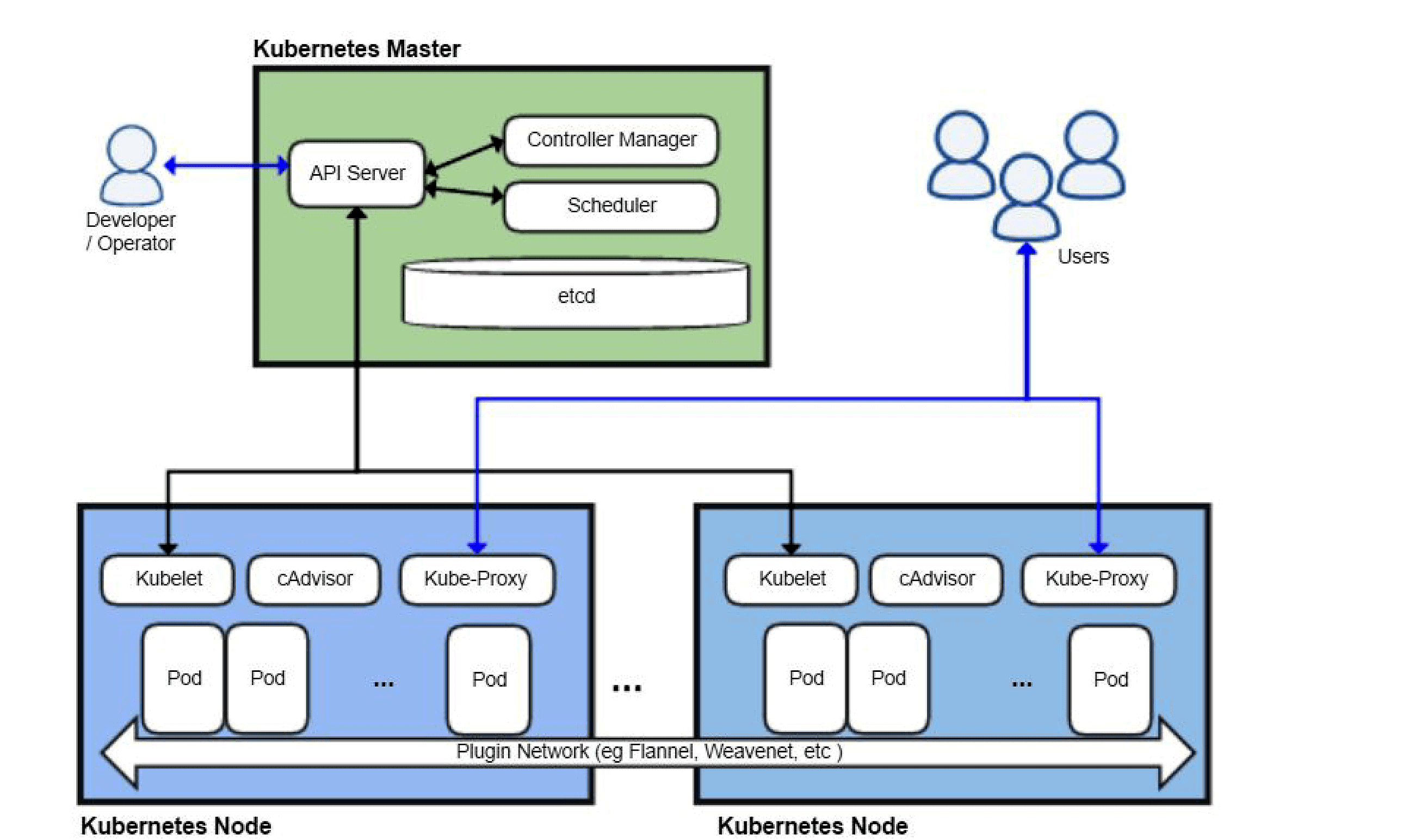
**Kubernetes**

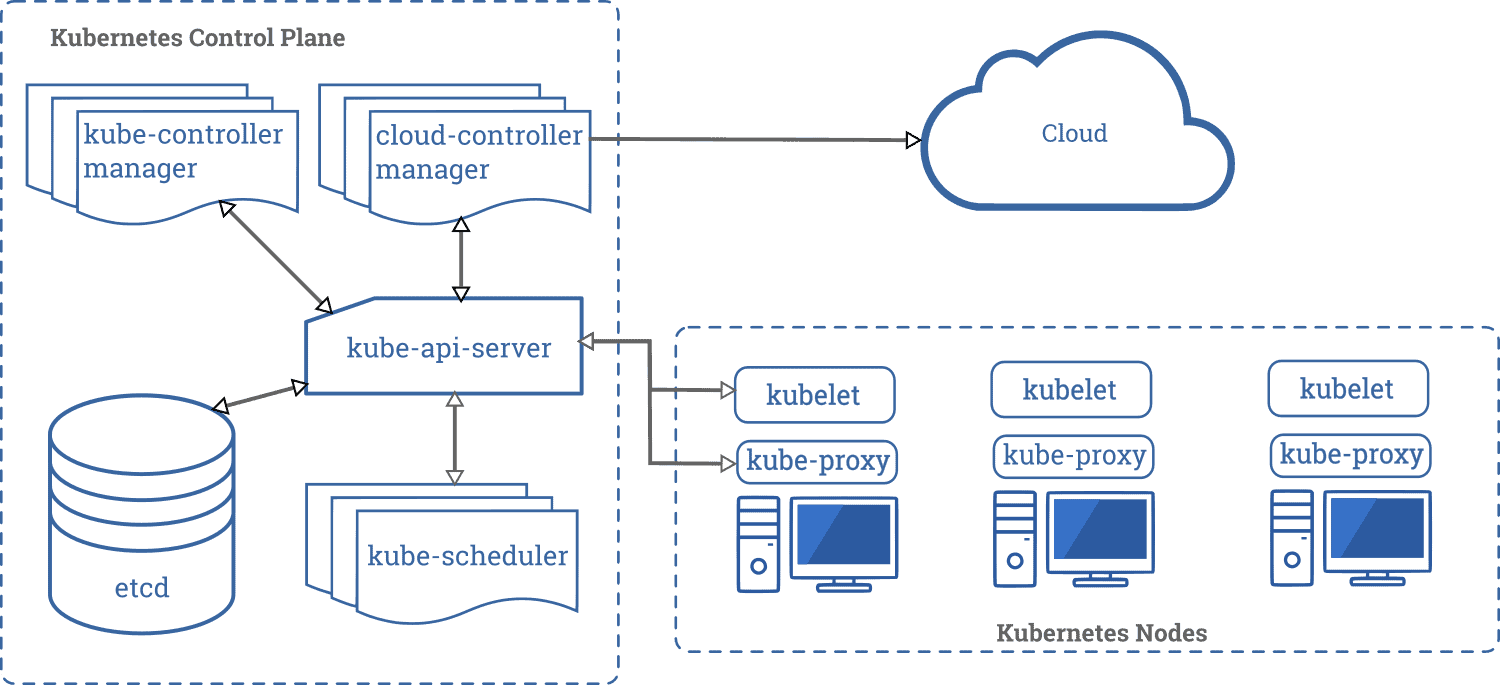
**“Kubernetes”** is a Greek word, which means helmsman or pilot which also gives us an idea of how the logo was made. Now, let us come to the technical part of it since Docker has its own limitations, Kubernetes comes into the picture to fill the gaps in the Docker containerization process. K8s is a complete containerization orchestration, which provides the ability to run dynamically scaling, containerized applications and utilizing an API for management. By doing a comparison of **Docker & Kubernetes** one can deduce the advantages of Kubernetes over other containerization orchestration.

**Kubernetes Architecture Diagram**

****

1. In the AWS Kubernetes architecture diagram above you can see, there are one or more master and multiple nodes. One or master is used to provide high availability.
2. The Master node communicates with Worker nodes using Kube API-server to kubelet communication.
3. In the Worker node, there can be one or more pods and pods can contain one or more containers.
4. Containers can be deployed using the image and can be deployed externally by the user.

**Kubernetes Architecture Components**

****

**Kubernetes Master Node**

**Master Node** is a collection of components like Storage, Controller, Scheduler, and API-server that makes up the control plan of Kubernetes. When you interact with Kubernetes by using CLI you are communicating with the Kubernetes cluster’s master node. All the processes run on a single node in the cluster, and this node is also referred to as the master.

**Master Node Components:**

1. **Kube API server**performs all the administrative tasks on the master node. A user sends the rest commands as YAML/JSON format to the API server, then it processes and executes them. The Kube API server is the front end of the Kubernetes control plane.
2. **etcd** is a distributed key-value store that is used to store the cluster state. Kubernetes stores the file in a database called the **etcd**. Besides storing the cluster state, etcd is also used to store the configuration details such as the subnets and the config maps.
3. **Kube-scheduler** is used to schedule the work to different worker nodes. It also manages the new requests coming from the API Server and assigns them to healthy nodes.
4. **Kube Controller Manager’s** task is to obtain the desired state from the API Server. If the desired state does not meet the current state of the object, then the corrective steps are taken by the control loop to bring the current state the same as the desired state.

