**\* \* \* \* \* Kubernetes \* \* \* \* \***

**09-28-k8s-25-Nov-24**

## Git Repo : https://github.com/ashokitschool/kubernetes\_manifest\_yml\_files.git

**=================================================================**

**\* \* \* \* \* \* \* \* \* \* \* \* Kubernetes (K8S) \* \* \* \* \* \* \* \* \* \* \* \***

**=================================================================**

=> It is free & open source s/w.

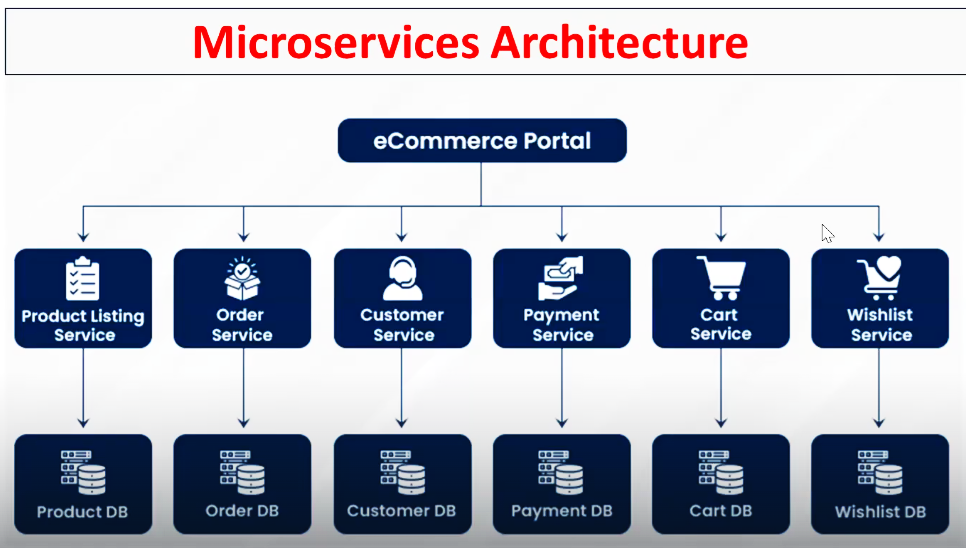
=> Google developed this k8s using "GO" programming language.

=> K8S provides Orchestration (Management) platform.

=> Managing multiple containers is not the easy job that’s why people are going with the orchestration job .

=> K8S is used to manage containers

(create/stop/start/delete/scale-up/scale-down)



**===============**

**Advantages**

**===============**

**1) Container Orchestration :** Manages containers.

**2Self Healing :** If any pod got crashed then it will replaced with new pod.

**3) Load Balancing :** Load will be distributed all containers which are up and running.

**4) ) Scalability / Auto Scaling :** Based on demand containers count will be increased or decreased.

* Horizontal : means increasing no. of containers .
* Verticle : means increasing power of the containers .

**=====================**

**Docker Vs Kubernetes**

**=====================**

Docker : Containerization platform

Note: Packaging our app code and dependencies as single unit for execution is called as Containerization.

Kubernetes : Orchestration platform

Note: Orchestation means managing the containers.

