**Kubernetes**

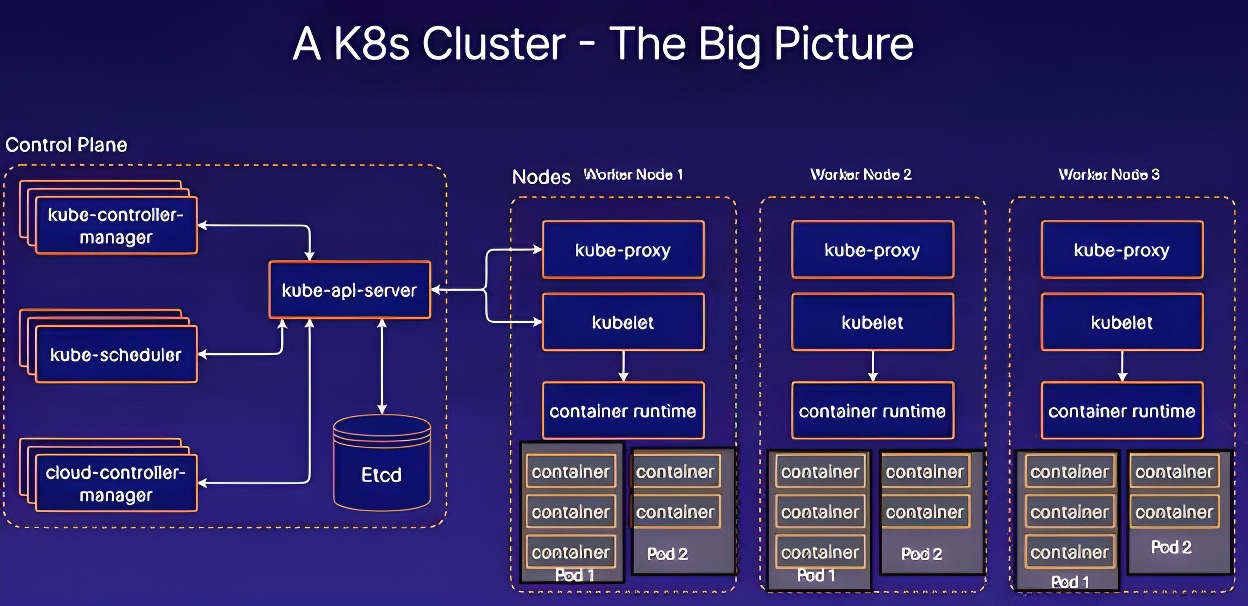
**Overview:**

Kubernetes (often abbreviated as K8s) is an open-source container orchestration platform designed to automate the deployment, scaling, management, and networking of containerized applications. It helps manage containerized applications across clusters of machines, ensuring high availability, scalability, and efficiency.

**Key Benefits of Kubernetes**

1. **Auto-Scaling**: Kubernetes can automatically scale up or scale down application instances based on demand.
2. **Self-Healing**: If a container fails or crashes, Kubernetes automatically restarts or replaces it.
3. **High Availability**: Ensures that your application is always available, even if a node or pod fails.
4. **Cost Efficiency**: Optimizes resource usage, cluster autoscaling to reduce infrastructure costs. Kubernetes can automatically scale the cluster (add or remove nodes) based on the workloads, which optimizes cost by ensuring that resources are only provisioned when needed.
5. **Load Balancing**: Kubernetes provides load balancing across application pods. It automatically distributes traffic to healthy pods within a service, ensuring even distribution and preventing overloading of individual containers.
6. **Cross-Platform Support**: Kubernetes is platform-agnostic, meaning it can run on any infrastructure—public clouds (AWS, Azure, Google Cloud), private clouds, or on-premise data centers.
7. **Security:** Kubernetes ensures security through features like Role-Based Access Control (RBAC), secrets management, network policies, and pod security policies, protecting applications from unauthorized access and vulnerabilities.
8. **Rolling Updates**: Kubernetes performs rolling updates to ensure zero downtime during deployments by gradually replacing old pods with new ones.
9. **Monitoring and Logging**: Integrated tools for real-time monitoring and centralized logging. Kubernetes integrates with various monitoring tools like Prometheus and Grafana to provide real-time visibility into application performance, resource usage, and cluster health.

