**KUBERNETES**

**What is Kubernetes?**

**Icon

Description automatically generated**

Kubernetes is a popular open-source platform for container orchestration that helps developers and operators to manage containerized applications. It provides several benefits, such as scalability, automation, high availability, and portability, making it easier to deploy, manage, and scale applications in a distributed environment.

Kubernetes is a powerful platform for managing containerized applications at scale. It provides a comprehensive set of tools and resources for deploying, scaling, and managing applications, and is highly extensible and customizable to meet the needs of any organization.

Kubernetes also provides a powerful API and command-line interface (CLI) for managing the cluster and its resources. Developers can use these tools to deploy, update, and scale applications, monitor resource usage and performance, and troubleshoot issues.

Kubernetes is highly extensible, with a large and growing ecosystem of plugins and extensions that can be used to add new features and functionality. This ecosystem includes tools for monitoring, logging, service discovery, load balancing, and more.

**History of Kubernetes:**

Kubernetes, also known as "K8s", is an open-source container orchestration system that was developed by Google in 2014. However, the origins of Kubernetes can be traced back to Google's internal container management system, Borg, which has been in use since the early 2000s.Google had been using container technology for a long time to improve the efficiency of its data centres. Containers are lightweight, portable, and scalable, which makes them ideal for running applications in a distributed system. However, managing many containers manually can be a complex and time-consuming task. To address this challenge, Google developed Borg, a container orchestration system that automates the deployment, scaling, and management of containers. Borg has been in use at Google for over a decade and has helped the company to manage billions of containers across thousands of servers.

In 2014, Google released a paper describing the principles and architecture of Borg, which caught the attention of the open-source community. This led to the creation of Kubernetes, an open-source project that was based on the principles of Borg and designed to provide the same level of container management automation to the broader community. The initial release of Kubernetes, version 0.1, was made in June 2014. It was primarily designed to run on the Google Cloud Platform, but it was also compatible with other cloud providers and on-premises data centres.

Kubernetes quickly gained popularity among developers and organizations that were looking to adopt container technology. In 2015, Google donated Kubernetes to the newly formed Cloud Native Computing Foundation (CNCF), a non-profit organization dedicated to promoting cloud-native computing.

The CNCF became the official steward of Kubernetes and helped to establish it as the de facto standard for container orchestration. The foundation also provided resources and support to the Kubernetes community, including funding for development, marketing, and events.

Over the next few years, Kubernetes continued to evolve and improve. New features were added to the platform, such as support for stateful applications, improved networking, and enhanced security. The ecosystem around Kubernetes also grew, with the development of new tools, plugins, and extensions that added additional functionality and made it easier to use.

In 2018, Kubernetes reached a major milestone with the release of version 1.0. This version marked the platform's maturity and stability, and it was declared "production-ready". Since then, Kubernetes has continued to grow in popularity and adoption, with thousands of companies and organizations using it to manage their containerized applications.

Today, Kubernetes is the most widely used container orchestration platform, with a large and active community of contributors and users. It has become an essential tool for modern cloud-native applications, providing a powerful and flexible way to manage containerized workloads at scale.

**Why Kubernetes?**



1.**Scalability**: Kubernetes makes it easy to scale applications up or down based on demand. You can easily add or remove resources to ensure that your application has the capacity it needs to handle traffic spikes or changes in demand. Kubernetes also provides features like auto-scaling, which automatically adjusts the number of replicas based on CPU utilization or other metrics.

2.**High Availability**: Kubernetes provides high availability by automatically monitoring and restarting failed containers. It ensures that your application is always available by automatically scheduling containers on healthy nodes and providing automatic failover in case of node failures.

3.**Portability**: Kubernetes is a platform-agnostic tool that can be deployed on any cloud or on-premises data centre. It provides a consistent environment for deploying and managing applications across different environments, making it easy to move workloads between different cloud providers or on-premises data centres.

4.**Resource Utilization**: Kubernetes ensures that your applications use resources efficiently by automatically scheduling containers based on available resources. It allows you to optimize the usage of resources like CPU and memory, reducing waste and saving costs.

5.**Automation:** Kubernetes provides a powerful set of automation tools that help to streamline the deployment and management of containerized applications. It automates tasks like container scheduling, networking, load balancing, and storage management, reducing the manual effort required to manage containers.

6.**Security**: Kubernetes provides a strong security model that ensures that your applications and data are secure. It provides features like network policies, container isolation, and secrets management to help protect against attacks and data breaches.

7.**Ecosystem**: Kubernetes has a large and active ecosystem of developers and users that provides a wide range of tools and extensions. These include monitoring tools, logging tools, service mesh, and many others, which help to extend the functionality of Kubernetes and make it easier to use.