**========================**

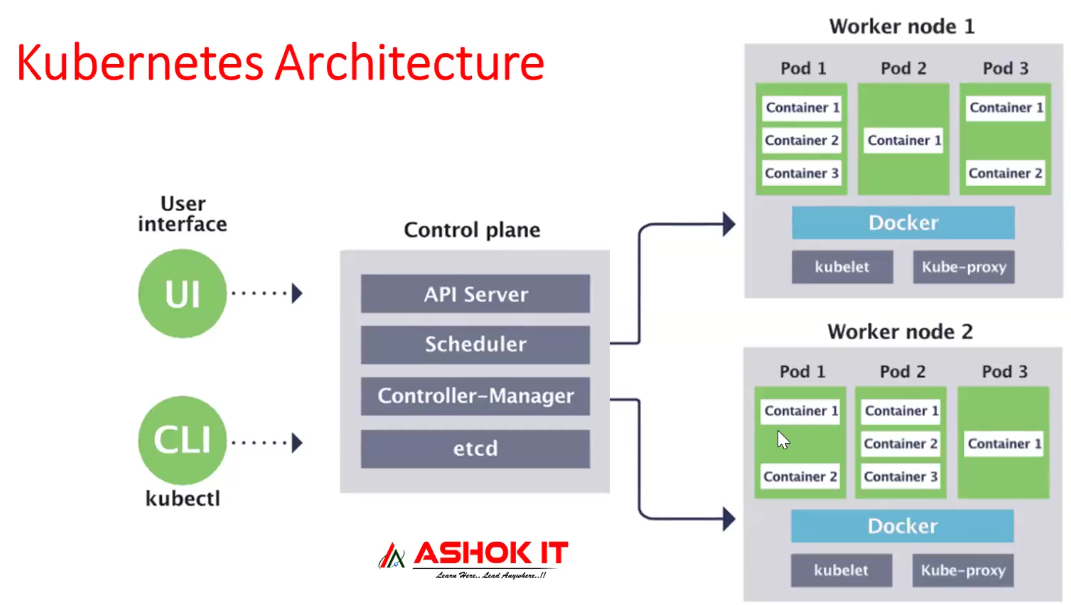
**Kubernetes Architecture**

**========================**

=> K8S will follow cluster architecture.

=> Cluster means group of servers will be available.

=> In K8s Cluster we will have master node (control plane) and worker nodes.

****

**========================**

**K8S Cluster Components**

**========================**

1) Control Node (Master Node)

- API Server

- Schedular

- Controller Manager

- ETCD

2) Worker Nodes

- Kubelet

- Kube Proxy

- Docker Runtime

- POD

- Container

* To deploy our application using k8s then we need to communicate with control node.
* We will use KUBECTL (CLI) to communicate with control plane.
* "API Server" will recieve the request given by "kubectl" and it will store that request in "ETCD" with pending status.
* "ETCD" is an internal database of k8s cluster.
* "Schedular" will identify pending requests available in ETCD and it will identify worker node to schedule the task.
* Note: Schedular will identify worker node using kubelet.

**Kubelet**

* "Kubelet" is called as Node Agent. It will maintain all the worker node information.
* "Kube Proxy" will provide network for the cluster communication.

**Controller Manager**

* "Controller Manager" will verify all the taks are working as expected or not.
* In Worker Node, "Docker Engine" will be available to run docker container.

**POD**

* In K8s, container will be created inside POD.
* POD is a smallest building block that we can create in k8s cluster to run our applications.
* Note: In K8s, everything will be represented as POD only.
* Pods are used to deploy applications and manage their lifecycle within a Kubernetes environment.

**==================**

**K8S Cluster Setup**

**==================**

* We can setup k8s cluster in multiple ways

1) **Self Managed k8S cluster** (We need to setup everything)

a) Mini Kube => Sinle Node Cluster => Only for beginners practice

b) Kubeadm => Multi Node Cluster (HA)

2) **Provider Managed K8S cluster** (ready made)

**a) AWS EKS**

**b) Azure AKS**

**c) GCP GKE**

**d) IBM IKE**

Note: Provider will give ready made cluster for us.

#### Note: Provider Managed Clusters are chargable #####

**===================================================**

**Assignment : Setup MiniKube software**

**===================================================**

Linux VM : https://github.com/ashokitschool/DevOps-Documents/blob/main/13\_MiniKube\_Setup.md

Windows : Download install docker desktop + download and install minkube

https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download

**Docker Desktop + minikube setup**

* Download docker desktop and enable all the setting from the youtube .
* While installing docker desktop enable both the ticks and install docker check is docker is running or not .
* Open powershell and run - docker run hello-world
* Install latest minikube for windows and go to the C drive/program files/kubernetes/minikube copy path and set environment varibles .
* Then create a new folder in D drive and enter and open cmd .
* Minikube status if minikube not available then enter below command
* Command - minikube start --driver=docker

**=====================**

**Kubernetes Resources**

**=====================**

1) PODS

2) Services

- ClusterIP

- NodePort

- LoadBalancer

3) Namespaces

4) ReplicationController (RC) & ReplicaSet (RS)

5) Deployment

6) DaemonSet

7) StatefulSet

8) PV & PVC

9) IngresController

10) HPA

11) HelmCharts

12) Metric Server

13) ConfigMap & Secret

14) Liveness & Readyness Probes

15) Taints & Tolerations

**==============**

**What is POD ?**

**==============**

=> POD is a smallest building block in the k8s cluster.

