Data Workshop on Kubernetes

Kubecon North America

Table of Contents

Author

Bhavin Shah
Sr. Technical Marketing Manager
Portworx by Pure Storage

Lab Overview

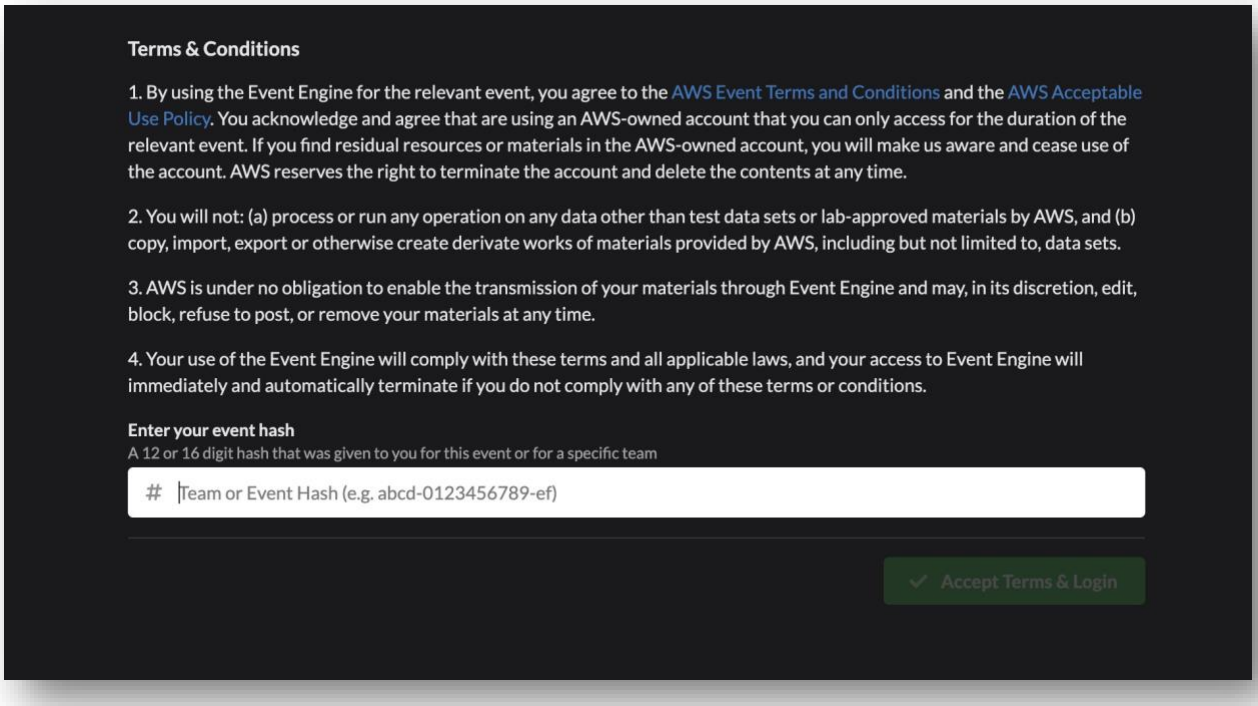
This lab guide is built for the Data Workshop on Kubernetes at Kubecon North America, to help users get hands-on experience with the #1 Kubernetes Data Platform in the ecosystem. As part of this lab, users will use an AWS Event Engine AWS account and the Portworx portfolio of products, which includes Portworx Backup, Portworx Enterprise and Portworx Data Services, to learn more about how they can run data services and stateful applications in Production on an Amazon EKS cluster. As part of this guide, users will deploy two Amazon EKS clusters, and use demo applications that use databases like PostgreSQL, MongoDB, MySQL, etc to experience scenarios like Accidental data corruption, Ransomware attacks, Availability Zone failures, Accidental Database deletion, avoiding noisy neighbor issues, automated storage capacity management, and single click day 0 and day 2 operations for data services on Kubernetes.

Lab Setup

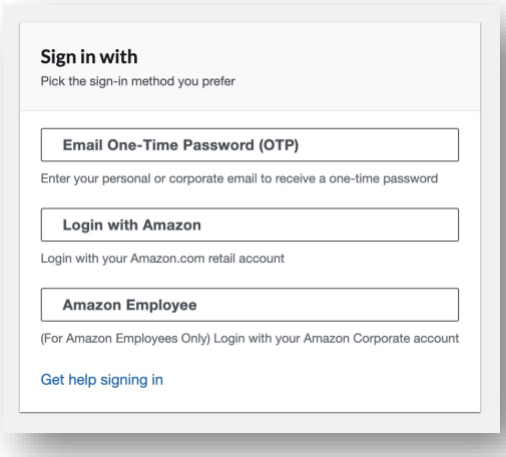
Accessing AWS Account

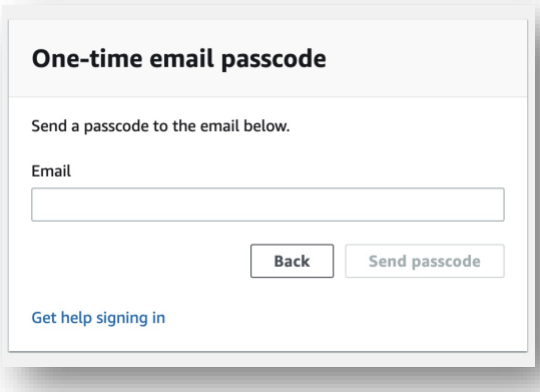
1. Navigate to the event engine platform using the link (https://dashboard.eventengine.run/login) and enter your 12- or 16-digit event hash.

Note: Use an incognito window to ensure that you aren't using your existing AWS account for this lab.

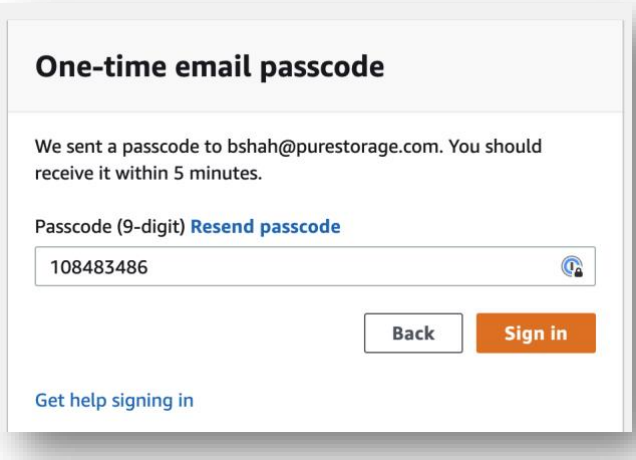


2. After entering your event hash, click *"Accept Terms & Login"*.
3. Next, click on *"Email One-Time Password (OTP)"* and enter your email address and hit *"Send Passcode"* to receive the OTP.

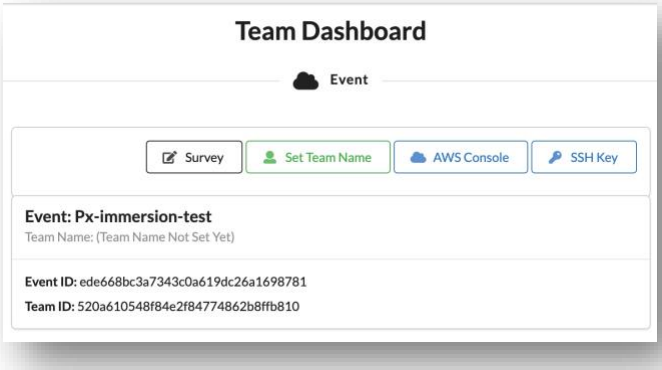




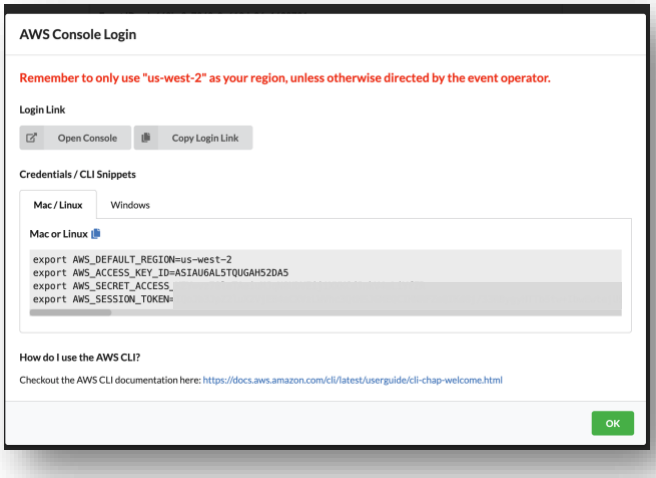
4. Check your email for the OTP. Copy the passcode and enter it in the Event Engine platform and hit *"Sign In"*.



5. Next, from the event engine dashboard, click on *"AWS Console"*.



6. Next, click on Open Console to access the AWS Management console.

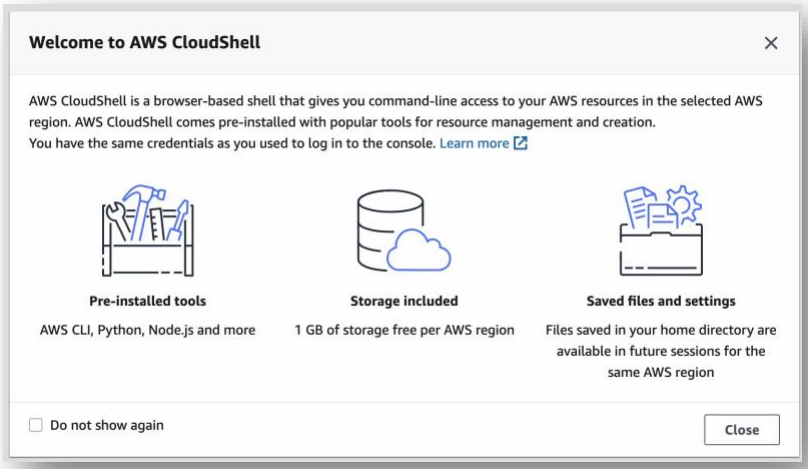


At this point, you have an AWS account that can be used for this Data workshop.

Deploying AWS Resources

1. Open an AWS CloudShell session from the top right of the AWS management console or, you can also search for CloudShell in the main search box on the left.

2. Read through the Welcome to AWS CloudShell box and click close.



Note: It takes a couple of minutes for the CloudShell session to become responsive.

Create AWS IAM User and policy

Once you have access to your AWS account using the Event Engine platform, we will go ahead and create an IAM user called *"workshop"*, whose credentials will be used to deploy Amazon EKS clusters and S3 buckets needed for the workshop.

1. Clone the PX-DataWorkshop repository on to your CloudShell session using the following command:

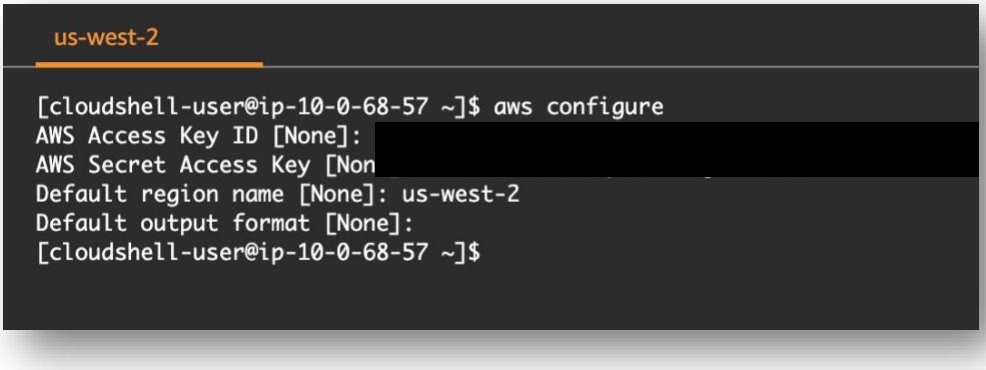
```
git clone https://github.com/bhavin04890/PX-DataWorkshop.git
```

2. Go to CloudShell and use the following commands

```
cd /home/cloudshell-user/PX-DataWorkshop

./create-aws-user.sh
```

3. Next, use the following commands to configure your AWS credentials using the AWS user **(workshop)** that you created in the previous step. Use the Access Key Id and Secret Access Key from the '***workshop-creds.txt***' as the input for the aws configure command and enter us-west-2 as the default region.

```
cat workshop-creds.txt

aws configure
```



4. Next, let's use the following command to update the IAM policy ARNs for our eks configuration files

```
cat iampolicyoutput.txt
```

```
vi eks-source-cluster.yaml
```

Note: Once you have updated the pxce2node policy ARN under the attachPolicyARNs section of the yaml file, Save the file using "esc → :wq"

```
vi eks-destination-cluster.yaml
```

Note: Once you have updated the pxce2node policy ARN under the attachPolicyARNs section of the yaml file, Save the file using "esc → :wq"

5. Next, let's create a new S3 bucket, that we will use to store our application snapshots.

```
./create-bucket.sh
```

6. Enter a unique name for your S3 bucket when the script prompts you to. If the script fails, because you didn't use a unique name, run the script again and enter a new name. Remember this name as we will use it later in the guide!
7. Next, let's run the pre-req.sh file, which will deploy the source and destination Amazon EKS clusters and applications that we need for the data protection section.

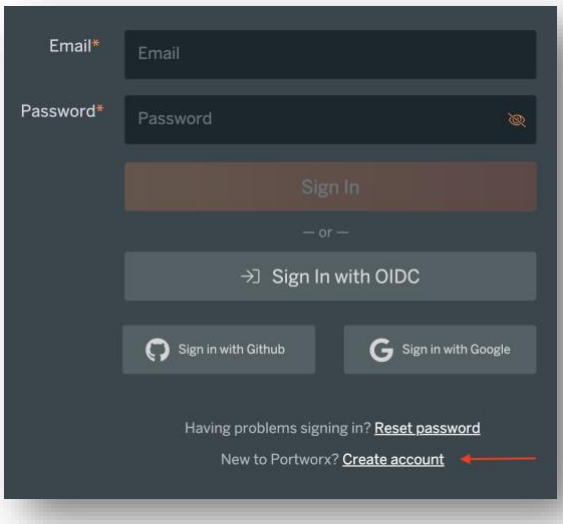
```
./pre-req.sh
```

While we wait for these resources to be deployed, we can proceed and create our BaaS accounts, to complete the pre-req steps.

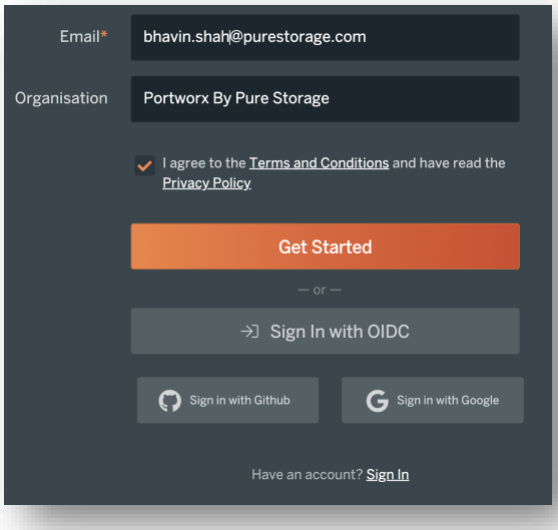
Create Portworx Central account

1. Navigate to Portworx Central (https://central.portworx.com/) and create a new account.

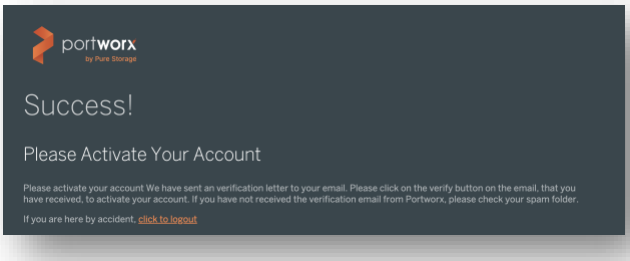
Note: Skip this section if you already have a Portworx Central account.



2. Enter your email address and name of your organization. (Note: Use your work email to register for Portworx Central)

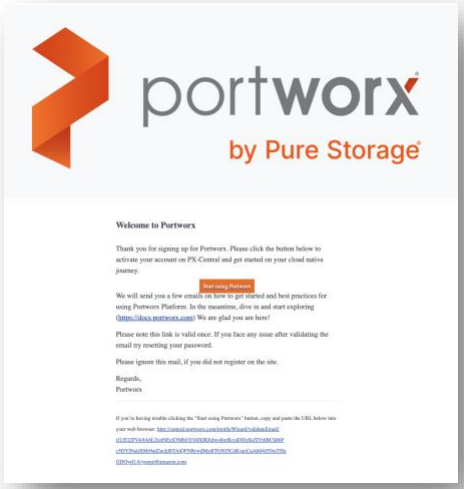


3. Activate your account by clicking the link in the email.

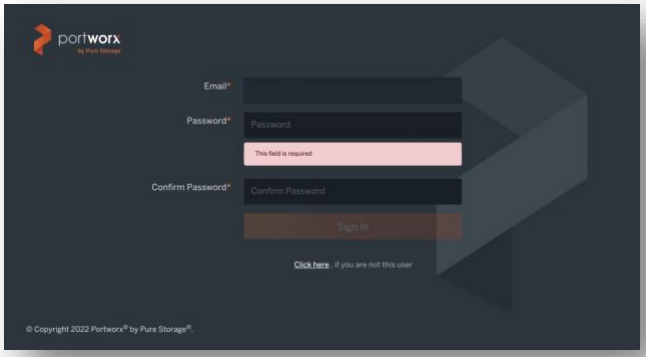


4. Check your email and verify your account by clicking *"Start using Portworx"*.

Note: It might take a couple of minutes to get the email.

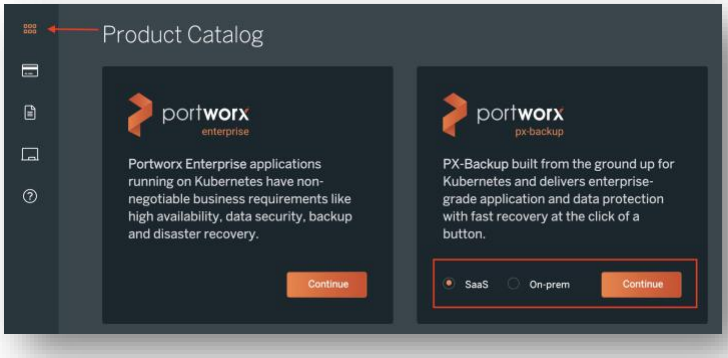


5. Set a new password for [Portworx Central] and click *"Sign In"* to confirm your account creation.

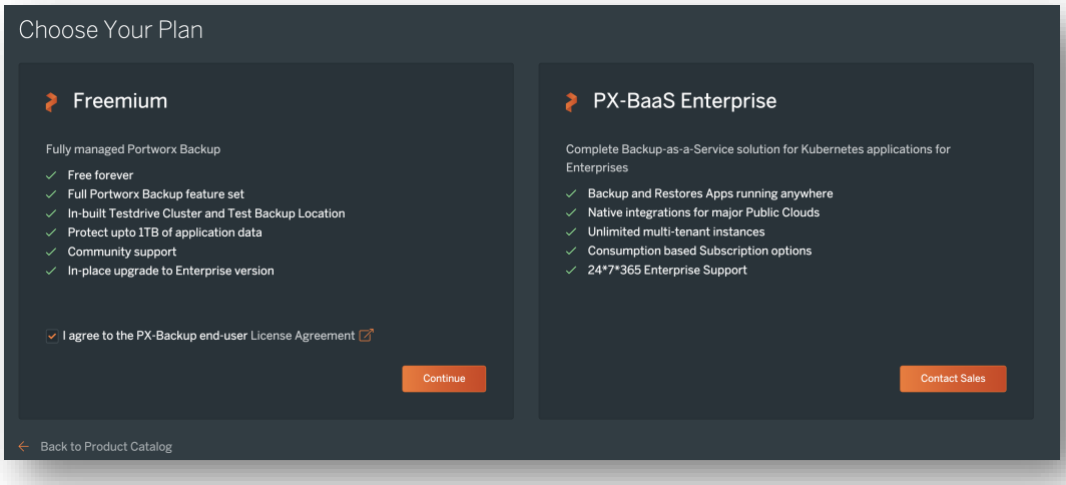


Create Backup As A Service Freemium Account

1. Navigate to the *"Product Catalog"* tab on the left pane and select Portworx Backup – SaaS option and click *"Continue"*.



2. Select the "Freemium" plan, check the box for the license agreement and click Continue.



At this point, the pre-requisites for the Data Workshop are done, and we will proceed to the slides portion and then come back for the hands-on lab again.

