



DATA PERSISTENCE

NEUEDA.COM

Objectives

- Data Persistence in OpenShift
- Persistent Volumes using NFS
- Persistent Volume Claims
- OpenShift Volumes
- Data Persistence Using Databases
- OpenShift Databases

Data Persistence in OpenShift

- In our previous lab we were able to deploy an application, but any data we put in was lost when the app updated.
- Actually, by design Kubernetes Pods are *ephemeral*.
 - Any given Pod could be replaced at any time.
- However persistent state is a critical aspect of many applications.

Example: Pods are Ephemeral

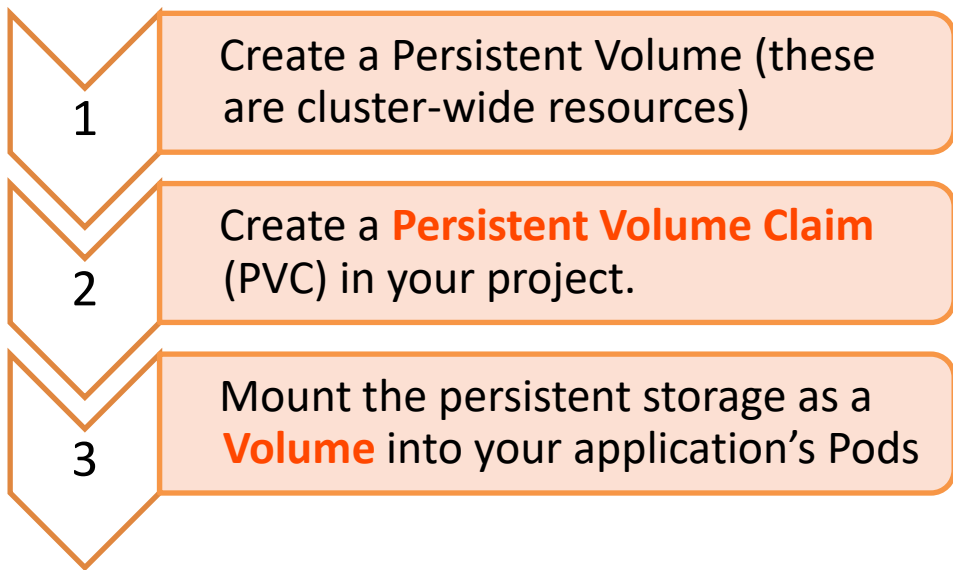
- Let's look at the Image Uploader deployment again
- Upload an image and see how it displays in the website
- Now scale the application down to 0 pods, then scale back up to 1 pod
- Open the URL again - no image!
- What happened? Pods (and their memory) are ephemeral
 - When the Pod is deleted, all its memory goes with it.

Handling Permanent Data

- OpenShift makes persistent storage available for
 - Data that is shared between Pods
 - Data that needs to persist past the lifetime of any particular Pod
- Permanent storage in pods is handled using **Persistent Volumes** (PVs)
- This can be supplied from a variety of storage solutions
 - NFS, HostPath (local directories), AWS EBS, Azure Disk, etc

Persistent Volumes using NFS

- In OpenShift the process of creating persistent file storage has three steps:



