# **MongoDB Deployment on Kubernetes Cluster via DeploymentSet and StatefulSet**

In this let’s see how we can deploy MongoDB on a Kubernetes cluster. We will deploy via both deployment sets and StatefulSets.

****MongoDB Deployment on Kubernetes using****

1. DeploymentSet
2. StatefulSet

****Deploy MongoDB via DeploymentSet****

1. ****pvc.yaml****

We need to create a pvc.yaml file first so that even if our pod gets restarted or deleted, our data won’t be lost.

---  
 apiVersion: "v1"  
 kind: "PersistentVolumeClaim"  
 metadata:   
 name: "mongodb-pvc"  
 namespace: "development"  
 labels:   
 app: "mongo-db"  
 spec:   
 accessModes:   
 - ReadWriteOnce  
 resources:   
 requests:   
 storage: 5Gi  
 storageClassName: gp2

Once the pvc.yaml file is created, we need to deploy the pvc.yaml file using the following command.

****kubectl apply -f pvc.yaml -n <namespace>****

The above command will work only if you have the kubectl tool installed. Also, if you have created a separate namespace for deploying MongoDB, you can defined it like above.

To check if the pvc has been created, run the following command.

****kubectl get pvc -n <namespace>****

****2. Deployment.yaml****

We need to create a deployment yaml file for deploying MongoDB on Kubernetes. The image that I am using for the deployment is “mongo”

If you want a particular version of MongoDB, you can checkout the official MongoDB image from DockerHub and check for tags.

---  
 apiVersion: "apps/v1"  
 kind: "Deployment"  
 metadata:   
 name: "mongo-db"  
 namespace: "development"  
 spec:   
 selector:   
 matchLabels:   
 app: "mongo-db"  
 replicas: 1  
 strategy:   
 type: "RollingUpdate"  
 rollingUpdate:   
 maxSurge: 1  
 maxUnavailable: 1  
 minReadySeconds: 5  
 template:   
 metadata:   
 labels:   
 app: "mongo-db"  
 spec:   
 containers:   
 -   
 name: "mongo-db"  
 image: "mongo"  
 imagePullPolicy: "Always"  
 env:   
 -   
 name: "MONGO\_INITDB\_ROOT\_USERNAME"  
 value: "xxxx"  
   
 -  
 name: "MONGO\_INITDB\_ROOT\_PASSWORD"  
 value: "xxxx"  
   
 ports:   
 -   
 containerPort: 27017  
 name: "mongodb"  
   
 volumeMounts:   
 -   
 name: "mongodb-persistent-storage"  
 mountPath: "/data/db"  
   
 volumes:   
 -   
 name: "mongodb-persistent-storage"  
 persistentVolumeClaim:   
 claimName: "mongodb-pvc"

We need to understand certain things while creating a deployment.yaml file.

a. Under volumeMounts, the mountPath should be “/data/db” only. Any other name will result in a loss of data if the pod restarts as data is persisted on the /data/db path only.

b. Under volumes, the claimName should be the same name that you have given in the pvc.yaml file. In my case, it is “mongodb-pvc”.

c. Under env variables, you need to mention: MONGO\_INITDB\_ROOT\_USERNAME and MONGO\_INITDB\_ROOT\_PASSWORD. The username and password can be anything of your choice.

Once the yaml for deployment is created, we need to apply the deployment using the following command:

****kubectl apply -f deployment.yaml -n <namespace>****

To check if the deployment has been created, run the following command:

****kubectl get deployments -n <namespace>****

3. ****Service.yaml****

We need to create a service file so that we can access our application. I am using nodeport since it will help us access the application outside the cluster.

---  
 apiVersion: "v1"  
 kind: "Service"  
 metadata:   
 name: "mongo-db"  
 namespace: "development"  
 labels:   
 app: "mongo-db"  
 spec:   
 ports:   
 -   
 name: "mongodb"  
 port: 27017  
 nodePort: 30332  
 type: "NodePort"  
 selector:   
 app: "mongo-db"

Once the yaml for service is created, we need to apply the service using the following command:

****kubectl apply -f service.yaml -n <namespace>****

To check if the service has been created, run the following command.

****kubectl get svc -n <namespace>****

Now, to connect to our MongoDB database via client tools like MongoDB Compass, we will require the IP address for the node on which the pod has been schedule.

a. To check the IP address for your pod, run the follwoing command.

