```
======================
Docker Vs Kubernetes
======================

1) Containerization : Running our application inside container

                Ex: Docker Containers

=> Packaging our app code and dependencies as single unit for execution is called as
Containerization.

2) Orchestration : Managing Containers

                Ex: Docker Compose, Docker Swarm, Kubernetes, Open Shift...


============
Kubernetes
============

=> It is an open source s/w

=> K8S is called as Orchestration Tool

=> K8S is used to manage our containers

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

=> K8S software developed by Google company

=> K8S software developed using Go Language.

=> Kubernetes is also called as K8S

====================
Advantages with K8S
====================

1) Auto Scaling

2) Self Healing

3) Load Balancing

==================
K8S Architecture
==================

=> K8S will follow cluster architecture

=> Cluster means group of servers

=> In K8S cluster we will have Master Node & Worker Nodes

Note: Master Node is called as Control Plane.

=> Master Node will recieve the request and will assign task to worker nodes.

=> Worker Nodes will perform the action based on task given by Master node.


======================
K8S Cluster Components
```

```
======================

1) Control Plane (Master Node)

                - API Server
                - Schedular
                - Controller Manager
                - ETCD

2) Worker Node

                - Kubelet
                - Kube Proxy
                - Docker Runtime
                - POD


Note: Kubectl is used to communicate with K8S cluster.

=> API Server will recieve the request given by Kubectl and will store the request into etcd.

=> ETCD is an internal db of k8s cluster.

=> Schedular will identify pending requests in ETCD and will identify worker node to to schedule the
task.

Noe: Schedular will communicate with the worker node using Kubelet.

=> Kubelet is called as Node Agent. It will maintain all the info related to worker node.

=> Kube Proxy will provide network for cluster communication.

=> Controller-manager will veriy all the tasks are working as expected or not.

=> In Every Worker Node, Docker Engine will be available to run Docker Container.

=> In K8S docker container will be created inside POD.

=> POD is a smallest building block that we can deploy in k8s cluster.

Note: Inside one POD we can create multiple containers.

Note: In K8S, everything will be represented as POD.

==================
K8S Cluster Setup
==================

1) Mini Kube  (Single Node Cluster) - Only For Practice

2) Kubeadm Cluster (Self Managed Cluster)

3) Provider Managed Cluster (Ex : AWS EKS, Azure AKS, GCP GKE....) - Realtime.

================
AWS EKS Cluster
================

EKS - Elastic Kubernetes Service

=> EKS provides readymade Kubernetes Control Plane.

=> EKS is highly scalable and robust solution to run k8s components.

Note: EKS is not free.
```

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

==========
Kubernetes Resources
==========

1) PODS
2) Services
3) Namespaces
4) ReplicationController (RC) - Outdated
5) ReplicaSet (RS)
6) Deployment
7) DaemonSet
8) StatefulSet
9) IngressController
10) HPA
11) Helm Charts
12) K8S Monitoring(Grafana & Premoethues)
13) EFK Stack (Log Monitoring)

==========
What is POD
==========

=> POD is a smallet building block in k8s cluter

=> Application will be deployed in K8S as a POD

=> We can create multiple pods for one application

=> To create POD we need docker image

=> For Every POD k8s will assign one IP address

=> When POD is damaged/crashed K8S will replace that pod (Self Healing)

=> When we create multiple PODS for one applications, load will be balanced by k8s.

Note: By default we can access PODS only with in the cluster.

=> To provide public access for our PODS we need to expose our PODS using K8S Service concept.

==========
K8S Services
==========

=> Service is used to expose PODS.

=> We have 3 types of Services in k8s.

1) Cluster IP
2) NodePort
3) Load Balancer

==========
What is Cluster IP ?
==========

=> POD is a short lived object

=> When pod is crashed/damaged k8s will replace that with new pod

=> When POD is re-created IP will be changed.

Note: It is not recommended to access pods using POD IP

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

=> Cluster IP is a static ip to access pods

=> Using Cluster IP we can access pods only with in the cluster

Ex: Database Pods, Cache Pods, Kafka Pods etc...

==========
What is NodePort service ?
==========

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

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

=> When we use Node Public IP to access our pod then all requests will go same worker node (burden will be increased on the node).

Note : To distribute load to multiple worker nodes we will use LBR service.

==========
What is Load Balancer Service ?
==========

=> It is used to expose our pods outside cluster using AWS Load Balancer

=> When we access load balancer url, requests will be distributed to all pods running in all worker nodes.

==========
K8s Namespaces
==========

=> Namespaces are used to group the resources.

