**Kubernetes (K8s) - Introduction & Architecture**

**What is Kubernetes (K8s)?**

Kubernetes (K8s) is an open-source platform for automating the deployment, scaling, and management of containerized applications. It provides a framework to run distributed systems resiliently, handling aspects like scaling, failover, and deployment patterns.

Real-Time Example:

* Imagine you're running a web application with a front-end and a back-end service. Kubernetes allows you to deploy and manage these components as containers across multiple machines while ensuring high availability and scaling.

**K8s Architecture**

Kubernetes follows a master-slave architecture where:

* **Master Node**: Manages the Kubernetes cluster and is responsible for the overall orchestration. It runs the following components:
  + **API Server**: The front-end of the Kubernetes control plane. It exposes the Kubernetes API and acts as the communication hub between different components.
  + **Controller Manager**: Manages controllers that ensure the desired state of the cluster (e.g., managing replication, ensuring nodes are running, etc.).
  + **Scheduler**: Assigns tasks (pods) to the appropriate worker nodes based on available resources.
  + **etcd**: A key-value store that holds the cluster state data.
* **Worker Nodes (Minions)**: The machines where the application containers are actually run. They contain the following components:
  + **Kubelet**: Ensures that containers are running in the Pod.
  + **Kube Proxy**: Manages network rules for pod communication.
  + **Container Runtime**: Runs containers (e.g., Docker).

Real-Time Example:

* In a production environment, if you're running a highly available web service, the **Master Node** coordinates how your application should run and how replicas of your application should be distributed across the **Worker Nodes**.

**Installation of Minikube**

Minikube is a tool that makes it easy to run Kubernetes clusters locally. Here's how to install it:

1. **Install Minikube**:
   * **On Windows/Mac/Linux**: Follow the installation instructions from the official Minikube website or use a package manager.
   * Command: minikube start
2. **Start the Cluster**:
   * minikube start will start a Kubernetes cluster with a single node.
3. **Check Cluster Status**:
   * kubectl cluster-info will show the cluster information.
4. **Accessing K8s Dashboard**:
   * minikube dashboard opens the Kubernetes web dashboard.

Real-Time Example:

* Minikube is useful for developers who want to try Kubernetes locally without setting up a full-fledged cloud infrastructure. You can use it to experiment with deploying applications and learning Kubernetes concepts on your laptop.

**K8s Cluster**

A Kubernetes cluster consists of the **Master Node** and **Worker Nodes**. It is the foundation for managing containerized applications, where multiple nodes are connected to form a unified system.

Real-Time Example:

* In a cloud environment like AWS, you can run multiple worker nodes on different EC2 instances, with Kubernetes managing the scheduling and scaling of your containers automatically.

**Pods**

A **Pod** is the smallest deployable unit in Kubernetes. It can host one or more containers. Pods share the same network namespace, meaning containers within a pod can communicate with each other using localhost. They also share storage volumes.

Real-Time Example:

* A **Pod** can host a web server container and a logging agent container. The web server handles requests, and the logging agent collects logs and writes them to a shared volume.

**Commands to work with Pods:**

* kubectl get pods - List all pods.
* kubectl describe pod <pod-name> - Get detailed information about a specific pod.

**Replicas and Replication Controller**

Kubernetes can ensure that a specified number of replicas of a pod are always running. This is done through the **Replication Controller** or more modernly using **ReplicaSets**.

* **ReplicaSet** ensures that a specified number of pod replicas are running at any given time.
* If a pod fails, the ReplicaSet creates a new pod to replace it.

Real-Time Example:

* If you want to run a web application with 3 replicas, the ReplicaSet will ensure that there are always 3 running pods, even if one crashes or gets terminated.

**Deployment**

A **Deployment** in Kubernetes is a higher-level concept used to manage the deployment and scaling of ReplicaSets. It allows you to define the desired state of your application (e.g., number of replicas, image version) and Kubernetes will ensure it is maintained.

Real-Time Example:

* If you're updating your web application, you can use a **Deployment** to rollout the new version. Kubernetes will create new pods with the updated version and remove the old pods without downtime.

**Commands to work with Deployments:**

* kubectl create deployment <deployment-name> --image=<image-name> - Create a new deployment.
* kubectl get deployments - List all deployments.

**Services**

A **Service** in Kubernetes is an abstraction that exposes a set of pods as a network service. It enables communication between pods and ensures that pods can be accessed reliably, even if the pods themselves are restarted.

There are several types of services in Kubernetes:

* **ClusterIP**: Exposes the service on an internal IP address.
* **NodePort**: Exposes the service on each node's IP at a static port.
* **LoadBalancer**: Exposes the service externally via a load balancer.
* **ExternalName**: Maps the service to an external DNS name.

Real-Time Example:

* A **Service** is used to expose the backend of your application to the front-end pods, ensuring that requests are directed to the right pod, even if the pods are dynamically created or removed.

**Commands to work with Services:**

* kubectl expose pod <pod-name> --port=<port> - Expose a pod as a service.
* kubectl get services - List all services.

**Ingress**

An **Ingress** is a collection of rules that allow external HTTP/S traffic to reach services within a Kubernetes cluster. Ingress controllers manage the routing of traffic to the appropriate services based on these rules.

Real-Time Example:

* Suppose you have multiple applications (e.g., a blog and an e-commerce site) running in a Kubernetes cluster, and you want to expose both to the internet. You can define an **Ingress** to route traffic to /blog to the blog application and /shop to the e-commerce site.

**Commands to work with Ingress:**

* kubectl apply -f ingress.yaml - Apply an Ingress resource.
* kubectl get ingress - List all ingress resources.

**Real-Time DevOps Use Case Example**

In a DevOps scenario, let's imagine you're working on an e-commerce application. You want to deploy the app in a Kubernetes cluster.

1. **Containerize the Application**: The front-end and back-end are containerized using Docker.
2. **Define Kubernetes Deployment**: Define a **Deployment** for the front-end and back-end services, specifying the number of replicas and container images.
3. **Set Up Services**: Expose the back-end service internally using a **ClusterIP** and the front-end using a **NodePort** or **LoadBalancer** so external users can access it.
4. **Configure Ingress**: Use **Ingress** to route traffic to different services based on the URL path.
5. **Scaling**: As traffic grows, Kubernetes automatically scales the replicas of the services.

This allows continuous delivery and scaling without manual intervention, which is one of the key benefits of using Kubernetes in a DevOps pipeline.

**Conclusion**

Kubernetes is a powerful tool for managing containerized applications at scale. By abstracting the underlying infrastructure, it enables you to focus on deploying and managing applications rather than managing the underlying servers. Kubernetes is essential in modern DevOps workflows due to its automation, scalability, and flexibility in deploying applications.

The key concepts covered:

* **Kubernetes Architecture** (Master Node and Worker Nodes)
* **Pods** (Basic deployable unit)
* **ReplicaSets and Deployments** (Scaling and updating applications)
* **Services** (Networking and load balancing)
* **Ingress** (Routing external traffic)