There are different types of control managers in Kubernetes architecture:

* **Node Manager,** it manages the nodes. It creates new nodes if any node is unavailable or destroyed.
* **Replication Controller,**it manages if the desired number of containers is running in the replication group.
* **Endpoints controller,** it populates the endpoints object that is, joins Services & Pods.

**Kubernetes Worker Node**

The worker nodes in a cluster are the machines or physical servers that run your applications. The Kubernetes master controls each node. there are multiple nodes connected to the master node. On the node, there are multiple pods running and there are multiple containers running in pods.

**Worker Node Components**

1. **Kubelet**is an agent that runs on each worker node and communicates with the master node. It also makes sure that the containers which are part of the pods are always healthy. It watches for tasks sent from the API Server, executes the task like deploying or destroying the container, and then reports back to the Master.
2. **Kube-proxy** is used to communicate between the multiple worker nodes. It maintains network rules on nodes and makes sure there are necessary rules defined on the worker node so the container can communicate to each in different nodes.
3. **Kubernetes pod** is a group of one or more containers that are deployed together on the same host. The pod is deployed with a shared storage/network, and a specification for how to run the containers. Containers can easily communicate with other containers in the same pod as though they were on the same machine.
4. **Container Runtime** is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containers.

**Kubernetes Objects**

Before one wants to work on K8s, one should know about the basic components in Kubernetes and also about objects in API. Basic objects and several higher-level abstractions are known as **controllers**. These are the building block of the application lifecycle.

Basic objects include:

* **Pod.** A group of one or more containers, a simple deployable unit of K8S.
* **Service.** An abstraction that defines a logical set of pods as well as the policy for accessing them.
* **Volume.** An abstraction that lets us persist data. This is necessary because containers are ephemeral—meaning data is deleted when the container is deleted.
* **Namespace.** A segment of the cluster dedicated to a certain purpose, for example, a certain project or team of devs.

Controllers, (higher-level abstractions) include:

* **ReplicaSet (RS).** Ensures the desired amount of pods is what’s running.
* **Deployment.** Offers declarative updates for pods and RS.
* **StatefulSet.** A workload API object that manages stateful applications, such as databases.
* **DaemonSet.** Ensures that all or some worker nodes run a copy of a pod. This is useful for daemon applications like Fluentd.
* **Job.** Creates one or more pods, runs a certain task(s) to completion, then deletes the pod(s).

**Microservice**

Monolithics had way too many backlogs, this is why Microservices have come into the picture. A traditional micro-service-based architecture would have multiple services making up one, or more, end products. Microservices are typically shared between applications and make the task of CI/CD easier for management.

**Images**

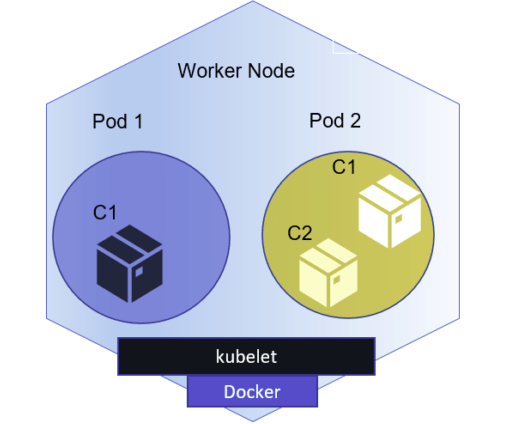
Typically, a docker container image – is an executable image containing everything you need to run your application, application code, libraries, runtime, environment variables, and configuration files. At runtime, a container image becomes a container that runs everything that is packaged into that image.

**Labels and Annotations**

Labels and Annotations in Kubernetes are one of the main components. They both provide a way for adding additional metadata to our Kubernetes Objects but there is one difference between them. Labels allow us to do a grouping of our objects so that we can perform queries for viewing and operating while Annotations are used for adding non-identifying metadata to Kubernetes objects.

