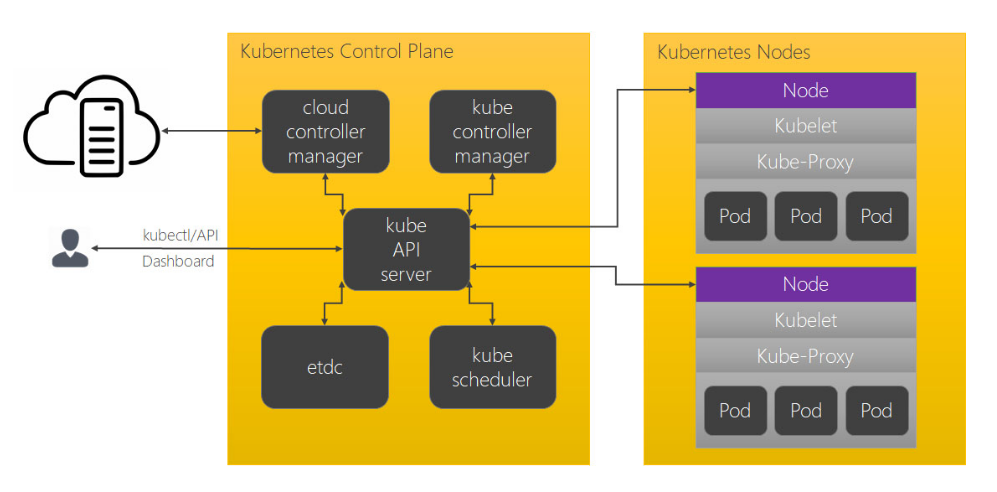
**Lab no -1\_rg\_kbnew**

**What is Kubernetes?**

**What is Kubernetes?**

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google’s experience running production workloads at scale with best-of-breed ideas and practices from the community.

****

### Kubernetes Components:

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.

### 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.

#### kube-apiserver:

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 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.

#### 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. 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, joins Services & Pods).
* **Service Account & Token Controllers**: Create default accounts and API access tokens for new namespaces.

#### cloud-controller-manager:

cloud-controller-manager runs controllers that interact with the underlying cloud providers. cloud-controller-manager runs cloud-provider-specific controller loops only and allows the cloud vendor’s code and the Kubernetes code to evolve independently of each other. The following controllers 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
* **Volume Controller**: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

#### 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 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.

**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 several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

### Addons:

Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

#### 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.

### Kubernetes Tools:

* **Kubectl**: kubectl is the command line tool for Kubernetes. It controls the Kubernetes cluster manager.
* **Kubeadm**: kubeadm is the command line tool for easily provisioning a secure Kubernetes cluster on top of physical or cloud servers or virtual machines (currently in alpha).
* **Kubefed**: kubefed is the command line tool to help you administrate your federated clusters.
* **Minikube**: minikube is a tool that makes it easy to run a single-node Kubernetes cluster locally on your workstation for development and testing purposes.
* **Dashboard**: Dashboard, the web-based user interface of Kubernetes, allows you to deploy containerized applications to a Kubernetes cluster, troubleshoot them, and manage the cluster and its resources itself.
* **Helm**: Kubernetes Helm is a tool for managing packages of pre-configured Kubernetes resources, aka Kubernetes charts.
* **Kompose**: Kompose is a tool to help Docker Compose users move to Kubernetes.
* **Kops**: Kops helps you create, destroy, upgrade, and maintain production-grade, highly available Kubernetes clusters in public cloud like AWS.
* **Prometheus**: Prometheus monitoring has fast become the go-to tool for Kubernetes monitoring tool.