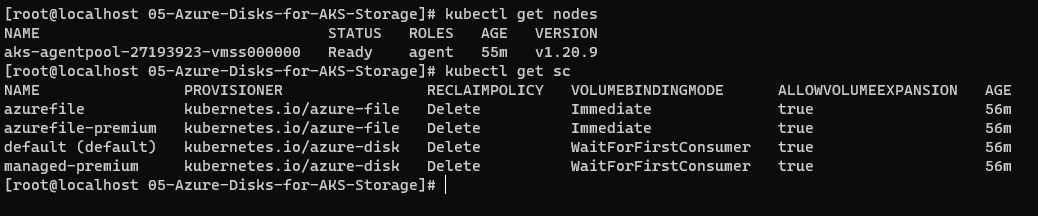
**Azure Disks**

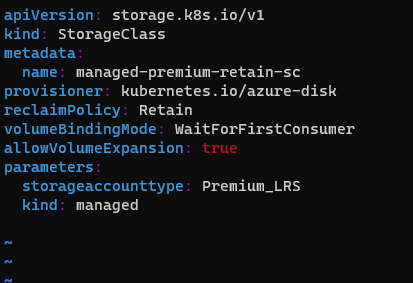
* Azure Disk Storage offers high performance, highly durable block storage for our mission and business-critical workloads.
* We can mount these volumes as devices on our Virtual Machines and Container instances.

**Azure provided default storage classes**



**Custom Storage Class**

* Create a custom storage class



**reclaimPolicy: Delete** –

With this setting, as soon as a PersistentVolumeClaim is deleted, it also triggers the removal of the corresponding PersistentVolume along with the Azure Disk.

**reclaimPolicy: retain** –

Disk is retained even when PVC is deleted (Recommended option)

**volumeBindingMode: Immediate** –

This setting implies that the PersistentVolumecreation, followed with the storage medium (Azure Disk in this case) provisioning is triggered as soon as the PersistentVolumeClaim is created.

**volumeBindingMode: WaitForFirstConsumer** –

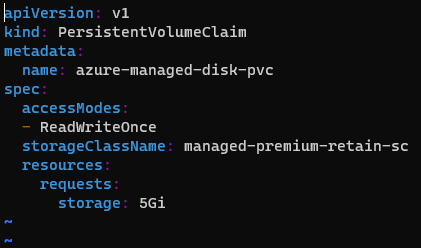
By default, the Immediate mode indicates that volume binding and dynamic provisioning occurs once the PersistentVolumeClaim is created. For storage backends that are topology-constrained and not globally accessible from all Nodes in the cluster, PersistentVolumes will be bound or provisioned without knowledge of the Pod's scheduling requirements. This may result in unschedulable Pods.

A cluster administrator can address this issue by specifying the WaitForFirstConsumer mode which will delay the binding and provisioning of a PersistentVolume until a Pod using the PersistentVolumeClaim is created. PersistentVolumes will be selected or provisioned conforming to the topology that is specified by the Pod's scheduling constraints.

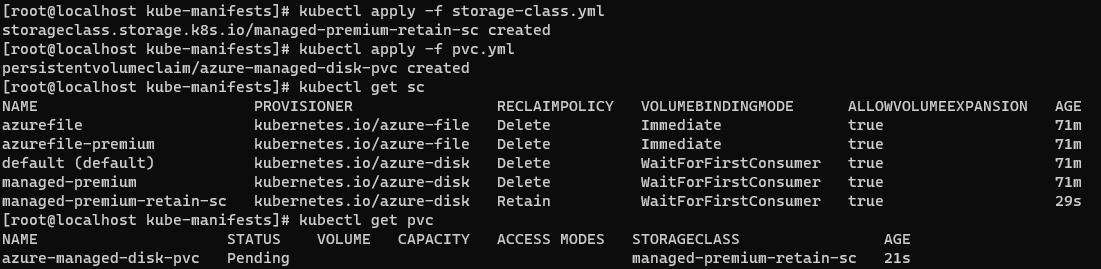
**storageaccounttype: Premium\_LRS** # or we can use Standard\_LRS

**kind: managed**  # Default is shared (Other two are managed and dedicated)

* Create a PVC –



* Provide storage class name managed-premium-retain-sc
* In resources we request 5Gi storage size.
* Deploy storage class and pvc

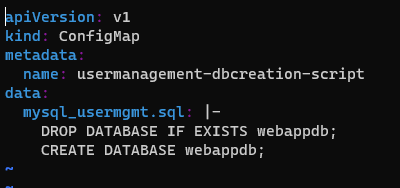


You can see the managed-premium-retain-sc storage class is created under the Kubernetes.io/azure disk and reclaimPolicy is Retain and Volumebinding is WaitForFirstConsumer.

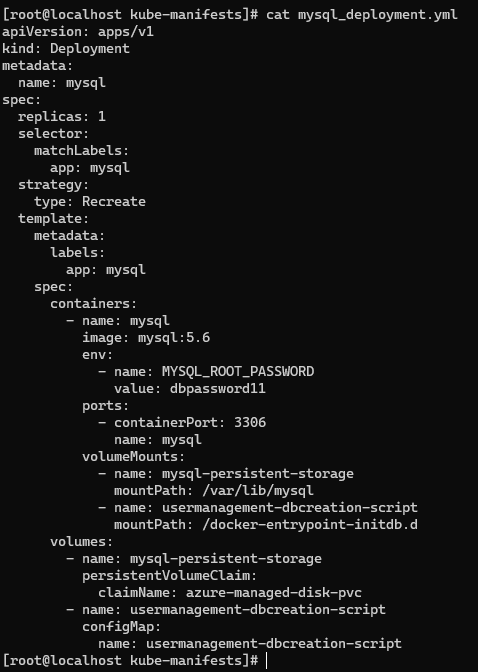
PVC azure-managed-disk-pvc is in pending state because it is waiting for your pod (WaitForFirstConsumer). Whenever the pod is provision at that time only this vpc is dynamically allocate PV for that respective pod. At that time what happens is that both will be created in the same physical location where they want to be place it. Where you will not have any major latency issue.

* Create a ConfigMaps

A ConfigMap is an API object used to store non-confidential data in key-value pairs. [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a [volume](https://kubernetes.io/docs/concepts/storage/volumes/).