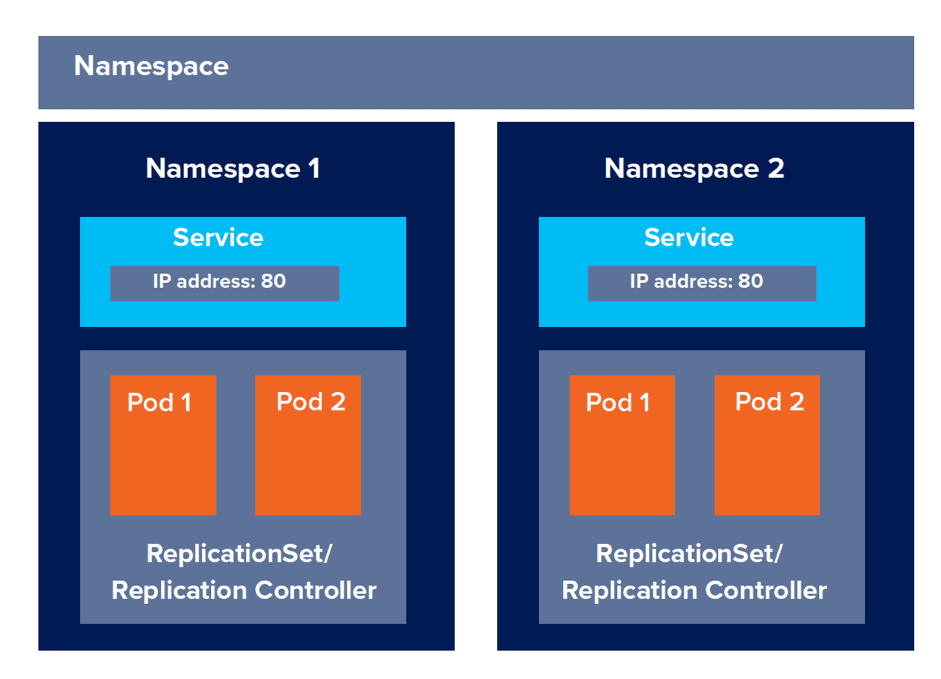
**Pods**

A single or group of containers that share storage and network with a K8s configuration, telling those containers how to behave. Pods share IP and port address space and can communicate with each other over local host networking. Each pod is assigned an IP address on which it can be accessed by other pods within a cluster. Applications within a pod have access to shared volumes – helpful when you need data to persist beyond the lifetime of a pod.



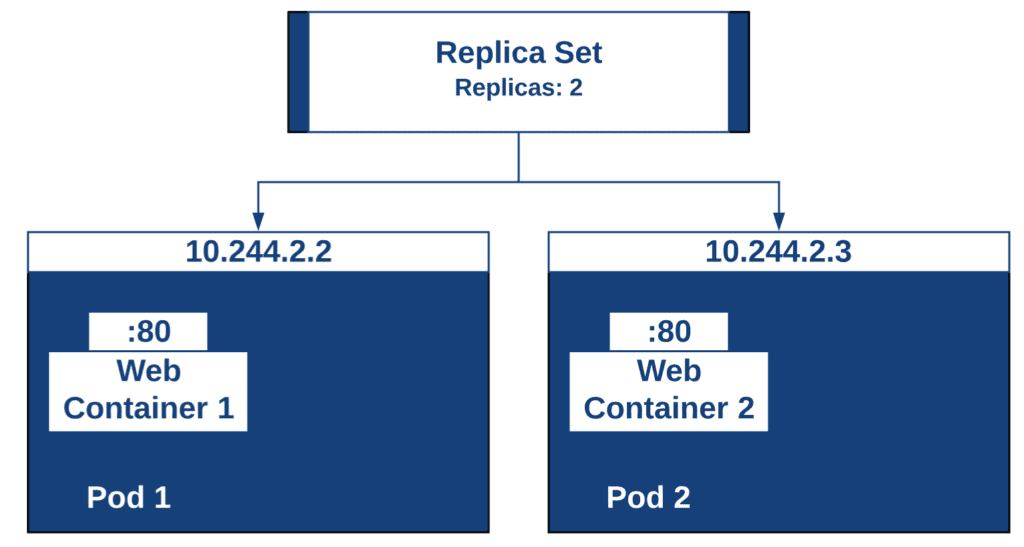
**Namespaces**

Namespaces are a way to create multiple virtual K8s clusters within a single cluster. Namespaces are normally used for wide-scale deployments where there are many users, teams, and projects.

