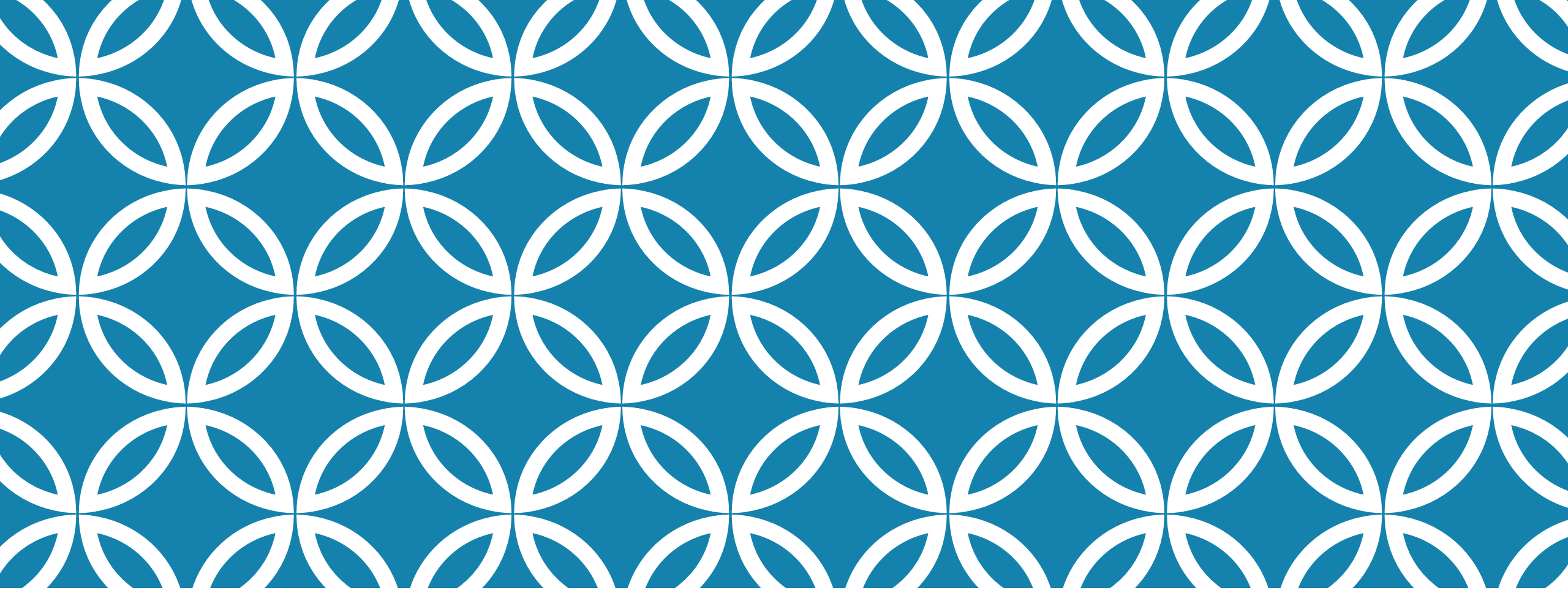


KUBERNETES MASTERCLASS 2

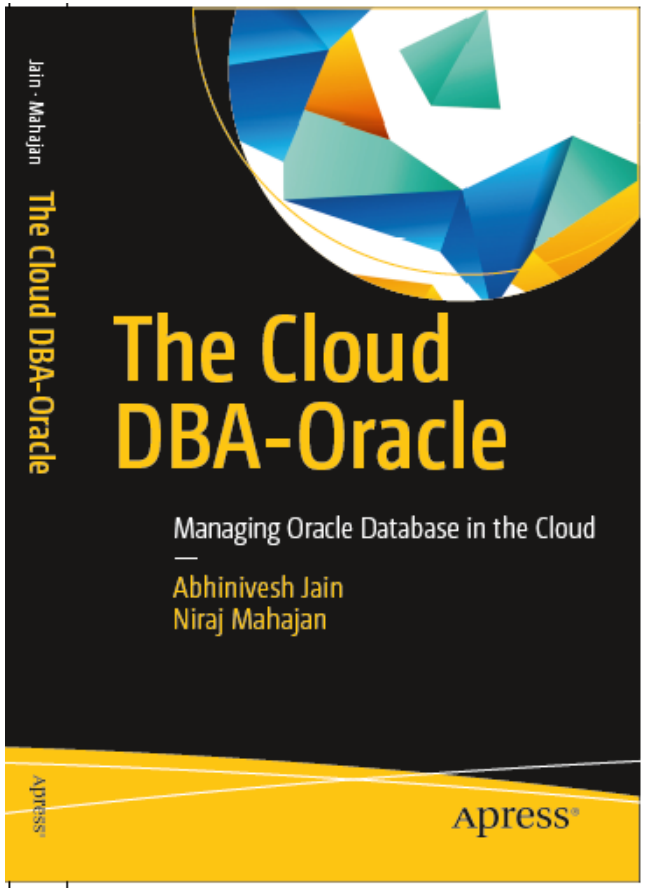
Abhinivesh Jain

ABOUT ME

Author, Speaker and Blogger

Open Source “Contributor”

Working as Distinguished Member of Technical Staff (DMTS)- Senior Member



@AbhiniveshJain

/abhiniveshjain

DMTS
DISTINGUISHED MEMBER OF TECHNICAL STAFF

AGENDA FOR MASTERCLASS-2

Kubernetes Building Blocks

- Namespace
- POD