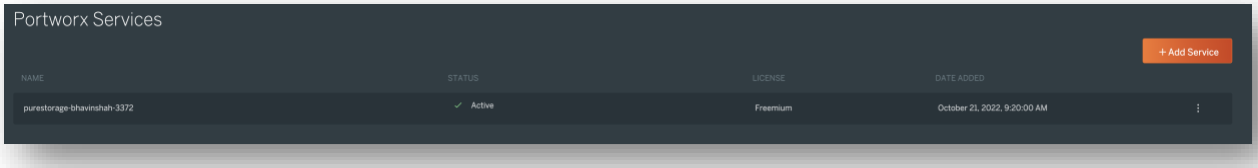
Portworx Backup As A Service (BaaS) overview

Portworx BaaS allows users to leverage a managed Kubernetes data protection tool to protect their containerized applications running on Amazon EKS clusters. As part of this section, we will go ahead and add our cloud accounts, backup targets, auto-discover Amazon EKS clusters.
Note: Ignore the test-drive cluster in the Freemium account. We are going to use the Amazon EKS clusters we deployed as part of pre-req for this workshop

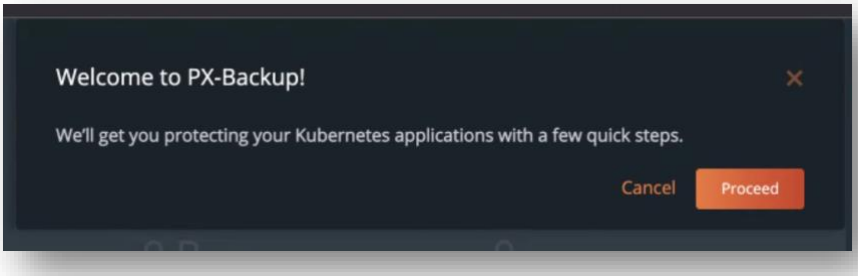
Configure cloud credentials

Credentials allow BaaS to authenticate with clusters for the purpose of taking backups and restoring to them, as well as with backup locations where backup objects are stored. To add cloud credentials, use the following steps:

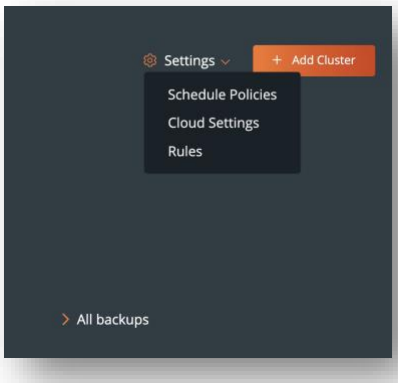
1. Log into Portworx Central (https://central.portworx.com/) and navigate to your BaaS Freemium instance.



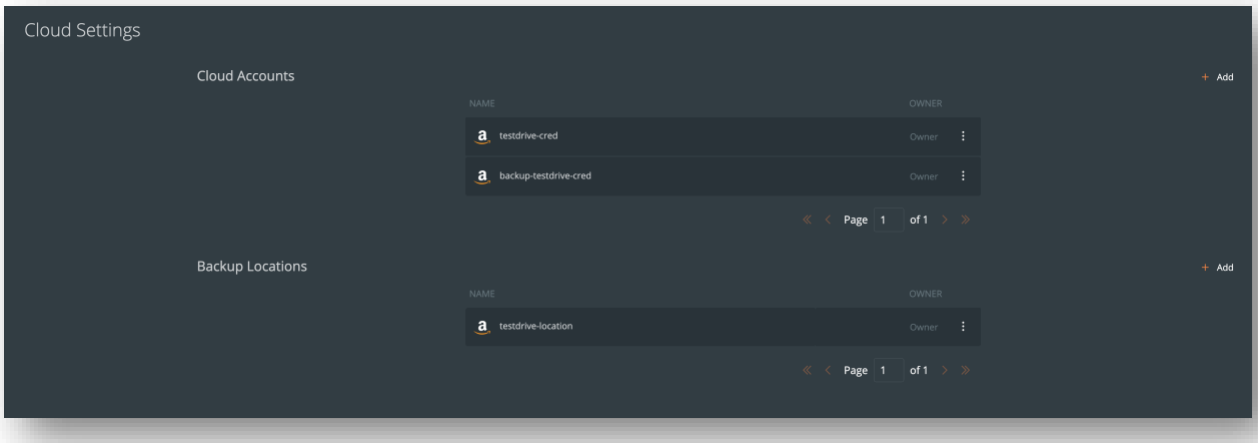
Note: Cancel and close out any tool tip suggestions from the BaaS Interface, like the screenshot below:



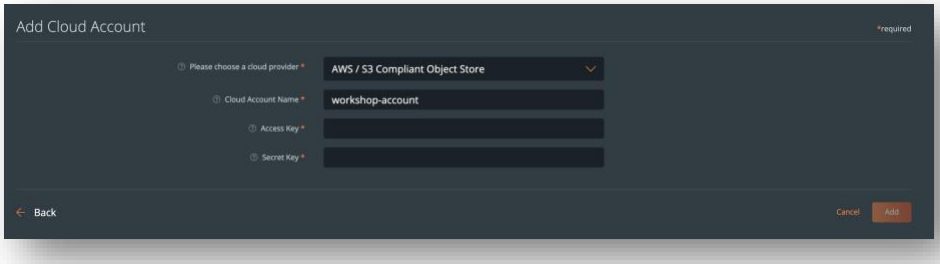
2. From the dashboard, click on **"Settings"** on the top right, and click **"Cloud Settings".**



3. Click on **"+ Add"** on the right to add a Cloud Account.

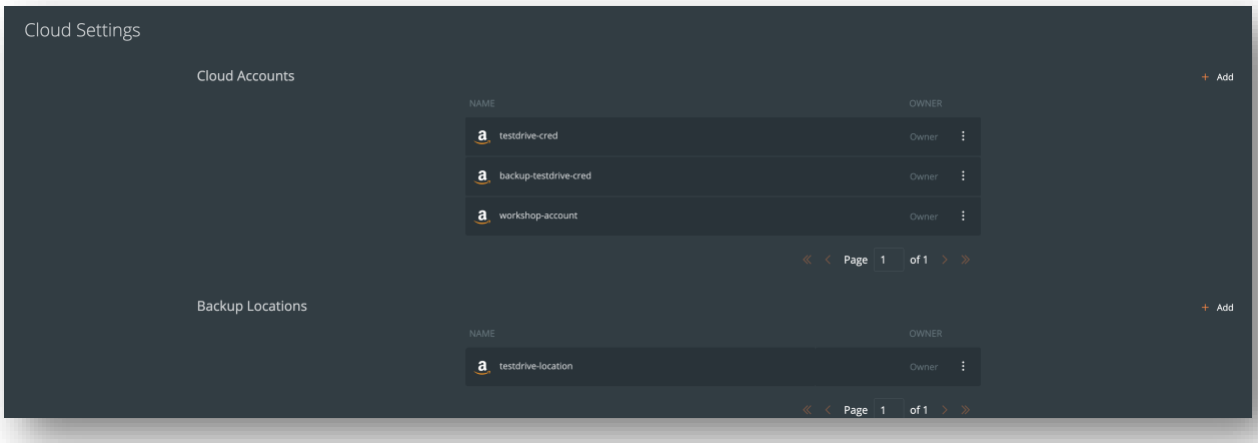


4. Select *"AWS / S3 Compliant Object Store"* as the Cloud Provider and give a name **(workshop-account)** for to the cloud account. This name is just used for managing credentials inside BaaS.



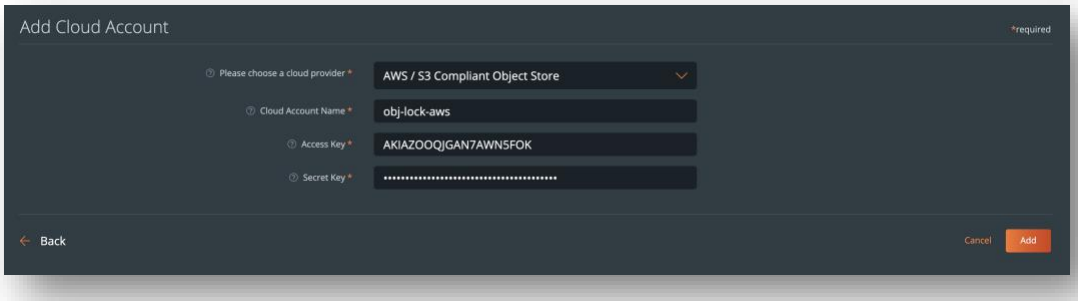
5. Enter the Access Key and Secret Key for the AWS IAM user **(workshop)** and click Add. Once complete, you should see your account listed under Cloud Accounts.

Note: To fetch the workshop user's Access Key and Secret Key, go back to the AWS Cloud Shell and use the command: "`cat workshop-creds.txt`"

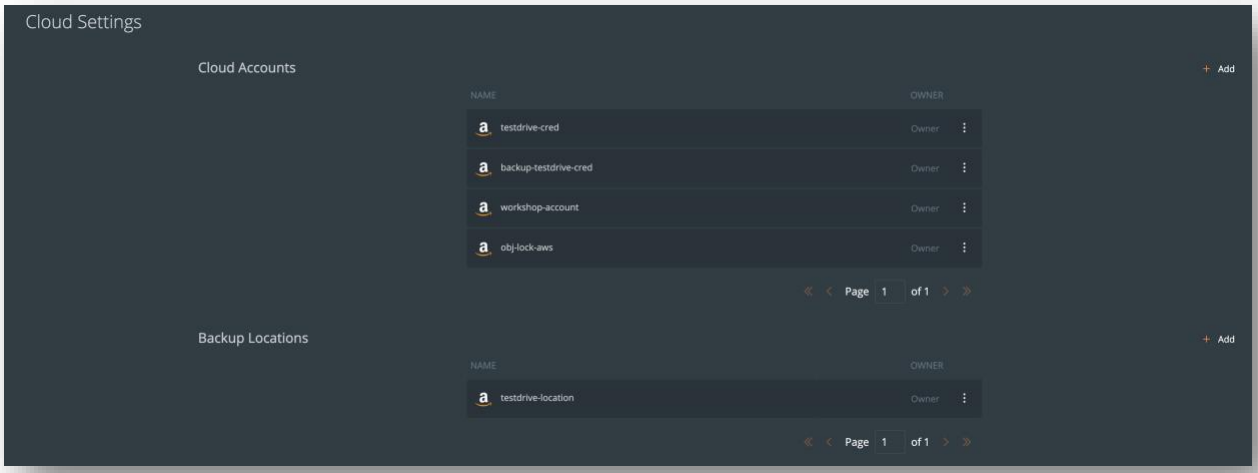


6. Repeat steps 3-5 above, to add another cloud account **(obj-lock-aws)**. This second cloud account will be used to store immutable application snapshots in an object lock enabled S3 bucket. Use the following credentials to access add the cloud account.

```
Access Key ID: AKIAZOOQJGAN7AWN5FOK
Secret Access key: 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
```



7. At this point, you have two AWS Cloud Accounts:
    a. workshop-account: Used to add Amazon EKS clusters and non-object lock enabled backup repository
    b. obj-lock-aws: Used to add object lock enabled S3-bucket as the backup repository.



Adding Backup targets

Backup locations are S3-compatible object storage bucket locations that can be added to BaaS and used to store end-to-end application snapshots of your containerized applications running on Amazon EKS.

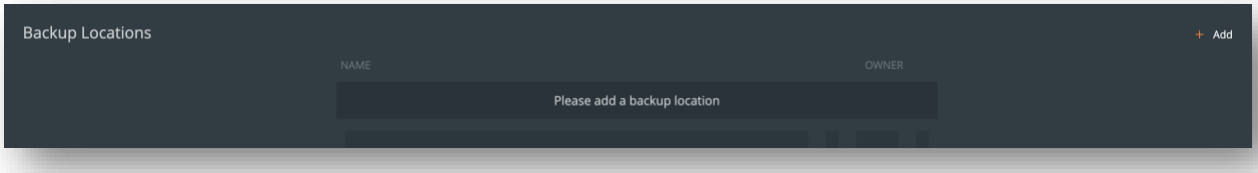
```
There are two type of backup targets / locations.

1. Object Lock enabled backup location – These backup locations can be used to
store Write Once Read Many (WORM) snapshot copies. Since, any backup snapshots
stored in these backup buckets are immutable, they help organizations protect
and recover from Ransomware attacks. During a Ransomware attack, hackers will
```

try to manipulate your backup snapshots. So, having an immutable backup snapshot, allows organizations to always have a golden copy of their application stored in a backup repository that can be used to recover applications to new Amazon EKS clusters.
2. **non-Object lock enabled backup locations** – These are standard backup locations that allow users to configure custom retention periods for their backup snapshots using BaaS. These backup locations can help organizations protect their applications from accidental corruption or data loss issues and allows them to restore applications quickly from a snapshot.

Organizations can typically choose to build a custom backup policy, where they can have an hourly backup, storing snapshots in a non-object lock enabled bucket with custom retention period, and maybe use the object-lock enabled bucket to store daily application snapshots

The steps to add an object lock enabled and a non-object lock enabled bucket to BaaS instance are the same. To add backup targets to your BaaS instance, use the following steps:

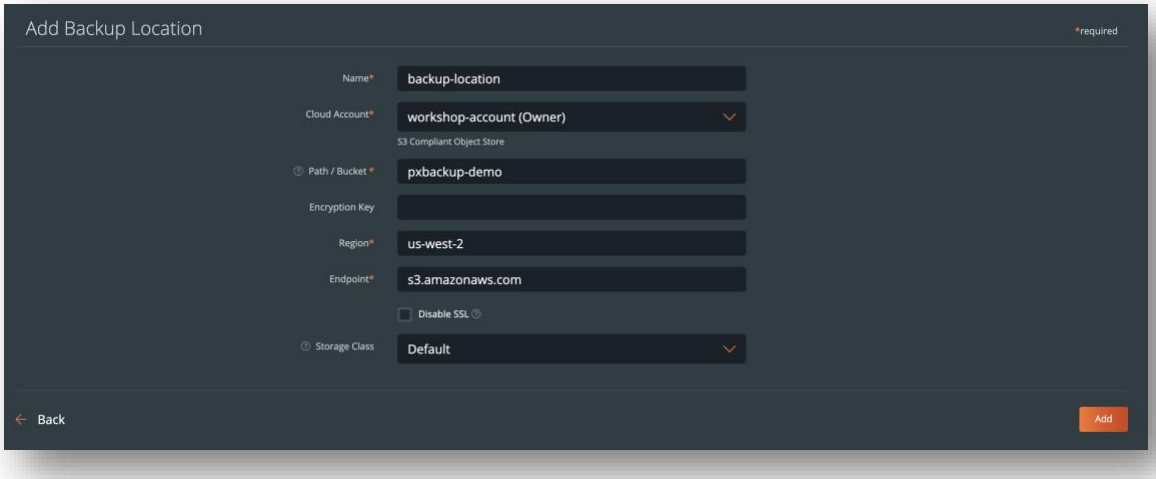
1. Click on "+ Add" on the Backup Locations section.



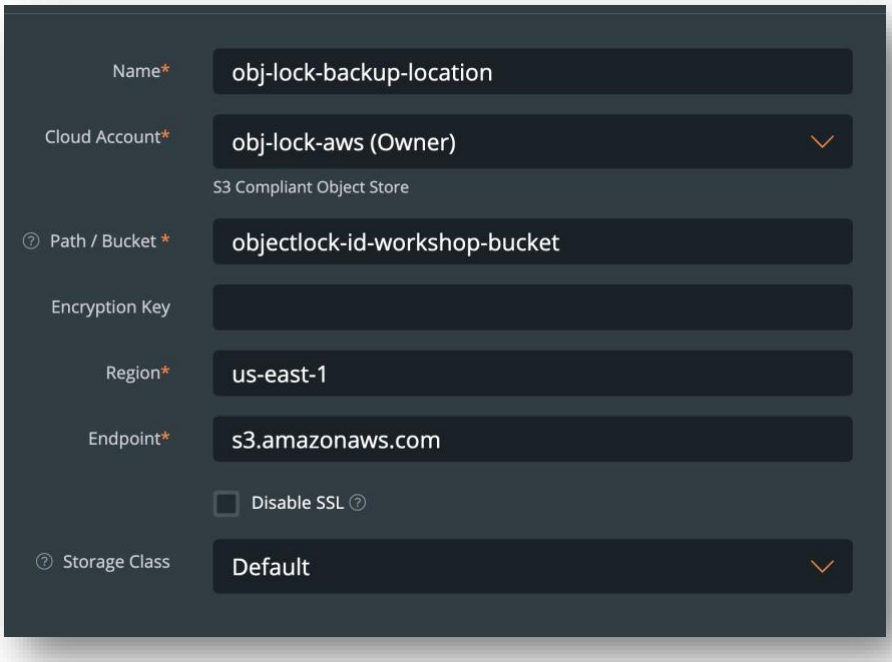
2. Enter a name for the backup location, select the Cloud Account (aws-immersion-account) that we added in the previous step, enter the name of the bucket, and the region where the bucket is available.
    a. **Name:** backup-location
    b. **Cloud Account:** workshop-account (Owner)
    c. **Path / Bucket:** <<name of the bucket your created as part of pre-req>>

Note: To get the name of your S3 bucket, go to AWS CloudShell and use the cmd *"aws s3 ls"*

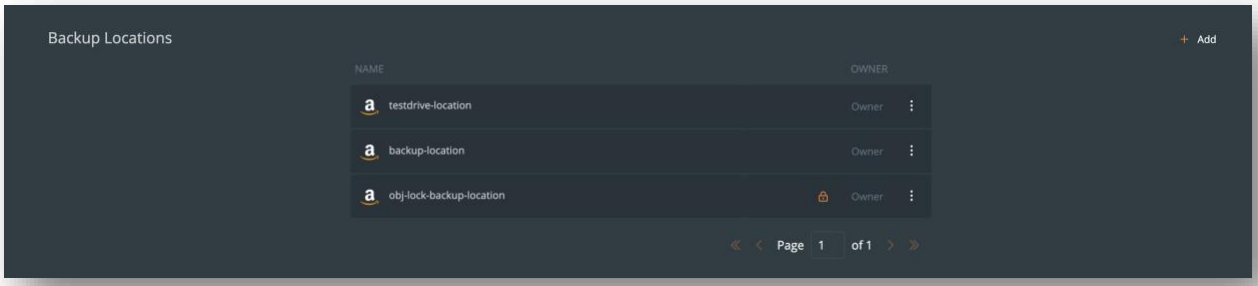
    d. **Region:** us-west-2
    e. **Endpoint:** s3.amazonaws.com
    f. **Storage Class:** Default



3. Click **Add** to add the backup location.
4. Repeat steps 2-4 to add another backup location. This time we will add the object lock enabled S3-bucket (name: objectlock-id-workshop-bucket) using the second cloud account (obj-lock-aws).
    a. **Name:** obj-lock-backup-location
    b. **Cloud Account:** obj-lock-aws (Owner)
    c. **Path / Bucket:** objectlock-id-workshop-bucket
    d. **Region:** us-east-1
    e. **Endpoint:** s3.amazonaws.com
    f. **Storage Class:** Default



5. Once the backup location has been added, you can look at all the available backup locations from the Cloud Settings page. The object lock backup location will be listed with the lock icon on the right. This helps users differentiate between the two types of backup targets supported by BaaS.

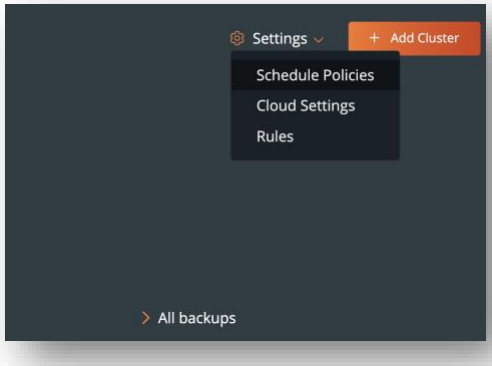


Schedule Policies

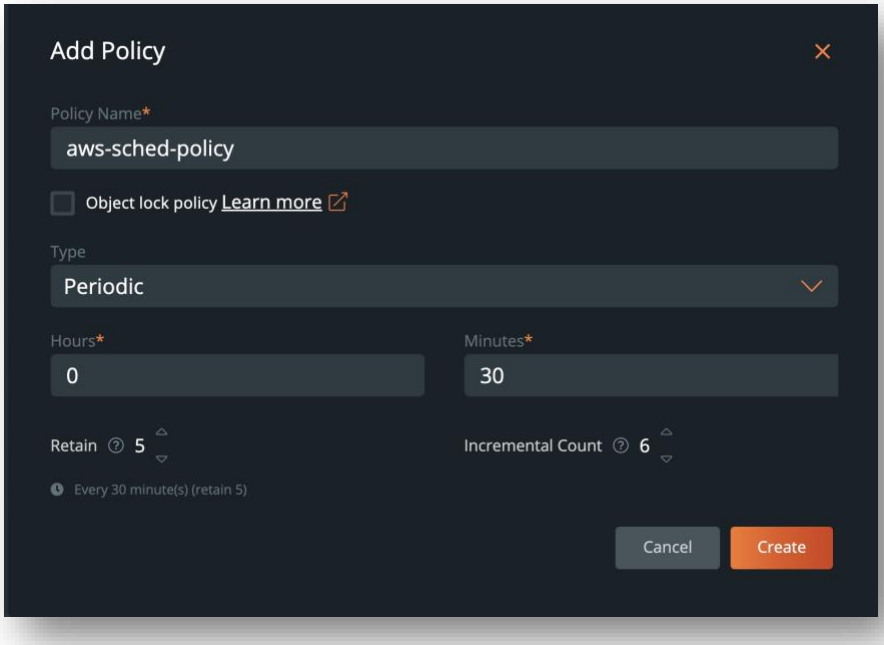
Portworx BaaS allows users to set their own schedule policies that can be used when creating backup jobs. BaaS allows users to either create ad-hoc, manual, one-time backup to create backup jobs that get triggered at a regular schedule. Using Schedule policies, users can control when they want to trigger backup jobs. These can be periodic, hourly, daily, weekly, and monthly policies.

Use the following steps to create a 30-min backup policy that we will use to create backup jobs:

1. Navigate to the dashboard and click on the *"Settings"* and *"Schedule Policies"* on the top right.



2. Click the *"+"* sign on the top right to add a new Schedule policy.
3. Give the schedule policy a name. If you are going to use this policy to create object-lock enabled backup jobs, then select the object lock policy checkbox. If you check the box, BaaS won't allow you to configure your own backup retention period but defer to the S3-bucket's retention setting.
    a. **Policy Name:** aws-sched-policy
    b. **Type:** Periodic
    c. **Hours:** 0
    d. **Minutes:** 30



4. Click Create to create the Schedule policy.

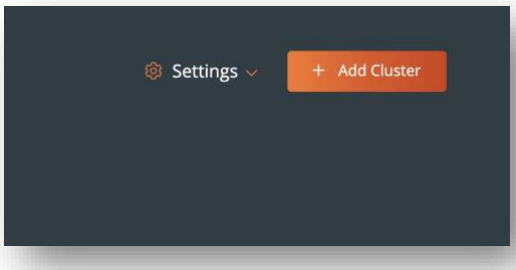
You can create more schedule policies that fit your specific application-level SLAs.