Ex: frontend app pods under one namespace

backend app pods under one namespace

db pods under one namespace

==========
Summary
==========

1) What is Containerization
2) What is Orchestration
3) K8S Introduction
4) K8S Advantages
5) K8S Architecture
6) AWS EKS Cluster Setup
7) What is POD
8) What is Service (Cluster IP, Node Port & LBR)
9) What is Namespaces

```
================================================================================

=> In K8S we will use manifest yml to deploy our applications

-------------------------
K8S Manifest YML Syntax
-------------------------

---
apiVersion :
kind:
metadata:
spec:
...


# Execute manifest yml using kubectl
$ kubectl apply -f <manifest-yml>

====================
K8S POD Manifest YML
====================

---
apiVersion: v1
kind: Pod
metadata:
 name: javawebapppod
 labels:
   app: javawebapp
spec:
  containers:
  - name: javawebappcontainer
    image: ashokit/javawebapp
    ports:
    - containerPort: 8080
...

# check pods running in cluster
$ kubectl get pods

# Create PODS using manifest YML
$ kubectl apply -f <pod-manifest-yml>

# Check POD running in which worker node
$ kubectl get pods -o wide

# Describe pod
$ kubectl describe pod <pod-name>

# Check POD Logs

$ kubectl logs <pod-name>


=============
K8S Service
=============

=> Service is used to expose our pods


---
apiVersion : v1
kind: Service
```

```
metadata:
 name: javawebsvc
spec:
 type: NodePort
 selector:
  app: javawebapp
 ports:
 - port: 80
   targetPort: 8080
   nodePort: 30070
...


Note: POD lable we will use as 'selector' in service (to identify the pods)


# Check k8s services running
$ kubectl get service

# create service using manifest yml
$ kubectl apply -f <service-manifest-yml>

=> Enable NodePort number in worker node security group inbound rules.

=> Access our application

        URL : http://node-public-ip:nodeport/java-web-app/


================================================================================
Q) Is it mandatory to specify Node Port Number in service manifest yml ?
================================================================================

Ans) No, if we don't specify k8s will assign one random port number in between
     30,000 - 32, 767.


=====================================
POD & Service in single manifest YML
=====================================

---
apiVersion: v1
kind: Pod
metadata:
 name: javawebapppod
 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
  ...


  $ kubectl apply -f <manifest-yml>

  $ kubectl get pods

  $ kubectl get svc

  Note: In the above manifest yml we have not configured Node Port Number. K8S will assign one randome
  number.

  => We need to enable that Node Port number in security group inbound rules.

  ===============
  K8S Namespaces
  ===============

  => Namespaces are used to group k8s resources logically

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

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

                  database-pods ===> create under one namespace

  => We can create multiple namespaces in k8s cluster

                  Ex: ashokit-frontend-ns, ashokit-backend-ns, ashokit-db-ns etc...

  => Each namespace is isolated with other namespaces.

  Note: When we delete a namespace, all the resources which are created under that namespace also gets
  deleted.


  # check namespaces
  $ kubectl get ns

  Note: When we don't specify any namespace then it will use 'default' namespace

  # Get all resources available under default namespace
  $ kubectl get all

  # Retrieve all resources available under particular namespace
  $ kubectl get all -n <namespace-name>

  # get pods of default namespace
  $ kubectl get pods

  # get pods of kube-system namespace
  $ kubectl get pods -n kube-sytem


  => We can create k8s namespace in 2 ways

                  1) Using kubectl command directley

                  2) Using k8s manifest yml



  Ex:-1 : $ kubectl create ns <namespace-name>
```

```
---------------------------
k8s namespace manifest yml
---------------------------

---
apiVersion: v1
kind: Namespace
metadata:
 name: ashokit-ns2
...

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


====================================
Namespace + pod + service - manifest
====================================

---
apiVersion: v1
kind: Namespace
metadata:
 name: ashokit-ns3
---
apiVersion: v1
kind: Pod
metadata:
 name: javawebapppod
 namespace: ashokit-ns3
 labels:
   app: javawebapp
spec:
  containers:
  - name: javawebappcontainer
    image: ashokit/javawebapp
    ports:
    - containerPort: 8080
---
apiVersion : v1
kind: Service
metadata:
 name: javawebsvc
 namespace: ashokit-ns3
spec:
 type: NodePort
 selector:
  app: javawebapp
 ports:
 - port: 80
   targetPort: 8080
...


$ kubectl get ns

$ kubectl apply -f <manifest-yml>

$ kubectl get all

$ kubectl get all -n ashokit-ns3


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

=> As of now, we have created POD manually using POD Manifest YML
```

```
=> If we create POD manually then we don't get self-healing capability

=> If POD is damaged/crashed/deleted then k8s will not create new POD.

=> If pod damaged then our application will be down.

Note: We shouldn't create POD directley to deploy our application.

Note: We need to use k8s resources to create pods.

=> If we create pod using k8s resources, then pod life cycle will be managed by k8s.


1) ReplicationController (Outdated)
2) ReplicaSet
3) Deployment
4) DaemonSet
5) StatefulSet

===========
ReplicaSet
===========

=> It is a k8s resource

=> It is used to create pods in k8s

=> It will take care of POD lifecycle.

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

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

=> 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: javawebapppod
    labels:
     app: javawebapp
   spec:
        containers:
        - name: javawebappconatiner
          image: ashokit/javawebapp
          ports:
                - containerPort: 8080
...
apiVersion: v1
kind: Service
metadata:
 name: javawebsvc
```

```
spec:
 type: LoadBalancer
 selector:
  app: javawebapp
 ports:
  - port: 80
    targetPort: 8080
...

$ kubectl get pods

$ kubectl get svc

$ kubectl get rs

$ kubectl apply -f <yml>

$ kubctl get all

$ kubectl delete pod <pod-name>

$ kubectl get pods


Note: If we configure service type as LoadBalancer then in AWS LBR will be created.

=> Using AWS LBR DNS url we can access our application.


$ kubectl scale rs javawebrs --replicas 4

$ kubectl get pods

$ kubectl scale rs javawebrs --replicas 3


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


=> K8S supports Auto Scaling when we use 'Deployment' to create pods.


================
K8S Deployment
================

=> It is one of the k8s resource/component

=> It is most recommended approach to deploy our applications in k8s.

=> Deployment will manage pod life cycle.

=> We have below advantages with K8s Deployment

        1) Zero downtime

        2) Auto Scaling

        3) Rolling Update & Rollback


=> We have below deployment strategies

                1) ReCreate
                2) RollingUpdate
```