=> In kubernetes everything will be represent as a POD .

=> Applications will be deployed as pods only in the k8s cluster.

=> We can create multiple pods for one single application.

= > POD is used to run the container .

=> To create a POD we will use manifest yml file.

=> In POD Manifest YML we will configure Docker Image.

=> If any POD is damanged then kubernetes will replace that with new POD (self healing).

=> If application running in multiple pods, then k8s will distribute load to all running pods (Load Balancing).

=> PODS count can be increased and decreased based on that load. K8s will take care of scalability.

=============

K8S Services

=============

=> K8S service is used to expose PODS.

=> We have 3 types of services in K8S

1) ClusterIP

2) NodePort

3) LoadBalancer

====================

What is ClusterIP ?

====================

=> When we deploy app in k8s, then pods will be created and every pod will have IP address.

=> POD is a short lived object.

=> When POD is crashed/damanged k8s will replace that with new POD.

=> When POD is re-created then new IP will be assigned.

Note: It is not recommended to access PODS using POD IP.

=> "Cluster IP" service is used to link all the PODS to single IP.

Note: Cluster IP service provides static IP to access PODS.

=> Cluster IP service we can access with in the cluster only.

============================

What is NodePort service ?

============================

=> NodePort service is used to expose our pods outside of the cluster.

=> Using NodePort service we can access our application pods with Node Public IP address.

Note: when we use node public ip to access our PODS then all requests will go to single node only then burden will be increased on that node.

=> If we want to distribute the load to all pods which are running in all worker nodes then we have to expose our PODS using "LoadBalancer" service.

=======================

What is Load Balancer

======================

=> It is used to expose our PODS outside of the cluster using Load Balancer.

=> When we access "Load Balancer" url, it will distribute the load to all pods which are running in all worker nodes.

=============================

How to create PODS in k8s ?

=============================

=> To create pods we will use YML (manfiest file).

---

apiVersion: <version-number>

kind: <k8s-resource-type>

metadata: <name>

spec: <container-info>

...

# Execute k8s manifest yml

kubectl apply -f <manifest-yml>

=====================================================================

Assignment : Deploy Java web app in k8s cluster & Access in browser

=====================================================================

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

...

# Check PODS available

kubectl get pods

# Exceute POD manifest YML

kubectl apply -f <yml>

# describe POD

kubectl describe pod <pod-name>

# get pod logs

kubectl logs <pod-name>

===========================

K8S Service Manifest YML

===========================

=> Using service we can expose PODS outside of cluster

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30070

...

# check k8s services

kubectl get svc

# execute service yml

kubectl apply -f <service-yml>

Note: Once service got created we can access our applicaiton in browser

# get application tunnel url

minikube service javawebappsvc

==========================

What is Nodeport number ?

==========================

=> If we don't specify node-port in service manifest yml then k8s will assign random node port number for the service in the range of 30,000 to 32,767.

================

K8S Namespaces

================

=> Namespaces are used to group k8s resources

database-pods ====> create under one namespace

backend-app-pods ====> create under one namespace

frontend-app-pods ====> create under one namespace

=> We can create multiple namespaces in k8s cluster

Ex : ashokit-a1-ns, ashokit-a2-ns

=> Each namespace is isolated with another namespace

# display all namespaces available

kubectl get ns

# get pods of default namespace

kubectl get pods

# get the pods available in kube-system namespace

$ kubectl get pods -n kube-system

=> in k8s, We can create namespace in 2 ways

1) using kubectl create ns command

2) using manifest yml

## Approach-1 :

$ kubectl create ns ashokit-ns

## Approach-2 :

---

apiVersion: v1

kind: Namespace

metadata:

name: ashokit-ns-2

...

$ kubectl apply -f <yml>

# delete namespace

$ kubectl delete ns <namespace-name>

Note: When we delete a namespace, all the resources belongs to that namespace gets deleted.

==========================

Namespace + Pod + Service

==========================

---

apiVersion: v1

kind: Namespace

metadata:

name: ashokit-ns

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

namespace: ashokit-ns

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

namespace: ashokit-ns

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30070

...