****kubectl describe pod <pod\_name> -n <namespace>****

There, you will find the IP address on which the pod has been scheduled.

---  
 Name: mongo-db  
 Namespace: 000000000030522-development  
 Priority: 0  
 Node: xxxx/xxxx  
 Start Time: Wed, 13 Oct 2021 11:28:32 +0000  
 Annotations: kubernetes.io/psp: eks.privileged   
 Status: Running

Here, on the “Node field”, the IP address will be present for your node. This will be quiet helpful to get the IP address for your node if you have a multi-node cluster.

The port number to connect will be your node-port. In our case, it is 30332.

Next, I will explain how we can connect to MongoDB Compass using the username and password.

****Deploy MongoDB via StatefulSet.****

****What Is StatefulSet?****

StatefulSet represents a set of pods unique, persistent and with a stable pod name. In the case of the DeploymentSet, whenever you deploy a pod, on every deployment or restart you will get a different pod name — something like “mongo-db-7f4b4b99b4-zmnw8”. Here, the pod name will change when it restarts. And on the next restart, it could be anything like “mongo-db-8eab2b99b4-cmda8”. If we want a stable pod name, StatefulSet can be used. Then, even when it restarts, we will get the same pod name (eg:- “mongo-db-0”).

StatefulSets are useful for stateful applications, which require a stable pod name.

1. ****pvc.yaml****

We need to create a pvc.yaml file first, so that even if our pod restarts or gets deleted, our data won’t be lost.

---  
 apiVersion: "v1"  
 kind: "PersistentVolumeClaim"  
 metadata:   
 name: "mongodb-pvc"  
 namespace: "development"  
 labels:   
 app: "mongo-db"  
 spec:   
 accessModes:   
 - ReadWriteOnce  
 resources:   
 requests:   
 storage: 5Gi  
 storageClassName: gp2

2. ****StatefulSet.yaml****

We need to create a Stateful yaml file for deploying MongoDB on Kubernetes. The image that I am using for the deployment is “mongo” Statefulset so that we get a unique pod name each time.

---  
 apiVersion: "apps/v1"  
 kind: "StatefulSet"  
 metadata:   
 name: "mongo-db"  
 namespace: "development"  
 spec:  
 serviceName: "mongo-db"   
 replicas: 1  
 selector:   
 matchLabels:   
 app: "mongo-db"  
 template:   
 metadata:   
 labels:   
 app: "mongo-db"  
 spec:   
 containers:   
 -   
 name: "mongo-db"  
 image: "mongo"  
 imagePullPolicy: "Always"  
 env:   
 -   
 name: "MONGO\_INITDB\_ROOT\_USERNAME"  
 value: "xxxx"  
   
 -  
 name: "MONGO\_INITDB\_ROOT\_PASSWORD"  
 value: "xxxx"  
   
 ports:   
 -   
 containerPort: 27017  
 name: "mongodb"  
   
 volumeMounts:   
 -   
 name: "mongodb-persistent-storage"  
 mountPath: "/data/db"  
   
 volumes:   
 -   
 name: "mongodb-persistent-storage"  
 persistentVolumeClaim:   
 claimName: "mongodb-pvc"

a. Under volumeMounts, the mountPath should be “/data/db” only. Any other name will result in a loss of data if the pod restarts as the data is persisted on /data/db path only.

b. Under volumes, the claimName should be the same name that you have given in the pvc.yaml file. In my case, this is “mongodb-pvc”.

c. Under env variables, you need to mention: MONGO\_INITDB\_ROOT\_USERNAME and MONGO\_INITDB\_ROOT\_PASSWORD. The username and password can be anything of your choice.

Once the yaml for statefulset is created, we need to apply the deployment using the following command:

****kubectl apply -f statefulset.yaml -n <namespace>****

To check if the deployment has been created, run the following command.

****kubectl get statefulsets -n <namespace>****

****3.**** ****HeadLessService.yaml****

In StatefulSet, instead of the normal service, we need to create a headless service.

A headless service is ****a service with a service IP, but instead of load-balancing it will return the IPs of our associated pods****. This allows us to interact directly with the pods instead of a proxy. It is as simple as specifying the cluster IP as none.

