## Activity 4.1 - Deploying webserver in Kubernetes Cluster using namespaces

1. Create a new directory say my-webserver inside the k8s-workspace directory and move to that directory.

mkdir my-webserver && cd my-webserver

1. Create a new file say my-webserver-pod.yml and add the below content to it -

apiVersion: v1 kind: Namespace metadata:

name: development

**---**

apiVersion: v1 kind: Pod metadata:

name: nginx-pod namespace: development labels:

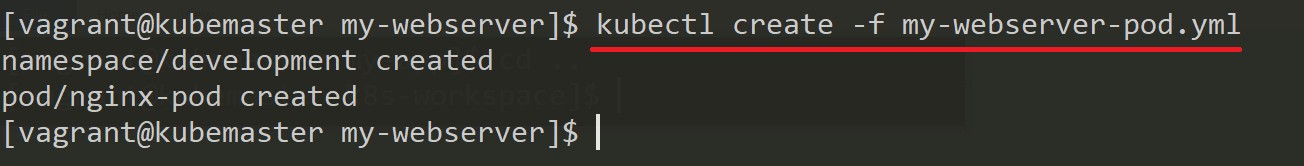
app: webserver spec:

containers:

- name: nginx-container image: nginx:alpine ports:

- containerPort: 80 protocol: TCP

1. To create the pod using the above given yaml file execute the below command -



kubectl create -f my-webserver-pod.yml

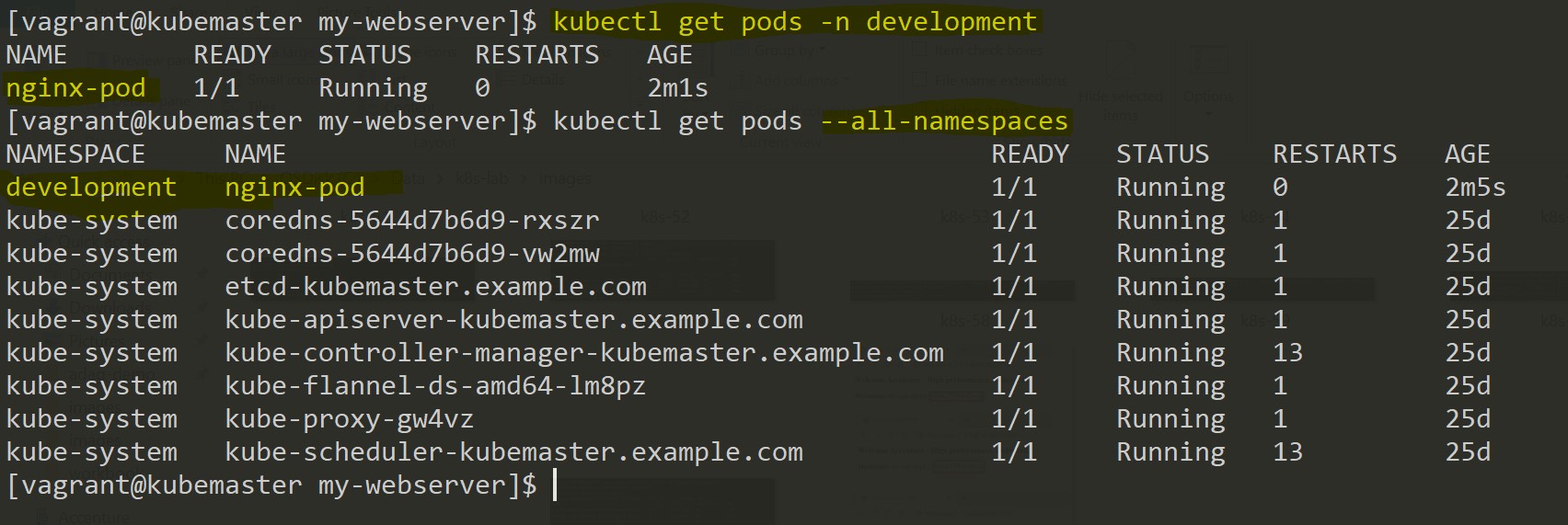
1. To view the existing namespaces in the cluster use the command -

kubectl get namespaces OR

kubectl get ns



1. To view the Pod in development namespace use the command -



kubectl get pods -n development

*# to view pods from all namespaces use the command*

kubectl get pods --all-namespaces

1. Now we will add a new replication controller object that will ensure that desire number of replicas are always present. Create a new file in the same working directory say my-webserver-rc.yml

apiVersion: v1

kind: ReplicationController metadata:

name: nginx-rc

namespace: development *# specify the namespace here*

spec:

replicas: 3 *# number of replicas to be present*

template: metadata:

name: nginx-rc-pod labels:

app: webserver *# matches with the previous created pod*

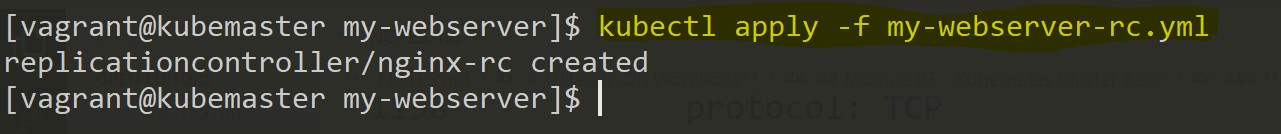
spec:

containers:

- name: nginx-container image: nginx:alpine ports:

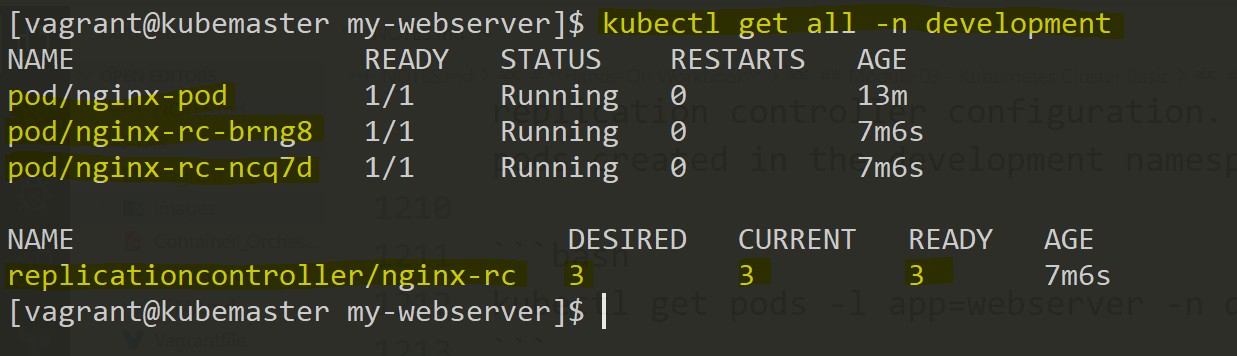
- containerPort: 80 protocol: TCP

1. To add this replication controller object execute the command -



kubectl apply -f my-webserver-rc.yml

1. As the desired state of webserver pods is 3 based on replication controller configuration. Check the resources created in the development namespace using the command -