$ kubectl apply -f <yml>

$ kubectl get ns

$ kubectl get pods -n ashokit-ns

$ kubectl get service -n ashokit-ns

$ kubectl get all -n ashokitns

$ minikube service javawebappsvc -n ashokit-ns

=========================

1) What is Orchestration

2) K8S Introduction

3) K8S Advantages

4) K8S Architecture

5) K8S Architecture components

6) K8S Cluster Setup

7) K8S Resources

8) What is POD

9) What is Service (ClusterIP, NodePort & LBR)

10) What is Namespace

====================================================================

=> If we create POD directley then k8s will not manage POD life cycle.

=> If POD is crashed/deleted then POD will not get re created.

=> If we want to get the benefit of k8s self-healing/auto-scaling/load-balancing then we should create POD using k8s resources.

==============

K8S Resources

==============

=> K8S resources are used to manage POD life cycle.

1) ReplicationController (RC) (Outdated)

2) ReplicaSet (RS)

3) Deployment

4) DaemonSet

5) StatefulSet

===========

ReplicaSet

===========

=> It is one of the k8s resource which is used to create & manage pods.

=> ReplicaSet will take care of POD life cycle.

Note: When POD is damaged/crashed/deleted then ReplicaSet will create new POD.

=> Always It will maintain given no.of pods count available for our application.

Ex=> replicas: 2

=> With this approach we can achieve high availability for our application.

=> By using RS, we can scale up and scale down our PODS count.

---

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: javawebrs

spec:

replicas: 2

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebpod

labels:

app: javawebapp

spec:

containers:

- name: javawebcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

namespace: ashokit-ns

spec:

type: LoadBalancer

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

$ kubectl get all

$ kubectl apply -f <yml>

$ kubectl get pods

$ kubectl get rs

$ kubectl delete pod <pod-name>

$ kubectl get pods

$ kubectl scale rs javawebrs --replicas 3

Note: When we execute above command replicaset will check how many pods are currently running based on that it will decide scale up or scale down.

Note: If we want to delete the pods then we have to delete the resource which created those pods.

$ kubectl delete rs javawebrs

Note: In ReplicaSet, scale up & scale down is manual process.

================================

What is diff between RC & RS ?

================================

=> RC will support only one label as selector to identify pods

Ex :

selector:

app: javawebapp

=> RS will support multiple Match Labels as selectors to identify pods.

Ex :

selector:

matchLabels:

app: javawebapp

===============

K8S Deployment

===============

=> It is one of the k8s resource/component.

=> It is the most recommended approach to deploy our apps in k8s cluster.

=> Deployment will manage pods life cycle.

=> We have below advantages with Deployment

1) Zero downtime

2) Auto Scaling

3) Rolling Update & Rollback

=> We have below strategies for deployment

1) ReCreate

2) Rolling Update

=> ReCreate means it will delete all existing pods and will create new pods.

Note: When we use ReCreate starategy we will be having application downtime.

=> RollingUpdate means it will delete old pod and creates new pod one by one.

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: javawebdeploy

spec:

replicas: 2

strategy:

type: ReCreate

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebpod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

----

apiVersion: v1

kind: Service

metadata:

name: javawebsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30070

...

==============

Auto Scaling

==============

=> It is the process of increasing or decreasing infrastructure resources based on demand.

=> Auto Scaling we can do in 2 ways

1) Horizontal Scaling

2) Vertical Scaling

=> Horizontal Scaling means increasing/decreasing number of "instances/servers/pods".

=> Vertical Scaling means increasing/decreasing capacity of the system

(ex: CPU, RAM etc.

=================================

Horizontal POD Autoscalar (HPA)

=================================

=> HPA is used to scale up/scale down no.of pod replicas based on the demand.

=> To scale up/scale down the PODS, HPA needs metrics from cluster nodes.

Note: To get metrics from the nodes we will use "Metrics Server". We need to deploy Metrics server POD in all worker nodes.

=> Metrics Server is an application that collect metrics from objects such as PODS and Nodes according to the state of CPU, RAM and keep them in time.

=> HPA will interact with Metrics Server to get worker node metrics and based on that information HPA will take decision for scale up or scale down.