---  
 apiVersion: "v1"  
 kind: "Service"  
 metadata:   
 name: "mongo-db"  
 namespace: "development"  
 labels:   
 app: "mongo-db"  
 spec:   
 ports:   
 -   
 name: "mongodb"  
 port: 27017  
 targetPort: 27017  
 clusterIP: "None"  
 selector:   
 app: "mongo-db"

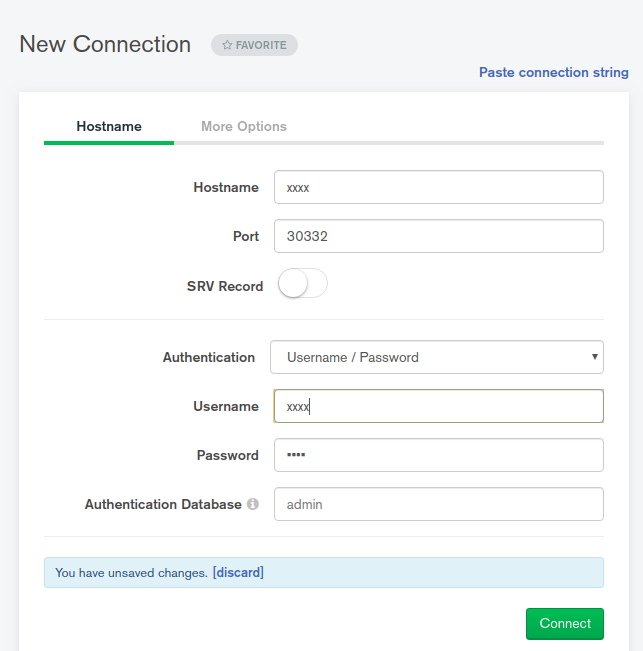
Once the yaml for headless service is created, we need to apply the headless service using the following command:

****kubectl apply -f service.yaml -n <namespace>****

****kubectl get svc -n <namespace>****

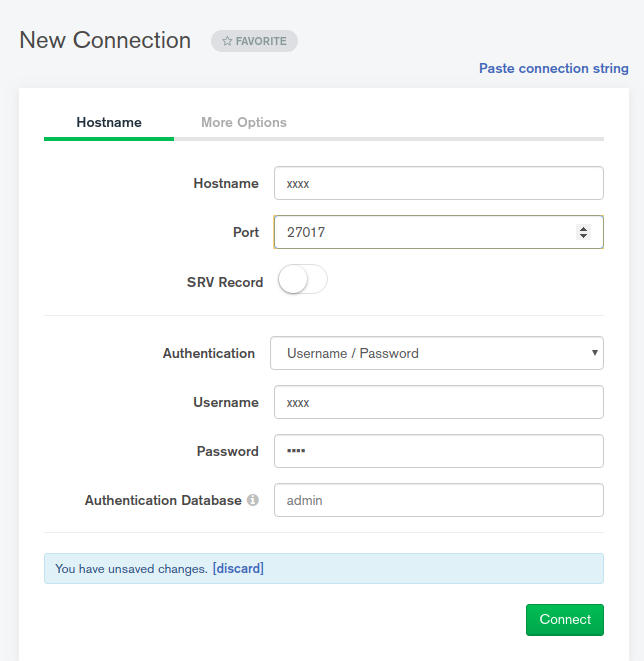
****Connecting the MongoDB pod with MongoDB Compass****

1. ****Connection for DeploymentSet****



Here, the hostname will be your IP address for the node and the username and password you have passed in the form of the env variable in the deployment.yaml file.

****2. Connection for StatefulSet****



Here, the IP address will be the IP address for the pod. This is unlike deploymentset, where we use the IP address for the node instead of the pod IP.

To check the pod IP address, run the following command:

kubectl get pod <pod\_name>-o wide -n <namespace>

If you want to use nodePort with StatefulSets, you can use the following configuration.

Here, instead of the headless service, we can create a normal service and add externalTrafficPolicy: “Local”.

---  
 apiVersion: "v1"  
 kind: "Service"  
 metadata:   
 name: "mongo-db"  
 namespace: "development"  
 labels:   
 app: "mongo-db"  
 spec:   
 externalTrafficPolicy: "Local"  
 ports:   
 -   
 name: "mongodb"  
 port: 27017  
 nodePort: 30332  
 type: "NodePort"  
 selector:   
 app: "mongo-db"

Now, we can access the MongoDB with the node IP address and nodePort, i.e., 30332.