Creating a Persistent Volume

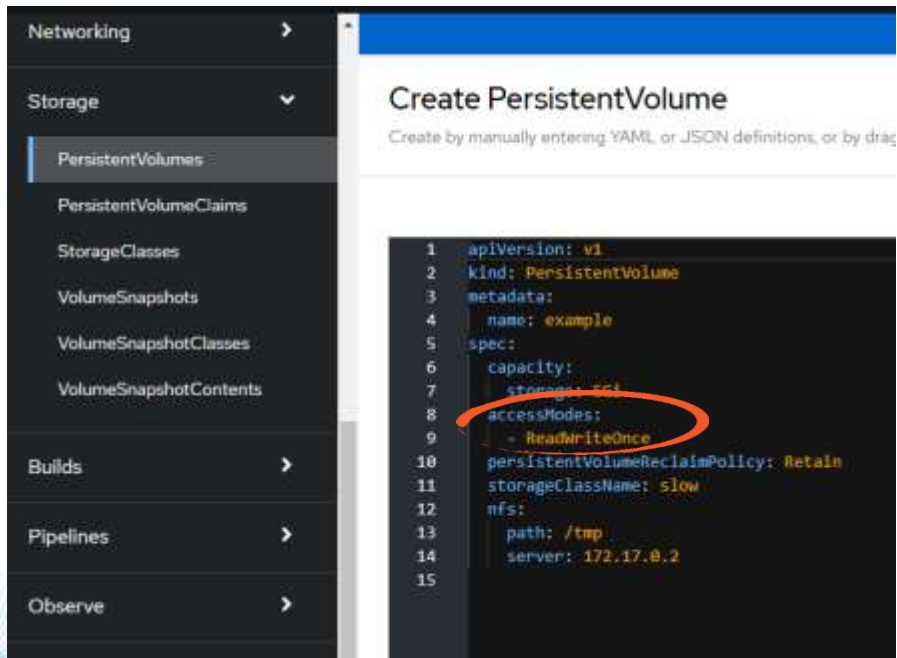
- Persistent Volumes are cluster-wide resources and really should only be allocated by the Cluster Administrator
- Go to the Administrator perspective in your Web Console and click on the Storage item.
 - See there is no “PersistentVolumes” option.
- In the Cluster Admin console you can see that option is available.

Creating a Persistent Volume

The screenshot shows the Red Hat OpenShift Container Platform web console. The left sidebar has a 'Storage' section with 'PersistentVolumes' highlighted. The main content area is titled 'PersistentVolumes' and features a 'Create PersistentVolume' button in the top right corner. Below the title is a search bar and a table of existing Persistent Volumes.

Name	Status	Claim	Capacity	Labels	Created
pvc-5b09e0dc-23c7-4dc6-a820-	Bound	pvc-91d2ae1d9d	1Gi	topology.k...=eu... topology.k...=eu...	4 May 2022, 09:45

Configuring a PersistentVolume: accessMode



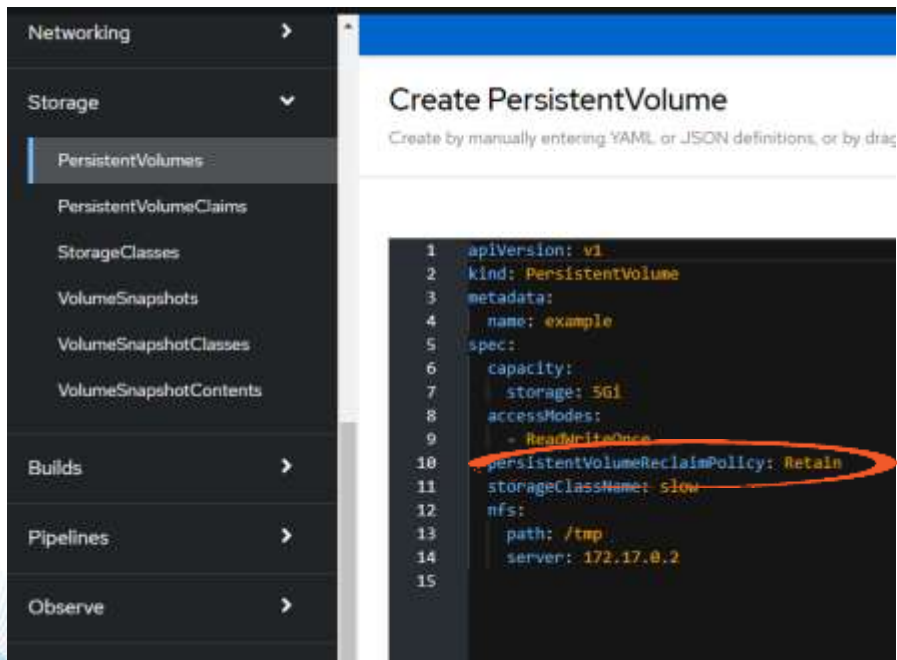
The screenshot displays the Kubernetes dashboard interface for creating a PersistentVolume. The left sidebar shows the 'Storage' section expanded, with 'PersistentVolumes' selected. The main panel shows the 'Create PersistentVolume' form, which includes a text area for entering YAML or JSON definitions. The following YAML code is shown in the text area:

```
1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: example
5  spec:
6    capacity:
7      storage: 1Gi
8    accessModes:
9      - ReadWriteOnce
10   persistentVolumeReclaimPolicy: Retain
11   storageClassName: slow
12   nfs:
13     path: /tmp
14     server: 172.17.0.2
```

The 'accessModes' field is highlighted with a red circle, indicating the configuration for the access mode.