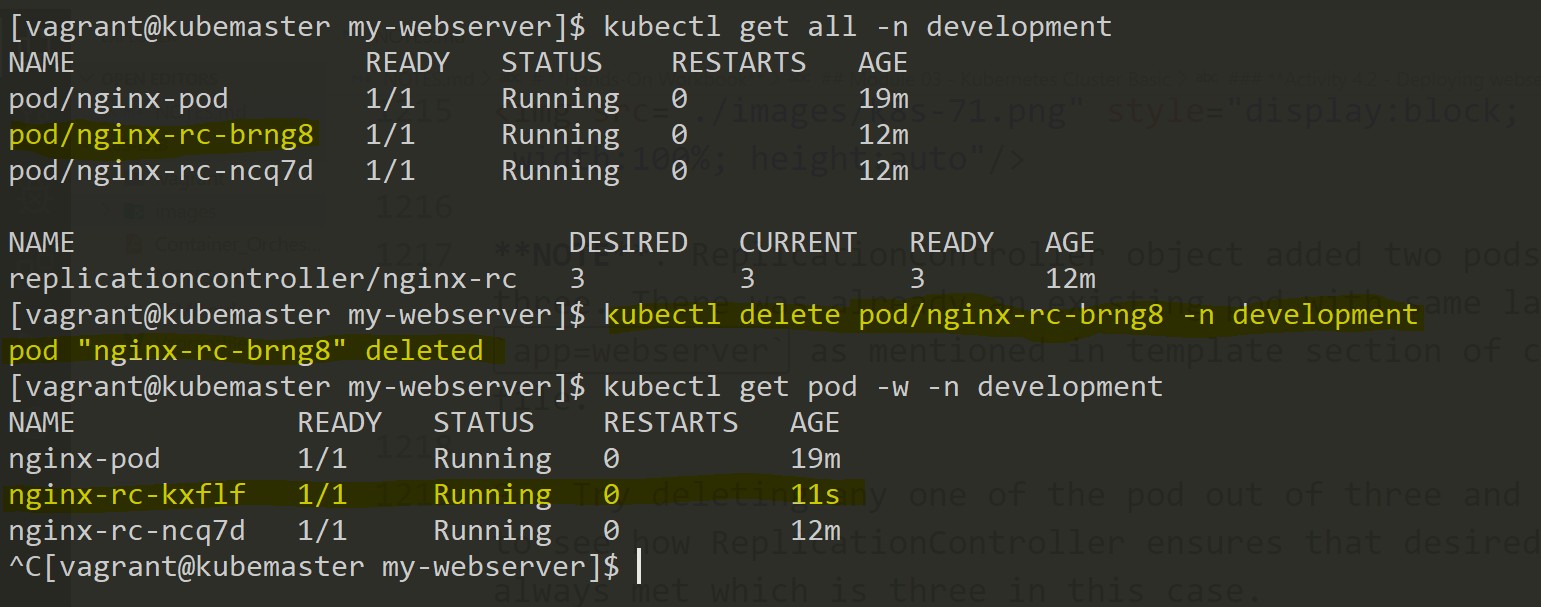


kubectl get all -n development

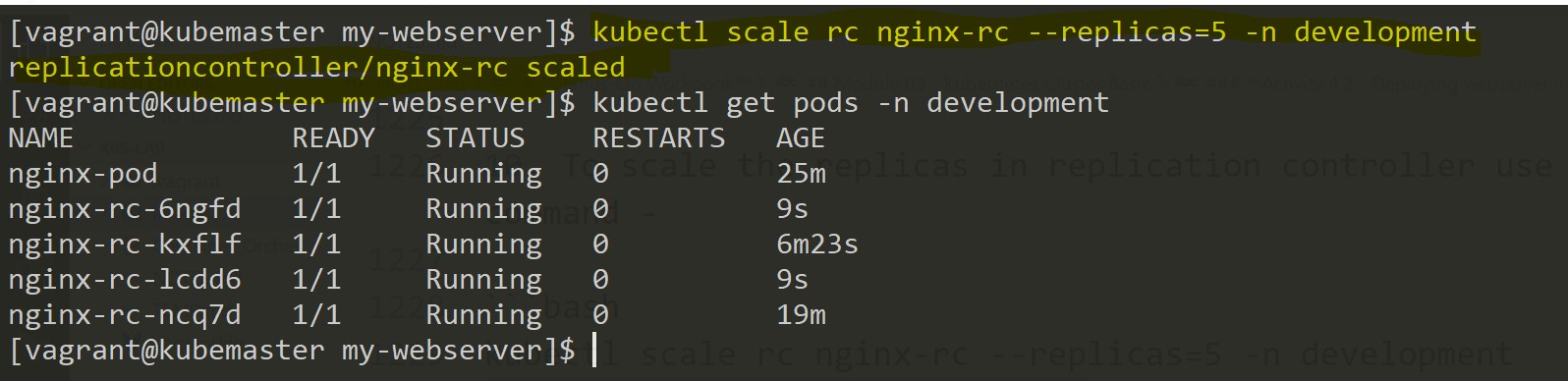
**NOTE**: ReplicationController object added two pods and not three. There was already an existing pod with same label app=webserver as mentioned in template section of configuration file.

1. Try deleting any one of the pod out of three and apply a watch to see how ReplicationController ensures that desired state is always met which is three in this case.

kubectl delete pod/<pod-name> -n development kubectl get pods -n development



1. To scale the replicas in replication controller use the command -



kubectl scale rc nginx-rc --replicas=5 -n development kubectl get pods -n development

1. Create a new my-webserver-svc.yml file in the same working directory and add the below content to it.

apiVersion: v1 kind: Service metadata:

name: nginx-svc namespace: development

spec:

type: NodePort ports:

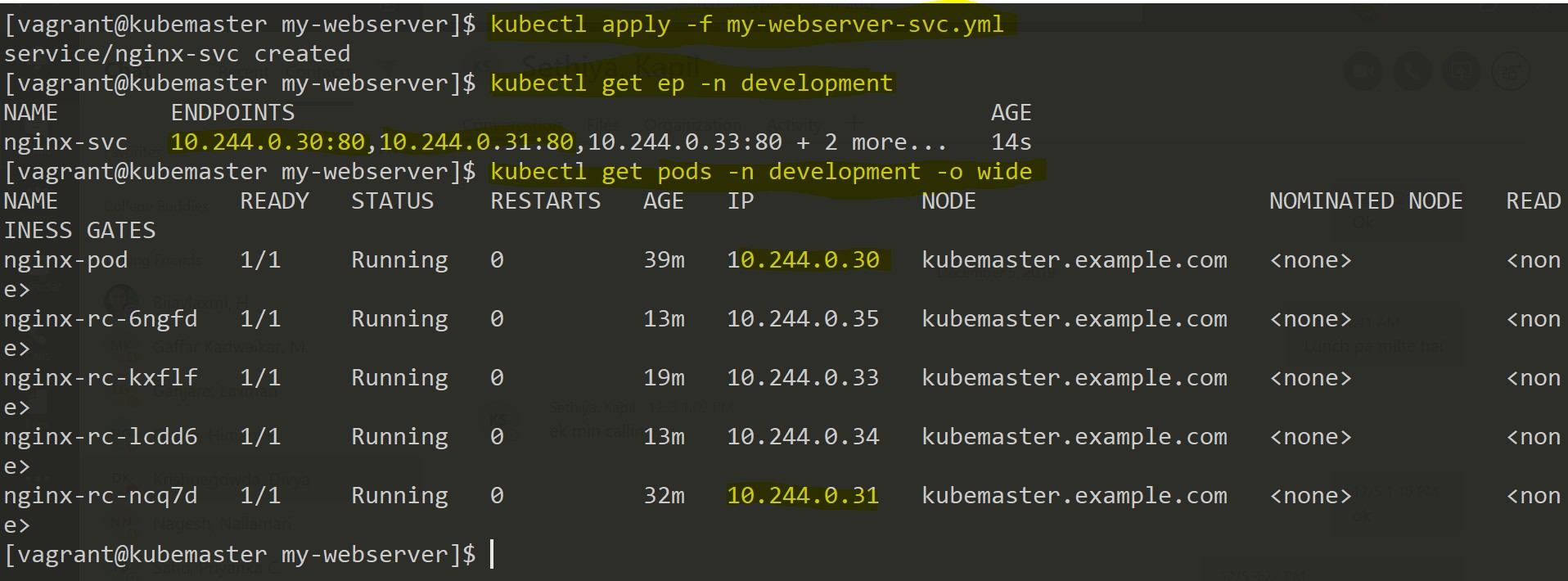
- port: 80 selector:

app: webserver

1. To create the service execute the below command -

kubectl apply -f my-webserver-svc.yml

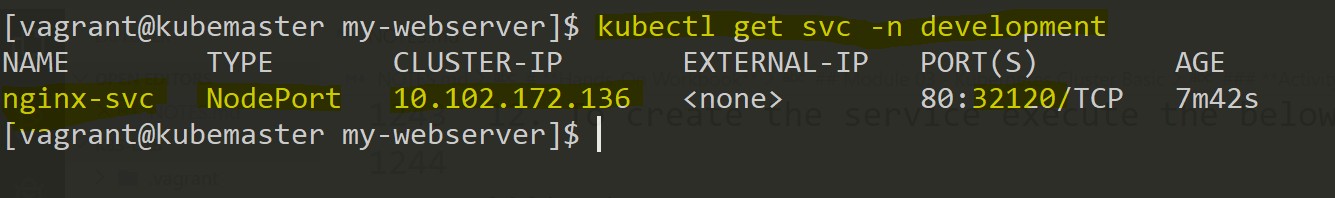
1. To view the service endpoints use the below command



kubectl get ep -n development

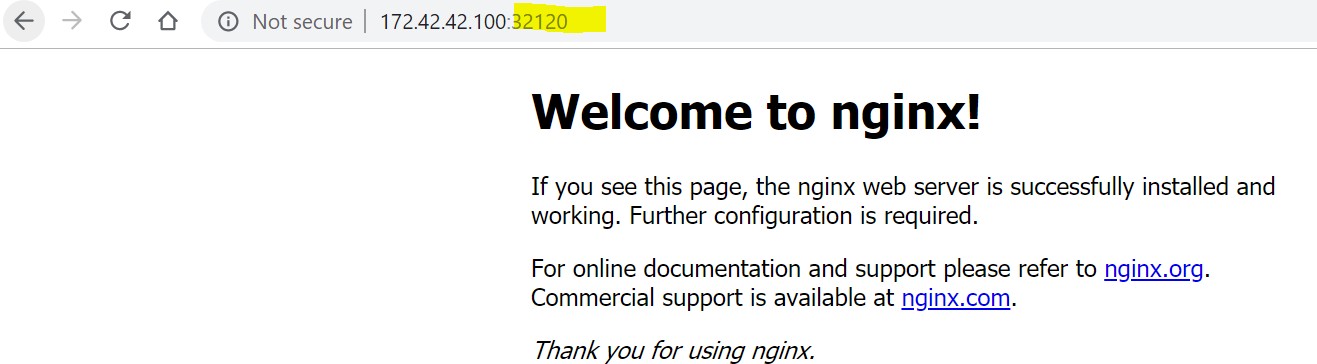
kubectl get pods -n development -o wide

1. To view the service created use the below command and note the port number:



kubectl get svc -n development

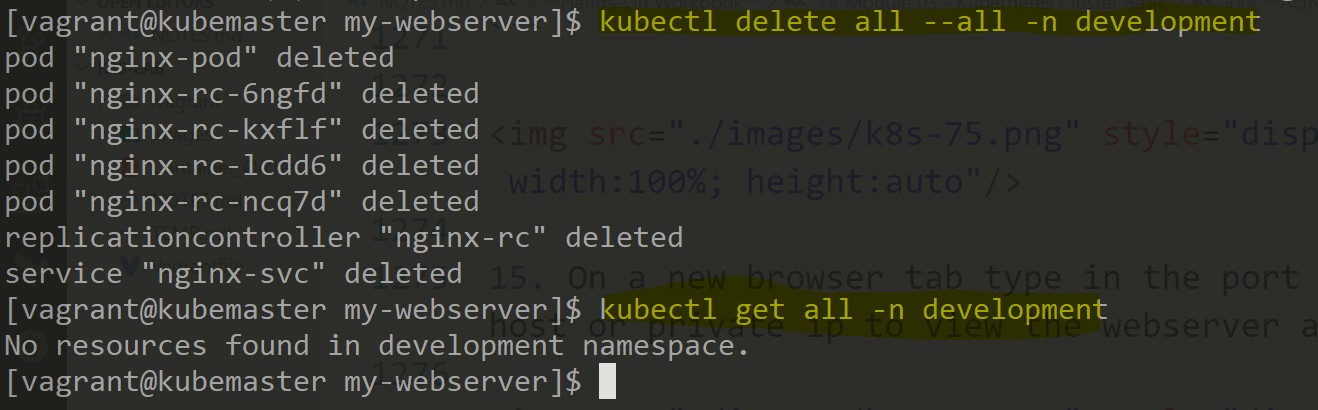
1. On a new browser tab type in the port number along with your host or private ip to view the webserver application:



1. To delete all the resources created in the development namespace execute the command -

kubectl delete all --all -n development kubectl get all -n development

1. Move back to the k8s-workspace directory using the command -



cd ..

## Activity 4.3 - Deploying MySQL onto Kubernetes Cluster with Secret and Persistent Volumes

1. Create a new working directory say my-db and move to that directory

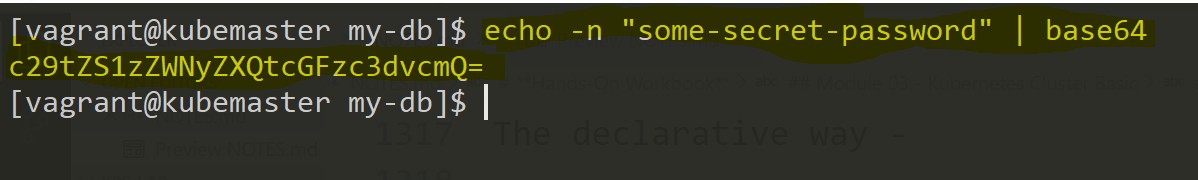
mkdir my-db && cd my-db

1. We will create a secret to provide the MySQL server root password. There are two ways in which this could be achieved by using command given below or writing configuration more declarative way.