Auto-discover Amazon EKS cluster

```
BaaS allows users to auto-discover their Amazon EKS clusters, rather than copy
and pasting kubeconfig files between the EKS cluster and the BaaS interface.
This allows users to leverage their cloud credentials to access their Amazon
EKS cluster, such that BaaS can inventory the cluster for applications, and
help users create backup jobs quickly. This also avoids a scenario where the
kubeconfig based credentials expire and your scheduled backup jobs start
failing.
```

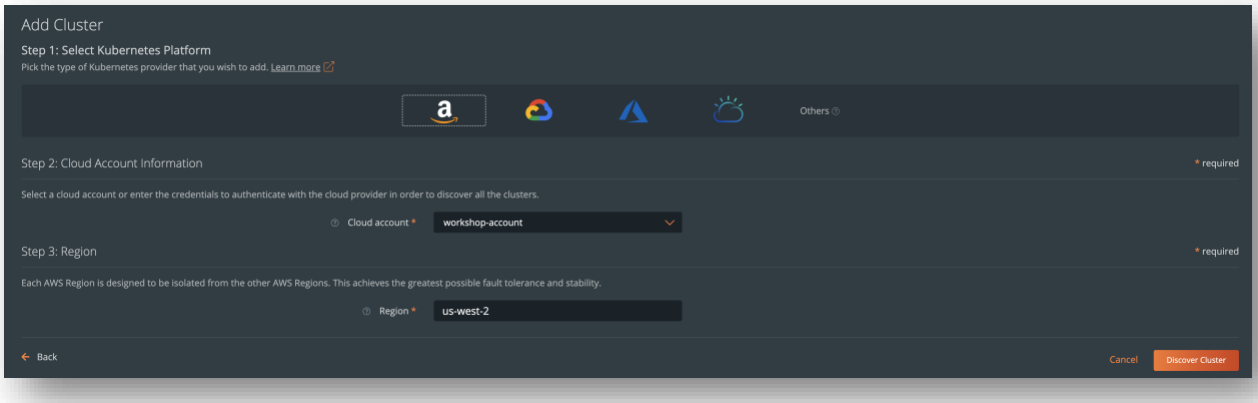
To auto-discover your Amazon EKS clusters, use the following steps:

1. Navigate to the dashboard and click on *"+ Add Cluster"* on the top right of your screen.

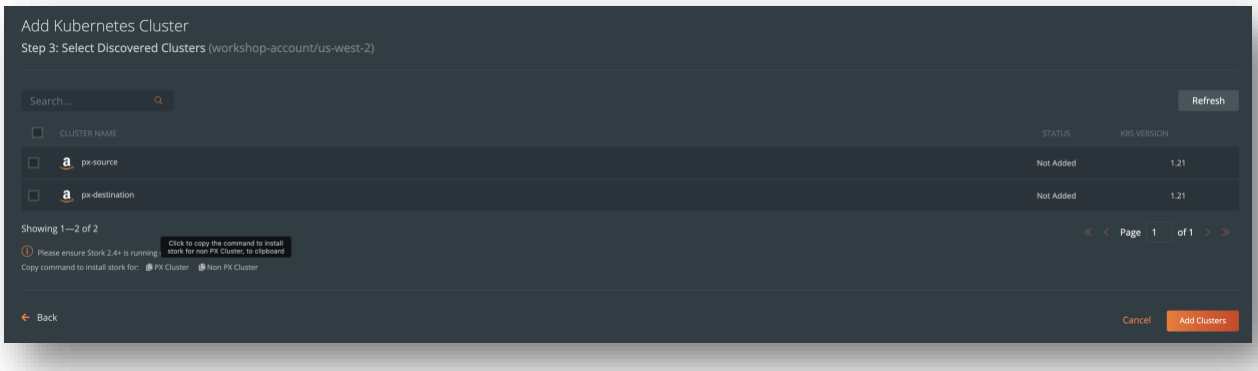


2. Select the Cloud Account we created earlier in the lab from the Cloud Account (workshop-account) drop down box and enter the region (us-west-2) where your Amazon EKS cluster is running and click *"Discover Cluster".*

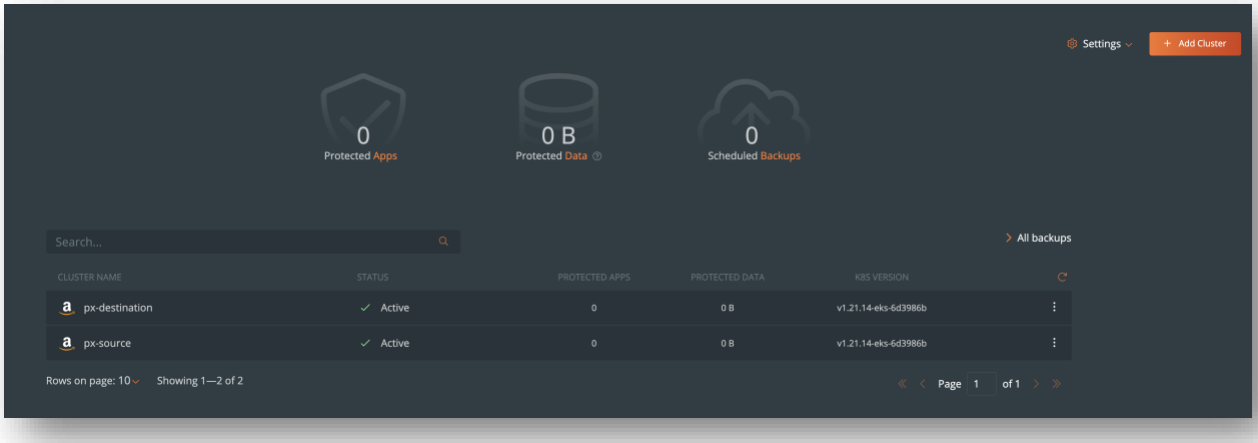
Note: If the cloud account list isn't loading, refresh the webpage.



3. Next, you will be shown a list of Amazon EKS clusters that you have access to. Select the clusters deployed as part of the pre-req (px-source and px-destination) and click *"Add Clusters".*



4. Once the clusters are added, it will show up on the dashboard.



In the next section, we will work with Backup and Restore using BaaS.

Kubernetes Backup and Restore

```
BaaS allows users to create backup jobs to protect their applications. These backup jobs can help users protect Kubernetes resources, application configuration and application data, and store an end-to-end application snapshot in an S3-compatible backup repository. Using Portworx BaaS, users can customize backup rules to include the following scenarios:
1.    Backup everything inside a Kubernetes namespace
2.    Backup a specific resource (like Persistent Volumes) inside a namespace
3.    Backup resources based on a resource label (e.g., app: demo) inside a namespace

This level of flexibility allows users to customize their backup jobs, to comply with their service level agreements (SLAs) or regulations.

In this section, we will cover a couple of different scenarios. We will create a backup for an application, store it in a non-object lock enabled bucket and restore it to the same Amazon EKS cluster and then we will create a backup for another application, store it in an object-lock enabled bucket and then restore it to a completely different cluster.
```

Kubernetes backup and restore – Same cluster

BaaS allows users to protect their applications from accidental deletions, data corruption and data loss, etc. by creating end-to-end application snapshots, and allowing users to restore these applications when needed.

In this section, we will access an application, generate some data, create a backup, simulate accidental deletion, and the use the application snapshot to restore our application back on the same Amazon EKS cluster.

1. From your AWS CloudShell, get the service endpoint (External-IP) for the Pacman application, using the following command:

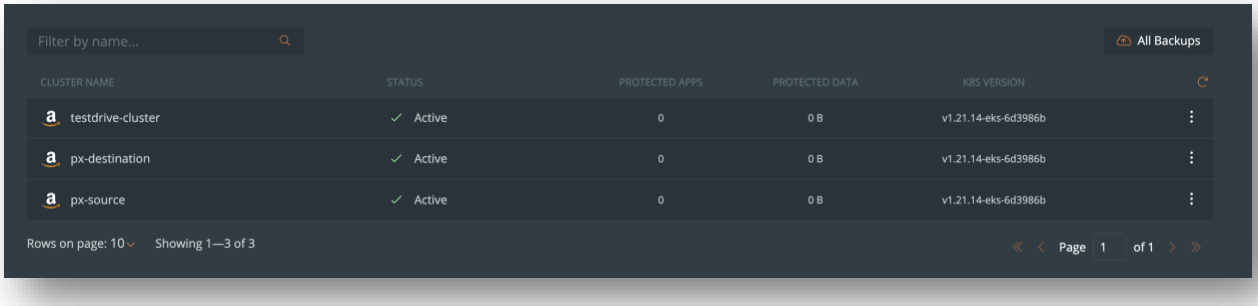
```
kubectl get svc -n pacman -l name=pacman
```

Note: If your CloudShell session has expired, reconnect to CloudShell, change your working directory to PX-DataProtection, and execute the following script.

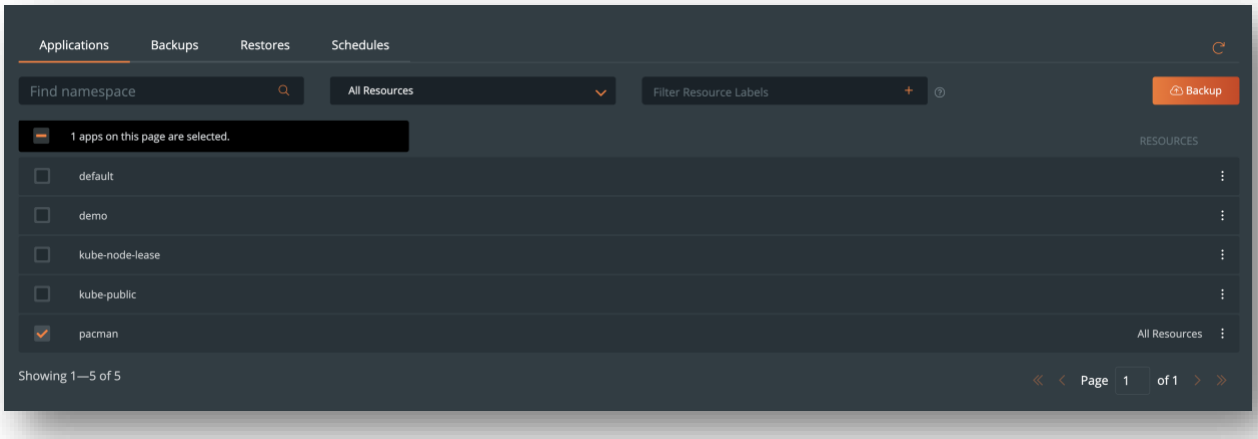
```
cd /home/cloudshell-user/PX-DataWorkshop
./connect-px-source.sh
```

2. Navigate to the application using a browser and play a game using your arrow keys to generate a high score entry. Enter a name for your score and click Save. Next, click “View Highscore List”

3. Now that you have some data stored, let's create a new backup job. Navigate to the BaaS dashboard and select the "px-source" cluster.

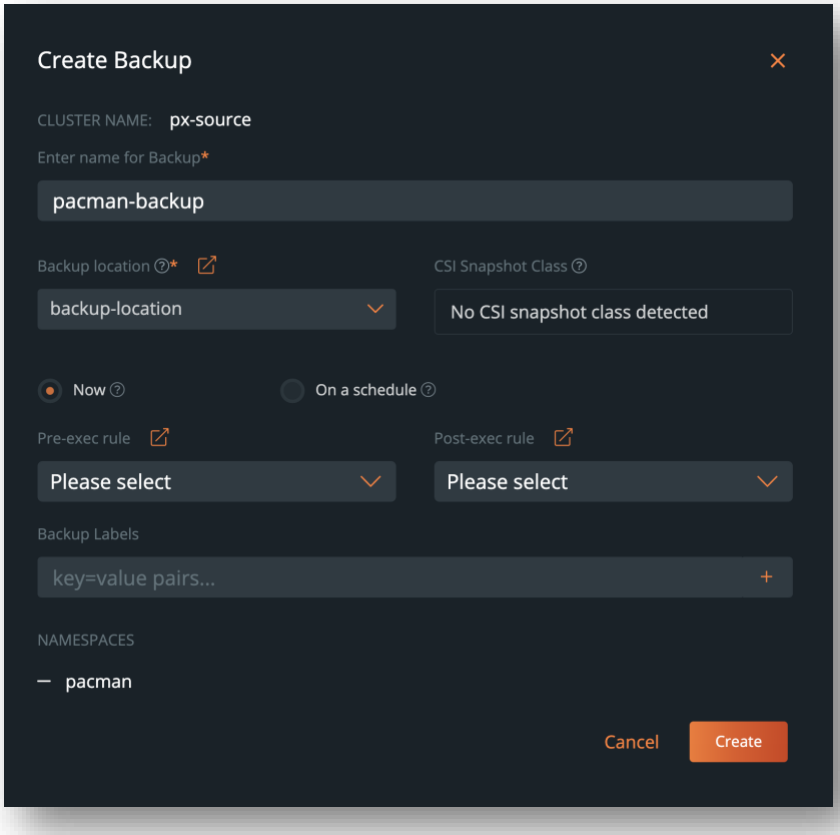


4. In this scenario, we are creating a backup for the entire pacman application, so select the *"pacman"* namespace, and click "**Backup**".

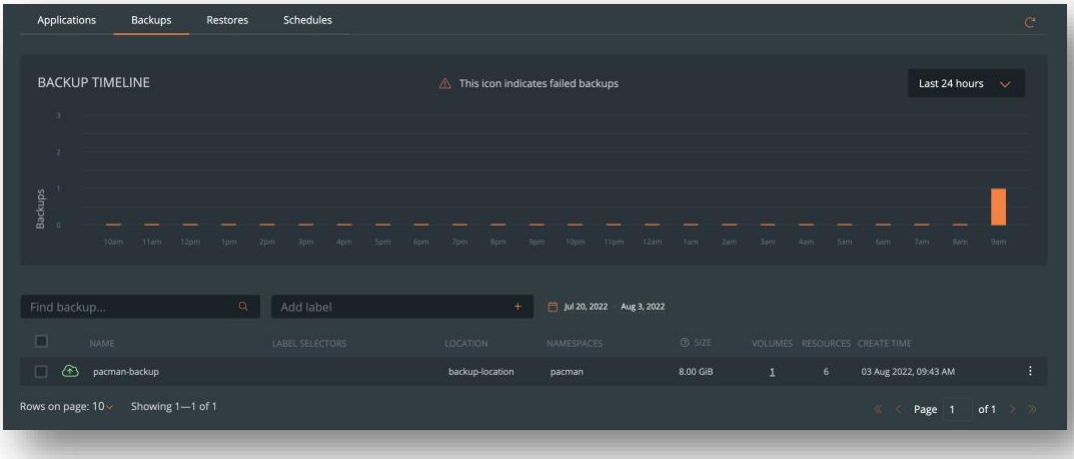


5. Enter a name for the backup and select your non-object lock enabled backup location. And then click **"Create"**.

- Name: pacman-backup
- Backup location: backup-location



6. Once created, BaaS will communicate with the source cluster and initiate an end-to-end application backup. You can monitor the backup job using the Backup tab.



7. Once the backup job is completed successfully, it turns green. Let's simulate a failure and delete your pacman application. To do this, navigate back to your AWS CloudShell, and use the following commands to delete the application.

```
./pacman-uninstall.sh
```

```
[cloudshell-user@ip-10-1-63-151 PX-DataWorkshop]$ ./pacman-uninstall.sh
warning: deleting cluster-scoped resources, not scoped to the provided namespace
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
podsecuritypolicy.policy "pacman" deleted
clusterrole.rbac.authorization.k8s.io "pacman-clusterrole" deleted
rolebinding.rbac.authorization.k8s.io "pacman-clusterrole" deleted
clusterrolebinding.rbac.authorization.k8s.io "pacman-clusterrole" deleted
secret "mongodb-users-secret" deleted
deployment.apps "mongo" deleted
deployment.apps "pacman" deleted
service "mongo" deleted
service "pacman" deleted
persistentvolumeclaim "mongo-storage" deleted
```

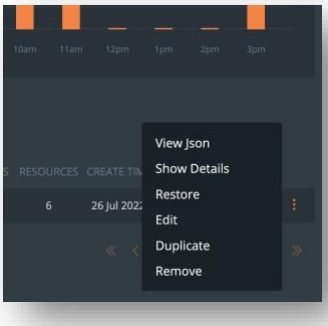
8. It takes a couple of minutes for the application to be deleted. Once deleted, use the following commands to validate that the application has been deleted.

```
kubectl get all -n pacman

kubectl get pvc -n pacman
```

9. Next, navigate back to the BaaS dashboard and click on the px-dataprotection-source cluster and then click on the Backups tab.

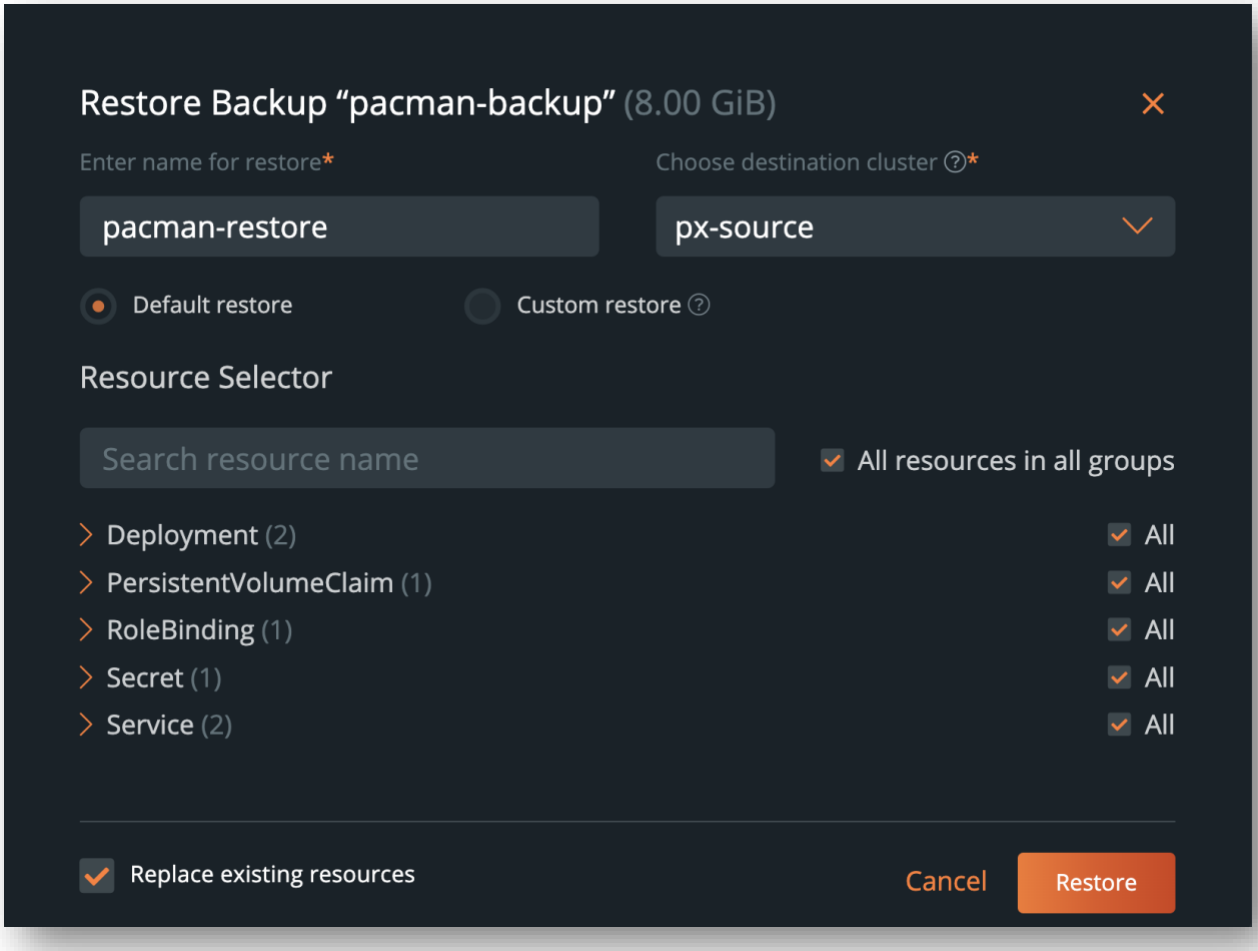
Note: If you get logged out of BaaS, navigate to Portworx Central and log in using your credentials.

10. Click the ":" on the right of your backup snapshot row and click **"Show Details"**. Here you can get additional details about the different resources backed up.