**Tools provided by Kubernetes:**

****

Kubernetes provides a set of tools and resources that allow developers to define, deploy, and manage containerized applications at scale. These tools include:

1.**Pods**: The smallest deployable units in Kubernetes. A pod is a group of one or more containers that share the same network namespace and can communicate with each other using localhost. Pods are scheduled onto nodes in a Kubernetes cluster, and the containers within a pod are always co-located and co-scheduled.

2.**Services**: A service is a logical grouping of pods that provides a stable network endpoint for accessing the application. Services can be exposed within the cluster or externally, allowing external clients to access the application.

3**.Replication Controllers/Replica Sets**: Replication controllers and replica sets ensure that a specified number of replicas of a pod are always running. They monitor the state of the pods and automatically create or delete replicas as necessary to maintain the desired state.

4.**Deployments:** Deployments provide declarative updates for pods and replica sets. They allow developers to define the desired state of the application and Kubernetes will automatically perform rolling updates to achieve that state.

5.**ConfigMaps/Secrets**: ConfigMaps and secrets allow developers to manage configuration data and sensitive information, such as passwords or API keys, separately from the application code.

6.**Stateful Sets**: Stateful Sets provide a way to manage stateful applications, such as databases, in Kubernetes. They ensure that each pod in the set has a unique identity and that stateful data is preserved across pod rescheduling and updates.

**Prerequisites:**

To deploy a web application to Kubernetes, you'll need the following prerequisites:

**Kubernetes cluster**: You need to have a Kubernetes cluster set up to deploy your application. This cluster can be set up on-premises or on a cloud provider like AWS, GCP, or Azure.

**Containerized application**: Your application should be containerized using a tool like Docker. This means your application code is packaged into a container image, along with all its dependencies.

**Container Registry**: You'll need a container registry to store your container images. You can use a public registry like Docker Hub or a private registry like Google Container Registry (GCR) or Amazon Elastic Container Registry (ECR).

**Kubernetes YAML files**: You need to create YAML files that define your application deployment, service, and any other required resources. These files describe how Kubernetes should deploy and manage your application.

**Knowledge of Kubernetes**: You need to have a good understanding of Kubernetes concepts and how to use kubectl, the Kubernetes command-line tool, to interact with your cluster. This includes understanding how to create and manage pods, deployments, services, and other resources.

**Ingress Controller**: You may need to install and configure an ingress controller, which is a Kubernetes resource that manages external access to your cluster. This allows you to expose your application to the internet and route traffic to the correct service.

**Monitoring and Logging**: You should have a system in place for monitoring your application and logging its output. Kubernetes provides built-in monitoring and logging solutions like Prometheus and Elasticsearch, or you can use third-party tools.

**Installation of Kubernetes on windows**:

**Install Docker**: Kubernetes relies on Docker to run containerized applications, so you'll need to install Docker first.

**Install Minikube**: You can download and install Minikube from the official website: <https://minikube.sigs.k8s.io/docs/>

**Install kubectl**: Once you have installed Minikube, you need to install kubectl, the Kubernetes command-line tool. You can download and install kubectl from the official website: <https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/>

**Start Minikube**: Use the following command to start Minikube:

minikube start

**Deploy a Pod**: You can deploy a simple nginx web server pod using the following YAML file:

apiVersion:v1

kind:Pod

metadata:

name:nginx-pod

spec:

containers:

-name:nginx

image:nginx

ports:

-containerPort:80

Save the YAML file as nginx-pod. yaml and then deploy it using the following command:

kubectl apply -f nginx-pod.yaml

This will deploy the nginx pod to your Kubernetes cluster running in Minikube.

**Access the Pod**: Once the pod is deployed, you can access it using the following command:

minikube service nginx-pod --url

This will output the URL for the pod's service, which you can open in your web browser to access the nginx web server.

**Commands Used for Kubernetes:**

Kubernetes provides a set of command-line tools that can be used to interact with the platform. Some of the most used commands include:

**kubectl create**: Used to create Kubernetes resources, such as pods, services, and deployments.

**kubectl get**: Used to retrieve information about Kubernetes resources.

**kubectl apply**: Used to apply changes to Kubernetes resources.

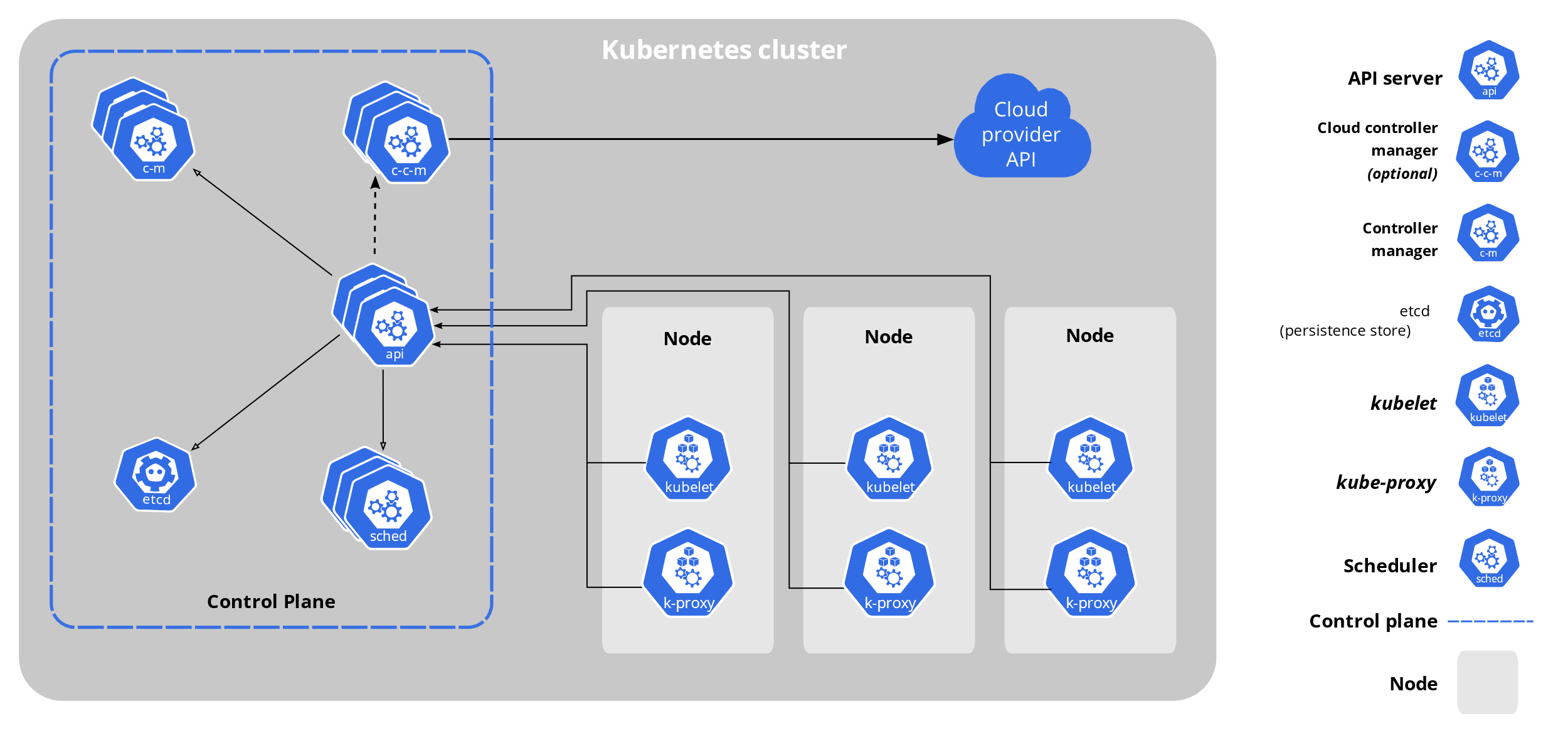
**kubectl delete**: Used to delete Kubernetes resources.

**kubectl logs**: Used to retrieve logs from Kubernetes pods.

**kubectl describe**: Used to retrieve detailed information about Kubernetes resources.

**kubectl exec**: Used to execute commands inside a Kubernetes pod.

**Kubernetes cluster :**

When you deploy Kubernetes, you get a cluster. 

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

**Components of Kubernetes Cluster:**

**Control Plane Components**

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for

example, starting up a new pod when a deployment's replicas field is unsatisfied).Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Creating Highly Available clusters with kubeadm for an example control plane setup that runs across multiple machines.

**kube-API server:** The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

The main implementation of a Kubernetes API server is kube-apiserver. kube-apiserver 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.

**Etcd** :Consistent and highly available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a back-up plan for those data.

**kube-scheduler**: Control plane component that watches for newly created Pods with no assigned node and selects a node for them to run on.Factors considered 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.

**kube-controller-manager:** Control plane component that runs controller processes. 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.

Some types of these controllers are:

**Node controller**: Responsible for noticing and responding when nodes go down.

**Job controller**: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

**Endpoint Slice controller**: Populates Endpoint Slice objects (to provide a link between Services and Pods).

**Service Account controller**: Create default Service Accounts for new namespaces.

**cloud-controller-manager:** A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API and separates out the components that interact with that cloud platform from components that only interact with your cluster.

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.

As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

The following controllers can have cloud provider dependencies:

**Node controller**: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding

**Route controller**: For setting up routes in the underlying cloud infrastructure

**Service controller**: For creating, updating and deleting cloud provider load balancers.

**Node Components :** Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

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

**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. kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

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

**Addons** :Addons use Kubernetes resources (Daemon Set, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, name spaced resources for addons belong within the kube-system namespace. Selected addons are described below; for an extended list of available addons, please see Addons.

**DNS** : While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.

**Web UI (Dashboard)**:Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

**Container Resource Monitoring** :Container Resource Monitoring records generic time-series metrics about containers in a central database and provides a UI for browsing that data.

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