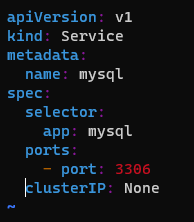
* This configmap is mounting to mysql-deployment as a volume mount.
* Whenever it is mounted to mysql-deployment and when mysql-deployment is created, it is going to create a database name webapp if it doesn’t exist.
* Reference link - <https://hub.docker.com/_/mysql>
* Create a mysql-deployment



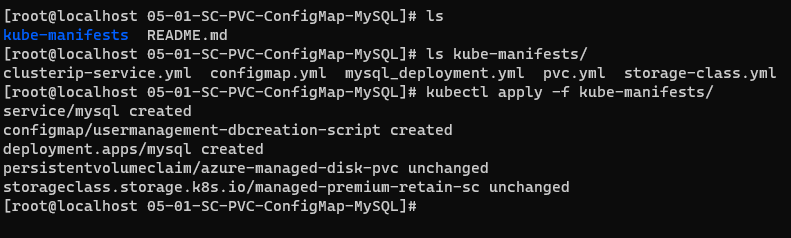
In this deployment pvc and configmap is mounted.

* Create MySQL ClusterIP Service manifest

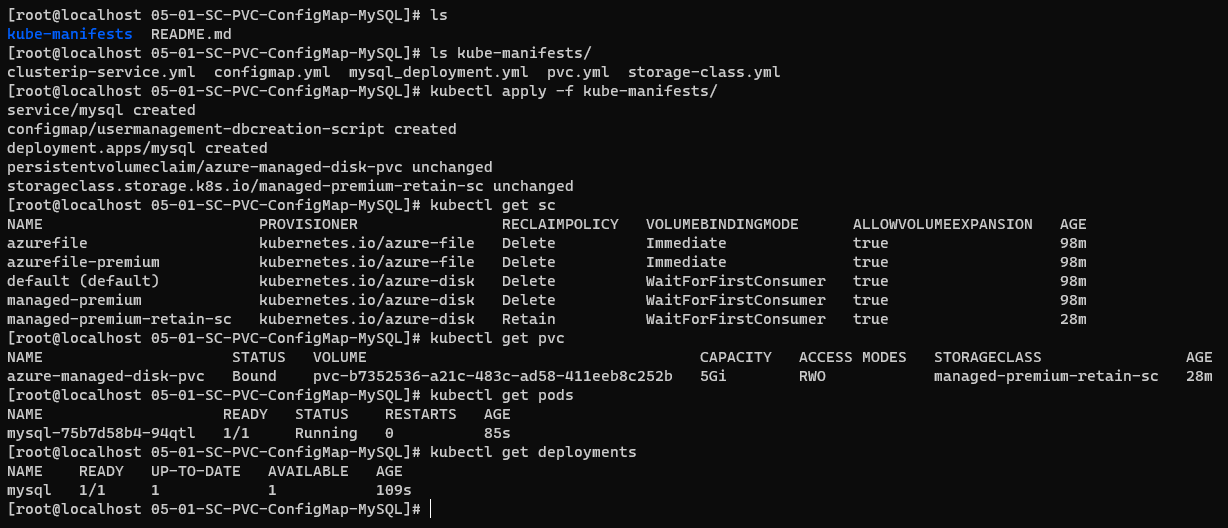
At any point of time, we are going to have only one mysql pod in this design so ClusterIP: None will use the Pod IP Address instead of creating or allocating a separate IP for MySQL Cluster IP service.