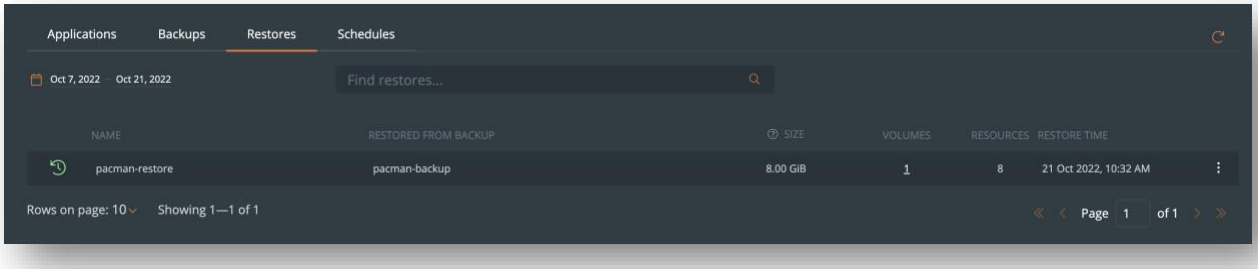


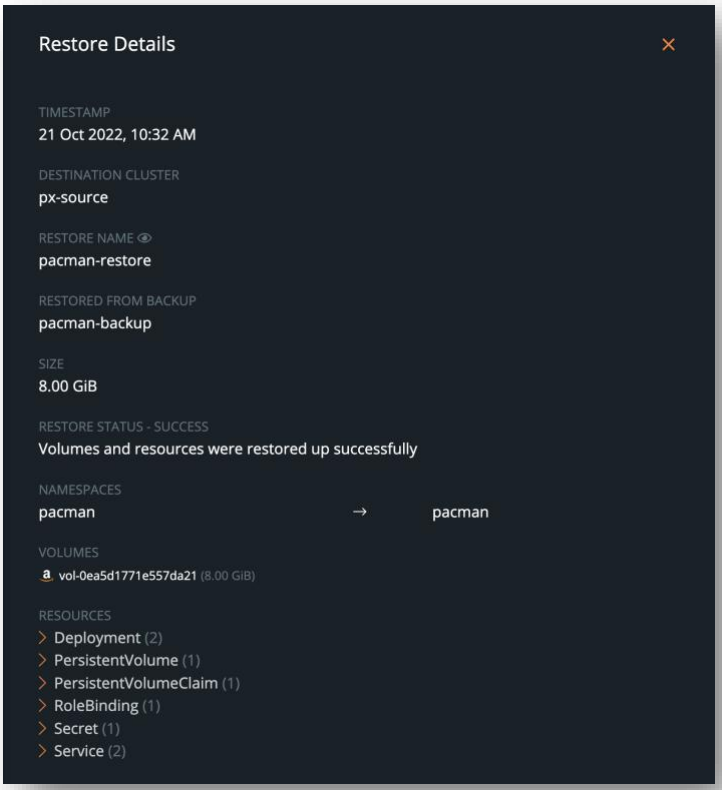
11. You can see that 2 Deployment objects, one persistent volume and one persistent volume claim. one role binding, one secret and 2 service objects were backed up as part of this snapshot.
12. Once you have reviewed the backup snapshot, click "Restore Backup". If you cancelled out of the previous screen, then select the backup snapshot again, click on the ":" on the right of your backup snapshot row and click "Restore".

13. Let's give your restore job a name (pacman-restore) and select the destination cluster. In this scenario, we are restoring the application back to the same cluster, so we will select the "px-source" cluster.



14. Verify that "All resources in all groups" checkbox is selected and click the "Replace existing resources" check box and hit Restore.
15. Once the restore is successful, you can click on the ":" on the restore object row and click on Show Details. This will give you a list of everything that was restored.





Note: If the restore is only partially successful, run the ./pacman-uninstall.sh script again, and wait 5 mins before running through steps 11-14.

16. Navigate back to the AWS CloudShell and use the following commands to validate the restore operation.

```
kubectl get all -n pacman

kubectl get pvc -n pacman
```

17. Use the following command to get the load balancer endpoint (External-IP) for the restored pacman application.

```
kubectl get svc -n pacman -l name=pacman
```

18. Click on High Scores and you can validate that the high scores you created earlier in the lab, are still persistent through an accidental deletion event.

This is how easy it is to use Portworx BaaS to protect your containerized applications running on Amazon EKS clusters!

Ransomware protection using Backup As A Service

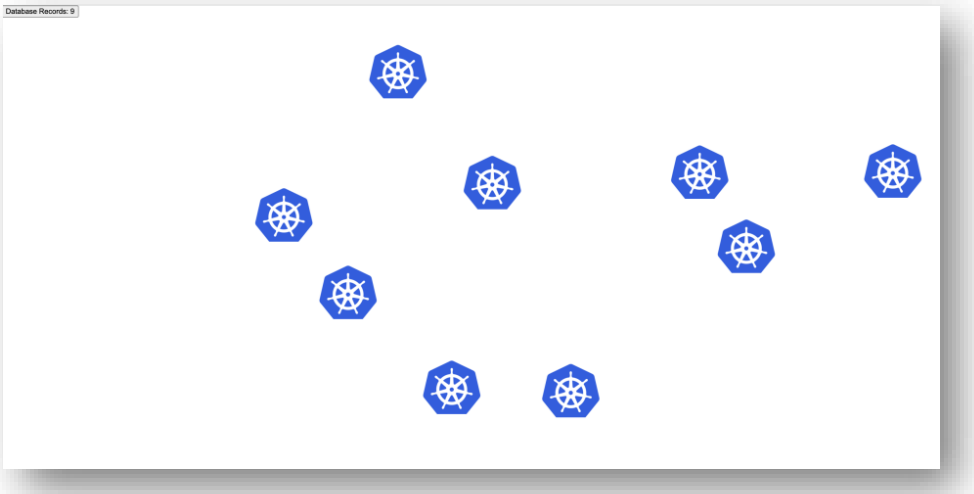
According to the Data Protection Trends report 2022, 76% of the organizations have suffered from at least one ransomware attack. These attacks are not limited to traditional or legacy infrastructure stacks, but also cover Kubernetes and modern applications running on Kubernetes. Organizations need a solution that can help them protect against such attacks and ensure that they have an insurance policy in place, to recover their applications securely on non-infected infrastructure. In this case, having an insurance policy is synonymous to having an immutable copy of your application snapshot stored offsite, so even if your primary infrastructure gets compromised, hackers can't modify your backups.
To support this use case, Portworx Backup As A Service allows users to leverage object lock enabled backup buckets to store their application snapshots. This allows app snapshots to be Write Once Read Many (WORM), and thus immutable from any ransomware attack.

In this scenario, we will look at how we can leverage BaaS to take a WORM snapshot for our application and then restore the application to a secondary Amazon EKS cluster.

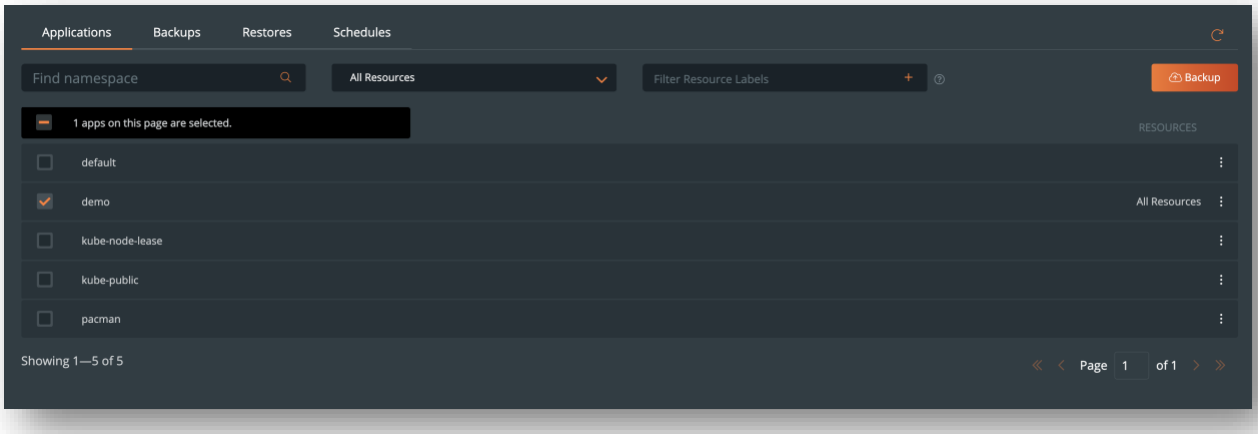
1. Navigate to the AWS CloudShell tab and use the following command to get the load balancer endpoint for our demo application (k8s-counter-service). This is a different application running in the *"demo"* namespace.

```
kubectl get svc -n demo k8s-counter-service
```

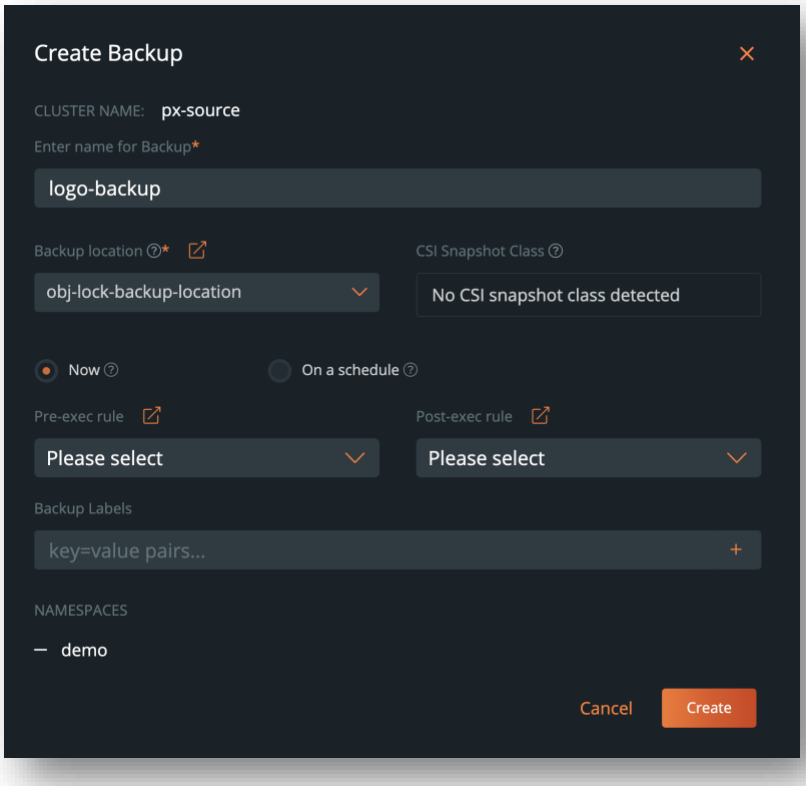
2. Copy the load balancer endpoint for the *"k8s-counter-service"* and navigate to the application using your web browser. This is a simple application, which generates a Kubernetes logo wherever your click on the screen and then updates the database record by storing the (x,y) coordinates in a backend Postgres database.



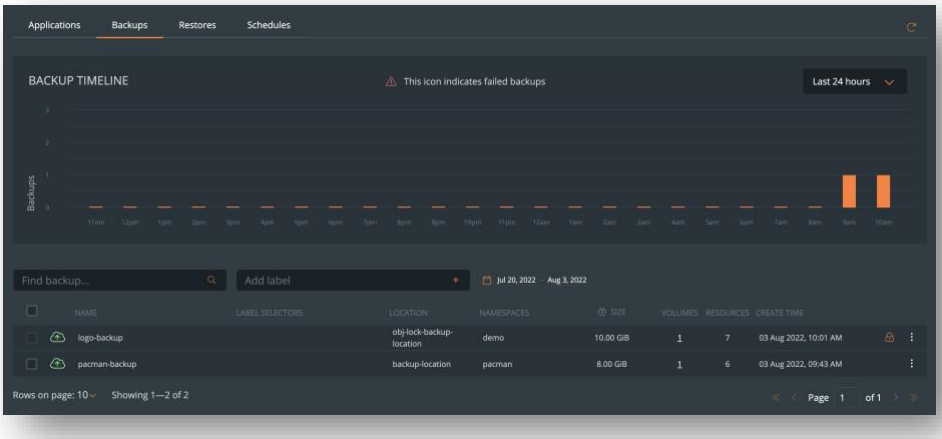
3. Once you have some data generated, let's head to the BaaS interface to create an immutable backup job. Navigate back to the BaaS Dashboard and click on the px-source cluster and select the **"demo"** namespace and click Backup.



4. Give the backup job a name and select the object lock enabled backup bucket as the backup location and hit *"Create"*.
    a. Enter name for Backup: logo-backup
    b. Backup location: obj-lock-backup-location



5. This will create an immutable backup snapshot and store it in the Object lock enabled S3 bucket. Once the backup is successfully complete, you will see a lock sign on the snapshot, indicating that this is a WORM backup snapshot.

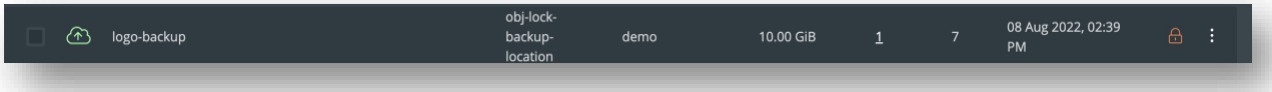


6. Next, let's go ahead and delete the application from our source cluster, simulating a ransomware attack, where the hacker has encrypted your primary application. In our simulation, we are just deleting the application completely from our source Amazon EKS cluster. To do this, navigate to the AWS CloudShell and run the following command:

```
./encryptlogo.sh

kubectl get all -n demo

kubectl get pvc -n demo
```

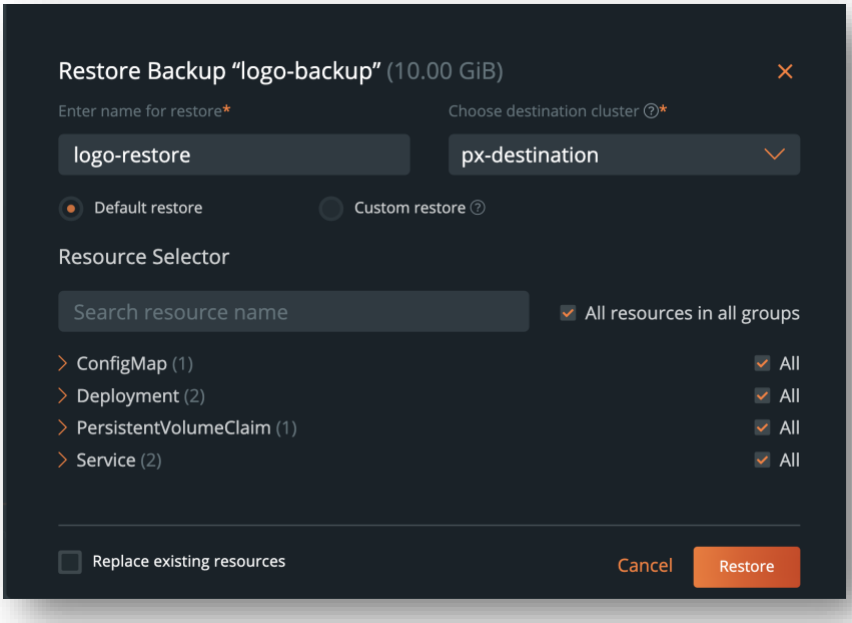
```
[cloudshell-user@ip-10-0-48-97 PX-DataProtection]$ ./encryptlogo.sh
deployment.apps "k8s-counter-deployment" deleted
service "k8s-counter-service" deleted
persistentvolumeclaim "postgres-data" deleted
configmap "example-config" deleted
deployment.apps "postgres" deleted
service "pg-service" deleted
Your Amazon EKS cluster has been under attack! All your applications have been encrypted!
```

7. Now, that your primary Amazon EKS cluster has been compromised, you will have to use your immutable application snapshot and restore your application to a second Amazon EKS cluster. To do that, let's navigate back to the BaaS dashboard, select the **"px-source"** cluster, backups tab, and select the object lock protected backup snapshot.
8. Click on the ":" on the backup snapshot row and click Restore.

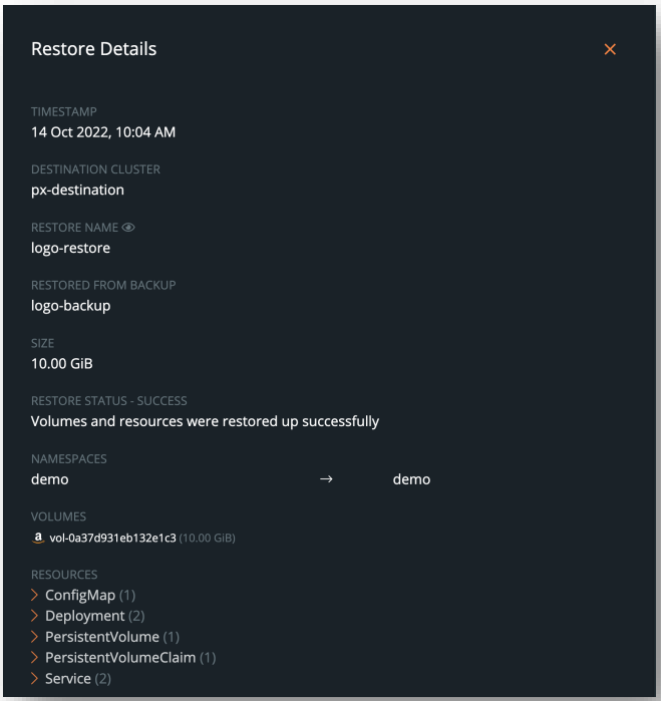


9. Give the restore job a name (logo-restore) and select the **"px-destination"** cluster as the destination cluster. Verify that all the resources are selected and click *"Restore"*.
    a. Enter name for restore: logo-restore

b. Choose destination cluster: px-destination



10. This initiates a restore operation on the second Amazon EKS cluster. Once the restore is successfully complete, click on the ":" and click Show Details. Here you can see a list of all the resources that were restored from your immutable backup to your new Amazon EKS cluster.



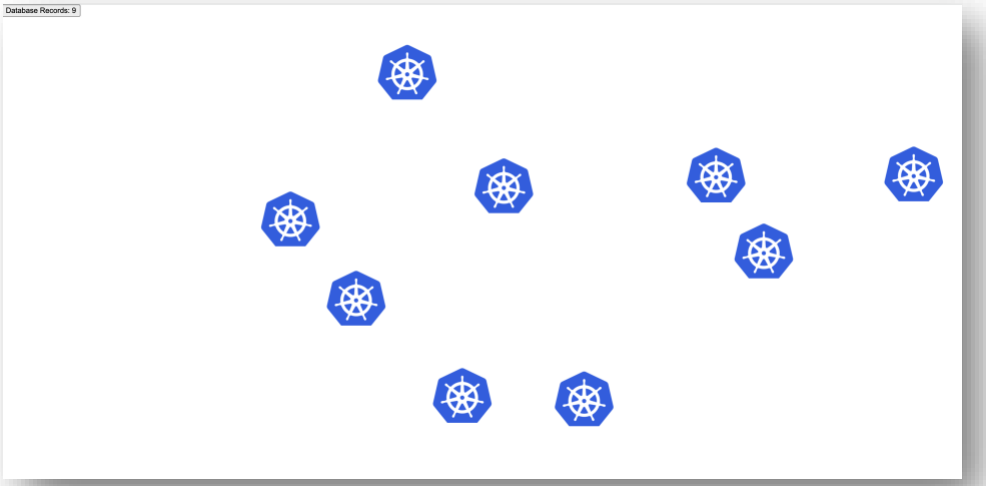
11. Navigate back to the AWS CloudShell, so we can log into our second cluster and validate that all our application data was restored. To do that use the following commands:

```
./connect-px-destination.sh

kubectl get all -n demo
kubectl get pvc -n demo
```

12. Get the load balancer endpoint using the following command and navigate to our application using a web browser.

```
kubectl get svc -n demo
```

13. Here, you should see the same number of Kubernetes logos at the exact same location, indicating that our application restore has been successful.



This is how users can leverage BaaS to protect their applications and recover them in case of a ransomware attack.

End of the Data Protection Lab

At this point, we are done with the Data Protection section of the Data Workshop. Use the following command, to get ready for the next section of the lab focused on Data Management for Kubernetes.

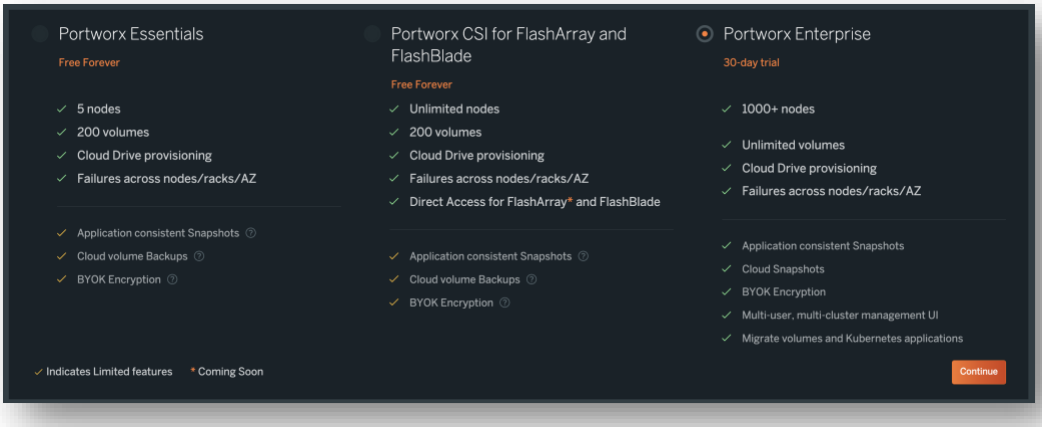
```
./wrapupdataprotection.sh
```

Before we proceed to the next section of the workshop, please navigate to the Portworx Data Services control plane (https://prod.pds.portworx.com/), so we can add everyone to the workshop organization.