****

**Replica Set**

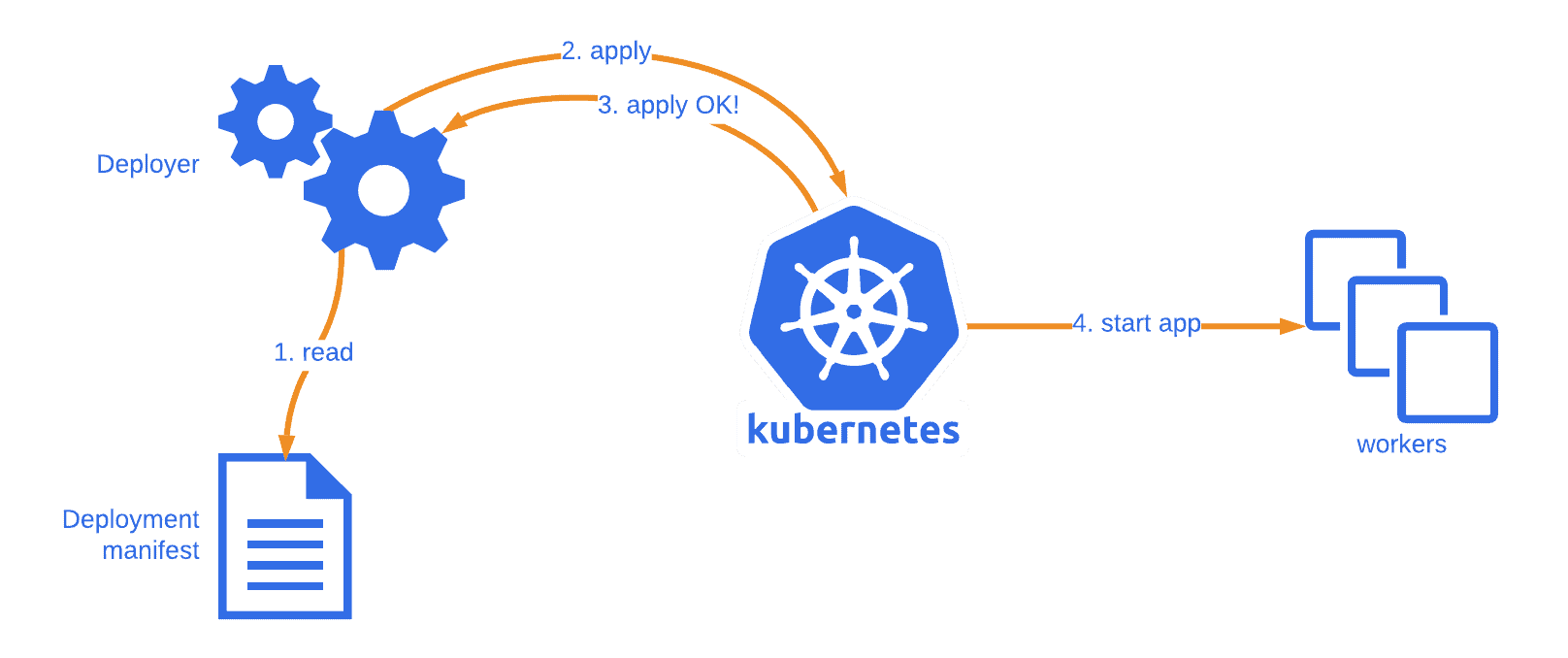
A Kubernetes replica set ensures that the specified number of pods in a replica set is running at all times.



A Replica Set allows you to define the number of pods that need to be running at all times and this number could be “1”. If a pod crashes, it will be recreated to get back to the desired state. For this reason, replica sets are preferred over a naked pod because they provide some high availability.