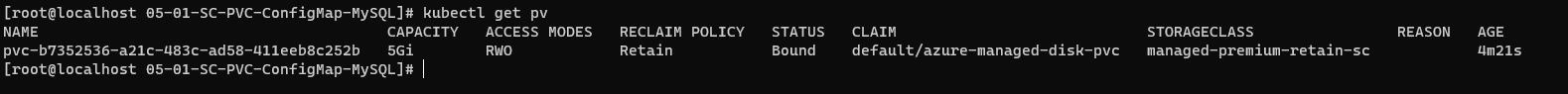


* Now Deploy all



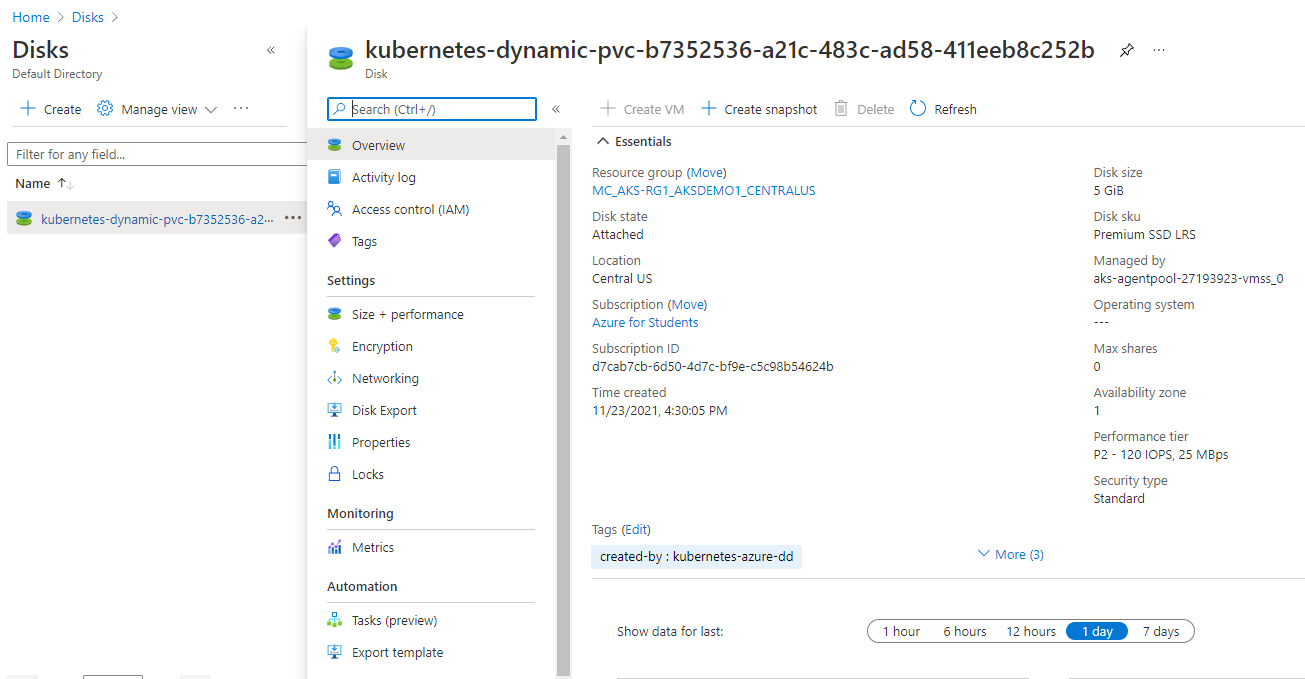
* Verify all



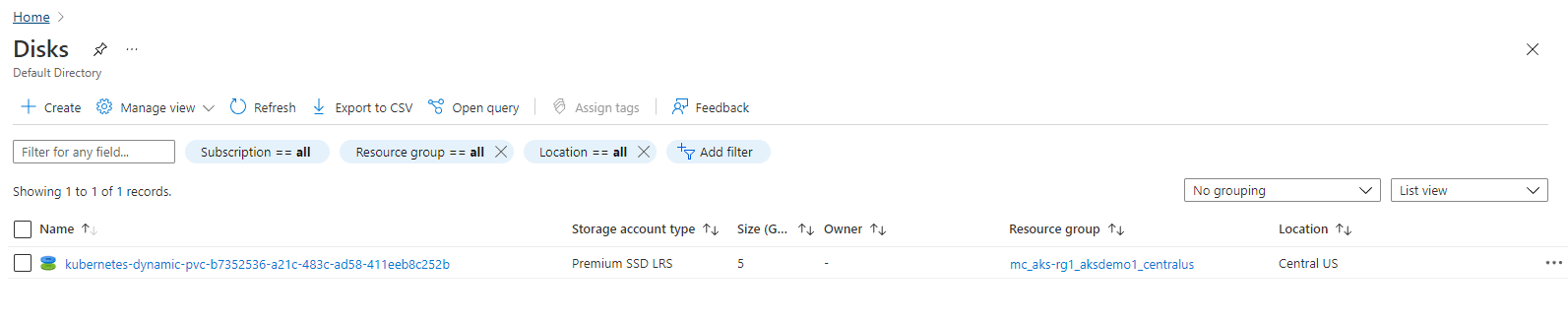


The PVC, PV is bounded

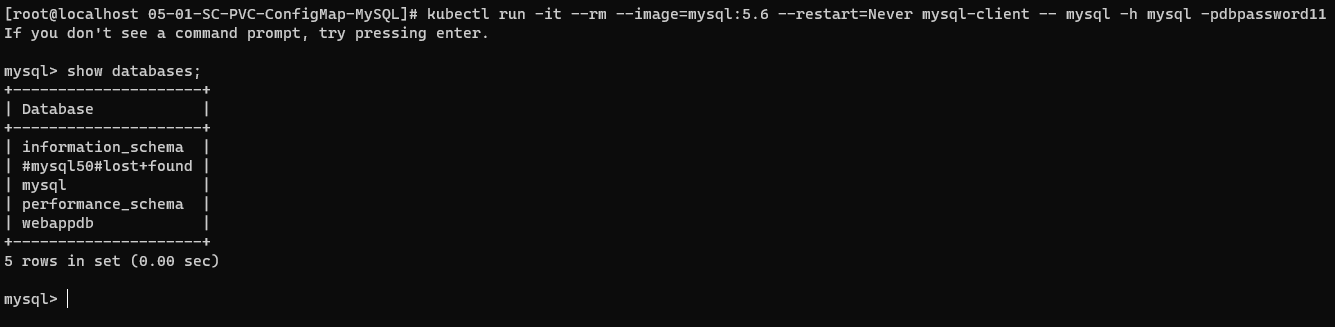
In azure Check the disk is created or not.



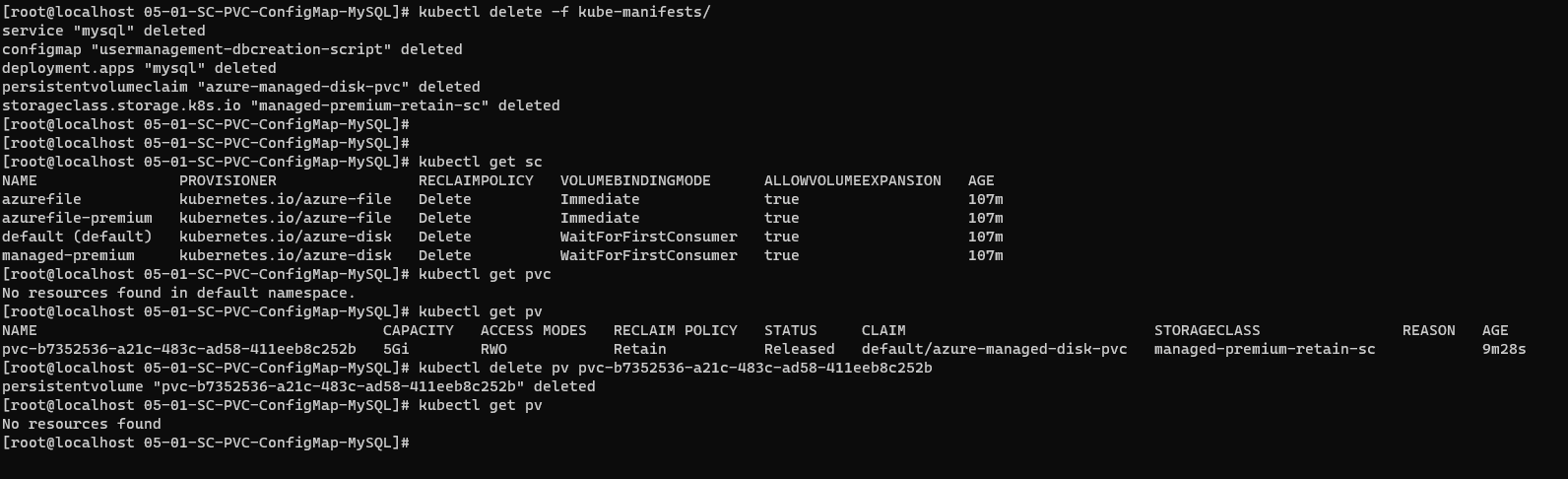
The disk resource group is tied to aks cluster. Which means when ever you delete AKS cluster then only this will be deleted. But whenever you try to delete all these objects also will not get deleted because of storage class concept is retained.