**Kubernetes Architecture**



Kubernetes architecture is composed of two main components:

1. Control Plane (Master Node)
2. Data Plane (Worker Node)

**Control Plane (Master Node)**

1. API Server
2. etcd
3. Scheduler
4. Controller Manager
5. Cloud Controller Manager

**Data Plane (Worker Node)**

1. Kube-proxy
2. Kubelet
3. Container runtime

**Data Plane (Worker Node):**

These are the nodes where the actual work happens. Each Node can have multiple pods and pods have containers running inside them. The worker nodes are responsible for running the actual workloads (containers) in the form of pods. They host the necessary components to run and manage containers. There are 3 processes in every worker Node that are used to Schedule and manage those pods.

1. **Kube-proxy:**

* The kube-proxy is responsible for network routing and load balancing between different services and pods across nodes.
* It maintains network rules for pod communication within the cluster. It can use IP tables or IPVS (IP Virtual Server) to direct traffic between services and pods, handling both internal cluster communication and external communication with the services.

1. **Kubelet:**

Kubelet is the node agent responsible for managing pod execution and ensuring that containers are running as expected. It interacts with the API server to receive instructions and reports back the status of the node. If a pod is not running as expected, Kubelet informs the control plane.

1. **Container Runtime:**

* The container runtime is responsible for running containers. Popular container runtimes include Docker, containerd, and CRI-O.
* The container runtime handles container lifecycle management (pulling images, starting and stopping containers) and communicates with the kubelet to ensure containers are running within the desired state.

**Control Plane (Master Node):** The Control Plane is responsible for the overall management of the Kubernetes cluster. It makes global decisions about the cluster, such as scheduling, scaling, and managing the state of the applications. The control plane consists of several components:

1. **API Server (kube-apiserver):**

* The API Server is the central component of the control plane and serves as the entry point for all the interactions within the Kubernetes cluster.
* It exposes the Kubernetes API, handles HTTP requests, and processes them.
* It authenticates and authorizes API requests, validates and processes API objects (such as Pods, Services, Deployments), and communicates with other components.
* All Kubernetes components communicate with each other through the API Server using RESTful API calls.

1. **Scheduler (kube-scheduler):**

* Responsible for assigning pods to worker nodes based on resource availability and constraints like CPU, memory, affinity rules, etc. It makes decisions based on the current state of the cluster.
* When API Server receives a request for Scheduling Pods then the request is passed on to the Scheduler. It intelligently decides on which node to schedule the pod for better efficiency of the cluster.

So who decides the information is **API server** but who acts on the information that is the **scheduler.**

1. **Etcd:**

A distributed key-value store used to store all cluster data, including the current and desired states of the Kubernetes objects (pods, services, etc). It acts as the source of truth for the cluster.

1. **Controller Manager (kube-controller-manager):**

The Controller Manager runs various controllers that regulate the state of the cluster. Controllers are responsible for ensuring that the desired state of the system matches the actual state.  
Examples of different Controllers are:

* **Node Controller:** Monitors the state of worker nodes and takes action if a node goes down.
* **Replication Controller:** Ensures that the desired number of pod replicas are always running.
* **Endpoint Controller**: Manages the Service endpoints.
* **Service Account and Token Controller:** Manages service accounts and tokens for API access.

1. **Cloud Controller Manager (CCM):**

* CCM separates cloud-specific logic from Kubernetes core, allowing Kubernetes to integrate with cloud provider APIs for resource management (e.g., load balancers, storage volumes, node management), without being tightly coupled to any particular cloud platform.
* It handles tasks such as node lifecycle, load balancer provisioning, and autoscaling.
* CCM allows Kubernetes to work across different cloud environments e.g.,
* **AWS** - Amazon Elastic Kubernetes Service (EKS),
* **Azure** - Azure Kubernetes Service (AKS),
* **GCP** - Google Kubernetes Engine (GKE)
* **on-prem environments**.

**Basic Terminology:**

**Pods:** A Pod is the smallest and simplest unit in Kubernetes, which can run one or more containers. Each pod gets its own IP address and can communicate with other pods inside the cluster.

**Service**: A service is an abstraction that defines a logical set of pods and a policy to access them, often used to expose an application to other applications or external clients. Services in Kubernetes include:

* **ClusterIP**: Exposes the service on an internal IP in the cluster.
* **NodePort**: Exposes the service on each node’s IP at a static port.
* **LoadBalancer**: Provisions an external load balancer to expose the service to the internet.

**Volumes**: Kubernetes provides different volume types for storing persistent data, including:

* **emptyDir**: Temporary storage that exists as long as the pod is running.
* **Persistent Volumes (PVs)**: Allows the use of storage beyond the pod lifecycle, often linked with cloud storage systems or local volumes.

**Namespaces**: Namespaces are used to divide cluster resources and allow multiple teams or projects to operate within the same Kubernetes cluster while maintaining resource isolation.