- Replicasets
- Deployment
- Services
- Persistent Volume & Persistent Volume Claim
- ConfigMaps & Secrets
- Stateful Sets
- Daemon Sets

Application Lifecycle management

- Deployment
- Rollback
- Upgrade
- Self Healing
- Scaling

Agenda of Masterclass-1

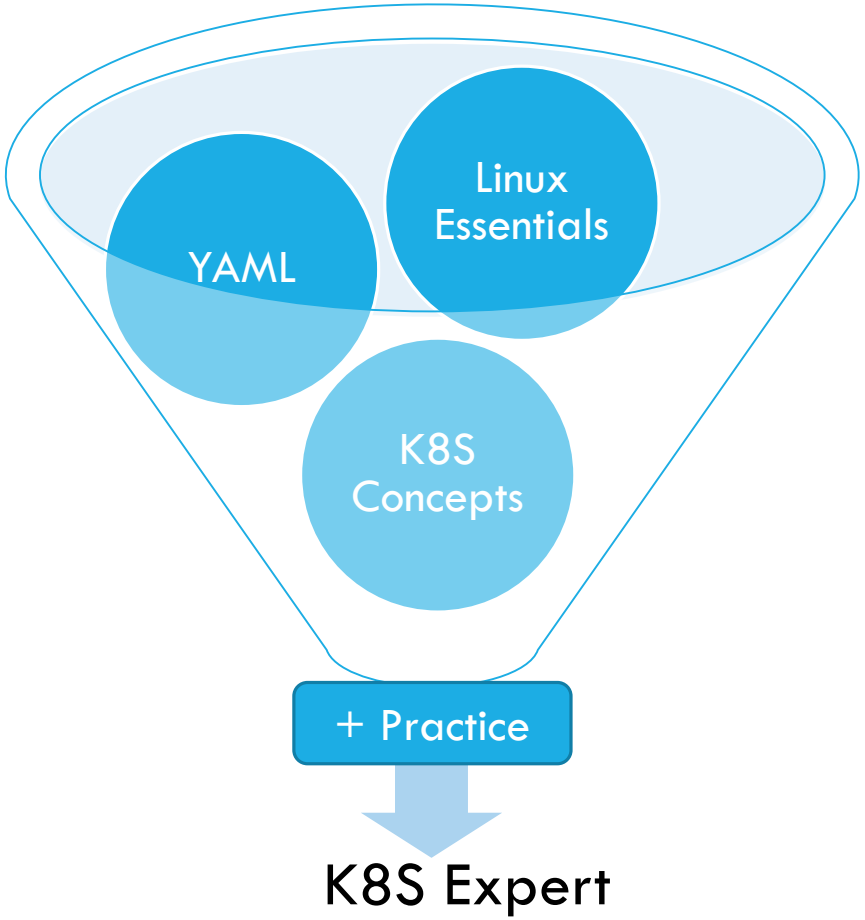
Container

- History
- Container Image
- Container Image registry
- Container Orchestration
- Sample application deployment