The imperative way -

kubectl create secret generic mysql-secret --from-literal=password=some-secret- password -n development

The declarative way -



*# first convert the text into base64 encoding using the command*

echo -n "some-secret-password" | base64

apiVersion: v1 kind: Secret metadata:

name: mysql-secret namespace: development labels:

app: mysql data:

password: c29tZS1zZWNyZXQtcGFzc3dvcmQ=

1. We will create a new PersistentVolume say with the name of mysql-pv of type hostpath using the below configuration -

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-pv namespace: development labels:

app: mysql spec:

storageClassName: local capacity:

storage: 8Gi accessModes:

- ReadWriteOnce hostPath:

path: /mnt/data

1. To create a new PersistentVolumeClaim say with the name of mysql-pvc that will be used by pod to claim volume for storing the data generated by database server use the below configuration.

apiVersion: v1

kind: PersistentVolumeClaim metadata:

name: mysql-pvc namespace: development labels:

app: mysql spec:

storageClassName: local accessModes:

- ReadWriteOnce resources:

requests: storage: 8Gi

1. Create a new file in the working directory say mysql-config.yml and combine all the configuration given in step 2, 3 and 4 together as -

**---**

apiVersion: v1 kind: Secret metadata:

name: mysql-secret namespace: development labels:

app: mysql data:

password: c29tZS1zZWNyZXQtcGFzc3dvcmQ=

**---**

apiVersion: v1

kind: PersistentVolume metadata:

name: mysql-pv namespace: development labels:

app: mysql spec:

storageClassName: local capacity:

storage: 8Gi accessModes:

- ReadWriteOnce hostPath:

path: /mnt/data

**---**

apiVersion: v1

kind: PersistentVolumeClaim metadata:

name: mysql-pvc namespace: development labels:

app: mysql spec:

storageClassName: local accessModes:

- ReadWriteOnce resources:

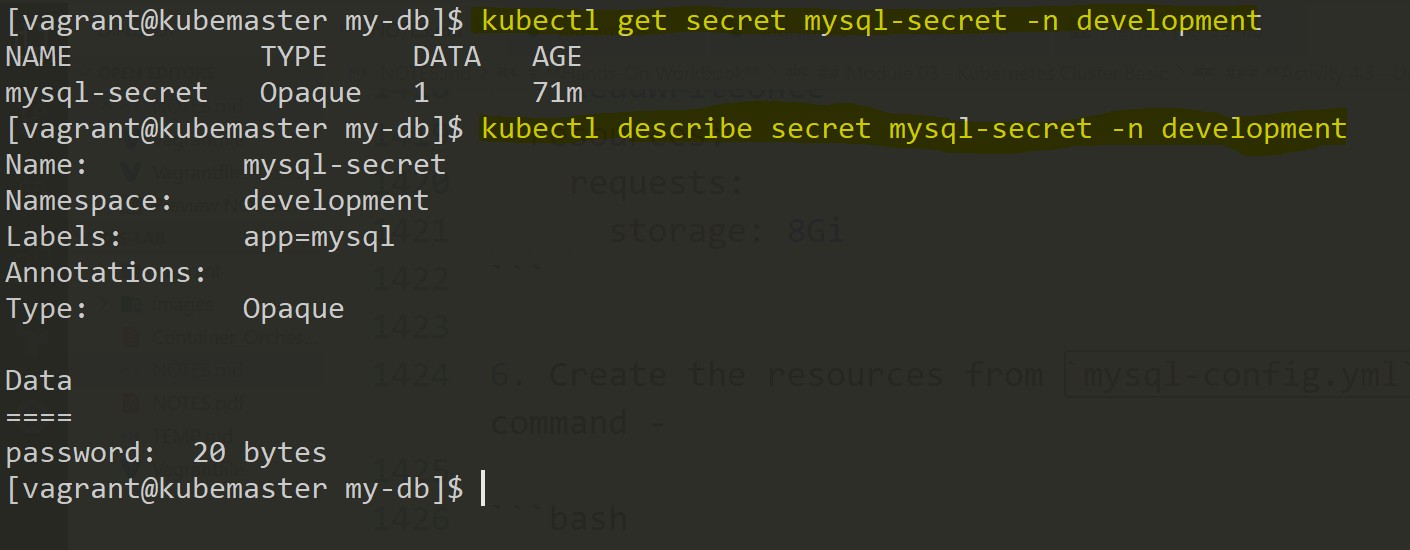
requests: storage: 8Gi

1. Create the resources from mysql-config.yml file using the command -

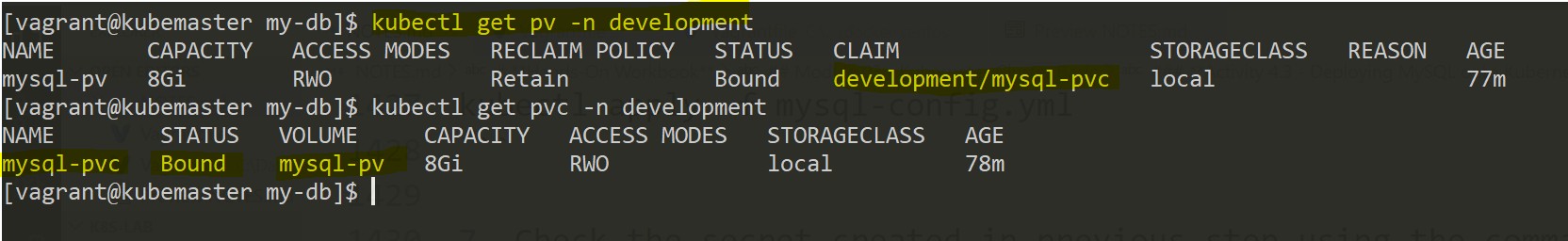
kubectl apply -f mysql-config.yml

1. Check the secret created in previous step using the command -

kubectl get secret mysql-secret -n development kubectl describe secret mysql-secret -n development



1. Check the persistent volume and claim created earlier which is bound to volume using the command -



kubectl get pv -n development kubectl get pvc -n development

1. Create a new file say mysql-pod.yml to launch a new pod. Add the below configuration to it -

apiVersion: v1 kind: Pod metadata:

name: mysql-pod namespace: development labels:

app: mysql spec:

containers:

- name: mysql image: mysql:5.6 env:

- name: MYSQL\_ROOT\_PASSWORD

valueFrom: secretKeyRef:

name: mysql-secret key: password

ports:

* containerPort: 3306 name: mysql

volumeMounts:

* name: mysql-storage

mountPath: /var/lib/mysql volumes:

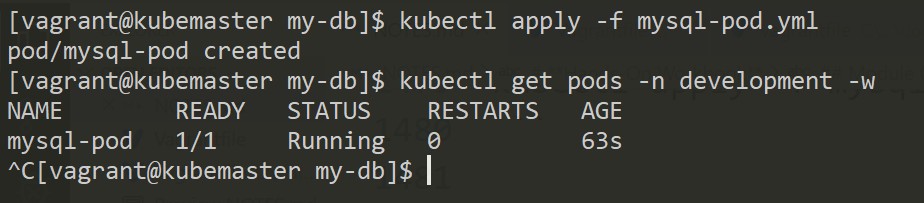
- name: mysql-storage persistentVolumeClaim:

claimName: mysql-pvc

1. To apply this configuration and launch a new mysql pod execute the below command -

kubectl apply -f mysql-pod.yml

1. Wait till the pod is fully up and running. Apply a watch to check the state of pod.



kubectl get pods -n development -w

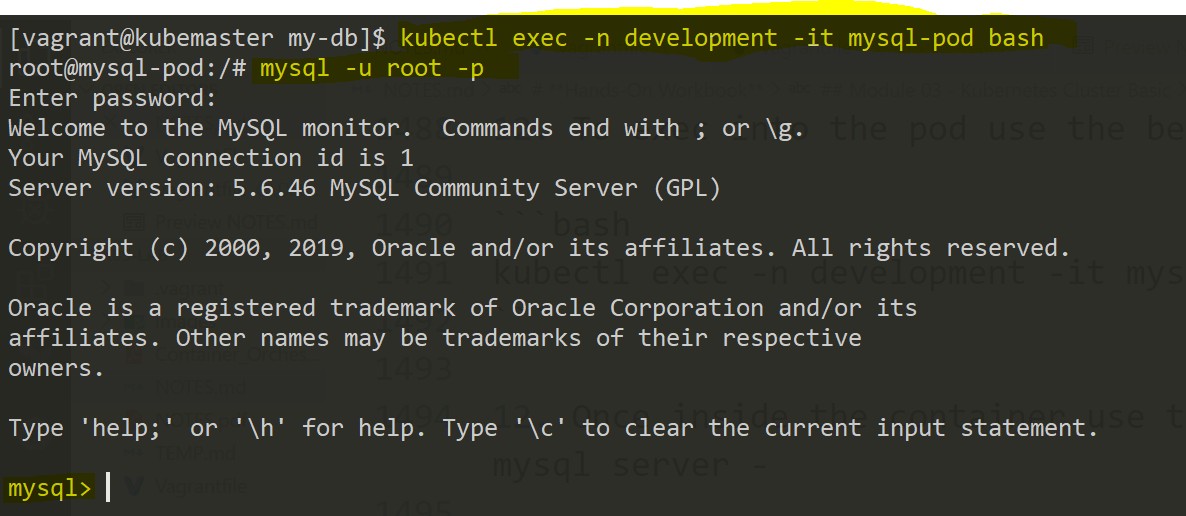
1. To exec into the pod use the below command -

kubectl exec -n development -it mysql-pod bash

1. Once inside the container use the below command to access the mysql server -

mysql -u root -p

Enter your password and get logged in to your mysql server.



1. Execute the below command to add some data to mysql database server

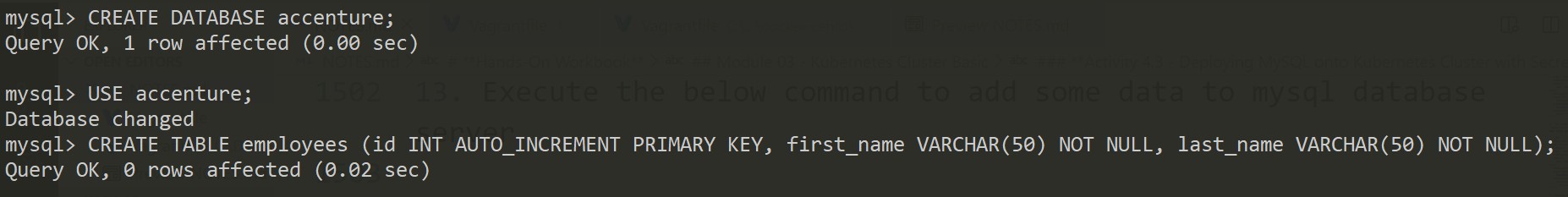
**CREATE DATABASE** accenture;

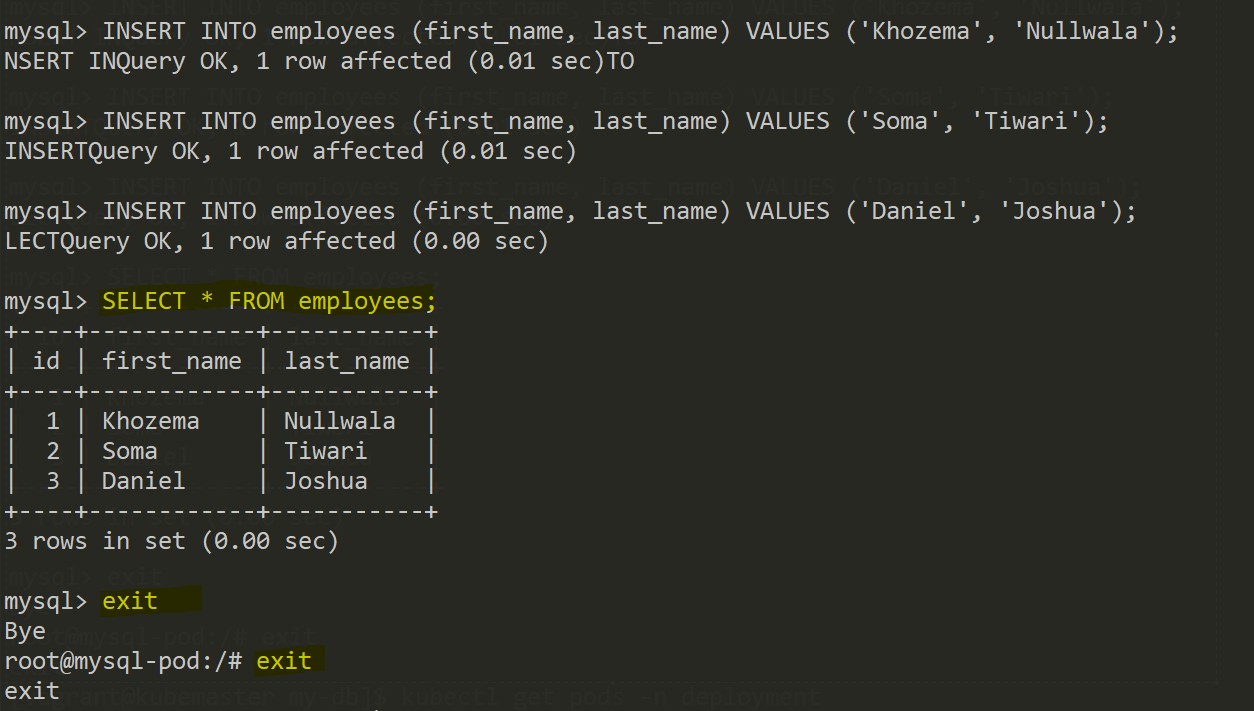
**USE** accenture;

**CREATE TABLE** employees (**id** INT AUTO\_INCREMENT PRIMARY **KEY**, first\_name VARCHAR(50)

**NOT** NULL, last\_name VARCHAR(50) **NOT** NULL);

**INSERT INTO** employees (first\_name, last\_name) **VALUES** ('Khozema', 'Nullwala'); **INSERT INTO** employees (first\_name, last\_name) **VALUES** ('Soma', 'Tiwari'); **INSERT INTO** employees (first\_name, last\_name) **VALUES** ('Daniel', 'Joshua'); **SELECT** \* **FROM** employees;