## Step-1 : Install Metrics API

## Step-2 : Deploy Sample Application as PODS

## Step-3 : Create Service

## Step-4 : Create HPA

## Step-5 : Increase the Load

## Step-6 : Observe HPA behaviour and HPA events

@@ Reference Video :: https://youtu.be/c-tsJrcB50I?si=93rv1P74TZX5F7ZC

@ Metrics API Git Repo : https://github.com/ashokitschool/k8s\_metrics\_server.git

@ HPA Git Repo : https://github.com/ashokitschool/kubernetes\_manifest\_yml\_files.git

================

AWS EKS

================

=> EKS stands for Elastic Kubernetes Service.

=> EKS is fully managed service in AWS cloud.

=> EKS provides ready-made k8s control plane for us.

=> EKS is most robust platform to run our k8s control plane.

Note: EKS is a paid service.

================

AWS EKS Setup

================

Git Repo :: https://github.com/ashokitschool/DevOps-Documents/blob/main/05-EKS-Setup.md

Note: Once practice completed then delete EKS cluster to avoid billing.

====================

ConfigMap & Secrets

====================

=> For every application we will have multiple environments in the real-time.

1) DEV

2) SIT

3) UAT

4) PILOT

5) PROD

=> DEV env used by developers to perform code integration testing.

=> SIT env used by testing team to perform "System Integration Testing".

=> UAT env used by client to perform "user acceptance testing".

=> PILOT env used for pre-production testing.

=> PROD env used for application live deployment.

Note: Every environment will have some properties to run the application.

1) Database props

2) SMTP Props

3) Kafka Props

4) Redis props

5) Payment Gateway Props

### We shouldn't configure the above properties in application. If we configure them in our application then our application will have hard coded properties to support for only one enviroment deployment.

=> If we want to deploy our application in another enviroment then we have to change properties and package our appliacation then create docker image then deploy in k8s cluster. It is time taking process and error prone.

Note: To make our application loosely copuled with enviroments we shouldn't configure properties with in the application.

## To make our application loosely coupled with environment properties we can use ConfigMap & Secret.

=> ConfigMap & Secret will allow you to de-couple application properties from docker images so that our application can be deployed in any enviroment.

Note: At the time of application deployment we can supply environment properties to the application using configmap and secret.

=> ConfigMap & Secret will store data in key-value format

=> To store non-senstive information we will use configmap and to store sensitive information we will use secret.

=======================

ConfigMap manifest YML

=======================

---

apiVersion: v1

kind: ConfigMap

metadata:

name: configmap-dev

data:

database\_url: "mongodb://localhost:27017"

database\_name: "exampledb"

...

$ kubectl get configmap

$ kubectl apply -f <configmap-yml-file>

============================

Reading Data From ConfigMap

============================

valueFrom:

configMapKeyRef:

name: configmap-dev

key: database\_url

valueFrom:

configMapKeyRef:

name: configmap-dev

key: database\_name

=====================

Secret manifest yml

=====================

---

apiVersion: v1

kind: Secret

metadata:

name: secret-dev

type: Opaque

data:

db\_username: YWRtaW4=

db\_password: cGFzc3dvcmQ=

...

$ kubectl get secret

$ kubectl apply -f <secret-manifest-yml>

============================

Reading Data From Secret

============================

valueFrom:

secretKeyRef:

name: secret-dev

key: db\_username

valueFrom:

secretKeyRef:

name: secret-dev

key: db\_password

===========

Assignment

===========

=> Deploy MySQL database in k8s cluster by using configMap and secret.

=> To do this task we will create below manifest yml files

a) mysql-config.yml

b) mysql-secret.yml

c) mysql-deployment.yml

d) mysql-service.yml

=============

HELM Charts

=============

=> HELM is a package manager for k8s cluster.

=> HELM is used to install required softwares in k8s cluster.

Ex : metrics-server, EFK, Promethues, Grafana etc..

=> Chart means collection configuration ymls.

Note: By using HELM charts we can deploy our applications also in k8s cluster.

