Kubernetes notes

1. Cluster:

- A Kubernetes cluster is a collection of nodes that work together as a single computing resource.

- It consists of one or more master nodes and multiple worker nodes.

2. Node:

- Each node in the cluster is a physical or virtual machine.

- Nodes run the necessary Kubernetes components to manage containers and pods.

- Nodes provide the runtime environment for containers and execute the workloads.

3. Pod:

- A pod is the smallest deployable unit in Kubernetes.

- Pods encapsulate one or more containers that share the same network namespace, IP address, and storage volumes.

- Pods are scheduled onto nodes by the **Kubernetes scheduler.**

4. Container:

- Containers are the actual runtime instances of the applications or services within a pod.

- **Each pod can contain one or more containers.**

- Containers are isolated from each other and share the resources allocated to the pod.

**Cluster has nodes. Each node hosts multiple pods. Each pod can run one or more containers, and containers execute the actual application code or services.**

1)We send config request to api server . Request can be in yaml format or json format.

2)The api server is the only entry point to the cluster.

* deployment:blueprint for creating pods
* etcd:hold current state of any kubernetes component
* minikube installation comes with docker installed to run those container inside the clusters
* we are hosting minikube in our machine as docker container itself
* kubectl installed as dependency in minikube
* kubectl get node

**Services:**

When we create a Service in Kubernetes, we specify a set of pods that the Service will load balance traffic to.

The Service automatically discovers the IP addresses of those pods and manages routing traffic to them.

However, we as a user or administrator do not typically interact directly with the IP addresses of individual pods within the

Service. Instead, we interact with the Service itself, which provides a stable endpoint for accessing the pod.

In yaml file,if we write

Port:80

targetPort:8080

inside ports and inside specification section defined in yaml file

It means:

traffic received on port 80 of the Service will be forwarded to port 8080 on the pods.

**ConfigMap:**

If we have an application running in Kubernetes, and this application needs some configuration settings to work properly. These settings might include things like database connection strings, API keys.

Now, instead of hardcoding these settings directly into our application's code or Docker image, which can make it inflexible and hard to update, Kubernetes offers a solution called ConfigMap.

A ConfigMap is like a simple key-value store.

It holds our configuration data in an organized way.

So, instead of scattering our settings across different places, we can gather them all into one ConfigMap.

**Steps:**

1. define a ConfigMap in Kubernetes, specifying all the configuration settings we need.

2. we connect our ConfigMap to our pods. This tells Kubernetes to make those settings available to your application.

3. Inside your application, we can access these settings just like regular environment variables. So, if our app needs the database connection string, it can fetch it from the ConfigMap.

This makes our app more adaptable because as we can change these settings without rebuilding our containers.

It is way to keep all our app's configuration settings organized and separate from the code.

**Using minikube:**

These tools allow us to create a single-node Kubernetes cluster on our local machine where we can deploy and manage pods.

**kubectl cluster-info**:to verify that cluster is running

If Minikube is running, we can interact with Kubernetes using:

**kubectl**, the Kubernetes command-line tool.

**Basic Commands:**

1. kubectl get pods: This command lists all the pods running in the current namespace.

2. kubectl get nodes: This command lists all the nodes (machines) in the Kubernetes cluster.

3. kubectl get deployments: This command lists all the deployments (applications) running in the cluster.

4. kubectl create: This command creates a new resource in the cluster. You need to specify the resource type and configuration.

Example: `kubectl create deployment my-deployment --image=my-image`

5. kubectl delete: This command deletes a resource from the cluster.

kubectl delete [resource type] [resource name]

Example: `kubectl delete pod my-pod`

6. kubectl describe: This command provides detailed information about a specific resource.

Example: `kubectl describe pod my-pod`

7. kubectl logs: This command retrieves the logs of a pod.

Example: `kubectl logs my-pod`

8. kubectl exec: This command allows you to execute a command inside a running pod.

example: kubectl exec -it [podName] - - [Command]

9.kubectl apply -f [yaml-file]:applies configs in yaml file to the cluster.Can deploy new resource or update the existing.

example: kubectl exec -it [podName] - - [Command]

10)Port forwarding:

Kubectl port-forward command.

It instructs Kubernetes to forward traffic from the specified pod's port to the local port we've chosen

`kubectl port-forward my-app 80:8080`,

it forwards traffic from port `8080` on our local machine to port `80` on the pod named `my-app`.

When we move to `localhost:8080` in our web browser, we'll effectively be accessing port `80` on the pod named `my-app` through port forwarding.

If the application running inside the pod is configured to serve content on port `80`, we'll see the response from that application as if we were accessing it directly from port `80` on our local machine.

So, accessing `localhost:8080` after running the port forwarding command will allow us to interact with the application running inside the pod named `my-app`.