Kubernetes

- History
- Architecture
- Managed k8s Providers
- AWS Offerings
- Launching EKS cluster
- Kubernetes building Blocks

KUBERNETES LEARNING FORMULA



KUBERNETES PLAYGROUNDS

- https://www.katacoda.com/courses/kubernetes/playground
- Run Minikube on local Windows machine using Virtual box/Hyper-V (on windows 10)
- Use any Linux server/virtual machine and install MiniKube
- Use any EC2 (free) tier machine and install Minikube
- Use any managed Kubernetes offering like- AWS EKS (it is charged on hourly basis so be careful)

KUBECTL AND KUBEADM- 2 KEY TOOLS IN K8S

Kubeadm

- Command line utility to provision kubernetes cluster

Kubectl

- Command line utility to manage kubernetes cluster resources
- Works using file present in $HOME/.kube/kubeconfig file, you can export KUBECONFIG variable
- Can call this utility with –kubeconfig flag
- Follows below syntax

```
kubectl [command] [TYPE] [NAME] [flags]
```

- TYPE could be given in 3 ways. (singular/plural/abbreviated)
- 2 ways to run commands-

1. Imperative (e.g. kubectl run nginx)
2. Declarative (e.g. kubectl create –f pod_definition.yaml)

KUBERNETES OBJECT TYPES RELEVANCE TO APPLICATION NEED

Application Need	Kubernetes Object Type
Running App	POD
Scaling of App	Replicaset
Application deployment	Deployment
Application upgrade/rollback	Deployment
Application exposure within/outside cluster	Service
Stateful Application deployment	Statefulset
Application configuration (environment variables)	ConfigMap
Application Configuration (Sensitive info like- Password)	Secrets

Application Need	Kubernetes Object Type
Running batch Jobs	Jobs
Running scheduled Jobs	CronJobs
Load balancing and traffic rules setup	Ingress
Persistent storage	Persistent Volume (PV) and Persistent Volume Claim (PVC)
Application self healing	-
Special purpose application hosting (like- monitoring/logging) hosting	Daemonset
Application version handling	Labels/Selectors
Application Environment separation	Namespace

LET'S DEEP DIVE INTO MOST COMMONLY USED OBJECTS (MY TOP 10)

1. Namespace
2. POD
3. Replicasets
4. Deployment
5. Services
6. Persistent Volume & Persistent Volume Claim
7. ConfigMaps
8. Secrets
9. Stateful Sets
10. Daemon Sets

NAMESPACES

- Way to divide cluster resources to multiple users. You can use Resource quota at Namespace level
- Name of resources should be unique within namespace but not across namespaces.
- Each k8s resource should be only in 1 namespace.
- Good for handling multi tenancy. E.g. Same cluster can be used by different departments or for different environments.

```
kubectl get namespace

kubectl create namespace <namespace-name>

kubectl get/describe object –n <namespace name>

kubectl get all –all-namespaces
```

NAMESPACES

- Nested Namespaces not allowed
- Namespaces that come with K8s
  - Default
  - Kube-system
  - Kube-public
  - It is never a good practice to use Default namespace in production setup
  - For Cross namespace communication, use service-name.namespace-name
  - kubens is good utility to switch between Kubernetes namespaces.

```
kubectl config set-context --current --namespace=<namespace-name>
```

NAMESPACES- QUIZ TIME

- How Many Namespaces should be created in any production environment?

A) 2

B) 10

C) 0

D) It Depends !!!

All Kubernetes Objects are inside Namespaces?