**Deployments**

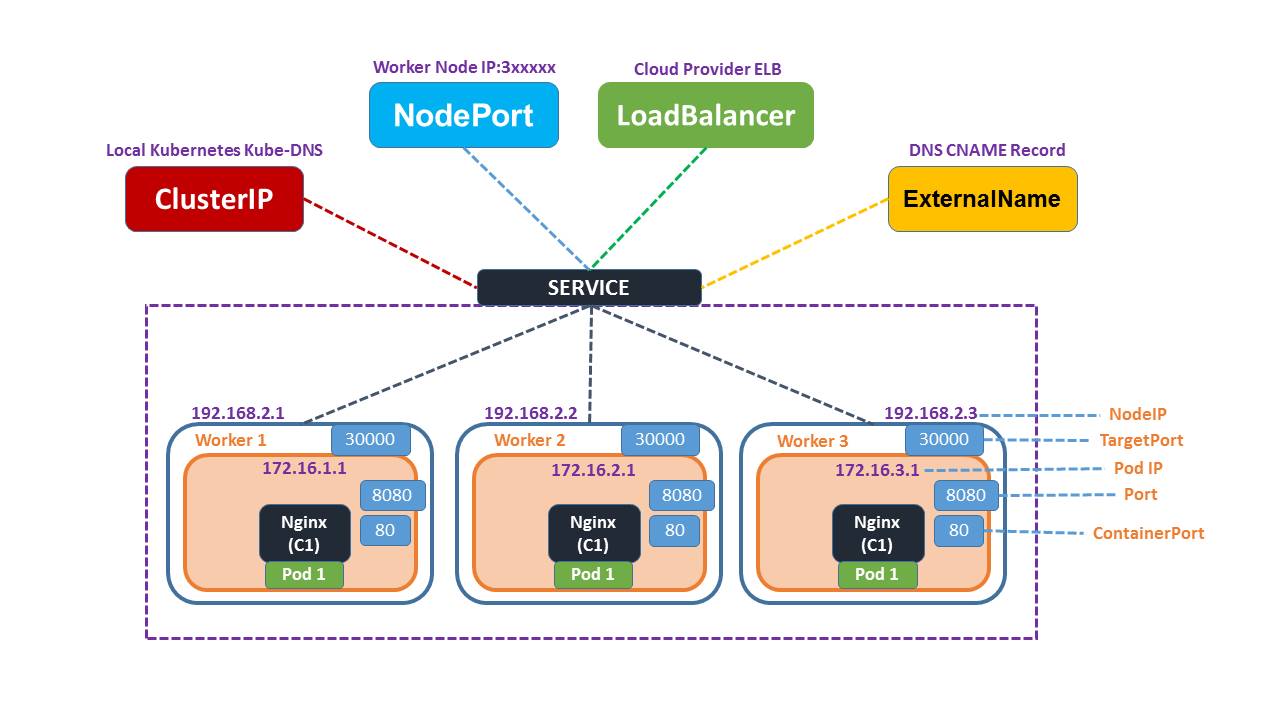
A way to define the desired state of pods or a replica set. Deployments are used to define HA policies for your containers by defining policies around how many of each container must be running at any one time.



In Kubernetes, most service-style applications use Deployments to run applications on Kubernetes. Using Deployments, you can describe how to run your application container as a Pod in Kubernetes and how many replicas of the application to run. Kubernetes will then take care of running as many replicas as specified.