3) Canary

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

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

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: javawebdeploy
spec:
 replicas: 2
 strategy:
  type: RollingUpdate
 selector:
  matchLabels:
   app: javawebapp
 template:
   metadata:
    name: javawebapppod
    labels:
     app: javawebapp
   spec:
        containers:
        - name: javawebappconatiner
          image: ashokit/javawebapp
          ports:
                - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
 name: javawebsvc
spec:
 type: LoadBalancer
 selector:
  app: javawebapp
 ports:
  - port: 80
    targetPort: 8080
...
```

```
$ kubectl delete all --all

$ kubectl get all

$ kubectl apply -f <yml>

$ kubectl get all
```

Note: Access app using LBR URL

```
$ kubectl scale deployment javawebdeploy --replicas 4

$ kubectl get pods

$ kubectl scale deployment javawebdeploy --replicas 3
```

```
==============================
Blue - Green Deployment Model
==============================

Blue Env Docker image : ashokit/javawebapp

Green Env Docker image : ashokit/javawebapp:v2

===================
Working Process
===================

Step-0 : Clone below git repo

URl : https://github.com/ashokitschool/kubernetes_manifest_yml_files.git

Step-1 : Navigate into blue-green directory

Step-2 : Create Blue Pods deployment

Step-3 : Create Live Service to expose blue pods

Step-4 : Access App using Live Service LBR DNS Url

   (blue pods response we should be able to see in browser)

Step-5 : Create Green Pods deployment

Step-6 : Check all pods running

Step-7: Make Green Pods Live (update selector and apply the yml)

Step-8 : Access App using Live Service LBR DNS Url

          (green pods response we should be able to see in browser)


=================
Package Managers
=================

=> In Linux os we have package manager to install required softwares/packages

                Ex: yum, apt ...

amazon linux vm :

                - sudo yum install java

                - sudo yum install git

                - sudo yum install maven

ubuntu linux vm :

                - sudo apt install java

                - sudo apt install git

                - sudo apt install maven


===============
What is HELM ?
================
```

=> HELM is a package manager which is used to install required softwares in k8s cluster

=> HELM will use charts to install required packages

=> Chart means collection of configuration files (manifest ymls)

####### HELM Charts ###########

=> Using HELM chart we can install promethues server

=> Using HELM chart we can install grafana server

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

# check helm repos
$ helm repo ls

# Before you can install the chart you will need to 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

$ kubectl top pods

$ kubectl top nodes

$ helm list

$ helm delete <release-name>
```

==========
Metric Server Unavailability issue fix
==========

URL : https://www.linuxsysadmins.com/service-unavailable-kubernetes-metrics/

```
$ kubectl edit deployments.apps -n kube-system metrics-server
```

=> Edit the below file and add new properties which are given below

---------- Existing File----------

```
 spec:
```

```
      containers:
      - args:
        - --cert-dir=/tmp
        - --secure-port=4443

---------------------------- New File--------------------
---
    spec:
      containers:
      - args:
        - --cert-dir=/tmp
        - --secure-port=4443
        - --kubelet-insecure-tls=true
        - --kubelet-preferred-address-types=InternalIP

-----------------------------------------------------------------

$ kubectl top pods

$ kubectl top nodes


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

-> Grafana is a multi-platform open source analytics and interactive visualization web application.

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

-> Grafana allows you to query, visualize, alert on and understand your metrics no matter where they
are stored. Create, explore and share dashboards.


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

# Search Repo
$ helm search repo prometheus-community

# 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 NodePort 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

# Check in which nodes our Prometheus and grafana pods are running
$ kubectl get pods -o wide

=> 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.
```