A) True

B) False

POD

- Collection of one or more Containers
- Usually there is 1:1 mapping of pod and container
- Multi-container POD also exists
- Acts like host for given application
- Containers inside POD has shared IP, shared storage and have shared Lifecycle

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: aws-pod
  labels:
    name: web
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
EOF
```

POD- QUIZ TIME

- Why I am not able to delete my POD? Whenever I delete, it comes back again.

A) POD is deployed using replicaset
B) POD is deployed using deployment
C) POD doesn't want to leave my cluster.
D) POD is non-deletable.

REPLICASET

Maintains specific number of PODs running at all time.

Deployment is next level object that manages replicaset and provides addition functionality

You can't change container image in replicaset

Replicaset and PODs are associated via Labels.

Replicaset can acquire pod that matches the label selector.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: aws-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
EOF
```

HORIZONTAL POD AUTOSCALER (HPA) USING REPLICASET

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Source: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/#example

DEPLOYMENT

Provides Declarative updates for PODs and Replicasets

Useful for Upgrade, Rollback and Scaling of app

Updating container image triggers deployment rollout.

Deployment controller changes from current state to desired state at a controlled rate. By Default, it ensures 75% of desired number of PODs are up.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aws-deployment
  labels:
    app: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
EOF
```

SERVICES

Helps in exposing application connectivity within/outside the cluster

Required for POD to POD communication and POD to external world communication.

3 Types

- Cluster IP
- NodePort
- Load Balancer

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
EOF
```

PERSISTENT VOLUME

Volume is a directory which is accessible to containers in a pod.

K8s supports several types of volumes, e.g. AWS EBS, Azuredisk, cephfs, csi, nfs, PVC etc.

Volume outlives the containers and data is preserved after pod restart

Persistent Volume is Kubernetes term, it refers to Volume that is managed by Kubernetes.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

VOLUME MOUNT EXAMPLE

PV aren't used directly in a POD, rather you create a PVC and refer that in POD. PVC example is shown below-

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 30Gi
EOF
```

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    app: db
spec:
  containers:
  - name: db
    image: mysql
    imagePullPolicy: Always
    ports:
    - containerPort: 3306
    volumeMounts:
    - name: mysql-data
      mountPath: /var/lib/mysql
  volumes:
  - name: mysql-data
    persistentVolumeClaim:
      claimName: db-pvc
EOF
```

CONFIGMAP

Used to keep Configuration data outside of application image. E.g. no need of 3 application images for 3 different environments.

Defined in terms of key-value pairs

Can be mounted in a POD to allow dynamic update of configmap values

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: web-configmap
data:
  DB_NAME: mysqldb
  DB_USER: root
  DB_PASSWORD: passwd
  DB_HOST: db-service
EOF
```

CONFIGMAP EXAMPLE

ConfigMap can be created from

- Directory
- Files
- Literal values

Multiple ConfigMap values can be referenced in one POD

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: aws-pod
  labels:
    app: web
spec:
  containers:
  - name: web
    image: nginx:latest
    ports:
    - containerPort: 3000
    envFrom:
     - configMapRef:
         name: web-configmap
    volumeMounts:
    - name: webconfigmap-vol
      mountPath: "/web-config"
      readOnly: true
  volumes:
  - name: webconfigmap-vol
    configMap:
      name: web-configmap
EOF
```

SECRETS

Used to keep Confidential information like password, Oauth tokens.