- The **accessMode** can be
 - Read/Write once (RWO) - Can be mounted as read/write by a single node in the cluster
 - Read-only many (ROX) - Can be mounted as read-only by many nodes
 - Read/Write many (RWX) - Can be mounted as read/write by multiple nodes in the cluster.

Configuring a PersistentVolume: Reclaim Policy



```
1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: example
5  spec:
6    capacity:
7      storage: 5Gi
8    accessModes:
9      - ReadWriteOnce
10   persistentVolumeReclaimPolicy: Retain
11   storageClassName: slow
12   nfs:
13     path: /tmp
14     server: 172.17.0.2
15
```

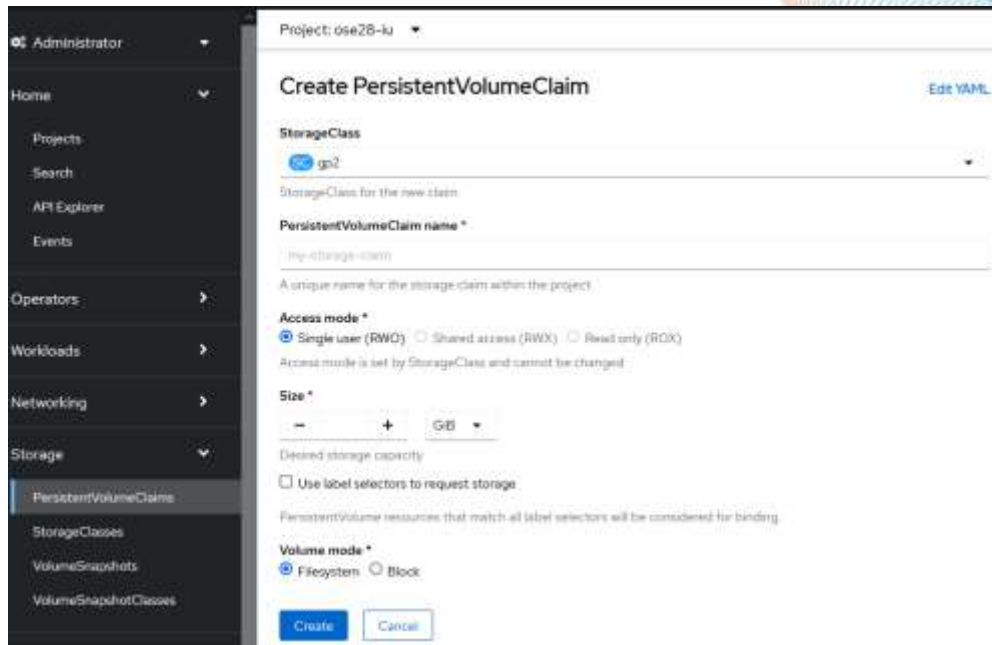
- The reclaim policy dictates how a PV handles reclaiming space after a storage claim on the PV is no longer required. It can be
 - Retain - all data is retaining in the PV, you have to reclaim space manually
 - Recycle - all data is automatically removed when the claim is deleted.

Using Persistent Storage

- To take advantage of a PV, you need to make a claim on that PV.
- PVs represent available storage, and PVCs represent an application's need for that storage.
- When you create a PVC, OpenShift looks for the best fit among the available PVs and reserves it for use by the PVC.
 - PV size vs PVC need - use the smallest available PV
 - Access Mode - use a PV with the same or greater access privileges than that required by the PVC

Adding a Persistent Volume Claim

- PersistentVolumeClaims can be created by accessing the PersistentVolumeClaims item under storage in the Administrator perspective



The screenshot displays the 'Create PersistentVolumeClaim' form in the Kubernetes Administrator UI. The left sidebar shows the navigation menu with 'Storage' expanded and 'PersistentVolumeClaims' selected. The main panel shows the form for creating a new claim. The 'StorageClass' is set to 'gp2'. The 'PersistentVolumeClaim name' is 'my-storage-claim'. The 'Access mode' is 'Single user (RWX)'. The 'Size' is '1 GB'. The 'Volume mode' is 'Filesystem'. The 'Create' button is highlighted.