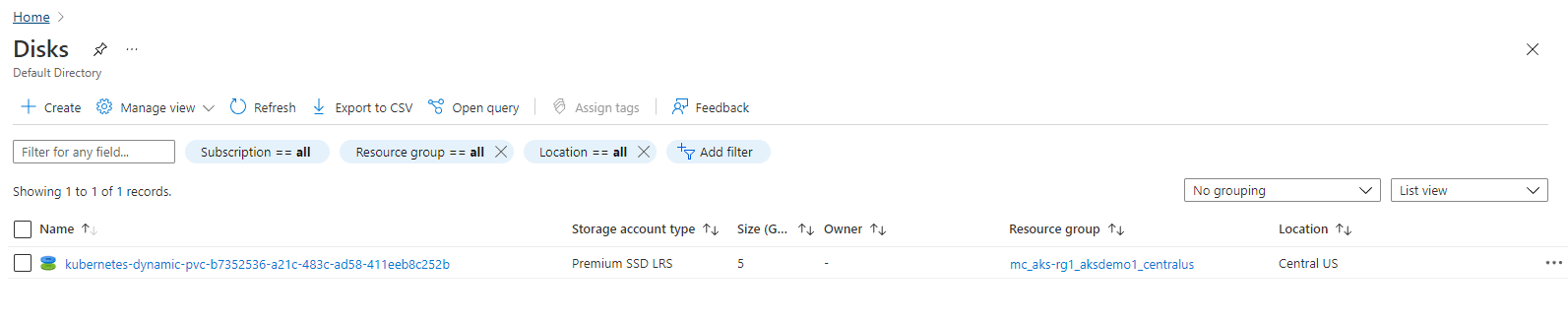
* Connect to MySQL Database and verify the database is created which we provided in ConfigMap



* Now delete all



And check the Azure disk



The disk still available.

This is a good option whenever you are using the custom storage classes to retain. SI by accidentally if you delete the pods or anything, still your disk is available with you.

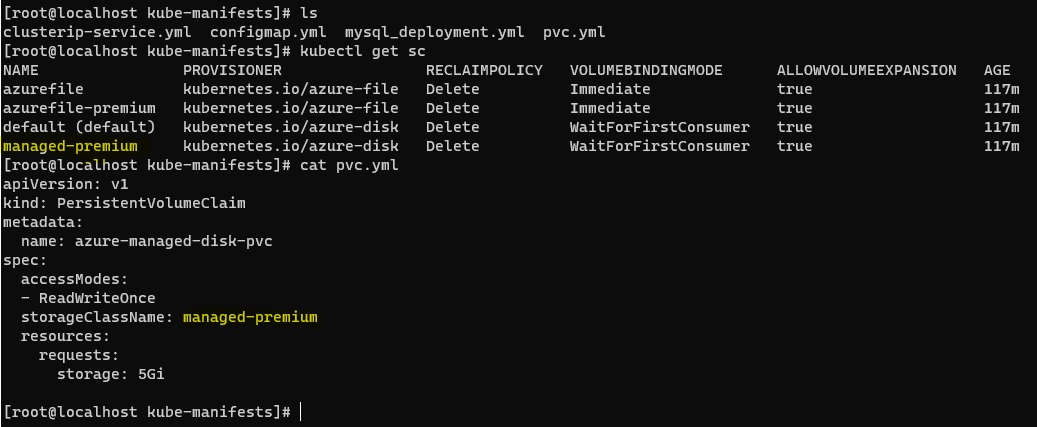
**AKS Provisioned Storage Class**

* Copy all templates from previous section
* Remove Storage Class Manifest
* Question-1: Why do we need to remove storage class Manifests?

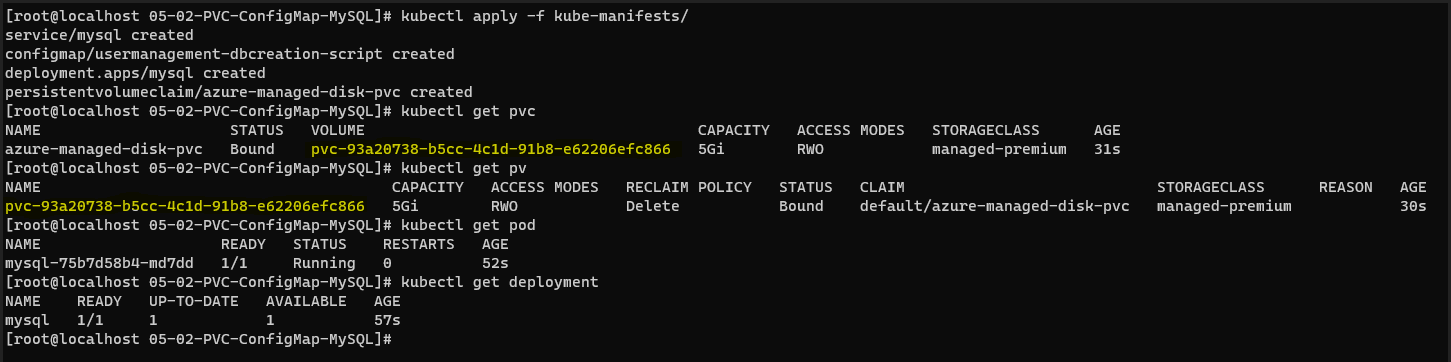
Ans = Azure AKS provisions two types of storage classes well in advance during the cluster creation process

* + managed-premium
  + default-
* We can leverage Azure AKS provisioned disk storage classes instead of what we created manually.