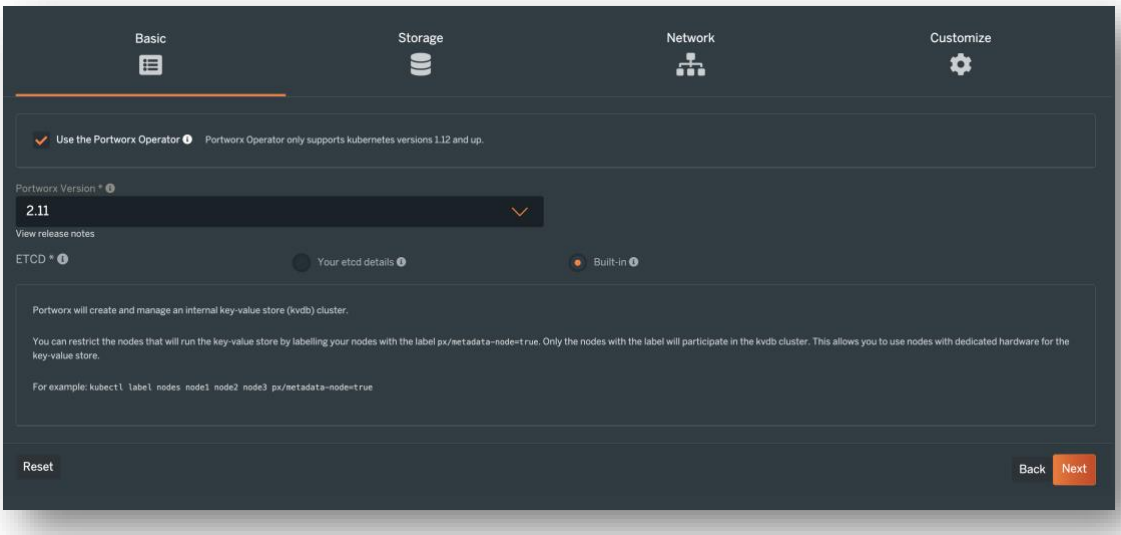
Deploying Portworx Enterprise on Amazon EKS

1. Navigate to the Portworx Central *"Product Catalog"* tab on the left pane and select Portworx Enterprise option and click *"Continue"*.

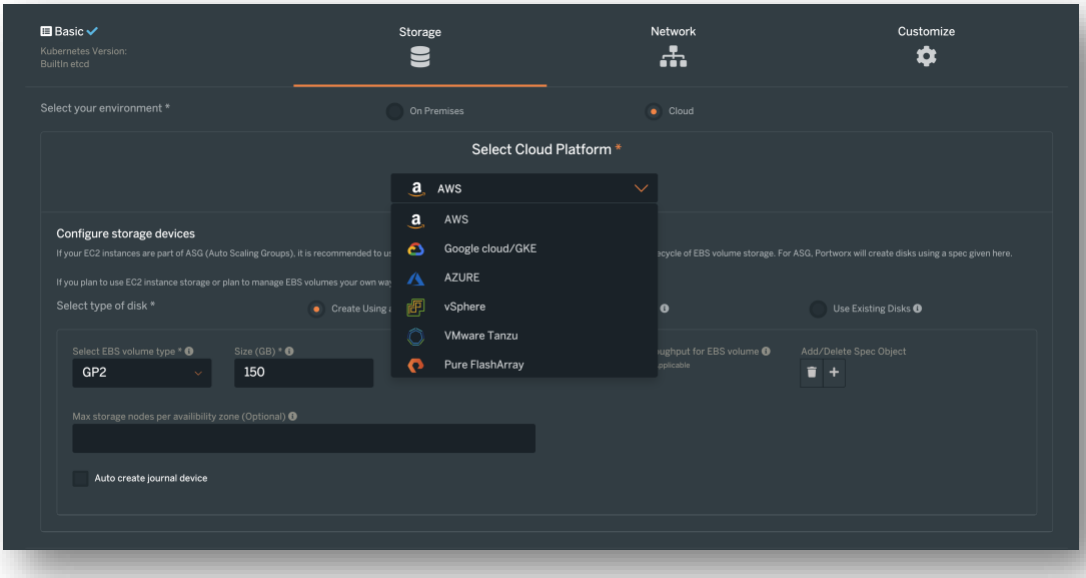
2. Select the "Portworx Enterprise – 30-day trial" click Continue.



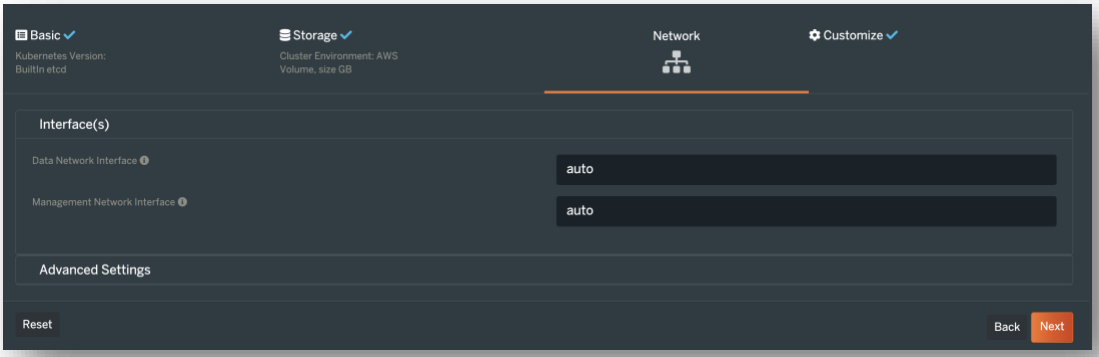
3. Check the box for "Portworx Operator" (If it's not checked already) and select 2.11 as the "Portworx Version" from the drop-down box. We will use the "Built-in" ETCD instance to use as the Key Value Database (KVDB) for Portworx. Click Next.



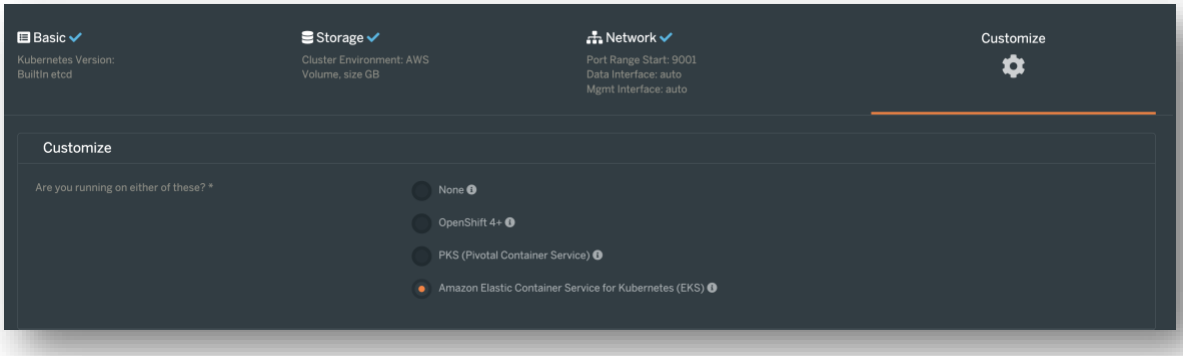
4. Select “Cloud” and select “AWS”. Click Next.

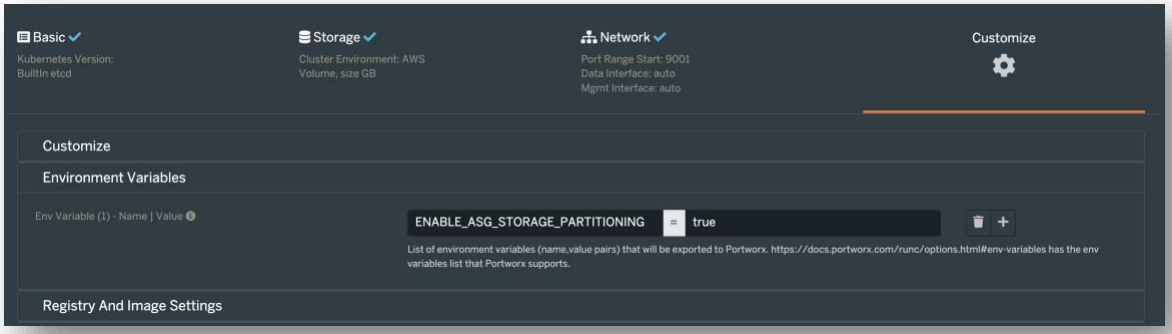


5. Leave the Network settings as default and click Next.



6. Select “Amazon Elastic Container Service for Kubernetes (EKS)” in the Customize tab, and then add an Environment Variable – “ENABLE_ASG_STORAGE_PARTITIONING: true”. Click Finish.



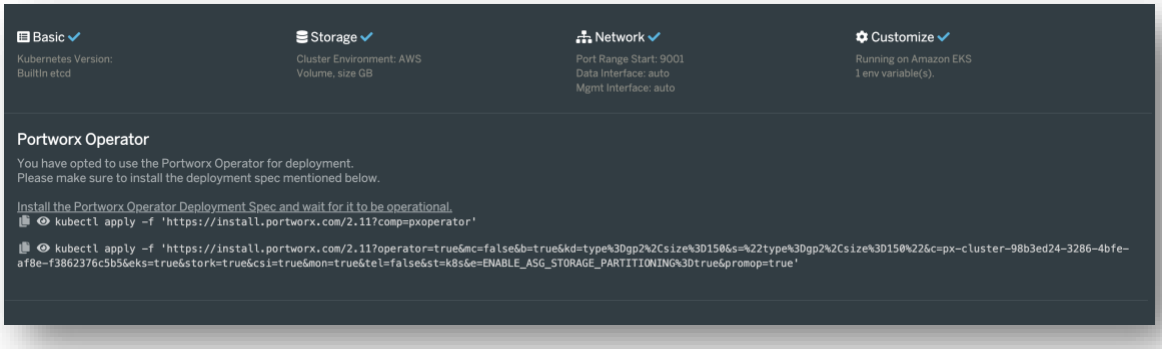


7. Read through the Portworx End User License Agreement and click Agree.
8. Copy both the 'kubectl' commands under the Portworx Operator section, we will use it to deploy Portworx Enterprise on our Amazon EKS cluster.
9. Next, navigate back to your AWS CloudShell session, and use the following command to connect to your Amazon EKS cluster.

Note: If your AWS CloudShell session has expired, use the following two commands to connect to your Amazon EKS cluster. You can also use these commands, to reconnect back to your Amazon EKS cluster at any point in the guide.

```
cd /home/cloudshell-user/PX-DataWorkshop
./connect-px-source.sh
```

10. Paste the first kubectl command from Portworx Central to deploy the Portworx Operator, wait a minute, and then paste the second kubectl command to deploy the Portworx Storage Cluster.




```
[cloudshell-user@ip-10-0-90-146 PX-DataMgmt]$ ./connect-eks.sh
2022-10-03 13:56:24 [✓]  saved kubeconfig as "/home/cloudshell-user/.kube/config"
[cloudshell-user@ip-10-0-90-146 PX-DataMgmt]$ kubectl apply -f 'https://install.portworx.com/2.11?comp=pxoperator'
serviceaccount/portworx-operator created
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
podsecuritypolicy.policy/px-operator created
clusterrole.rbac.authorization.k8s.io/portworx-operator created
clusterrolebinding.rbac.authorization.k8s.io/portworx-operator created
deployment.apps/portworx-operator created
[cloudshell-user@ip-10-0-90-146 PX-DataMgmt]$ kubectl apply -f 'https://install.portworx.com/2.11?operator=true&mc=false&b=true&kd=
type%3Dgp2%2Csize%3D150&s=%22type%3Dgp2%2Csize%3D150%22&c=px-cluster-98b3ed24-3286-4bfe-af8e-f3862376c5b5&eks=true&stork=true&csi=t
rue&mon=true&tel=false&st=k8s&e=ENABLE_ASG_STORAGE_PARTITIONING%3Dtrue&promop=true'
storagecluster.core.libopenstorage.org/px-cluster-98b3ed24-3286-4bfe-af8e-f3862376c5b5 created
[cloudshell-user@ip-10-0-90-146 PX-DataMgmt]$
```

11. To monitor the Portworx Enterprise deployment, you can use the following command:

```
kubectl get pods -n kube-system -w
```

Note: You can use “Ctrl + C” to exit out of the kubectl watch command.

12. Once all the Portworx storage cluster pods are 2/2, your Portworx storage cluster is ready to go!

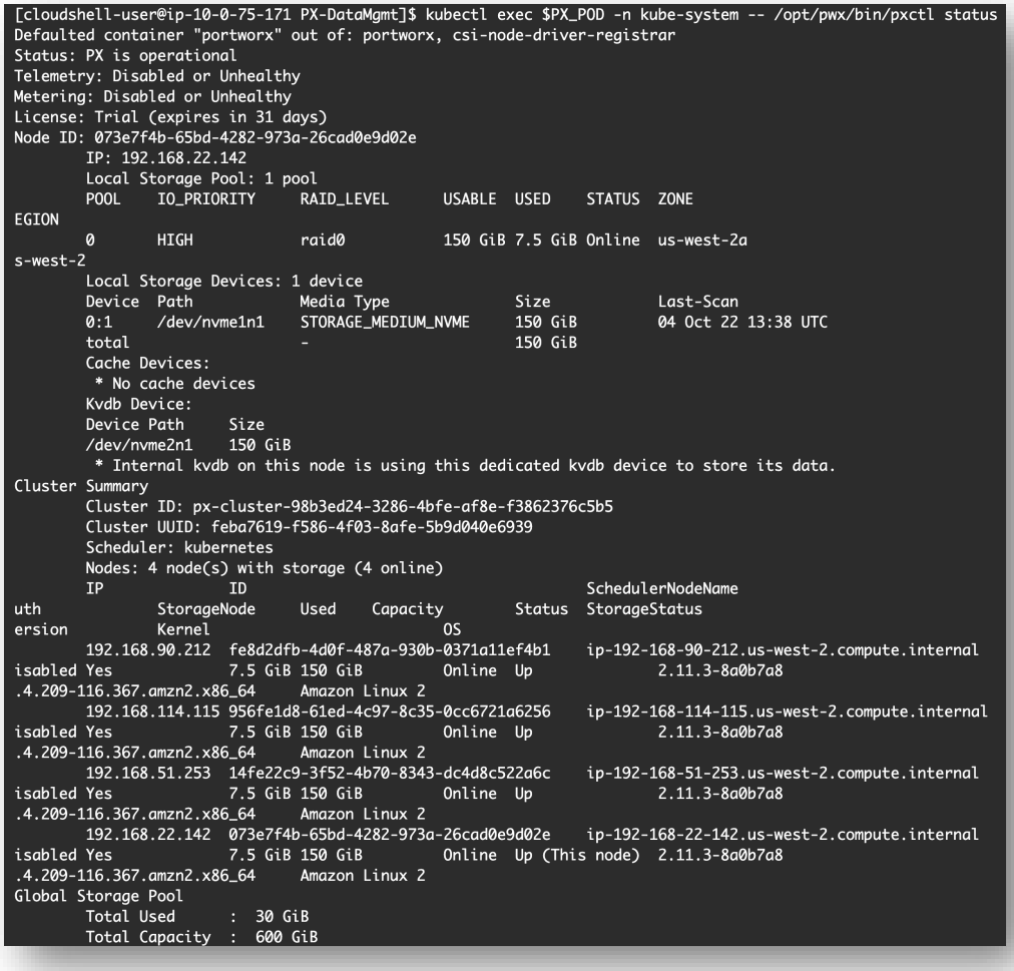
```
px-cluster-98b3ed24-3286-4bfe-af8e-f3862376c5b5-5zfgm   2/2     Running   0          3m1s
px-cluster-98b3ed24-3286-4bfe-af8e-f3862376c5b5-fm6fk   2/2     Running   0          3m1s
px-cluster-98b3ed24-3286-4bfe-af8e-f3862376c5b5-g997r   2/2     Running   0          3m1s
px-cluster-98b3ed24-3286-4bfe-af8e-f3862376c5b5-zrjng   2/2     Running   0          3m1s
```

13. Use the following command to look at the Portworx StorageCluster status and the pre-created storageclasses.

```
kubectl get storagecluster -n kube-system

kubectl get sc
```

Note: Copy the following two commands together and hit Enter.

```
PX_POD=$(kubectl get pods -l name=portworx -n kube-system \
-o jsonpath='{.items[0].metadata.name}')

kubectl exec $PX_POD -n kube-system -- /opt/pwx/bin/pxctl status
```



14. Install Grafana on your Amazon EKS cluster using the following command:

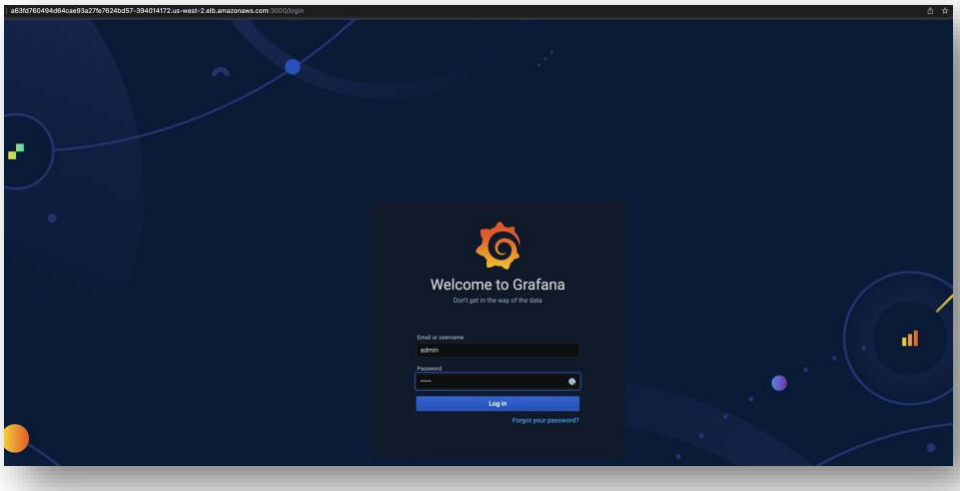
```
./install-grafana.sh
```

15. Navigate to the Grafana UI using the load balancer endpoint for Grafana over port 3000, and login using the default credentials (admin/admin).

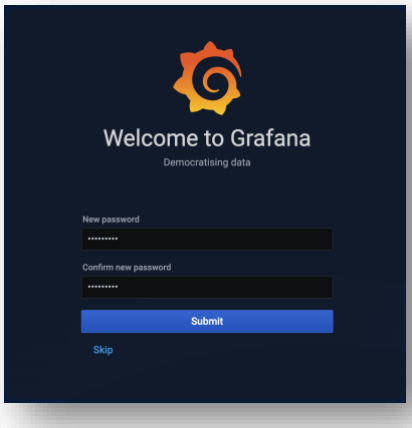
Note: Append port 3000 at the end of the load balancer URL (See example below) and navigate to the Grafana dashboard

```
http://adc886f8c913040bf8da8ade62f267c7-1960369372.us-west-
2.elb.amazonaws.com:3000/
```

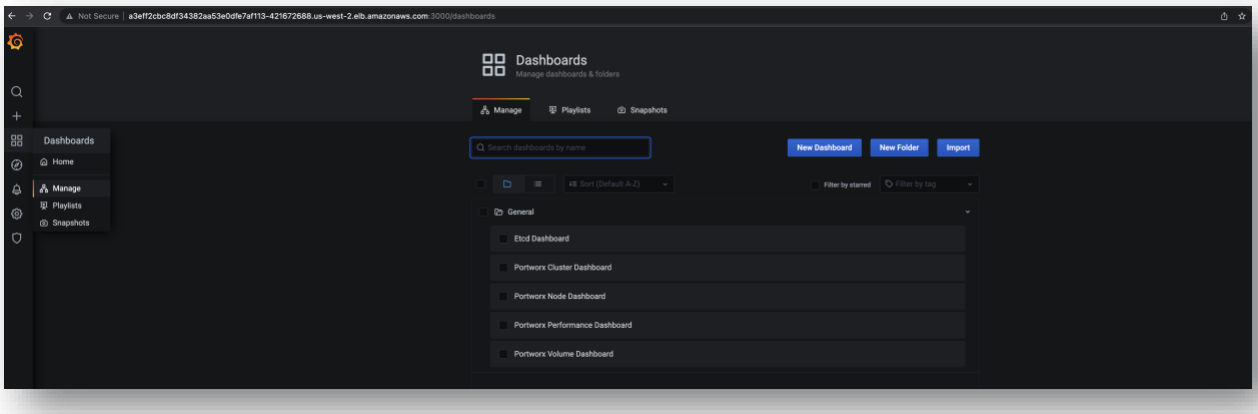
Note: If you are having issues connecting to the Amazon ELB endpoint, please disconnect from any VPNs you might be using. It might take a couple of minutes for the Grafana dashboard to be accessible.

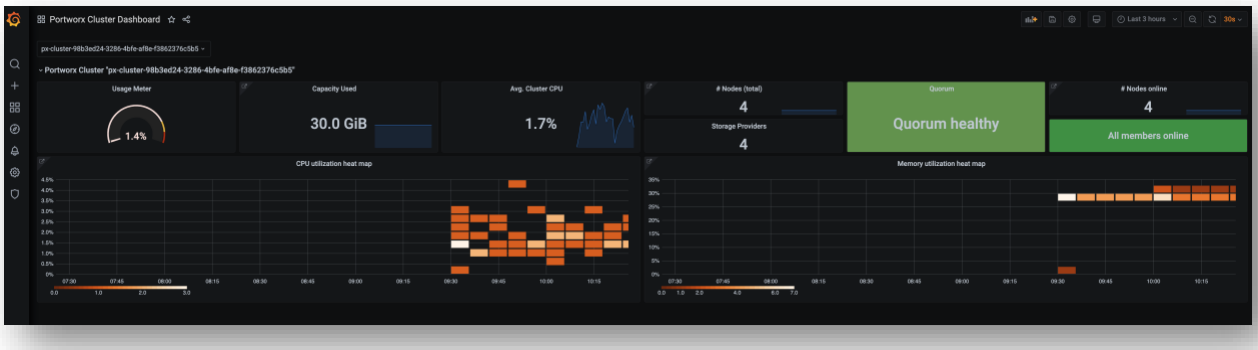


16. Set a new password for your Grafana instance on the next screen and then log into the Grafana instance.



17. Find the Portworx cluster dashboard by clicking on the Dashboard → Manage in the left pane, and then click on Portworx Cluster Dashboard.