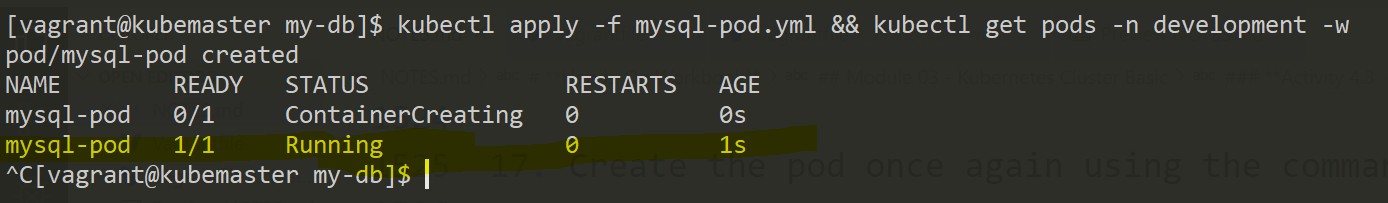
1. Exit from mysql server and come out of the exec container image by typing exit
2. Delete the mysql pod created using the command -



kubectl delete -f mysql-pod.yml

**NOTE**: The above command deletes the mysql server but the data generated is persisted that could be again attached to a new server.

1. Create the pod once again using the command -



kubectl apply -f mysql-pod.yml && kubectl get pods -n development -w

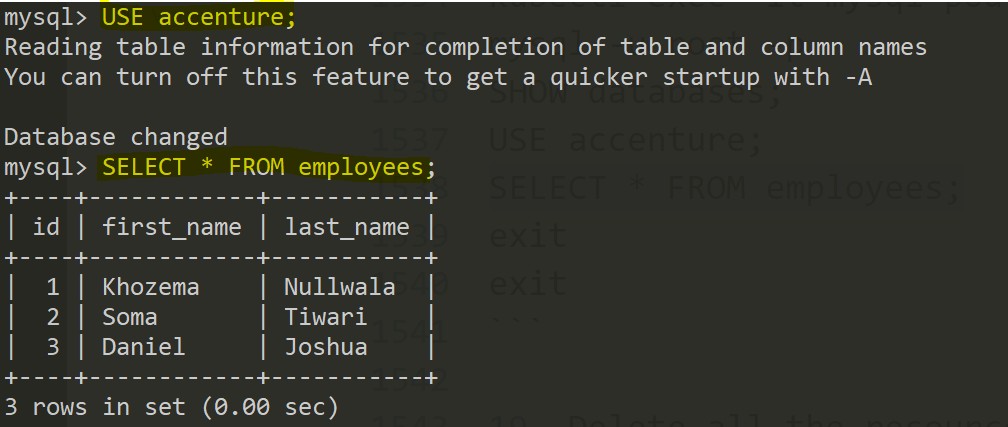
1. Execute the below command one by one and observe the data fetched through SQL query -

kubectl exec -it mysql-pod bash -n development mysql -u root -p

SHOW databases; USE accenture;

SELECT \* FROM employees; exit

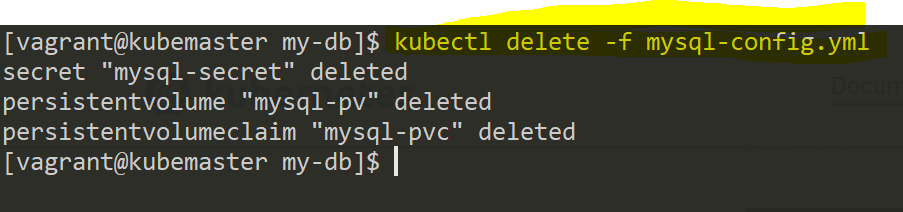
exit



1. Delete mysql pod created using the below command -

kubectl delete all -n development -l app=mysql

1. Delete the volume and secrets created using the command -



kubectl delete -f mysql-config.yml

1. Move to the k8s-workspace directory by using the command -

cd ..

## Activity 4.4 - Working with StatefulSet

1. 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.

1. Create a new directory say statefulset-demo inside the k8s-workspace directory.

mkdir statefulset-demo && cd statefulset-demo

1. Create a new nginx-statefulset.yml file and add the below content to it.

**---**

kind: PersistentVolume apiVersion: v1 metadata:

name: my-pv spec:

storageClassName: manual capacity:

storage: 1Gi

accessModes: ["ReadWriteOnce", "ReadWriteMany"] hostPath:

path: "/tmp/data"

**---**

kind: PersistentVolumeClaim apiVersion: v1

metadata: name: my-pvc

spec:

storageClassName: manual resources:

requests: storage: 1Gi

accessModes: ["ReadWriteOnce", "ReadWriteMany"]

**---**

apiVersion: v1 kind: Service metadata:

name: nginx labels:

app: nginx spec:

ports:

- port: 80

name: web clusterIP: None selector:

app: nginx

### ---

apiVersion: apps/v1 kind: StatefulSet metadata:

name: web spec:

selector: matchLabels:

app: nginx serviceName: "nginx" replicas: 2 template:

metadata: labels:

app: nginx spec:

terminationGracePeriodSeconds: 10 containers:

* name: nginx

image: k8s.gcr.io/nginx-slim:0.8 ports:

* + containerPort: 80 name: web

volumeMounts:

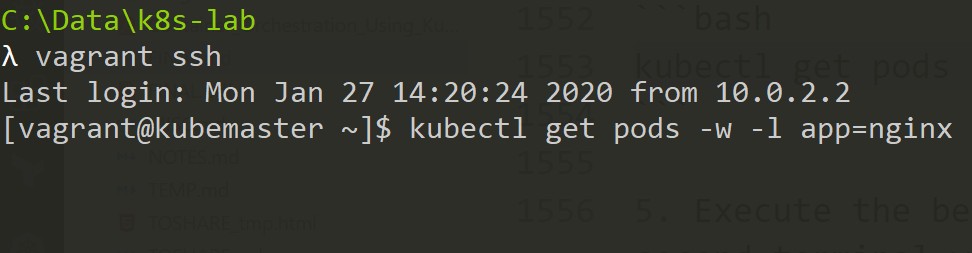
* + name: www

mountPath: /usr/share/nginx/html volumes:

* name: www persistentVolumeClaim:

claimName: my-pvc

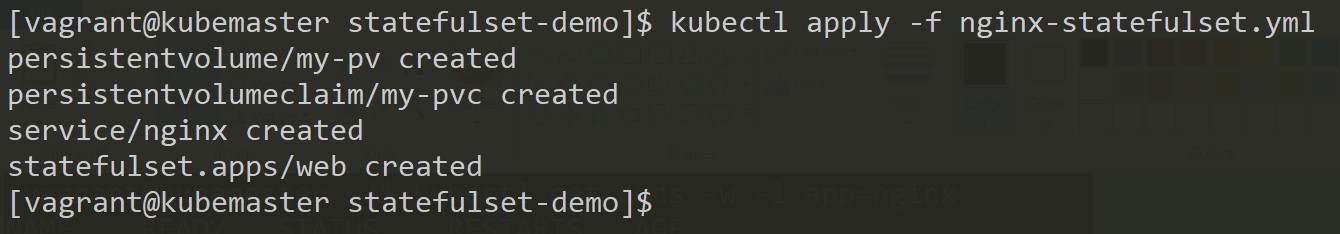
1. Open a new terminal window and log into your virtual machine using the command vagrant ssh like before and then use the command to add a watch on the pod creation of StatefulSet

