# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology,Pune-37

*(An autonomous Institute of Savitribai Phule Pune University)*



**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | A |
| **Batch** | 2 |
| **GR-no** | 12111192 |
| **Rollno** | 2 |
| **Name** | Aakash Anil Matkar |

**Assignment 7 - Deploy a stateless/stateful application on Kubernetes cluster.**

**Theory:**

Tools Required:

* **Kubernetes ClusteR**
* **Kubectl**
* **DockeR**
* **A Stateless Application (e.g., Nginx server)**
* **A Stateful Application (e.g., MySQL database)**

Steps:

1. **Setup Kubernetes Cluster: Set up a Kubernetes cluster on your local machine or cloud provider. You can use tools like Minikube for local setup or services like Google Kubernetes Engine (GKE) for cloud setup.**
2. **Install Kubectl: Install** kubectl **which is a command line tool for controlling Kubernetes clusters.**
3. **Create Docker Images: Create Docker images for your stateless and stateful applications. For a stateless application, you can use an Nginx server. For a stateful application, you can use a MySQL database.**
4. **Deploy Stateless Application:**
   * **Write a deployment YAML file for the stateless application.**
   * **Use** kubectl create -f <nginx-deployment.yaml> **to create a deployment.**
   * **Verify the deployment using** kubectl get deployments**.**
5. **Deploy Stateful Application:**
   * **Write a StatefulSet YAML file for the stateful application.**
   * **Use** kubectl create -f <mysql-statefulset.yaml> **to create a StatefulSet.**
   * **Verify the StatefulSet using** kubectl get statefulsets**.**
6. **Testing:**
   * **Test the stateless application by sending HTTP requests and checking the responses.**
   * **Test the stateful application by writing and reading data, and verifying if the data persists.**
7. **Cleanup: Delete the deployments and StatefulSets once done with testing.**

Deliverables:

* **Dockerfiles for stateless and stateful applications.**
* **Deployment YAML file for stateless application.**
* **StatefulSet YAML file for stateful application.**
* **Screenshots of the** kubectl get deployments **and** kubectl get statefulsets **commands output.**
* **A brief report explaining the steps performed, issues encountered, and how they were resolved.**

**How to Create a StatefulSet in Kubernetes**

In this section, you will learn how to create a Pod for MySQL database using the StatefulSets controller.

**Create a Secret**

|  |
| --- |
| To start, you will need to create a Secret for the MySQL application that will store sensitive information, such as usernames and passwords. Here, I am creating a simple Secret. However, in a production environment, using the HashiCorp Vault is recommended. Use the following code to create a Secret for MySQL: |
| apiVersion: v1 kind: Secret metadata:  name: mysql-password type: opaque stringData:  MYSQL\_ROOT\_PASSWORD: password |
| Save the code using the file name mysql-secret.yaml and execute the code using the following command on your Kubernetes cluster: |
| kubectl apply -f mysql-secret.yaml |
| Get the list of Secrets: |
| kubectl get secrets |
| **Create a MySQL StatefulSet Application**  Before creating a StatefulSet application, check your volumes by getting the persistent volume list: |
| kubectl get pv  NAME CAPACITY ACCESS MODES RECLAIM STATUS  pvc-e0567 10Gi RWO Retain  Bound |
| Next, get the persistent volume claim list: |

|  |
| --- |
| kubectl get pvc  NAME STATUS VOLUME  CAPACITY ACCESS  mysql-store-mysql-set-0 Bound pvc-e0567d43ffc6405b 10Gi RWO |
| Last, get the storage class list: |
| kubectl get storageclass  NAME PROVISIONER  RECLAIMPOLICY  linode-block-storage linodebs.csi.linode.com Delete  linode-block-storage-retain (default)  linodebs.csi.linode.com Retain |
| Then use the following code to create a MySQL StatefulSet application in the Kubernetes cluster: |
| apiVersion: apps/v1 kind: StatefulSet metadata:  name: mysql-set spec:  selector: matchLabels:  app: mysql serviceName: "mysql" replicas: 3 template:  metadata: labels:  app: mysql spec:  terminationGracePeriodSeconds: 10 containers:   * name: mysql image: mysql:5.7 ports:   + containerPort: 3306 volumeMounts:   + name: mysql-store mountPath: /var/lib/mysql   env:  - name: MYSQL\_ROOT\_PASSWORD  valueFrom: secretKeyRef:  name: mysql-password key: MYSQL\_ROOT\_PASSWORD  volumeClaimTemplates:  - metadata: |