In the next section, we will deploy a couple of demo applications, to learn how Portworx can help you customize your StorageClass definitions and provision ReadWriteOnce and ReadWriteMany volumes from a unified storage pool.

Dynamic storage provisioning using Portworx Storage Classes

In this section, we will deploy two different storage classes, one for block and one for file persistent volumes, and then deploy demo applications that use those storage classes to deploy persistent volumes on demand.

1. Use the following commands to look at the StorageClass configuration and deploy them against your Amazon EKS cluster.

```
cat block-sc.yaml
```

```
kubectl apply -f block-sc.yaml
```

```
cat file-sc.yaml
```

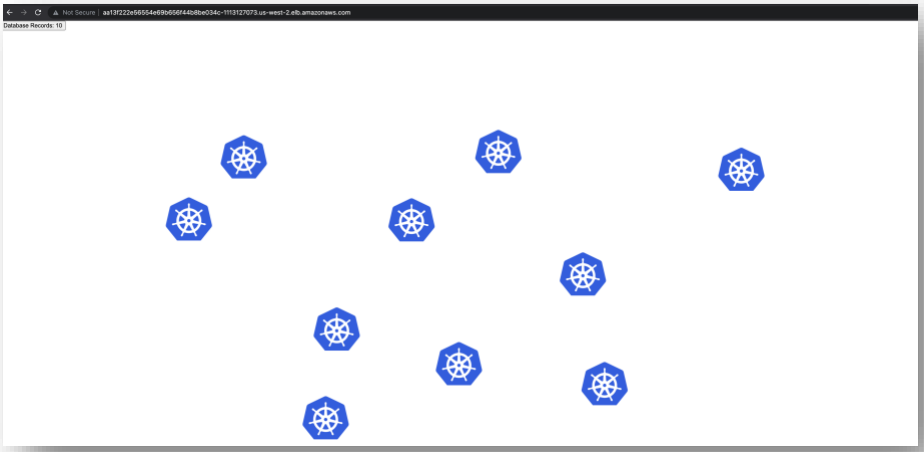
```
kubectl apply -f file-sc.yaml
```

Deploying Block-based (ReadWriteOnce) application on Amazon EKS

1. Let’s deploy a simple demo application that has a PostgreSQL database for backend and a simple nginx based frontend component.

```
./k8s-app.sh
```

2. Once the script completes execution, copy, and navigate to the Amazon ELB endpoint using your browser and click a few times on the screen, to generate Kubernetes logos. The coordinate locations (X,Y) for these logos are stored in the backend Postgres database.



3. Next, let's use the following command to inspect the Postgres volume. You will see how the Portworx persistent volume has been provisioned, the size, the file system format, the read and write throughout and IOPS, and how many replicas and where they are stored in your Amazon EKS cluster.

```
cat inspect-postgres-vol.sh
```

```
./inspect-postgres-vol.sh
```

```
[cloudshell-user@ip-10-0-81-111 PX-DataMgmt]$ ./inspect-postgres-vol.sh
        Volume                  :  419191747958813387
        Name                    :  pvc-75d1a2c5-e097-4397-ad62-54ad15628c38
        Size                    :  5.0 GiB
        Format                  :  ext4
        HA                      :  3
        IO Priority             :  LOW
        Creation time           :  Oct 3 17:40:10 UTC 2022
        Shared                  :  no
        Status                  :  up
        State                   :  Attached: 2a12a8af-433d-4aec-8e7f-26eb136b1f6e (192.168.79.238)
        Last Attached           :  Oct 3 17:49:40 UTC 2022
        Device Path             :  /dev/pxd/pxd419191747958813387
        Labels                  :  app=postgres,io_profile=auto,namespace=demo,pvc=postgres-data,repl=3
        Mount Options           :  discard
        Reads                   :  151
        Reads MS                :  140
        Bytes Read              :  2695168
        Writes                  :  2175
        Writes MS               :  1053
        Bytes Written           :  8376320
        IOs in progress         :  0
        Bytes used              :  35 MiB
        Replica sets on nodes:
                Set 0
                  Node          : 192.168.1.216 (Pool fa784f7a-502e-4877-bdaf-a8b6e583fffc )
                  Node          : 192.168.101.128 (Pool 65cdcd98-9777-467a-891b-61f23370826b )
                  Node          : 192.168.79.238 (Pool 5d81f81a-a948-4665-a165-7d17f1dd34bd )
        Replication Status      :  Up
        Volume consumers        :
                - Name          : postgres-7957478b7d-c274l (79b6d9fb-981f-4881-90e4-60b76b931f5a) (Pod)
                  Namespace     : demo
                  Running on    : ip-192-168-79-238.us-west-2.compute.internal
                  Controlled by : postgres-7957478b7d (ReplicaSet)
```

4. Show the entries in the Postgres table using the following commands

```
POD=$(kubectl get pods -l app=postgres -n demo | grep 1/1 | awk '{print $1}')
kubectl exec -it $POD -n demo -- bash
psql -U $POSTGRES_USER
\c postgres
Select * from mywhales;
\q
```

```
exit
```

Deploying File-based (ReadWriteMany) application on Amazon EKS

```
Applications like Jenkins, Wordpress, etc need a shared persistent volume between different application pods. This is where the ReadWriteMany capability of Portworx Enterprise helps users leverage a single solution for both Block and File needs. Users can create custom Kubernetes StorageClass for block and file based applications and use the same underlying storage pool for provisioning block and file persistent volumes.
```

In this section, we will deploy a Highly available Jenkins deployment using a simple helm chart and a single ReadWriteMany persistent volume.

1. Let's look at the PVC config file and the script we will use to deploy Jenkins, and then run that script.

```
cat jenkins-pvc.yaml
```

```
cat jenkins-deploy.sh
```

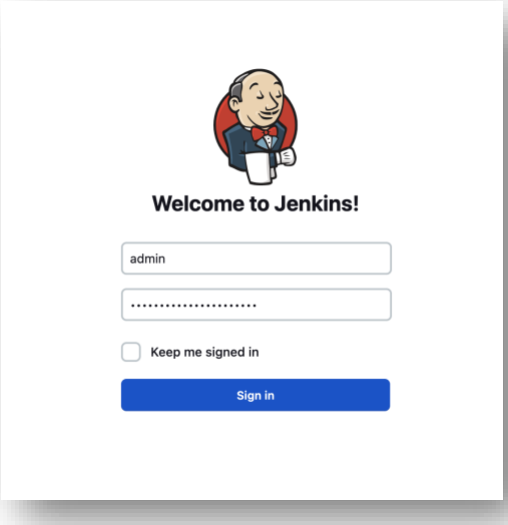
Before we execute the jenkins-deploy.sh command, let's run the install-helm.sh command, to ensure the jenkins deployment succeeds.

```
./install-helm.sh
```

```
./jenkins-deploy.sh
```

```
[cloudshell-user@ip-10-0-86-13 PX-DataMgmt]$ ./jenkins-deploy.sh
namespace/jenkins created
persistentvolumeclaim/jenkins-claim created
"jenkins" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "jenkins" chart repository
Update Complete. ⎈Happy Helming!⎈
NAME: jenkins
LAST DEPLOYED: Fri Oct  7 14:38:20 2022
NAMESPACE: jenkins
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'admin' user password by running:
  kubectl exec --namespace jenkins -it svc/jenkins -c jenkins -- /bin/cat /run/secrets/additional/chart-admin-password && echo
2. Get the Jenkins URL to visit by running these commands in the same shell:
  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        You can watch the status of by running 'kubectl get svc --namespace jenkins -w jenkins'
  export SERVICE_IP=$(kubectl get svc --namespace jenkins jenkins --template "{{ range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
  echo http://$SERVICE_IP:80/login

3. Login with the password from step 1 and the username: admin
4. Configure security realm and authorization strategy
5. Use Jenkins Configuration as Code by specifying configScripts in your values.yaml file, see documentation: http:///configuration-as-code and examples: https://github.com/jenkinsci/configuration-as-code-plugin/tree/master/demos

For more information on running Jenkins on Kubernetes, visit:
https://cloud.google.com/solutions/jenkins-on-container-engine

For more information about Jenkins Configuration as Code, visit:
https://jenkins.io/projects/jcasc/


NOTE: Consider using a custom image with pre-installed plugins
statefulset.apps/jenkins patched
Jenkins Endpoint:
NAME      TYPE           CLUSTER-IP      EXTERNAL-IP                                                               PORT(S)        AGE
jenkins   LoadBalancer   10.100.38.226   abd60cb0b1493475980e198bce19bf34-691572150.us-west-2.elb.amazonaws.com   80:31108/TCP   72s
Jenkins Username:
admin
Jenkins Password:
xCZ87ZgfUgjj3Ux0q2yLya
```

2. Navigate to Jenkins dashboard and log in using the credentials on the screen displayed on the screen.



3. Next, go back to your CloudShell session and inspect the persistent volume using the following script.

```
cat inspect-jenkins-vol.sh

./inspect-jenkins-vol.sh
```

```
[cloudshell-user@ip-10-0-86-13 PX-DataMgmt]$ ./inspect-jenkins-vol.sh
Defaulted container "portworx" out of: portworx, csi-node-driver-registrar
Defaulted container "portworx" out of: portworx, csi-node-driver-registrar
       Volume                  :  667463062575129140
       Name                    :  pvc-dcb276a1-bda8-4f30-8ccb-02fb26a46376
       Size                    :  15 GiB
       Format                  :  ext4
       HA                      :  2
       IO Priority             :  LOW
       Creation time           :  Oct 7 14:38:01 UTC 2022
       Shared                  :  v4 (service)
       Status                  :  up
       State                   :  Attached: ea8f8b4f-80bf-4e89-b364-d37700d9b361 (192.168.34.121)
       Last Attached           :  Oct 7 14:38:22 UTC 2022
       Device Path             :  /dev/pxd/pxd667463062575129140
       Labels                  :  namespace=jenkins,pvc=jenkins-claim,repl=2,sharedv4=true,sharedv4_svc_type=ClusterIP
       Mount Options           :  discard
       Sharedv4 Client Mount Options           :  actimeo=60,port=2049,proto=tcp,retrans=4,soft,timeo=600,vers=4.0
       Reads                   :  59
       Reads MS                :  75
       Bytes Read              :  1138688
       Writes                  :  16156
       Writes MS               :  22728
       Bytes Written           :  573964288
       IOs in progress         :  0
       Bytes used              :  268 MiB
       Replica sets on nodes:
              Set 0
                Node           : 192.168.73.149 (Pool da0757fb-bb24-4db0-95b2-aa92ff5bfe9f )
                Node           : 192.168.34.121 (Pool 2454bc18-57aa-4e58-91b9-67f66db19b64 )
       Replication Status      :  Up
       Volume consumers        :
              - Name           : jenkins-0 (95eba763-59a5-4974-b29e-65276ca37860) (Pod)
                Namespace      : jenkins
                Running on     : ip-192-168-34-121.us-west-2.compute.internal
                Controlled by  : jenkins (StatefulSet)
              - Name           : jenkins-1 (1e478b55-d007-47f3-a71d-938735789720) (Pod)
                Namespace      : jenkins
                Running on     : ip-192-168-73-149.us-west-2.compute.internal
                Controlled by  : jenkins (StatefulSet)
              - Name           : jenkins-2 (fd075b72-260b-4997-9a98-1c3af03b637d) (Pod)
                Namespace      : jenkins
                Running on     : ip-192-168-73-149.us-west-2.compute.internal
                Controlled by  : jenkins (StatefulSet)
```

4. Next, let's describe a couple of jenkins pods and verify that the same pvc is mounted on multiple pods.

```
kubectl describe pods jenkins-0 -n jenkins

kubectl describe pods jenkins-2 -n jenkins
```

```
  jenkins-home:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  jenkins-claim
    ReadOnly:   false
```

As you can see all three Jenkins pods have mounted the jenkins-claim persistent volume to the jenkins-home directory and are simultaneously accessing and writing data to the persistent volume. In this scenario, if you lose any Jenkins pod, the user will still be able to access the application using the surviving pods and access the same build pipelines and plugins.

5. In scenarios where you need to scale up your Jenkins deployment, to ensure that you can keep up with developer demands, you can use the following command to add two more pods to the Jenkins StatefulSet. These new pods will also mount the same persistent volume and will have access to the same application data.

Note: It will take a couple of minutes for the two new pods to be online and running.

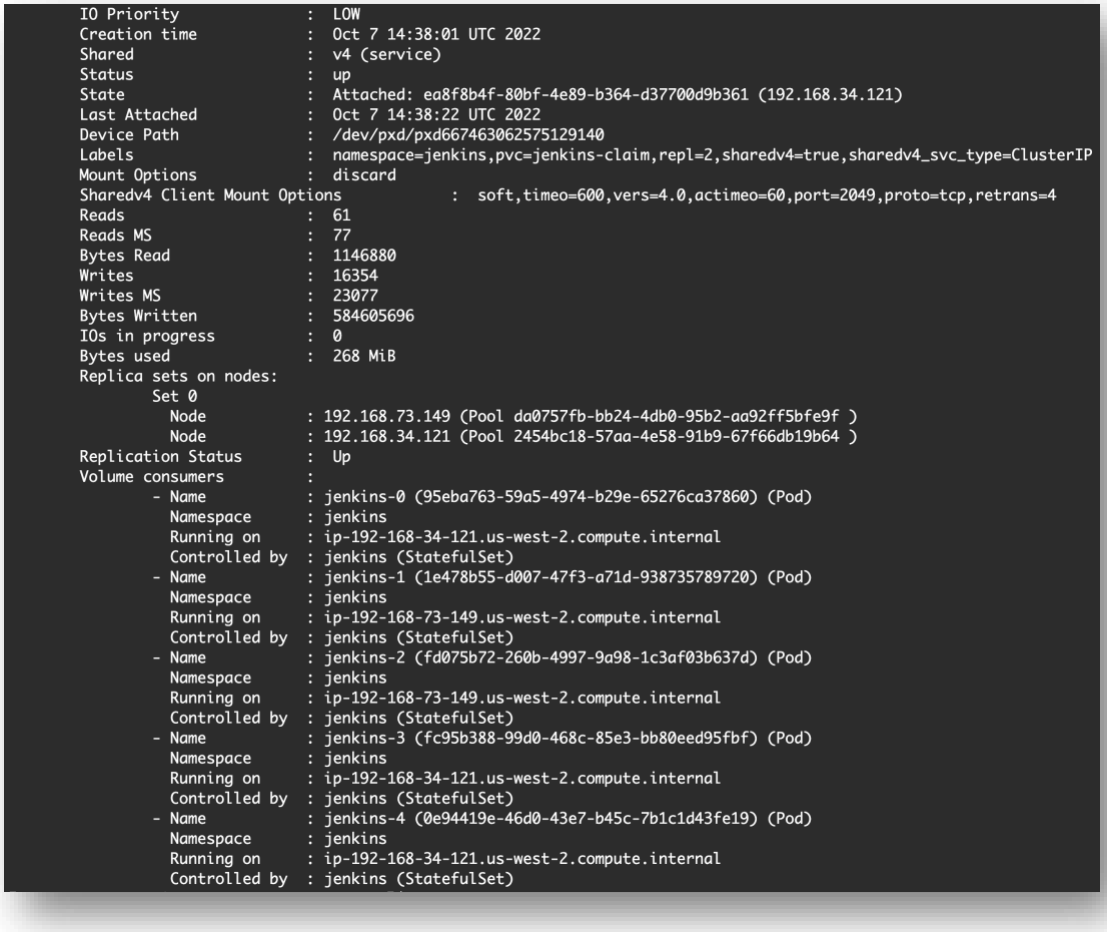
```
kubectl patch statefulsets jenkins -p '{"spec":{"replicas":5}}' -n jenkins

watch kubectl get sts,pods -n jenkins
```

Note: Use Ctrl + C to exit out of the watch command.

```
kubectl get all -n jenkins
```

6. To validate that all five pods have the persistent volume mounted, let's use the inspect-jenkins-vol script again and look at the output.

```
./inspect-jenkins-vol.sh
```




```
kubectl describe pods jenkins-3 -n jenkins

kubectl describe pods jenkins-4 -n jenkins
```

This is how easy it is to leverage Portworx to deploy different types of storage for different application needs on the same Amazon EKS cluster.

Cross Availability Zone (AZ) application availability

Portworx allows you to spread volume replicas across different Amazon Availability Zones (AZs). So, even if you lose a Kubernetes worker node, your pod can be rescheduled to another Amazon EKS worker node and continue using the persistent volume replica on the new node. This process makes the AZ-level fault tolerance faster, as your pods don't have to wait for administrators to manually restore the EBS volume from a snapshot and mount it on the new Amazon EKS worker node.

1. For this exercise, we will use the simple K8s logo application that we deployed in the previous section. To look at the application components, use the following command:

```
kubectl get pods -n demo -o wide -l app=postgres
```

2. We have our postgres pod and it's persistent volume, that is running in one of the AZs in the us-west-2 region. Using the following commands, we will look at where the pod is running currently, simulate an AZ failure, and see how Amazon EKS recovers it instantly to a different AZ in the region. This failover operation is almost instantaneous as Portworx already have a replica of the persistent volume running in the new AZ.

```
cat get-node-az.sh

./get-node-az.sh
```

```
cat simulate-node-failure.sh

./simulate-node-failure.sh
```

```
kubectl get pods -n demo -o wide -l app=postgres
```

```
./get-node-az.sh
```

3. Navigate back to the application endpoint using the following command and refresh the page, to ensure your application is online and you can see all the different logos.

```
kubectl get svc -n demo
```

4. To verify that the Postgres pod got rescheduled to a different AWS AZ based Amazon EKS worker node, with a local persistent volume replica, use the following commands:

```
kubectl get pods -n demo -o wide
```

```
./inspect-postgres-vol.sh
```

5. Next, let's uncordon the Amazon EKS worker node using the following commands

```
kubectl get nodes

kubectl uncordon <<cordoned-node>>
```

Working with Portworx Snapshots

```
Portworx allows you to create snapshots for your persistent volumes, to protect
you from scenarios where you accidently delete a database or drop a table in
the database. Using Portworx snapshots, you can either store those snapshots
locally on the cluster, or you can also offload them to an Amazon S3 bucket.
These snapshots can either be for one persistent volumes, individually, or it
can be for multiple persistent volumes that belong to the same application at
the same time. Portworx also allows you to specify pre- and post-snapshot rules,
so your snapshots are always application consistent.
```

In this scenario, we are going to take a simple snapshot of the persistent volume that is being used by our Postgres database.

1. To take a snapshot, we will create a new VolumeSnapshot object using a yaml file below:

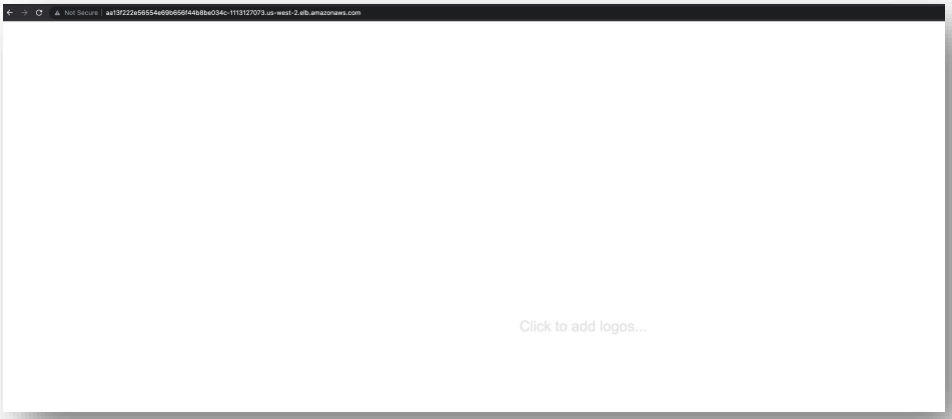
```
cat pg-snapshot.yaml

kubectl apply -f pg-snapshot.yaml

kubectl get stork-volumesnapshots,volumesnapshotdatas -n demo
```

2. Now that we have a snapshot for our persistent volume, let's accidently perform a "Drop table" operation in our Postgres database. This will remove all the location entries for our Kubernetes logos.

```
POD=$(kubectl get pods -l app=postgres -n demo | grep 1/1 | awk '{print $1}')
kubectl exec -it $POD -n demo -- bash
psql -U $POSTGRES_USER
\c postgres
drop table mywhales cascade;


\q
```

```
exit
```

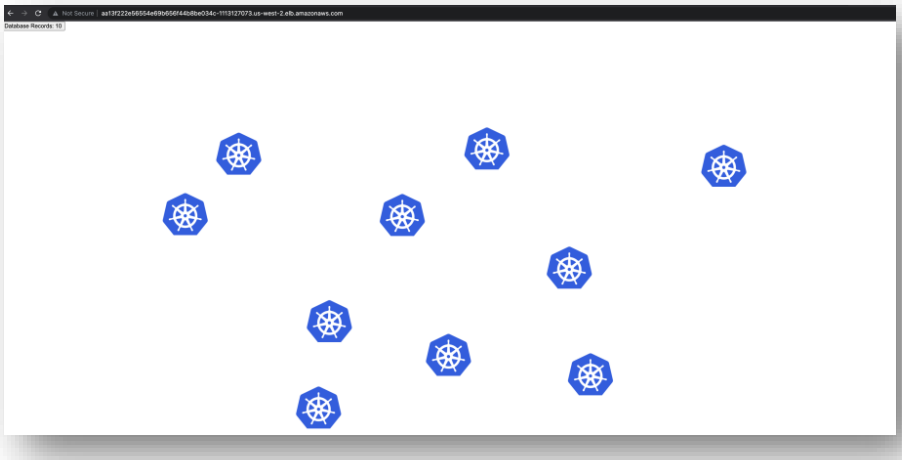
3. Once the entries have been deleted, we can go back to the UI and try to refresh the page. And, as you can see in the screenshot below, the UI is completely empty, as we deleted all the location information for the Kubernetes logos.