**Services**

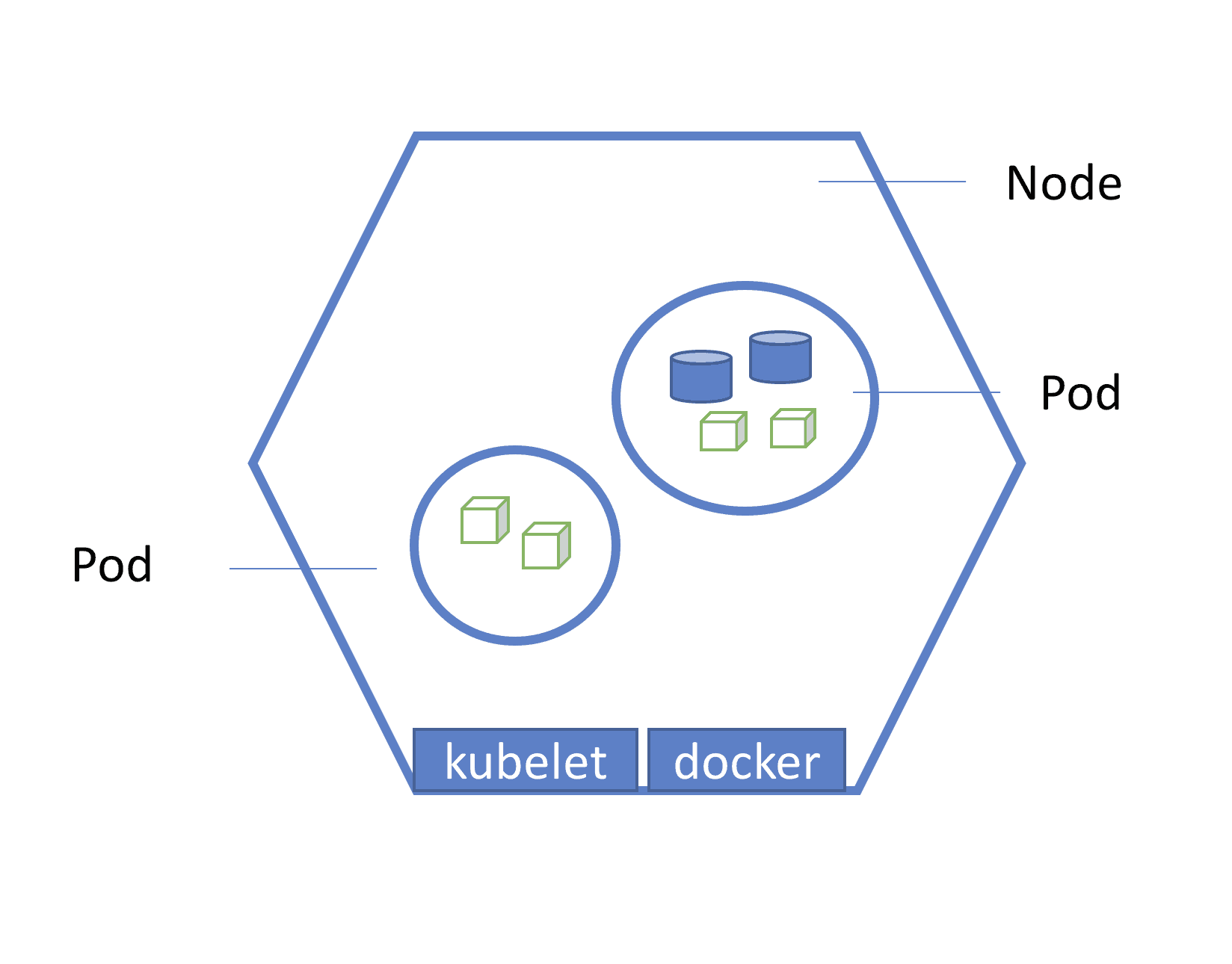
Coupling of a set of pods to a policy by which to access them. Services are used to expose containerized applications to origins from outside the cluster.

****

[Service is both](https://kubernetes.io/docs/concepts/services-networking/service/) an abstraction that defines a logical set of pods and a policy for accessing the pod set. There are 4 types of services viz. NodePort. ClusterIP, LoadBalancer and ExternalName.

**Nodes**

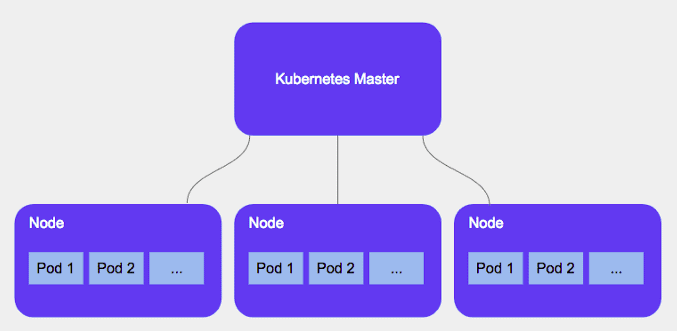
Kubernetes run your workload by placing containers into Pods to run on *Nodes*. A node may be a virtual or physical machine, depending on the cluster.

****

A Node is a worker machine and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master. A Node can have multiple pods. So, it’s a one-to-many relationship. The Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster.