Project: ose28-lu

Create PersistentVolumeClaim

[Edit YAML](#)

StorageClass
gp2

StorageClass for the new claim

PersistentVolumeClaim name *

my-storage-claim

A unique name for the storage claim within the project

Access mode *

☒ Single user (RWX) ☐ Shared access (RWX) ☐ Read only (ROX)

Access mode is set by StorageClass and cannot be changed

Size *

1 GB

Desired storage capacity

☐ Use label selectors to request storage

PersistentVolume resources that match all label selectors will be considered for binding.

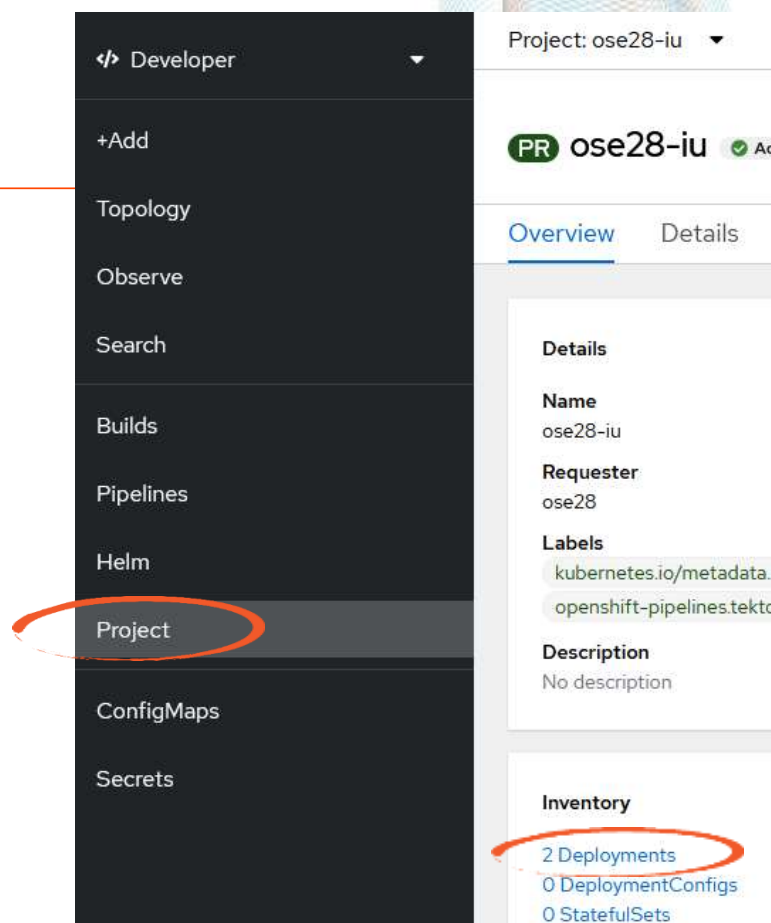
Volume mode *

☒ Filesystem ☐ Block

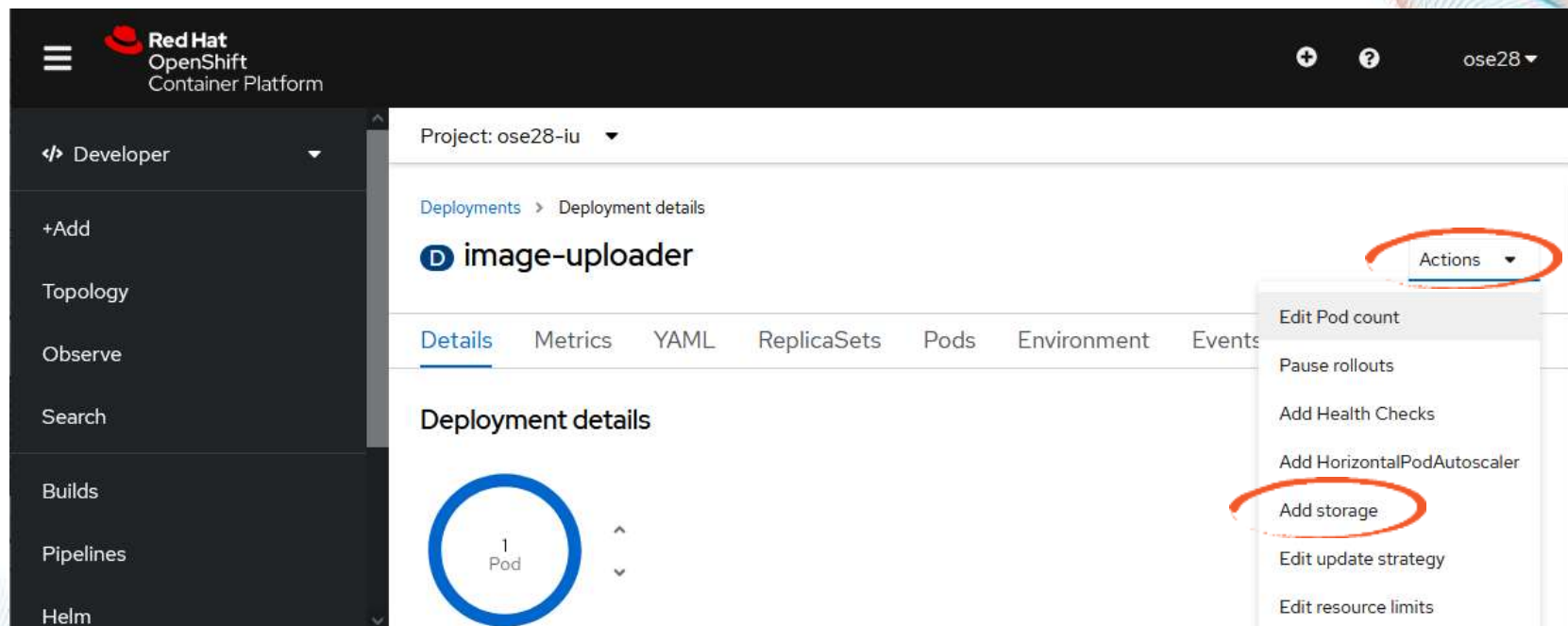
[Create](#) [Cancel](#)

OpenShift Volumes

- A *volume* is any filesystem, file or data mounted into an application's Pods to provide persistent data.
- To attach a PVC as a volume in our app using the Web Console you need to get to the project overview page and select the Deployment.



Creating a Volume



The screenshot displays the Red Hat OpenShift Container Platform interface. On the left is a dark sidebar with navigation options: Developer, +Add, Topology, Observe, Search, Builds, Pipelines, and Helm. The main header shows the Red Hat logo and 'OpenShift Container Platform' on the left, and a user profile 'ose28' on the right. The main content area is titled 'Project: ose28-iu' and shows 'Deployments > Deployment details' for a deployment named 'image-uploader'. Below this, there are tabs for 'Details', 'Metrics', 'YAML', 'ReplicaSets', 'Pods', 'Environment', and 'Events'. The 'Details' tab is active, showing 'Deployment details' and a circular progress indicator with '1 Pod'. An 'Actions' dropdown menu is open on the right, with the 'Add storage' option highlighted by a red circle. Other options in the menu include 'Edit Pod count', 'Pause rollouts', 'Add Health Checks', 'Add HorizontalPodAutoscaler', 'Edit update strategy', and 'Edit resource limits'.

Creating a Volume

- Just select your existing PVC
- The Mount Path is the path within each pod at which the persistent volume should be available - it is application dependent.
- On “Save” the volume is attached.

Red Hat OpenShift Container Platform

Project: ose28-lu

Add Storage to [image-uploader](#)

PersistentVolumeClaim *

☒ Use existing claim

[PVC](#) ose28-pvc

☐ Create new claim

Mount path *

/opt/app-root/src/uploads

Mount path for the volume inside the container.

☐ Mount as read-only

Subpath

Optional path within the volume from which it will be mounted into the container. Defaults to the root of the volume.

The volume will be mounted into all containers. You can [select specific containers instead](#).