4. Next, let's try to restore our Postgres database using the persistent volume snapshot that we took in the first step. Once we create a new persistent volume from the snapshot, we will also scale down and scale up the front end component of our application, so it reconnects to our database.

```
cat restore-from-snap.sh

./restore-from-snap.sh
```

5. Next, navigate back to the UI and refresh the page. You should be able to see all the Kubernetes logos at their original location.



This is how Portworx allows you to protect your applications running on Amazon EKS from accidental deletions or data corruptions scenarios.

Building Multi-tenant Amazon EKS clusters

We see two different types of deployment architectures amongst our customers. The first type runs individual Kubernetes clusters for each application team. These clusters tend to have a smaller footprint, maybe 10 Amazon EKS worker nodes. The second type runs bigger Kubernetes clusters that are multi-tenant and run multiple applications in parallel. Regardless of the type of organization, ensuring that all applications on your Amazon EKS cluster get the required number of resources is important. Kubernetes allows you to set pod-based CPU and memory requests and limits, that help ensure that there isn't a noisy neighbor issue. Portworx allows you to do the same thing for your storage resources. With Application IO control, Portworx allows administrators to set limits to the read and write IOPS or Bandwidth each persistent volume can consume, at Day 0 as part of StorageClass configuration, or on Day 2, using our pxctl cli utility.

In this section, we will run a script that create an FIO job to generate random IO against our Portworx storage cluster, and we will see how we can update the IOPS limit in real time, and see the limit enforced almost instantly.

1. Let's look at the io-gen.sh file which runs that FIO job against the default namespace of our Amazon EKS cluster.

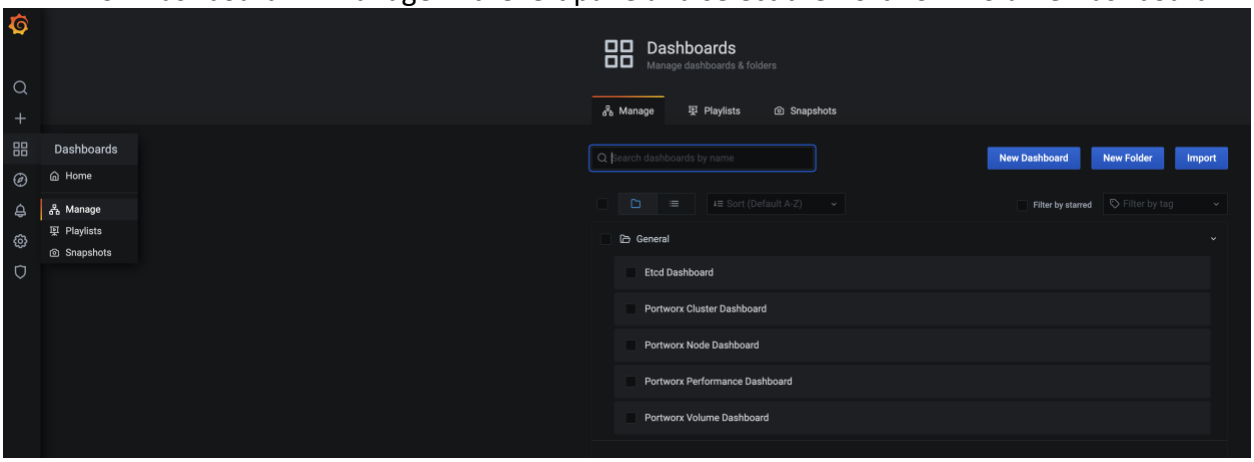
```
cat io-gen.sh

./io-gen.sh
```

2. Look at the PVC and get the volume ID.

Note: Rerun the following command till you see Bound on the persistent volume claim

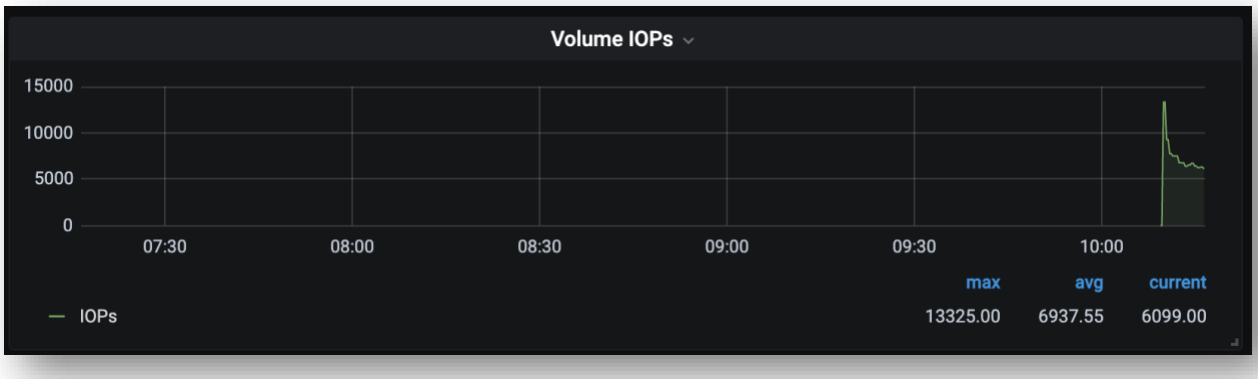
```
kubectl get pvc
```

3. Go back to the Grafana UI and navigate to the Portworx Volume Dashboard by clicking on Dashboard → Manage in the left pane and select the Portworx Volume Dashboard.

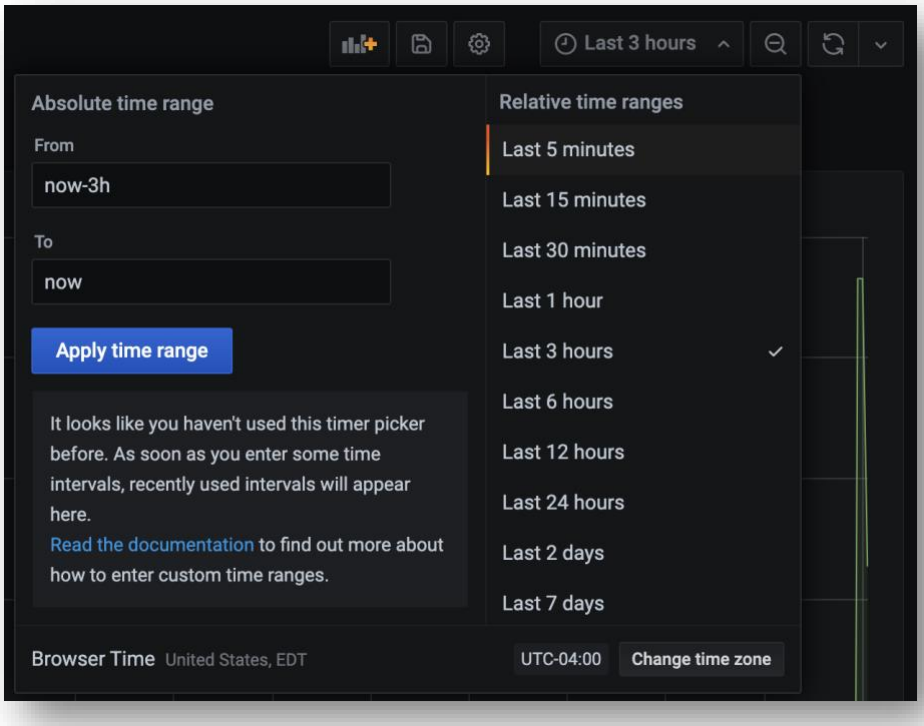


4. On the Portworx Volume Dashboard, select our FIO volume from the Volume Name drop down box.

5. Find the Volume IOPS pane in the dashboard and click on View.



6. Change the timeline view to last 5 mins using the drop-down box on the top right.



7. Look at the IOPS graph, your IOPS should be more 5000.

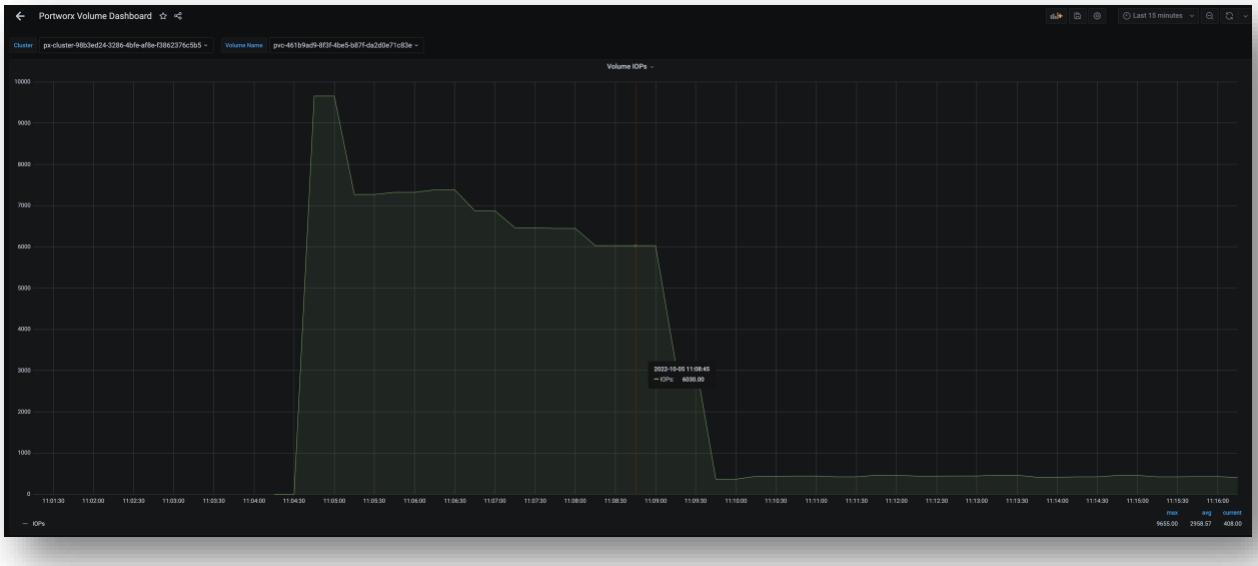
8. Navigate back to AWS CloudShell, and use a simple pxctl command to update the max read and write IOPS to 750 each. We will use a simple script called update-iops.sh to set these maximum limits for our persistent volume.

```
cat update-iops.sh

./update-iops.sh
```

9. Once the parameters are set, we will navigate back to the volume dashboard in Grafana and refresh to look at the new IOPS number.

Note: Grafana graphs update every minute, so we will have to wait for a couple of minutes to see the drop in IOPS.

```
Defaulted container "portworx" out of: portworx, csi-node-driver-registrar
        Volume                  :  1104100817062829620
        Name                    :  pvc-fca3db97-db62-488b-9fb3-9fc6b7c42fdb
        Size                    :  47 GiB
        Format                  :  ext4
        HA                      :  3
        IO Priority             :  LOW
        Creation time           :  Oct 7 14:57:14 UTC 2022
        Shared                  :  no
        Status                  :  up
        State                   :  Attached: e9e2b43d-105e-4feb-a588-c0c90fe1385b (192.168.14.189)
        Last Attached           :  Oct 7 14:57:16 UTC 2022
        Device Path             :  /dev/pxd/pxd1104100817062829620
        Labels                  :  io_profile=auto,namespace=default,pvc=kubestr-fio-pvc-pjq9j,repl=3
        Mount Options           :  discard
        Max Read IOPS           :  750
        Max Write IOPS          :  750
        Reads                   :  66
        Reads MS                :  38
        Bytes Read              :  1167360
        Writes                  :  1332402
        Writes MS               :  16577265
        Bytes Written           :  6983413760
        IOs in progress         :  1
        Bytes used              :  5.4 GiB
        Replica sets on nodes:
                Set 0
                  Node          : 192.168.14.189 (Pool 94b84295-0a83-471e-8fee-872643850f48 )
                  Node          : 192.168.120.99 (Pool e9c1dc43-f9eb-43c6-9307-3a74ad864938 )
                  Node          : 192.168.34.121 (Pool 2454bc18-57aa-4e58-91b9-67f66db19b64 )
        Replication Status      :  Up
        Volume consumers        :
                - Name          : kubestr-fio-pod-82ts9 (3cfa3872-c9f3-4c5c-af5c-eee8406ee626) (Pod)
                  Namespace     : default
                  Running on    : ip-192-168-14-189.us-west-2.compute.internal
```



This is how Portworx allows administrators to set either IOPS or bandwidth limits, so that they don’t have to worry about the noisy neighbor issues, where one application consumes more than its share of the resources and starves the other applications running on the same Amazon EKS cluster.

Automated Storage Capacity Management – Portworx Autopilot

```
Monitoring and managing storage utilization for your applications is a crucial factor to ensure applications uptime. If applications run out of storage, they will go offline, and this is true even for modern applications running on Amazon EKS. Portworx Autopilot allows administrators to set policies in place where Portworx will monitor the utilization of your individual persistent volumes and your storage pool and perform expansion operations on your behalf. This allows administrators to offload the storage management onto Portworx and ensure application uptime.
```

In this scenario, we will go through a simple Postgres and pgbench deployment, where pgbench will generate 70GB work of data against a 10G persistent volume, and we will see how you can use Portworx Autopilot rules, to automate the persistent volume expansion operations for your Postgres PVC.

1. Let's start by looking at all the configuration files for autopilot.

```
cat autopilotrule.yaml

cat autopilot-script.sh

cat autopilot-postgres.yaml

cat autopilot-app.yaml
```

2. Let's use the autopilot-script.sh file to configure Autopilot, and use the autopilot rule to perform volume expansion operation:

```
./autopilot-script.sh
```

Note: Ignore the AlreadyExists error during the script execution.

3. Once the script has been executed, you can monitor the random data generation by looking at the logs of the pgbench container.

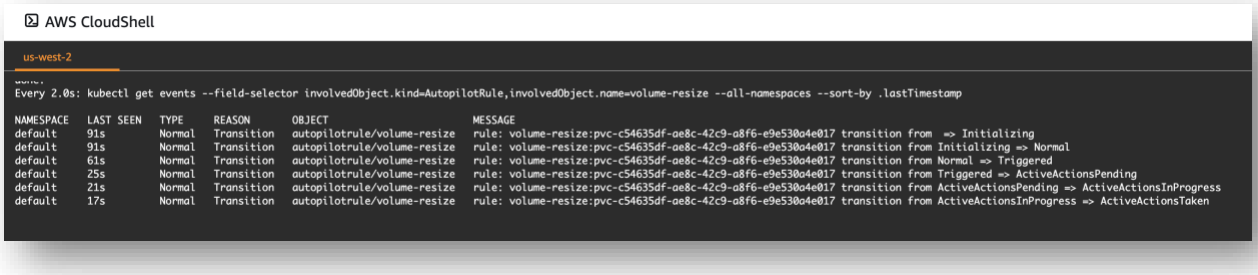
```
kubectl get pvc -n pg1
```

```
POSTGRES_POD=$(kubectl get pods -n pg1 | grep 2/2 | awk '{print $1}')

kubectl logs $POSTGRES_POD -n pg1 pgbench
```

4. Next, let's use the watch command to monitor how Autopilot rule works:

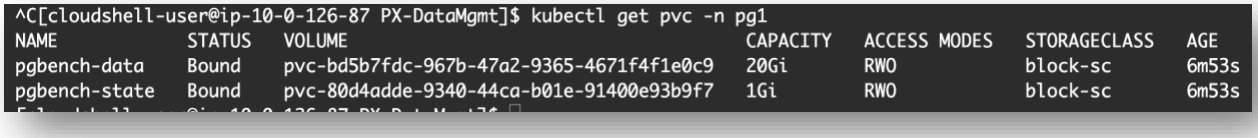
```
watch kubectl get events --field-selector \
involvedObject.kind=AutopilotRule,involvedObject.name=volume-resize \
 --all-namespaces --sort-by .lastTimestamp
```

Note: Once you see ActiveActionsTaken in the event output, click Ctrl + C to exit the watch command.



5. Once you see ActiveActionsTaken, we can check the size of the PVC again.

```
kubectl get pvc -n pg1
```



6. As you can see the size of the volume has now been increased to 20Gi to accommodate for the increased application requirement. Following the rule, these expansion operations will continue till you hit a max size of 100GB.

This is how Portworx allows you to reduce the operational burden when it comes to Day 2 operations for Amazon EKS clusters

End of the Data Management Lab

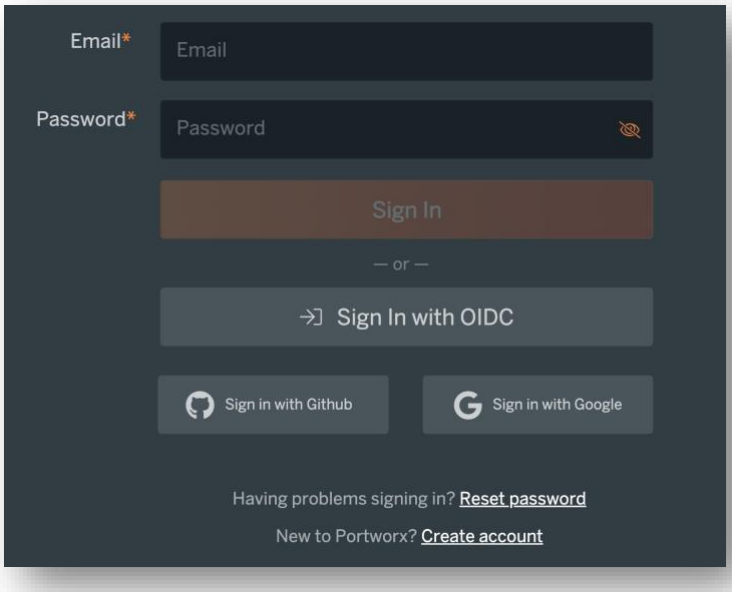
At this point, we are done with the Data Management section of the Data Workshop. Use the following command, to get ready for the next section of the lab focused on Data Services on Kubernetes.

```
./wrapupdatamanagement.sh
```

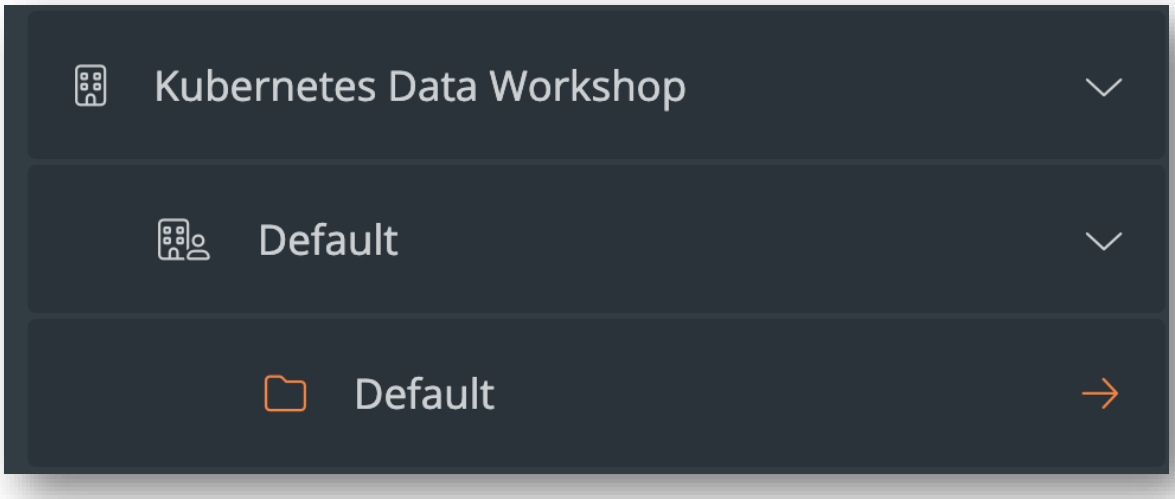
Portworx Data Services

Using the Portworx central credentials, login to the Portworx Data Services (PDS) portal (https://prod.pds.portworx.com/).