Here are a few things to note:

name: mysql-store spec:

accessModes: ["ReadWriteOnce"] storageClassName: "linode-block-storage-retain" resources:

requests: storage: 5Gi

|  |
| --- |
| 1. The kind is a StatefulSet. kind tells Kubernetes to create a MySQL application with the stateful feature. 2. The password is taken from the Secret object using the secretKeyRef. 3. The Linode block storage was used in the volumeClaimTemplates. If you are not mentioning any storage class name here, then it will take the default storage class in your cluster. 4. The replication count here is 3 (using the replica parameter), so it will create three Pods named mysql-set-0, mysql-set-1, and mysql-set- 2.   Next, save the code using the file name mysql.yaml and execute using the following command: |
| kubectl apply -f mysql.yaml |
| Now that the MySQL Pods are created, get the Pods list: |
| kubectl get pods  NAME READY STATUS RESTARTS AGE  mysql-set-0 1/1 Running 0  142s  mysql-set-1 1/1 Running 0  132s  mysql-set-2 1/1 Running 0  120s |
| **Create a Service for the StatefulSet Application**  Now, create the service for the MySQL Pod. Do not use the load balancer service for a stateful application, but instead, create a headless service for the MySQL application using the following code: |
| apiVersion: v1 kind: Service metadata:  name: mysql labels:  app: mysql spec:  ports:  - port: 3306 |

|  |
| --- |
| clusterIP: None selector:  app: mysql |
| Save the code using the file name mysql-service.yaml and execute using the following command: |
| kubectl apply -f mysql-service.yaml |
| Get the list of running services: |
| kubectl get svc |
| **Create a Client for MySQL**  If you want to access MySQL, then you will need a MySQL client tool. Deploy a MySQL client using the following manifest code: |
| apiVersion: v1 kind: Pod metadata:  name: mysql-client spec:  containers:  - name: mysql-container image: alpine  command: ['sh','-c', "sleep 1800m"]  imagePullPolicy: IfNotPresent |
| Save the code using the file name mysql-client.yaml and execute using the following command: |
| kubectl apply -f mysql-client.yaml |
| Then enter this into the MySQL client: |
| kubectl exec --stdin --tty mysql-client -- sh |
| Finally, install the MySQL client tool: |
| apk add mysql-client |
| **Access the MySQL Application Using the MySQL Client**  Next, access the MySQL application using the MySQL client and create databases on the Pods.  If you are not already in the MySQL client Pod, enter it now: |
| kubectl exec -it mysql-client /bin/sh |
| To access MySQL, you can use the same standard MySQL command to connect with the MySQL server: |
| mysql -u root -p -h host-server-name |

|  |
| --- |
| For access, you will need a MySQL server name. The syntax of the MySQL server in the Kubernetes cluster is given below: |
| stateful\_name-ordinal\_number.mysql.default.svc.cluster.local  #Example  mysql-set-0.mysql.default.svc.cluster.local |
| Connect with the MySQL primary Pod using the following command. When asked for a password, enter the one you made in the “Create a Secret” section above. |
| mysql -u root -p -h mysql-set-  0.mysql.default.svc.cluster.local |
| Next, create a database on the MySQL primary, then exit: |
| create database erp;  exit; |
| Now connect the other Pods and create the database like above: |
| mysql -u root -p -h mysql-set- 1.mysql.default.svc.cluster.local  mysql -u root -p -h mysql-set- 2.mysql.default.svc.cluster.local |

# Conclusion

In this lab we learned about Kubernetes’s two main controllers for creating Pods: Deployments and StatefulSets. A Deployment object is well suited for stateless applications, and the StatefulSets controller is well suited for stateful applications. If you are planning to deploy stateful applications, such as MySQL and Oracle, then you should use the StatefulSets controller instead of the Deployment object.

The StatefulSets controller offers an ordinal number feature for each Pod starting from zero. This helps stateful applications easily set up a primary-replica architecture, and if a Pod dies, a new Pod is re-created using the same name. This is a very useful feature and does not break the chain of stateful application clusters. If you are scaling down, then it deletes in the reverse order.

Use the StatefulSets controller in the Kubernetes cluster for deploying stateful applications, such as Oracle, MySQL, Elasticsearch, and MongoDB. While cloning and syncing data must still be completed manually, StatefulSets go a long way in easing the complexity involved in deploying stateful applications.