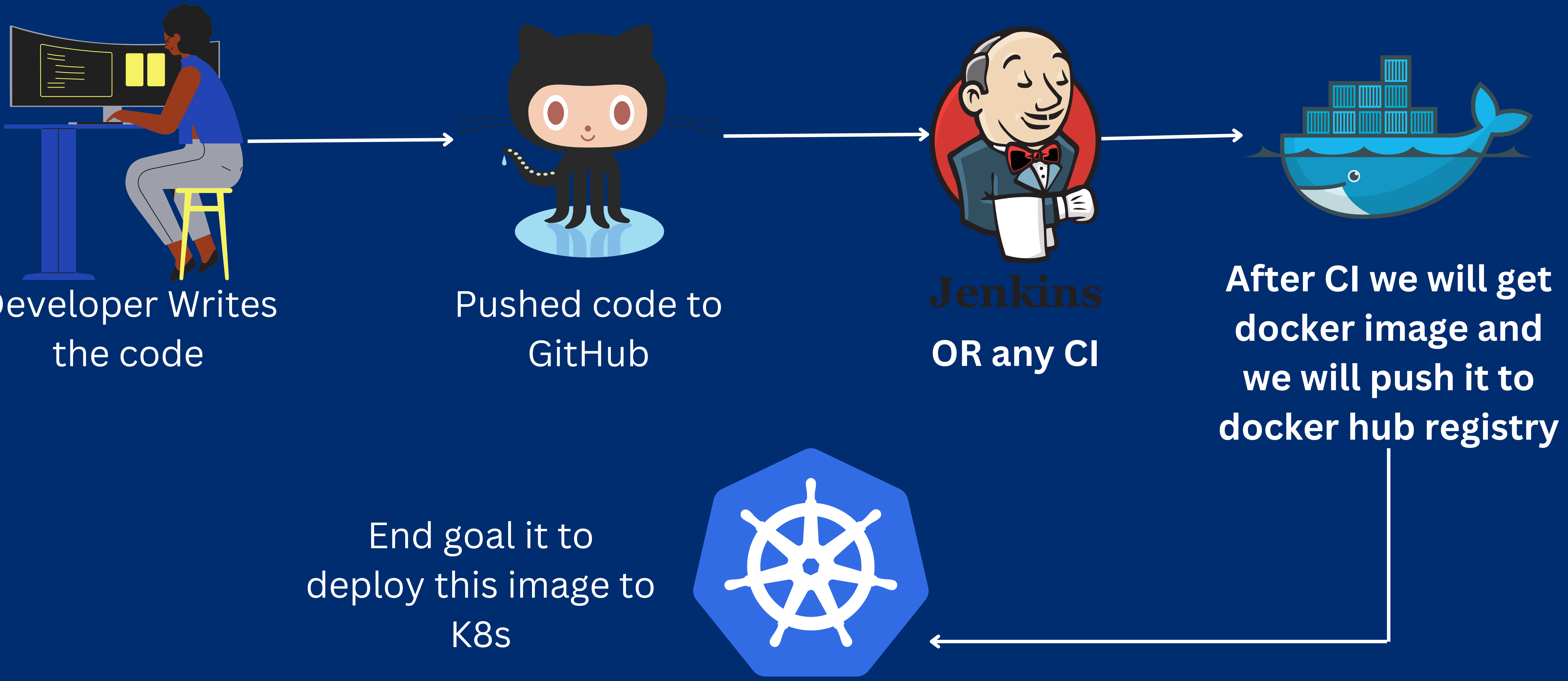
What is Replicaset?

A ReplicaSet in Kubernetes is used to ensure that a specified number of replicas of a pod are running at any given time. It provides a way to manage and scale the number of identical pods that make up a service or application.

What is Deployment?

In Kubernetes, a Deployment is an object that manages a set of replicas of a pod template. Deployments provide a way to manage updates and rollbacks to applications running on a Kubernetes cluster.

General Workflow



Developer Writes the code

Pushed code to GitHub

OR any CI

After CI we will get docker image and we will push it to docker hub registry

End goal it to deploy this image to K8s

How to pull image from Docker Hub Private Repo

Create a Kubernetes secret that holds the credentials to access the private Docker registry.
Here's an example command to create a Kubernetes secret with the Docker registry credentials:

```
kubectl create secret docker-registry <SECRET_NAME> --docker-server=
<DOCKER_REGISTRY_SERVER> --docker-username=
<DOCKER_REGISTRY_USERNAME> --docker-password=
<DOCKER_REGISTRY_PASSWORD> --docker-email=<DOCKER_REGISTRY_EMAIL>
```

Here docker registry server is https://index.docker.io/v1/

Ref https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/