====================

Helm Installation

====================

$ curl -fsSl -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3

$ chmod 700 get\_helm.sh

$ ./get\_helm.sh

$ helm

-> check do we have metrics server on the cluster

$ kubectl top pods

$ kubectl top nodes

1) execute manifest ymls -- we are done with this process

2) use helm chart

# check helm repos

$ helm repo ls

# Add the metrics-server repo to helm

$ helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/

# Install the chart

$ helm upgrade --install metrics-server metrics-server/metrics-server

=========================

Kubernetes Monitoring

=========================

=> We can monitor our k8s cluster and cluster components using below softwares

1) Prometheus

2) Grafana

=============

Prometheus

=============

-> Prometheus is an open-source systems monitoring and alerting toolkit

-> Prometheus collects and stores its metrics as time series data

-> It provides out-of-the-box monitoring capabilities for the k8s container orchestration platform.

=============

Grafana

=============

-> Grafana is an analysis and monitoring tool

-> It provides visulization for monitoring

-> It provides charts, graphs, and alerts for the web when connected to supported data sources.

Note: Graphana will connect with Prometheus for data source.

#########################################

How to deploy Grafana & Prometheus in K8S

##########################################

-> Using HELM charts we can easily deploy Prometheus and Grafana

##########################################################

Install Prometheus & Grafana In K8S Cluster using HELM

##########################################################

# Add the latest helm repository in Kubernetes

$ helm repo add stable https://charts.helm.sh/stable

# Add prometheus repo to helm

$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

# Update Helm Repo

$ helm repo update

# install prometheus

$ helm install stable prometheus-community/kube-prometheus-stack

# Get all pods

$ kubectl get pods

Node: You should see prometheus pods running

# Check the services

$ kubectl get svc

# By default prometheus and grafana services are available within the cluster as ClusterIP, to access them outside lets change it to LoadBalancer.

# Edit Prometheus Service & change service type to LoadBalancer then save and close that file

$ kubectl edit svc stable-kube-prometheus-sta-prometheus

# Now edit the grafana service & change service type to LoadBalancer then save and close that file

$ kubectl edit svc stable-grafana

# Verify the service if changed to LoadBalancer

$ kubectl get svc

=> Access Promethues server using below URL

URL : http://LBR-DNS:9090/

=> Access Grafana server using below URL

URL : http://LBR-DNS/

=> Use below credentials to login into grafana server

UserName: admin

Password: prom-operator

=> Once we login into Grafana then we can monitor our k8s cluster. Grafana will provide all the data in charts format.

==========

DaemonSet

==========

=> DaemonSet is a Kubernetes feature that lets you run a Kubernetes pod on all cluster nodes.

=> Every time when a new node is added to a cluster, the pod is added to it, and when a node is removed from the cluster, the pod is removed.

=> In a Kubernetes architecture, DaemonSets are used for deploying background services across clusters, providing support services for every nodeâ€”such as system operations services, collecting logs, monitoring frameworks like Prometheus, and storage volumes.

Note: We can use DaemonSet to deploy FluentD pods.

=============

StatefulSet

=============

=> StatefulSet is the workload API object used to manage stateful applications.

=> StatefulSets are valuable for applications that require one or more of the following.

Stable, unique network identifiers.

Stable, persistent storage.

Ordered, graceful deployment and scaling.

Ordered, automated rolling updates.

===========================

EFK Setup in K8S Cluster

===========================

=> The EFK stack in Kubernetes is a popular logging solution composed of three components: Elasticsearch, Fluentd, and Kibana.

=> It is used to collect, store, and visualize logs generated by applications and Kubernetes clusters.

===============================

Blue - Green Deployment Model

===============================

====================

Ingress Controller

====================

=> It is a k8s resources and it is used to map extranal traffic to internal services running in the k8s clsuter.

=> It supports path based routing

Path = /hoteks ===> forward that request to "hotels-service"

Path = /flights ===> forward that request to "flighs-service"

Reference Video : https://youtu.be/Fa7f0cHgx2k

=============

K8S Summary

=============

1) K8s Introduction

2) K8S Architecture

3) K8S Cluster Setup

- MiniKube

- AWS EKS

4) K8S Terminology

5) PODS

6) Services

7) Namespaces

8) ReplicationController

9) ReplicaSet

10) Deployment

11) HPA

12) Blue Green Deployment

13) ConfigMap & Secret

14) MySQL DB Deployment in k8s

15) DaemonSet

16) StatefulSet

17) EFK Stack

18) HELM Charts

19) Ingress Controller

========= Pending Topics ===========

20) PV & PVC

21) Taints & Tolerations

22) Liveness & Readyness

23) Grafana & Promethues