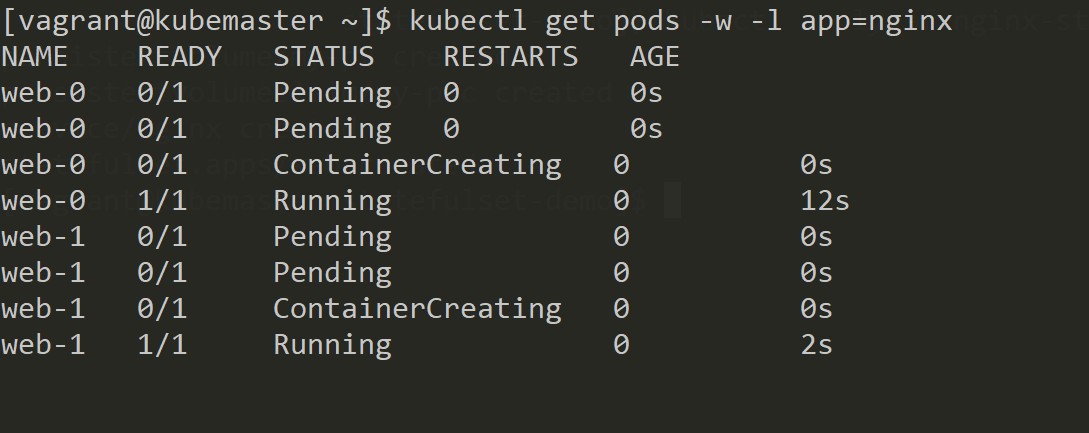


kubectl get pods -w -l app=nginx

1. Execute the below command to create the resource in second terminal with statefulset-demo as present working directory -

kubectl apply -f nginx-statefulset.yml

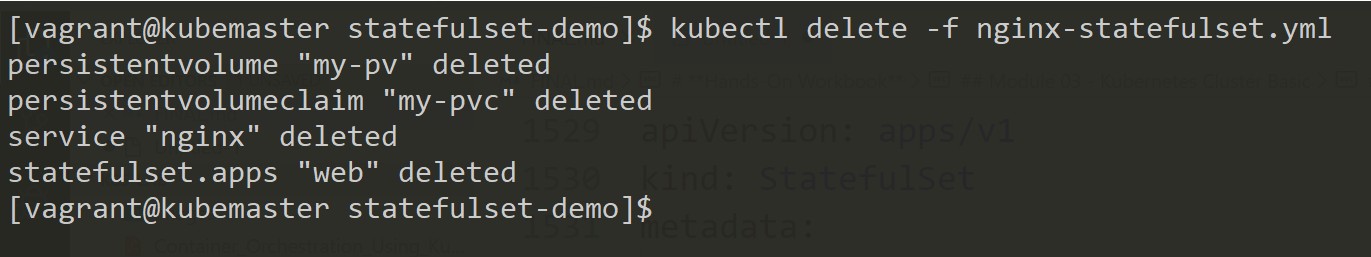




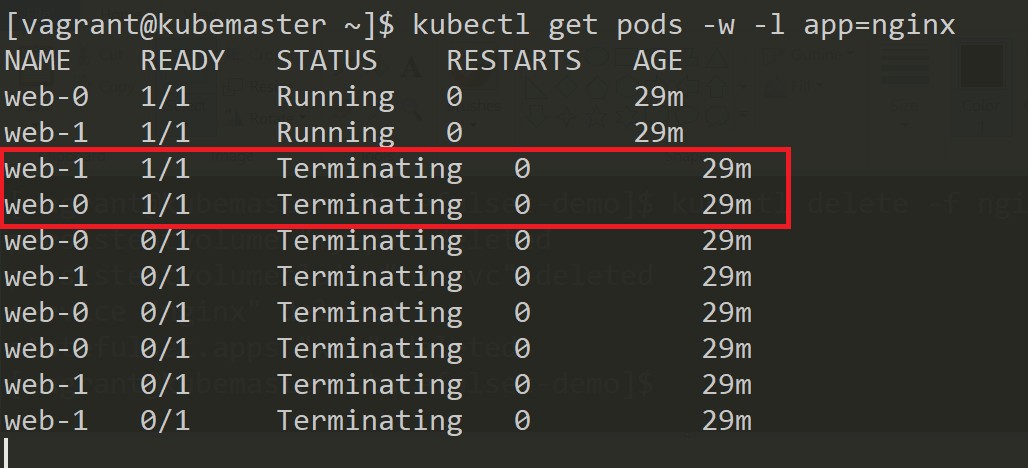
1. Notice that for a StatefulSet with N replicas, when Pods are being deployed, they are created sequentially, in order from {0..N-1}.

**NOTE**: web-1 Pod is not launched until the web-0 Pod is Running and Ready

1. Statefulset pods are deleted in reverse order with respect to its ordinal index, and waits for each to be completely shutdown before deleting the next.
2. Execute the below command to check the termination in other terminal in reverse order -



kubectl delete -f nginx-statefulset.yml



1. Move to the k8s-workspace directory by using the command -

cd ..

## Activity 4.5- Deploying multi-tier app on to Kubernetes cluster

Before continuing with this activity refer the [Docker Voting App Reference](https://github.com/dockersamples/example-voting-app). The different components of the Voting App is given below -

**Vote** — front end that enables a user to choose between a cat and a dog.

**Redis** — database where votes are stored.

**Worker** — service that get votes from Redis and store the results in a Postgres database.

**Db** — the Postgres database in which vote results are stored.

**Result** — front end, displaying the results of the vote.

* 1. Create a new working directory say my-multi-tier-app and move to that directory using the command -

mkdir my-multi-tier-app && cd my-multi-tier-app

* 1. To set up redis create a new file say redis-deploy.yml with the below content in it –

**---**

apiVersion: v1 kind: Service metadata:

labels: app: redis

name: redis namespace: development

spec:

clusterIP: None ports:

- name: redis-service port: 6379

targetPort: 6379 selector:

app: redis

**---**

apiVersion: apps/v1 kind: Deployment metadata:

name: redis labels:

app: redis namespace: development

spec:

replicas: 1 selector:

matchLabels: app: redis

template: metadata:

labels: app: redis

spec:

containers:

- name: redis

image: redis:alpine ports:

- containerPort: 6379 name: redis

* 1. To set up postgres as our db to store the vote result create a new file say postgres-deploy.yml with the below content in it -

### ---

apiVersion: v1

kind: PersistentVolume metadata:

name: postgres-pv namespace: development

spec:

accessModes:

- ReadWriteOnce capacity:

storage: 1Gi hostPath:

path: /mnt/postgres

### ---

apiVersion: v1

kind: PersistentVolumeClaim metadata:

name: postgres-pv-claim namespace: development

spec:

accessModes:

- ReadWriteOnce resources:

requests: storage: 1Gi

### ---

apiVersion: v1 kind: Service metadata:

labels: app: db

name: db

namespace: development spec:

clusterIP: None ports:

- name: db port: 5432

targetPort: 5432 selector:

app: db

### ---

apiVersion: apps/v1 kind: Deployment metadata:

name: db labels:

app: db

namespace: development spec:

replicas: 1 selector:

matchLabels: app: db

template: metadata:

labels: app: db

spec:

containers:

* name: db

image: postgres:9.4 env:

* + name: PGDATA

value: /var/lib/postgresql/data/pgdata ports:

* + containerPort: 5432 name: db

volumeMounts:

* + name: db-data

mountPath: /var/lib/postgresql/data volumes:

* name: db-data persistentVolumeClaim:

claimName: postgres-pv-claim

* 1. To set up result front-end create a new file say result-deploy.yml with the below content in it -

**---**

apiVersion: v1 kind: Service metadata:

name: result labels:

app: result namespace: development

spec:

type: NodePort

ports:

- port: 5001

targetPort: 80 name: result-service

selector: app: result

**---**

apiVersion: apps/v1 kind: Deployment metadata:

name: result labels:

app: result namespace: development

spec:

replicas: 1 selector:

matchLabels: app: result

template: metadata:

labels:

app: result spec:

containers:

- name: result

image: dockersamples/examplevotingapp\_result:before ports:

- containerPort: 80 name: result

* 1. To set up vote front-end create a new file say vote-deploy.yml with the below content in it -

**---**

apiVersion: v1 kind: Service metadata:

name: vote labels:

apps: vote namespace: development

spec:

type: NodePort ports:

- port: 5000

targetPort: 80 name: vote-service

selector: app: vote

**---**

apiVersion: apps/v1 kind: Deployment

metadata: name: vote labels:

app: vote namespace: development

spec:

replicas: 2 selector:

matchLabels: app: vote

template: metadata:

labels: app: vote

spec:

containers:

- name: vote

image: dockersamples/examplevotingapp\_vote:before ports:

- containerPort: 80 name: vote

* 1. To set up worker create a new file say worker-deploy.yml with the below content in it -

**---**

apiVersion: v1 kind: Service metadata:

labels:

apps: worker name: worker

namespace: development spec:

clusterIP: None selector:

app: worker

**---**

apiVersion: apps/v1 kind: Deployment metadata:

labels:

app: worker name: worker

namespace: development spec:

replicas: 1 selector:

matchLabels: app: worker

template: metadata:

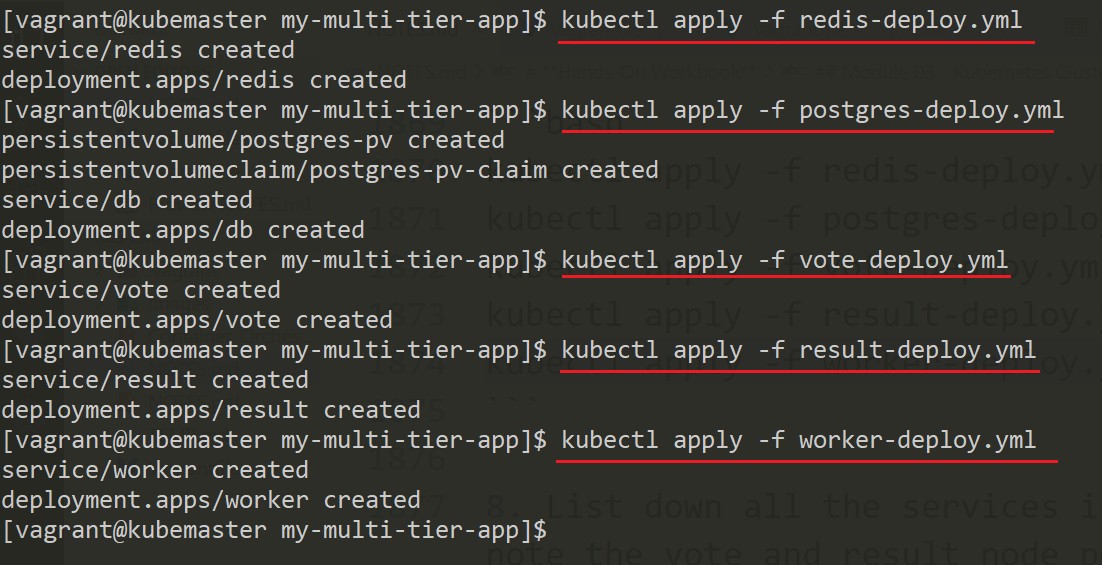
labels:

app: worker spec:

containers:

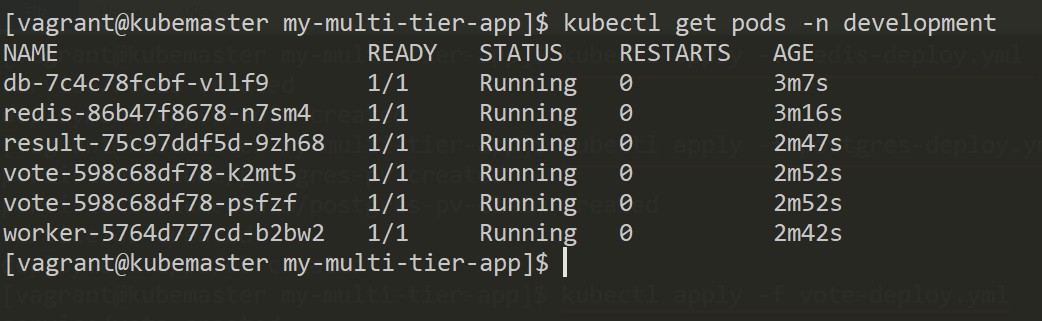
- image: dockersamples/examplevotingapp\_worker name: worker

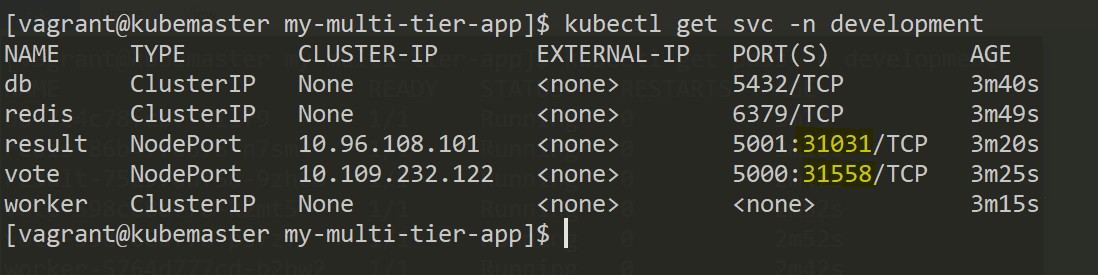
* 1. Execute the below command one by one to deploy our resources -



kubectl apply -f redis-deploy.yml kubectl apply -f postgres-deploy.yml kubectl apply -f vote-deploy.yml kubectl apply -f result-deploy.yml kubectl apply -f worker-deploy.yml

* 1. List down all the services in the development namespace and note the vote and result node port number.





* 1. On a new browser tab window access the voting front-end app and click on your vote