[Save](#) [Cancel](#)

Lab 3: Add Persistent Data to the Image Uploader

- This is a quick lab to implement what we have just seen in your own project.
- Go to [/labs/3-image-uploader-persistent-storage.md](#)

Volumes Separate Data

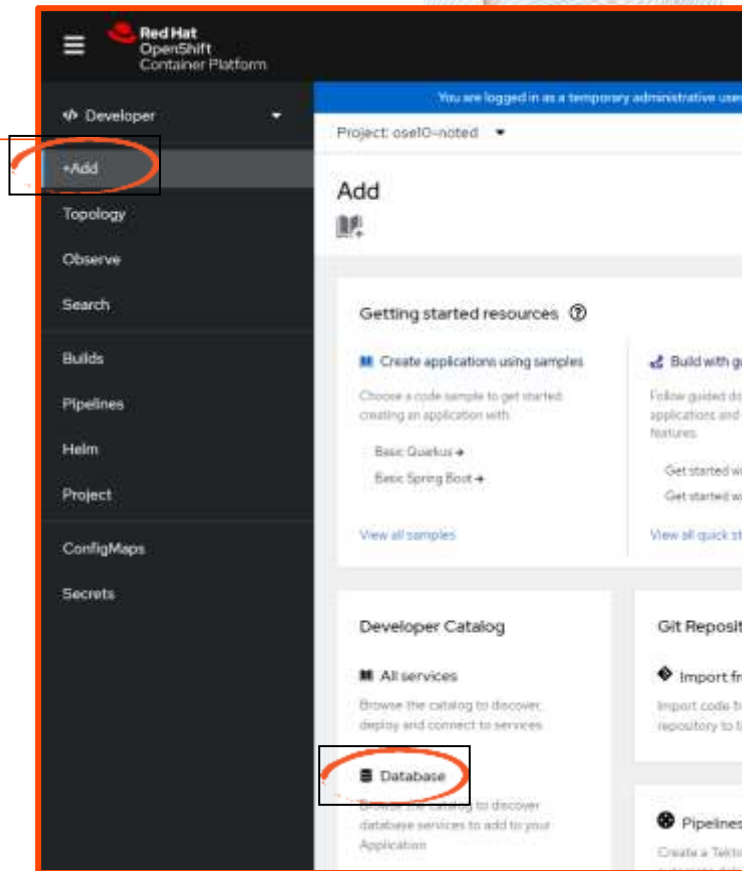
- Suppose everyone set up Volumes on our cluster - why don't you see everyone else's data?
 - Each application deployment uses its own NFS volume to store data.
 - Each NFS volume is then mounted into its own application's mount space.
 - So the data is always separated on the cluster.

Data Persistence Using Databases

- The Persistent Volume technology is only one way to data persistence.
- You can also hook your application to a database
 - If the database is not running on your cluster, then it supplies its own data persistence
 - If the database is running on your cluster, then data persistence is achieved behind the scenes.