**Cluster**

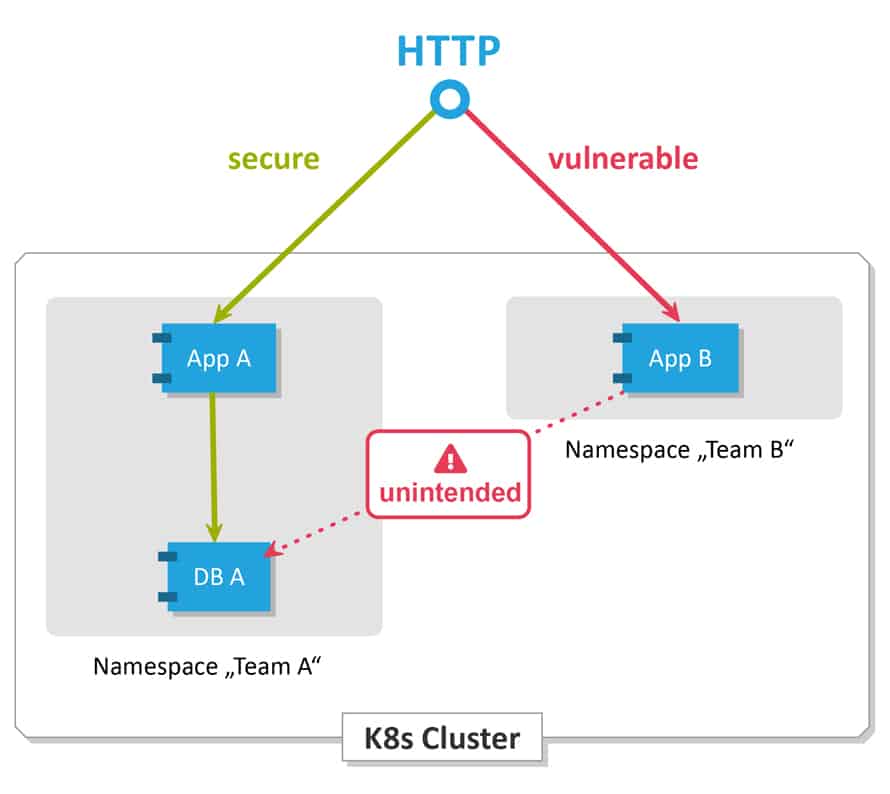
A Kubernetes Cluster is a group of nodes or machines running together. It consists of two types of servers, a Master Node, and a Worker Node. These servers can be Virtual machines (VM) or physical servers (Bare metal). Together, these servers form a Kubernetes cluster and are controlled by the services that make up the Control Plane.



**Kubernetes Security**

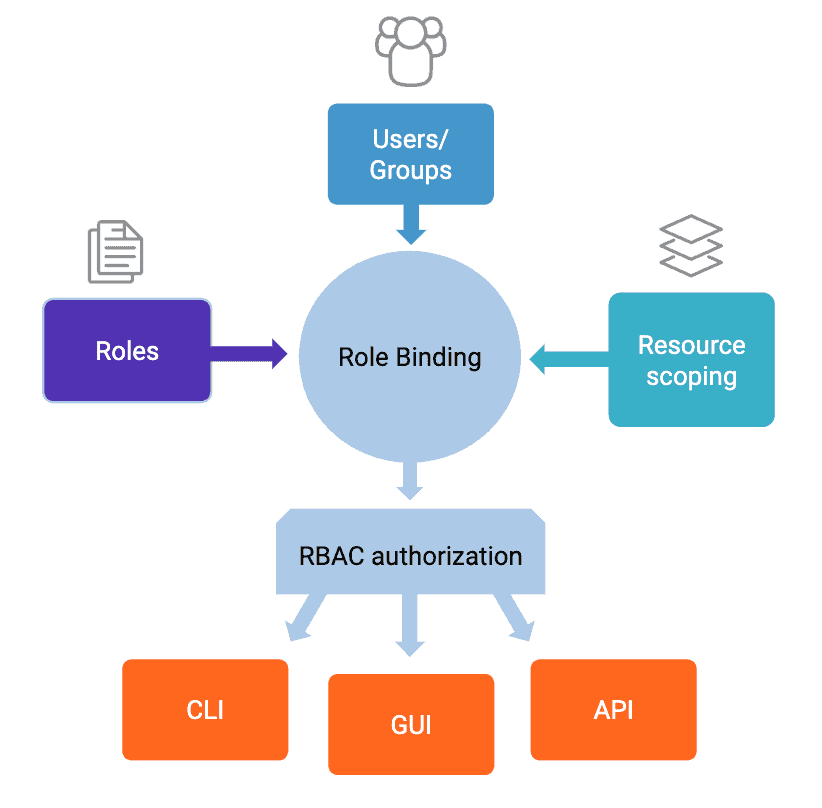
**Kubernetes security** is important throughout the container **lifecycle** due to the distributed, **dynamic nature** of a **Kubernetes cluster**. Kubernetes offers a rich set of controls that can be used to **effectively** secure clusters and their applications. Different security approaches are required for each of the **three phases** of an application lifecycle: build, deploy, and runtime.

**Network Policies**



One important configuration that demands attention from a security perspective is the **network policy** feature. These are **Kubernetes** assets that control the traffic between **pods**. Kubernetes network policy lets developers **secure** access to and from their applications. This is how we can **restrict** a user for access.

**RBAC – Role-Based Access Control**



It is a technique of **regulating access** to a computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of a **private user** to perform a **selected** task, like reading, creating, or modifying a file.