Defined in terms of key-value pairs

Uses base64 encoding, To convert any value to base64, use below command-

echo -n ‘give-value-here' | base64

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: db-config
type: Opaque
data:
  mysql_database: c2hkcmV0cmliZQ==
  mysql_user: c2hkcmV0cmliZQ==
  mysql_password: c2hkcmV0cmliZQ==
  mysql_root_password: V2kwkm9ANTIz
EOF
```

STATEFUL SETS

Deployment and Replicaset are good for stateless application whereas for Stateful application, Statefulsets should be used

Manages PODs and scaling of PODs but guarantees about ordering and uniqueness of each POD

For a StatefulSet with N replicas, when Pods are being deployed, they are created sequentially, in order from {0..N-1}.

When Pods are being deleted, they are terminated in reverse order, from {N-1..0}

DAEMON SETS

Ensures that all (or some) Nodes run a copy of a Pod.

Typical use cases-

- Cluster storage daemon
- Log collection daemon
- Node monitoring daemon

Toleration Key	Effect	Version	Description
node.kubernetes.io/not-ready	NoExecute	1.13+	DaemonSet pods will not be evicted when there are node problems such as a network partition.
node.kubernetes.io/unreachable	NoExecute	1.13+	DaemonSet pods will not be evicted when there are node problems such as a network partition.
node.kubernetes.io/disk-pressure	NoSchedule	1.8+	
node.kubernetes.io/memory-pressure	NoSchedule	1.8+	
node.kubernetes.io/unschedulable	NoSchedule	1.12+	DaemonSet pods tolerate unschedulable attributes by default scheduler.
node.kubernetes.io/network-unavailable	NoSchedule	1.12+	DaemonSet pods, who uses host network, tolerate network-unavailable attributes by default scheduler.

CONNECTING THE DOTS

How to use all these concepts in real life?

Case Study

Create a 2 tier application using Web and DB Tier. You can use Nginx and MySQL.

Web application should be accessible from port 80 but web container should run on port 8080. Application should have self healing and Web tier should have scaling capabilities. All configurations should be stored outside the PODs and credentials should be stored in encrypted form. Rolling upgrade with Zero downtime should be allowed. Pause and resume of app upgrade should be allowed. Rollback of upgrade should be allowed.

CONNECTING THE DOTS

Solution Components

1. POD (Web and DB)- Do we need to create these separately?...NO
2. Secrets
3. Service
4. Deployment (POD, Replicaset)
5. PV
6. PVC
7. Labels and Selector

CONNECTING THE DOTS

Order of object creation-

1. Secrets
2. PV
3. PVC
4. Deployment (POD, Replicaset)
5. Service

CONNECTING THE DOTS

Requirement Mapping

Requirement	Solution Component
Save configuration outside of POD and in encrypted form	Secrets
Self Healing, Rolling upgrade with Zero downtime, Pause and Resume of upgrade, Rollback of upgrade	Deployment
Scaling	Deployment Scaling
Port 80:8080 mapping	Service
Persistent data in database	PV and PVC

WHAT NEXT?

Create 2 Tier app similar to the case study details and show the steps required for meeting various requirements.

USEFUL LINKS

https://redhat-developer-demos.github.io/kubernetes-tutorial/kubernetes-tutorial/pod-rs-deployment.html

https://kubernetes.io/docs/reference/kubectl/overview/

https://collabnix.github.io/kubelabs/

https://kubernetesbyexample.com/

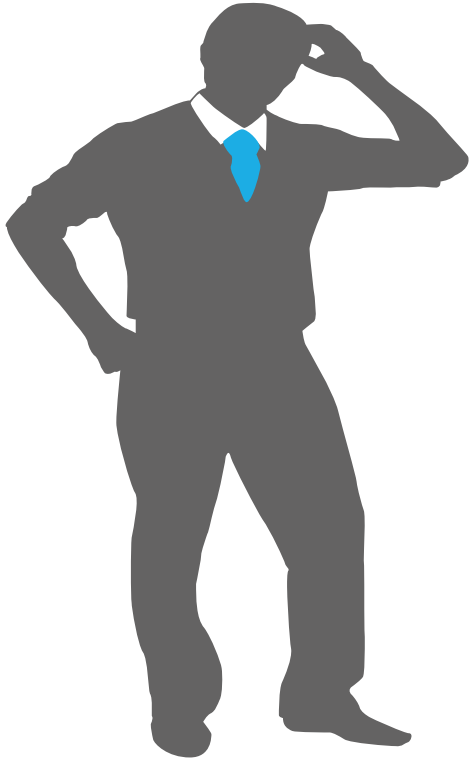
And the most important one is

https://kubernetes.io/docs/concepts/

MASTERCLASS IS JUST TIP OF THE ICEBERG (K8S)

Image courtesy : http://preparedtoanswer.org

QUESTIONS???



THANKS FOR ATTENDING THIS MASTERCLASS.