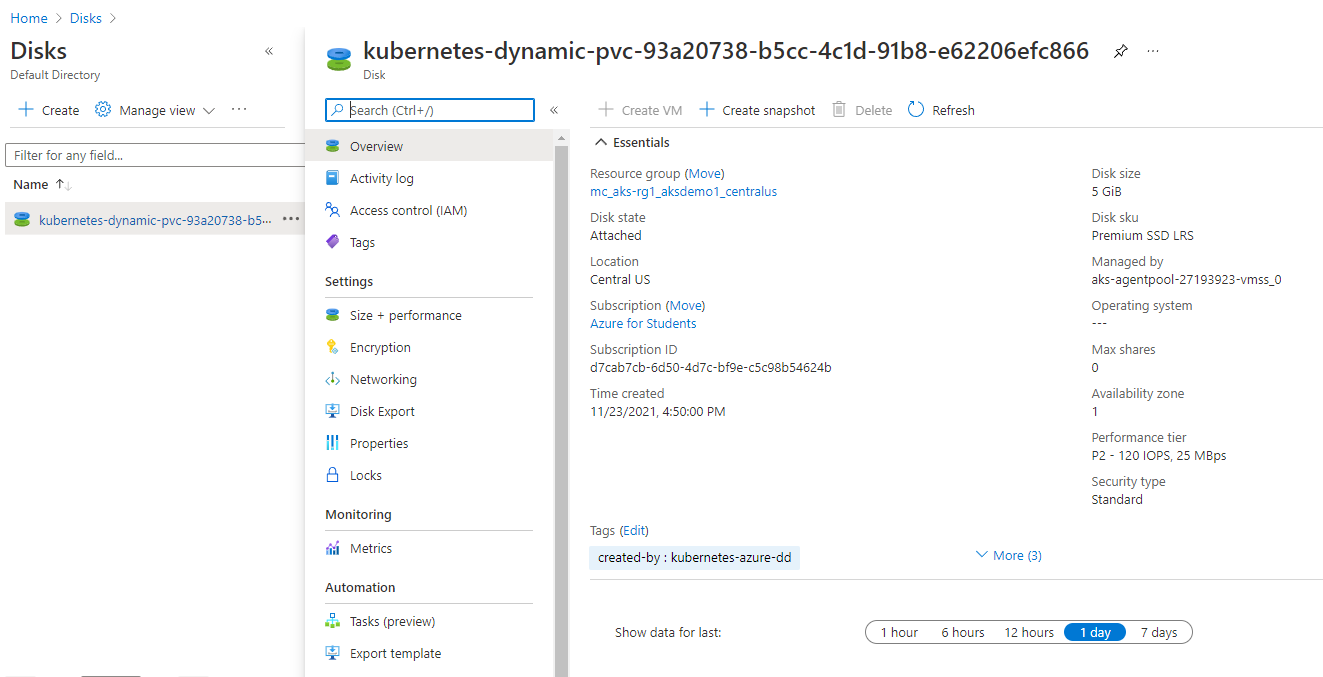
1. First update the pvc.yml and change the storageclassName to managed-premium



1. Now deploy all



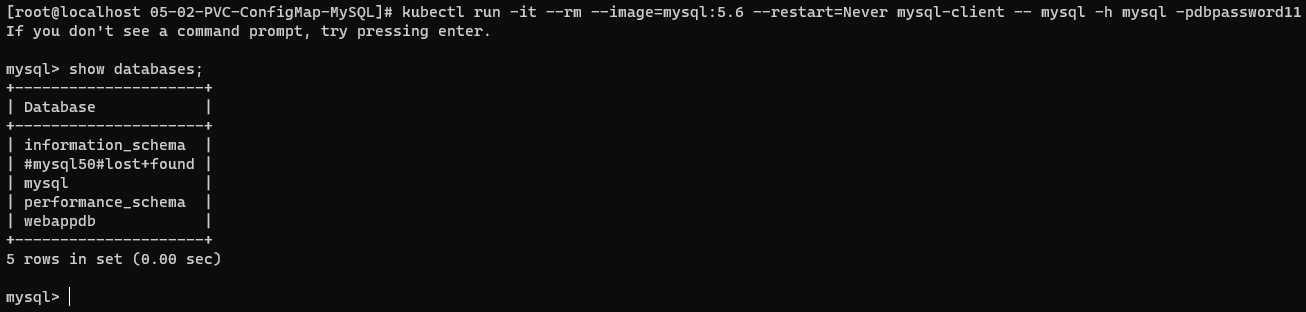
1. It creates a disk



You can see the disk name and PV name is same.



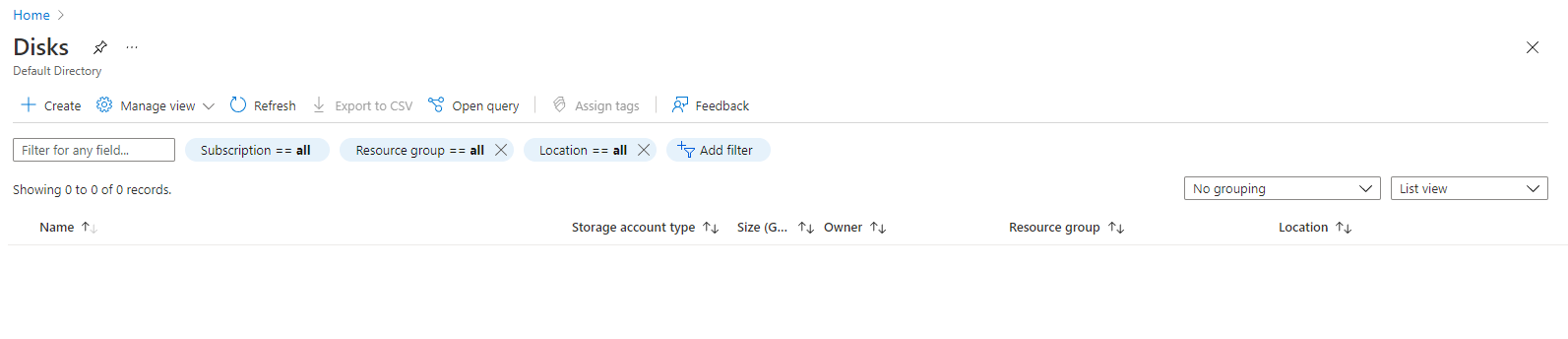
1. Connect to MySQL Database and verify the database is created which we provided in ConfigMap.