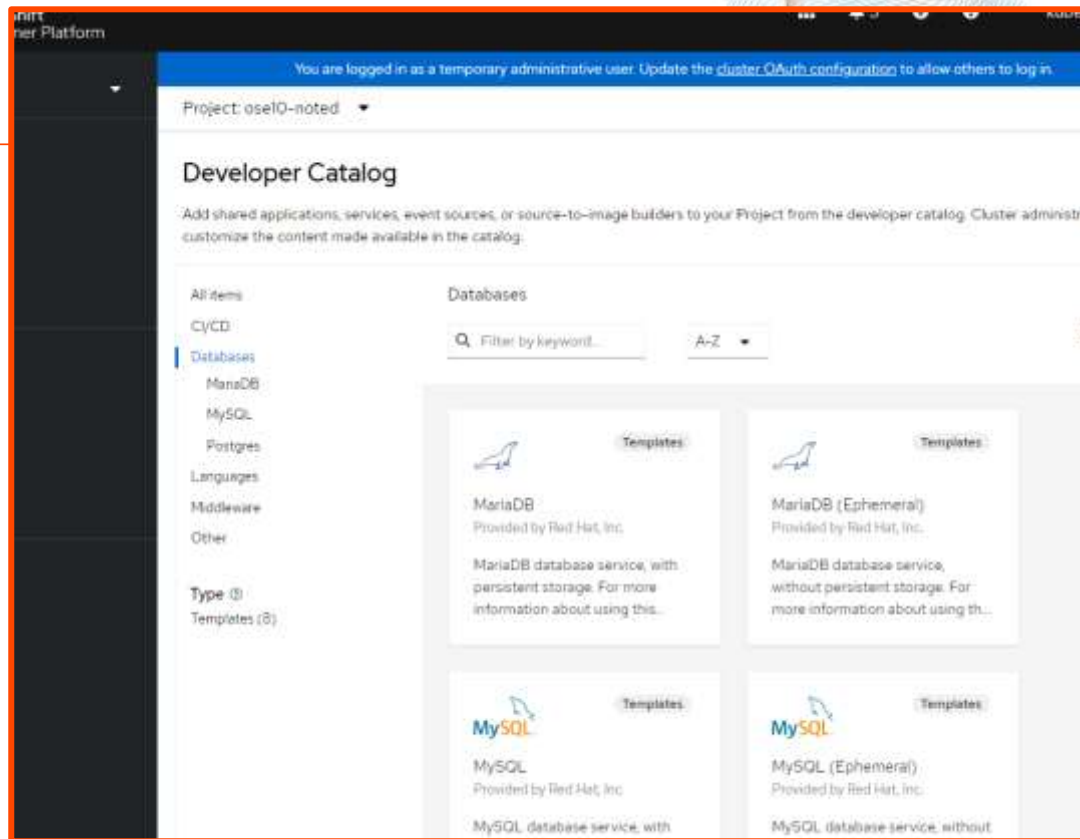
OpenShift Databases

- OpenShift has pre-configured MySQL, MariaDB and PostgreSQL databases available in the Developer Catalogue



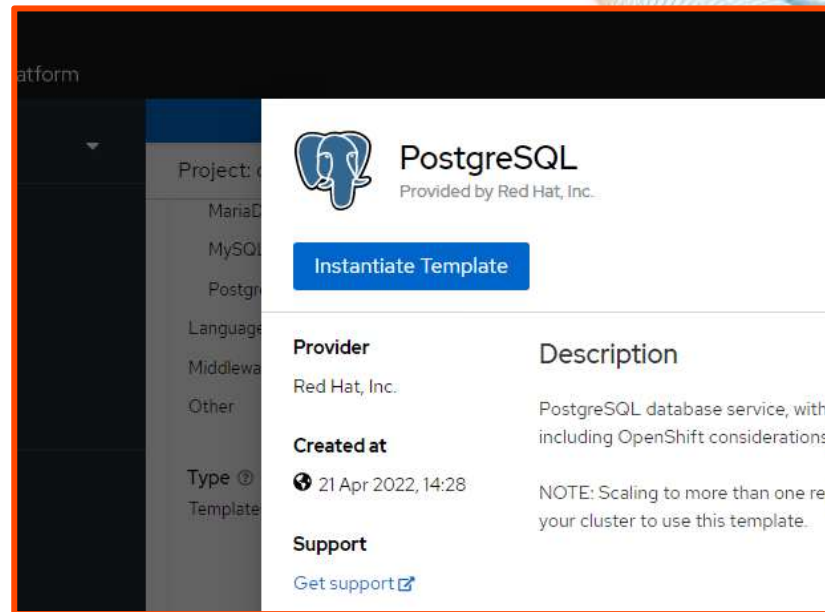
OpenShift Databases

- The databases all offer ephemeral (for testing) and persistent options.



OpenShift Databases

- During Instantiation you can supply a username, password and other configuration details.
- If you use a template you will need to manually configure your app to inject the environment variables needed to connect.



Demo: Petclinic

- In this demo we will deploy a Spring Boot app and a MySQL database on the same cluster and link the two.

Connecting to Databases

- The database deployment we saw is quite manual, OpenShift has a number of more automatic connection methods.
- One of the simplest of these is using a **Secrets** object.
- Secrets objects store one or more values that are intended to be obscured (like passwords, or certificate files).
- Secrets objects can be made available to Pods in a namespace without divulging the contents to users.
- We will see Secrets for connecting a database later on.

Summary

- Data Persistence in OpenShift
- Persistent Volumes using NFS
- Persistent Volume Claims
- OpenShift Volumes
- Data Persistence Using Databases
- OpenShift Databases

Questions and Comments?