1. Now Delete everything

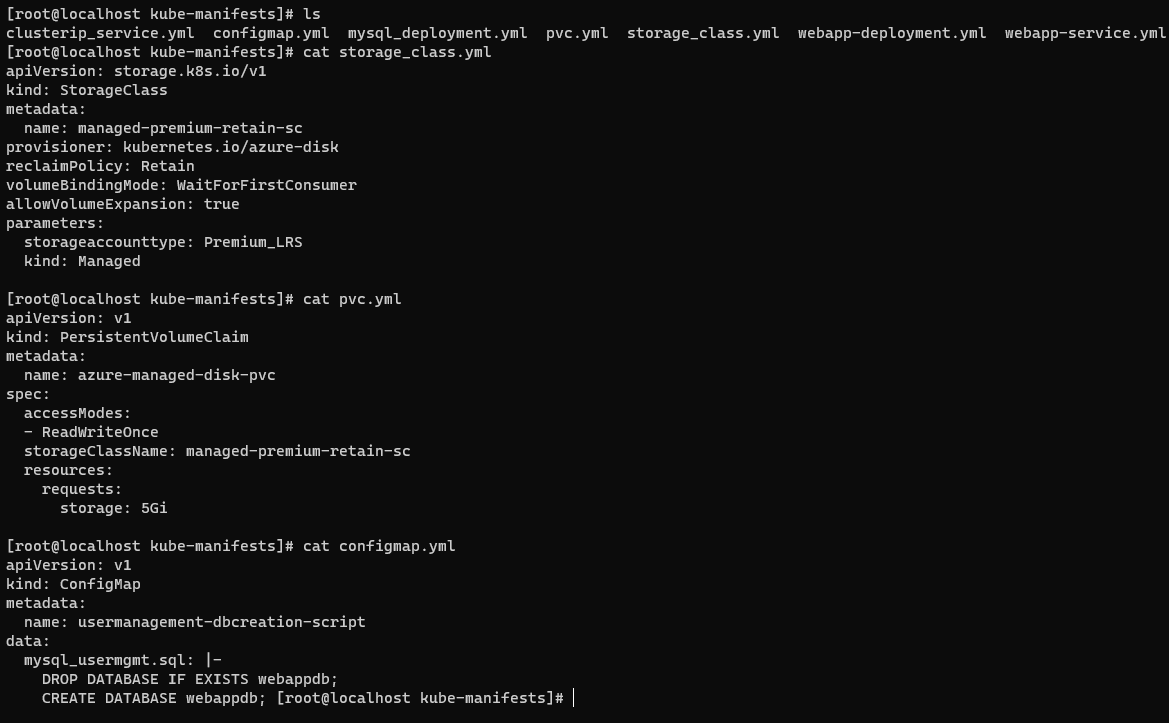


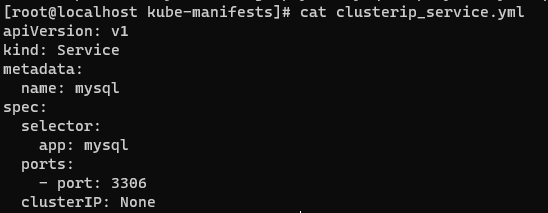
Now this time when PV is deleted then automatically the Azure disk is also deleted.



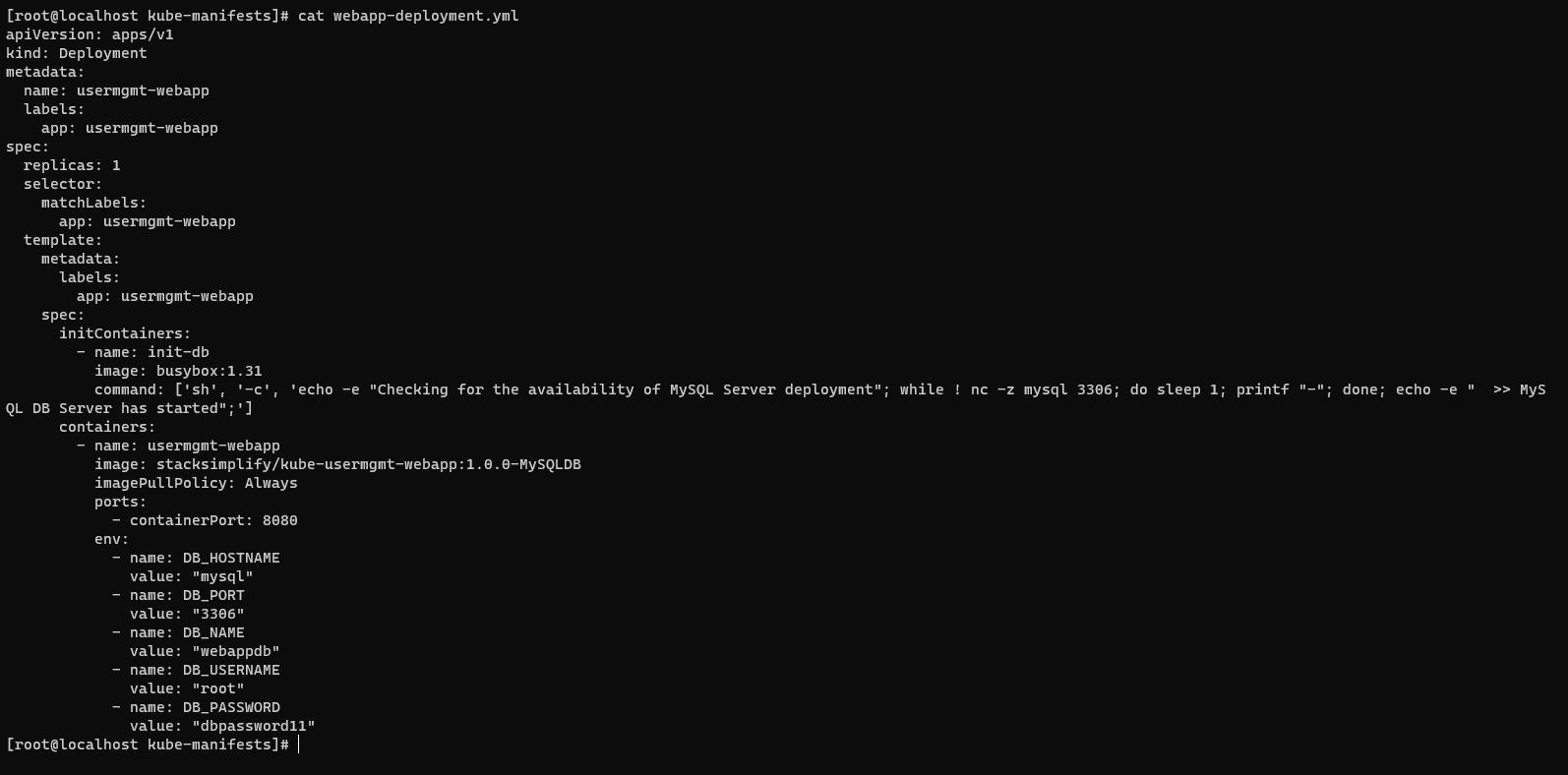
**Deploy UserManagement Web Application with MySQL Database**

1. Copy all templates from first Custom Storage section



1. Create User Management Web Application k8s Deployment manifest



Environment Variables

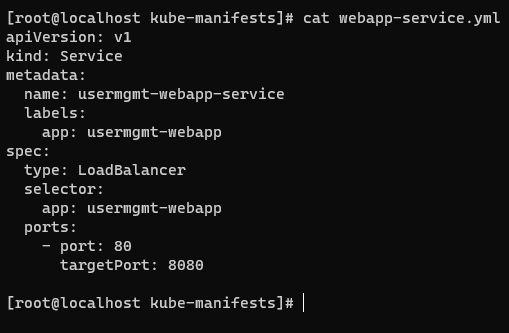
|  |  |
| --- | --- |
| Key Name | Value |
| DB\_HOSTNAME | mysql |
| DB\_PORT | 3306 |
| DB\_NAME | webappdb |
| DB\_USERNAME | root |
| DB\_PASSWORD | dbpassword11 |

Problem Observation:

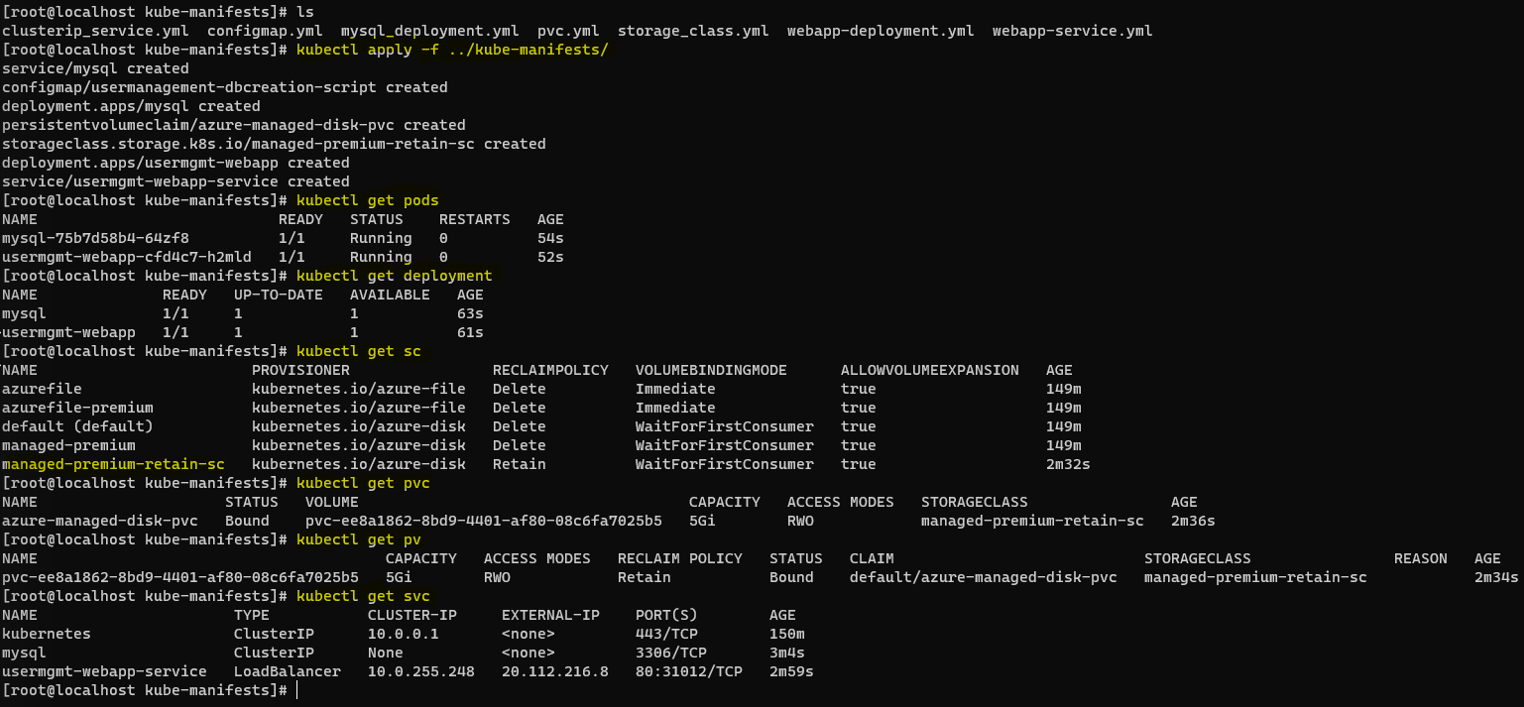
If we deploy all manifests at a time, by the time mysql is ready our User Management Web Application pod will be throwing error due to unavailability of Database.

To avoid such situations, we can apply initContainers concept to our User management Web Application Deployment manifest.

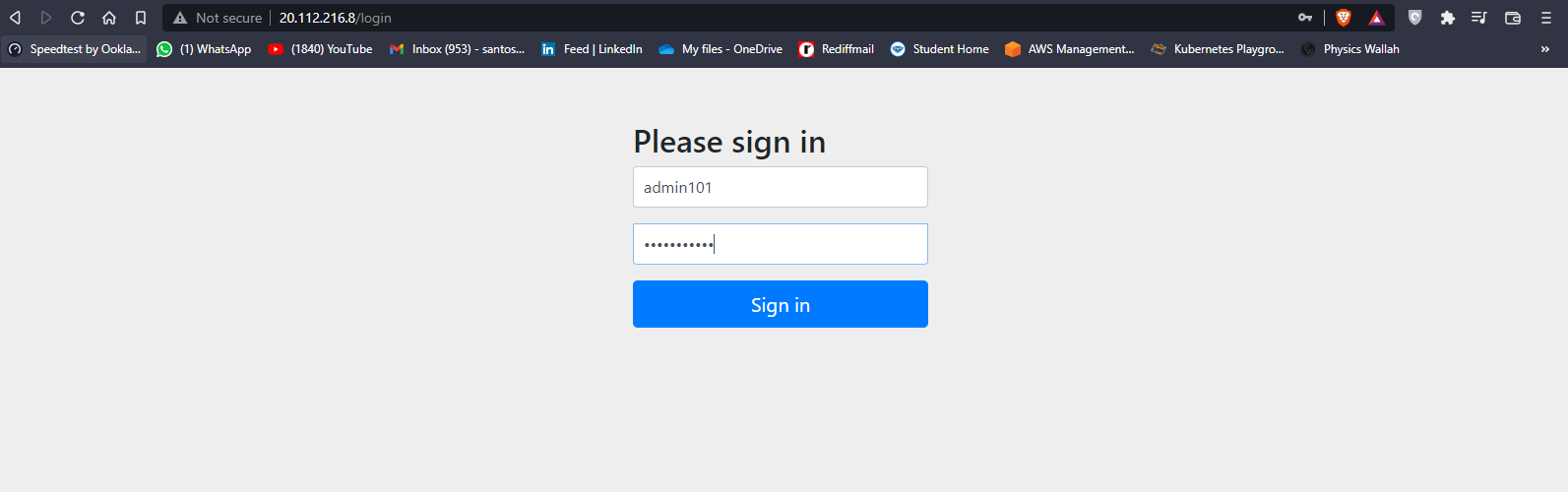
1. Create User Management Web Application Load Balancer Service manifest



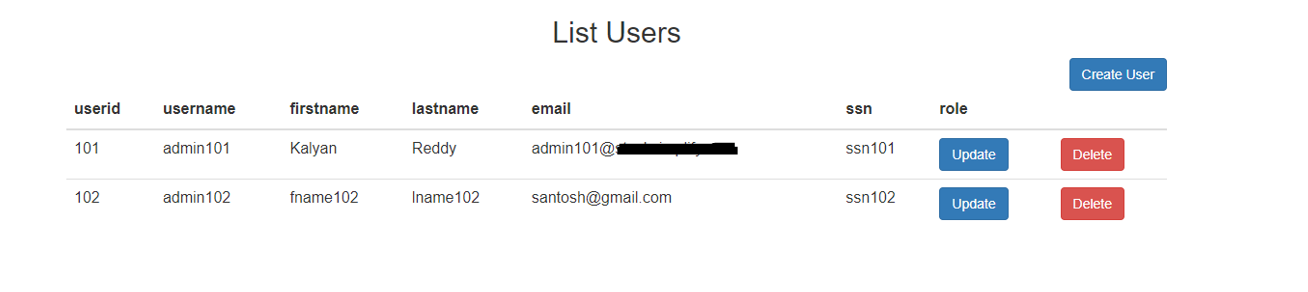
1. Now deploy all

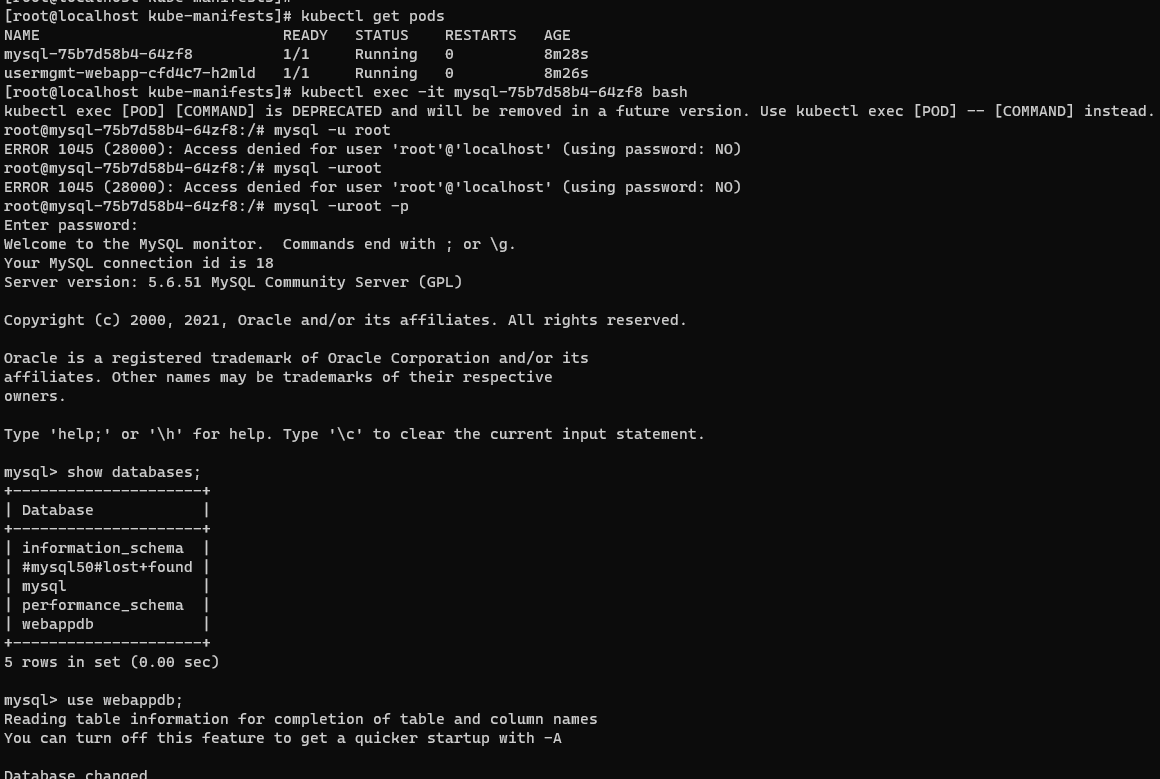


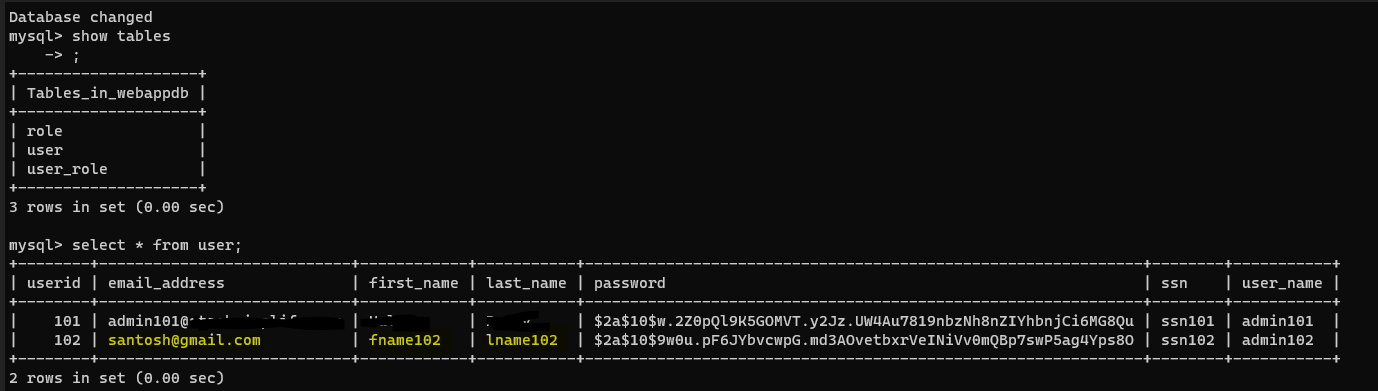
1. Verify all



Add user and check in mysql database the user is present or not.







The web application successfully deployed with custom storage class!!