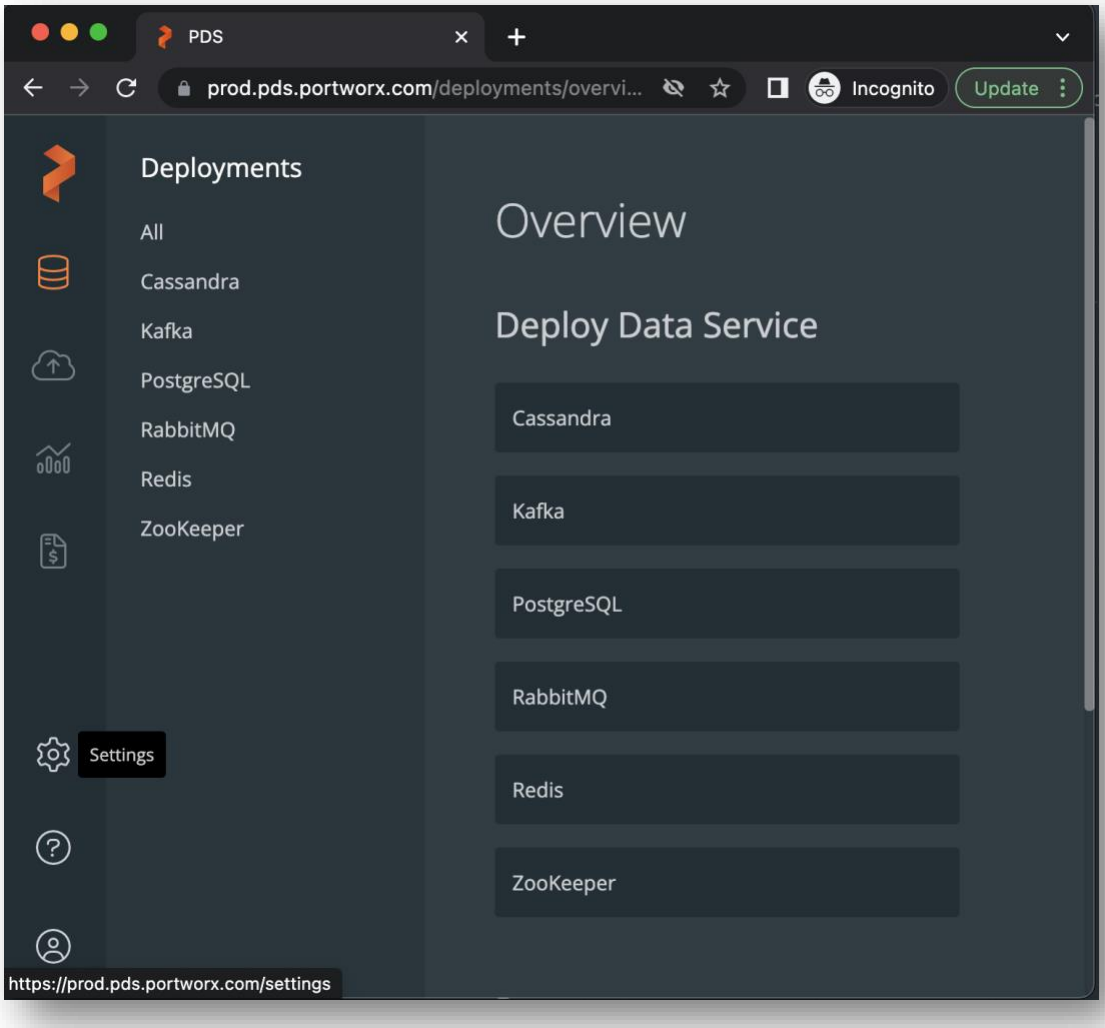
1. Use your Portworx Central credentials to login to the PDS User Interface.



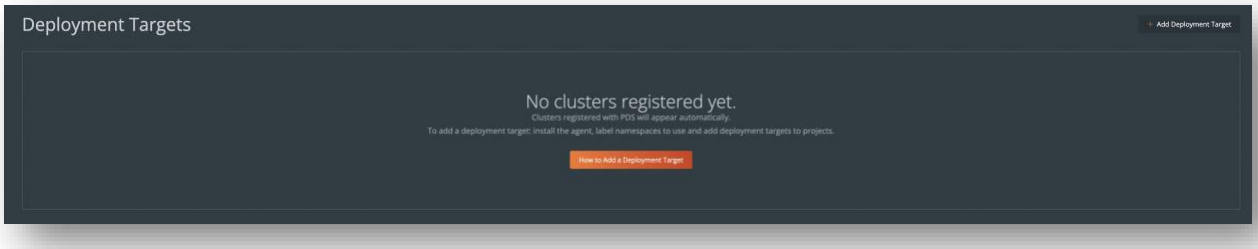
2. We have created a new project for this data workshop "Kubernetes Data Workshop", so find it under Accounts → Tenant → Project.



3. After logging into PDS, the first thing we will do is add our Amazon EKS clusters as a deployment target. From the dashboard, go to settings.



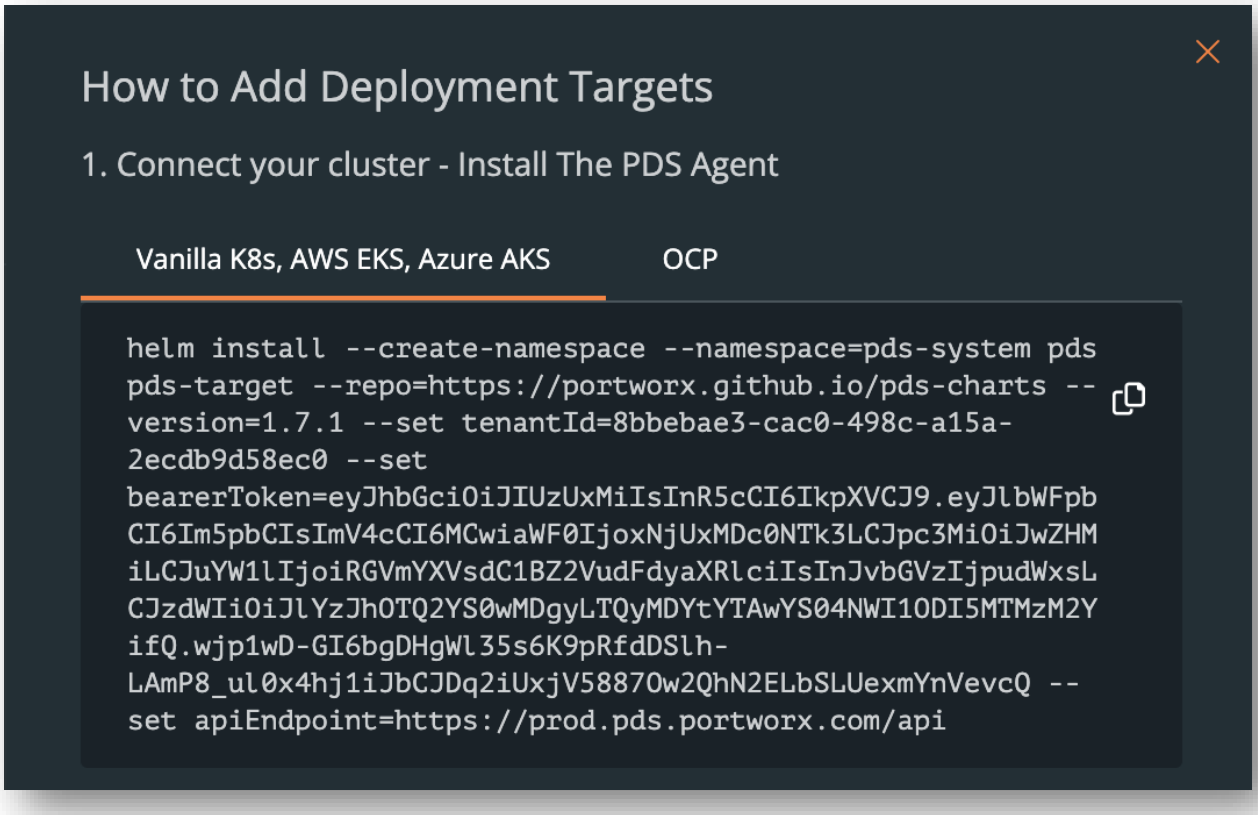
4. Under Settings, click on Deployment Targets and click “Add Deployment Target” on the right.



5. Copy the HELM command under the first tab (Vanilla K8s, AWS EKS, Azure AKS).

Note: Before we paste the HELM command against our Amazon EKS cluster using CloudShell, add a new flag to the HELM command, to reflect your cluster name. (--set clusterName=<<yourname-px-source>>).
Example below:

```
helm install --create-namespace --namespace=pds-system pds pds-target --repo=https://portworx.github.io/pds-charts --version=1.8.1 --set tenantId=b4f008dd-d663-4742-b6fc-1efc6e3d08f0 --set bearerToken=token --set apiEndpoint=https://staging.pds.portworx.com/api --set clusterName=<<yourname-px-source>>
```



6. Navigate to AWS CloudShell and run the following two commands to connect back to your EKS cluster and ensure that HELM is installed.

```
./connect-px-source.sh
./install-helm.sh
```

7. To verify installation

```
kubectl get pods -n pds-system -w
```

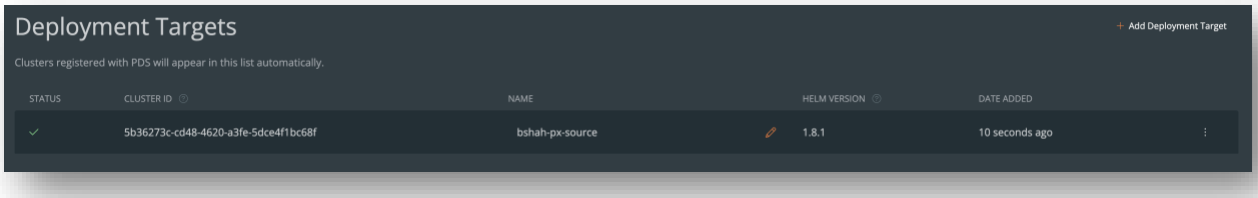
Note: Use Ctrl + C to exit out of the watch command.

```
NAME: pds
LAST DEPLOYED: Fri Oct 14 18:24:13 2022
NAMESPACE: pds-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
[cloudshell-user@ip-10-1-13-199 PX-DataWorkshop]$
[cloudshell-user@ip-10-1-13-199 PX-DataWorkshop]$
```

8. Use the following command to command to create namespaces that we will use to deploy data services using PDS

```
kubectl apply -f pds-namespaces.yaml
```

9. Verify that your cluster has been successfully added to the PDS UI.



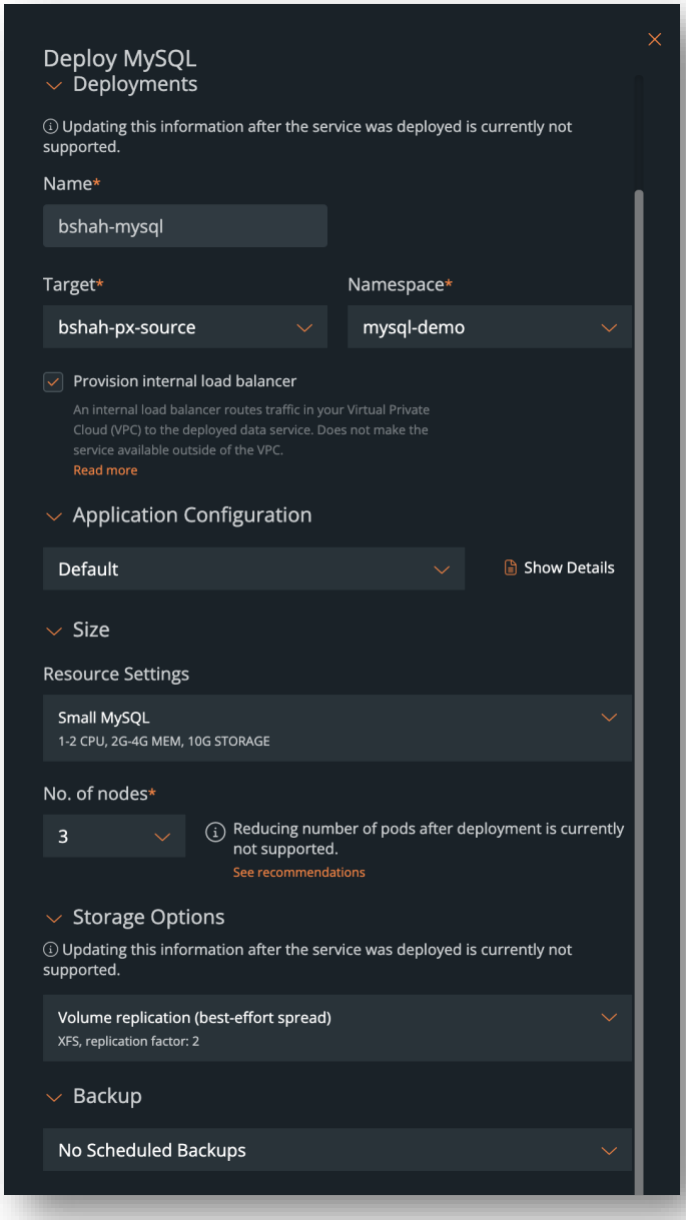
This is how easy it is to add Deployment targets to Portworx Data Services. You don't need to copy Kubeconfig files, or open Firewall ports. We will deploy all the agents on your Target cluster using the HELM command than then establish a secure reverse tunnel from your target cluster back to the PDS Control Plane.

Deploy MySQL cluster

In this section, we will look at how you can easily deploy a highly available MySQL cluster on Amazon EKS without knowing anything about Kubernetes Operators or Custom Resources.

1. From the PDS Dashboard, select MySQL as the data service that you want to deploy
2. In the MySQL deployment wizard, enter the following values and hit Deploy.
    a. Version: 8.0.30
    b. Name: <<your-name-mysql>>
    c. Target: <<your-name-px-source>>
    d. Namespace: mysql-demo
    e. Keep the box for Provision Internal Load Balancers selected

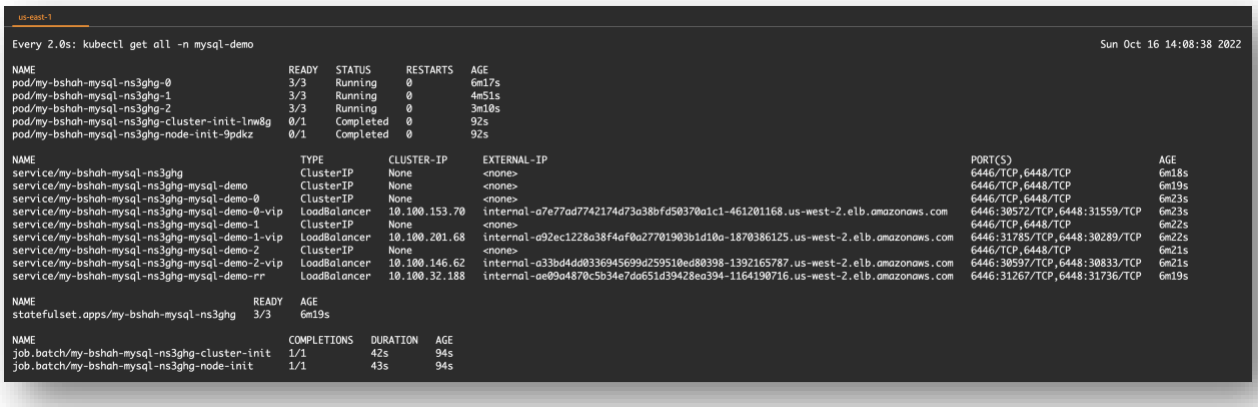
f. Application Configuration: Default
g. Size:
    i. Resource Settings: Small
    ii. No. of Nodes: 3
h. Storage Options: Volume replication (best-effort spread)
i. Backups: No Scheduled Backups



3. This initiates a MySQL deployment. To monitor the deployment, navigate back to the Amazon EKS cluster using AWS CloudShell and check the resources in the mysql-demo namespace

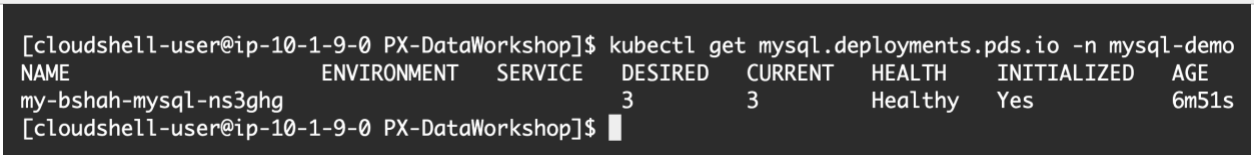
```
watch kubectl get all -n mysql-demo
```

Note: Ctrl + C to exit the Watch command



10. PDS deploys a custom resource using the Operator deployed during the cluster addition. Check the custom mysql resource using the following command:

```
kubectl get mysql.deployments.pds.io -n mysql-demo
```

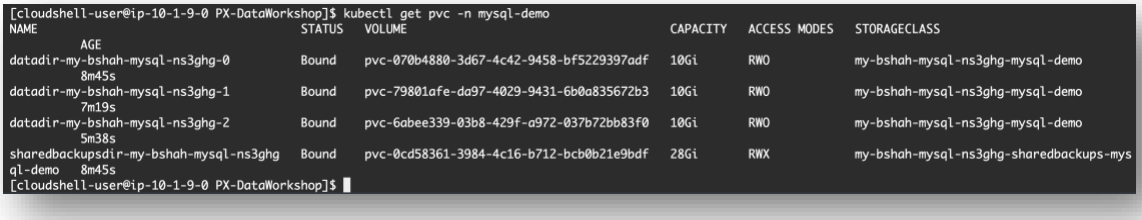


11. For each data service deployment, PDS will deploy a custom storage class. Check the Storage Class using the following command:

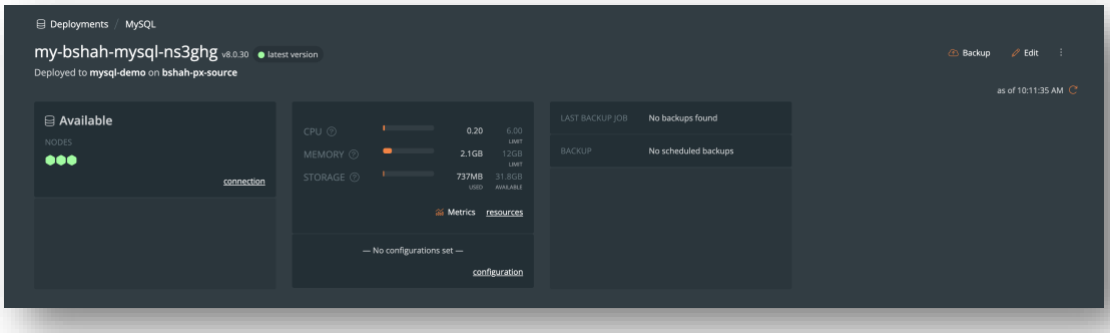
```
kubectl get sc
```

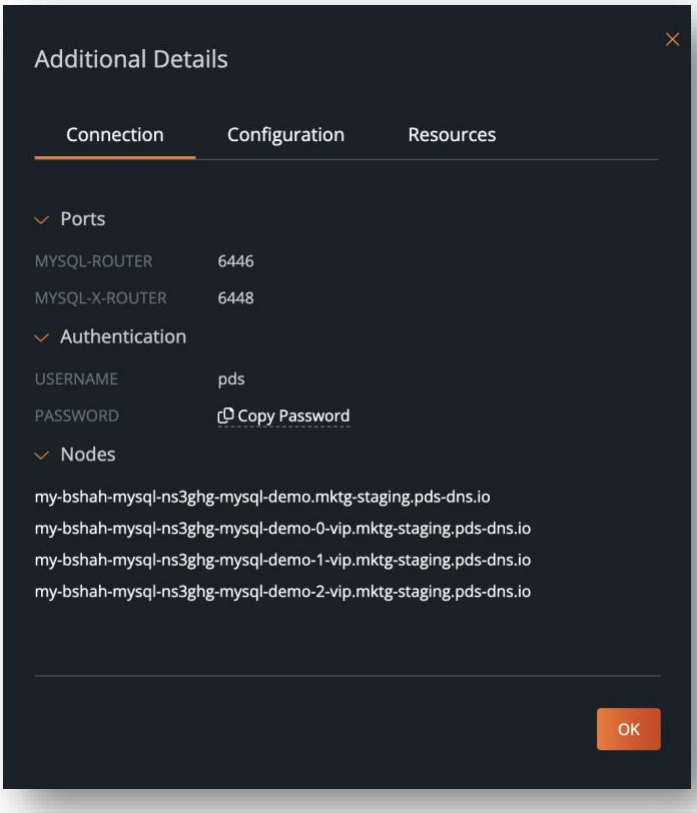
12. Using the above Storage Class, PDS dynamically deploys persistent volumes in the mysql-demo namespace. Check the PVCs deployed using the following command:

```
kubectl get pvc -n mysql-demo
```



13. You can also monitor your data service deployment using the PDS UI. Let's do that and copy the connection string, so we can start using this MySQL cluster.





14. To deploy an app using the mysql endpoint, go back to AWS CloudShell and edit the mysql-app.yaml file and update the PMA_HOST and MYSQL_PASSWORD variables to match the connection details from the PDS UI.

```
vi mysql-app.yaml
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phpmyadmin-deployment
  labels:
    app: phpmyadmin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: phpmyadmin
  template:
    metadata:
      labels:
        app: phpmyadmin
    spec:
      containers:
        - name: phpmyadmin
          image: phpmyadmin/phpmyadmin
          ports:
            - containerPort: 80
          env:
            - name: PMA_HOST
              value: my-bshah-mysql-ns3ghg-mysql-demo.mktg-staging.pds-dns.io
            - name: PMA_PORT
              value: "6446"
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secrets
                  key: ROOT_PASSWORD
            - name: MYSQL_USER
              value: pds
            - name: MYSQL_PASSWORD
              value: "PETiE83OwP2Ydj4oepgJ6XzAYKqLU1V1hr83m5eG"
---
```

15. Next, apply this mysql-app.yaml file to the mysql-demo namespace

```
kubectl apply -f mysql-app.yaml -n mysql-demo
```

This will deploy a simple phpMyAdmin deployment that you can use to interact with your MySQL instance.

```
[cloudshell-user@ip-10-1-9-0 PX-DataWorkshop]$ kubectl apply -f mysql-app.yaml -n mysql-demo
secret/mysql-secrets created
deployment.apps/phpmyadmin-deployment created
service/phpmyadmin-service created
```

16. Use the following commands to look at the deployed resources and fetch the application endpoint and navigate to it using the your browser session.

```
kubectl get all -n mysql-demo
```

```
kubectl get svc -n mysql-demo phpmyadmin-service
```

```
[cloudshell-user@ip-10-1-9-0 PX-DataWorkshop]$ kubectl get svc -n mysql-demo phpmyadmin-service
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)        AGE
phpmyadmin-service  LoadBalancer   10.100.236.199   a6f9371d746ee4a0fb24985d106525a8-307973487.us-west-2.elb.amazonaws.com   80:31197/TCP   3m6s
```

17. You can login to phpMyAdmin using the credentials for MySQL from the PDS UI.

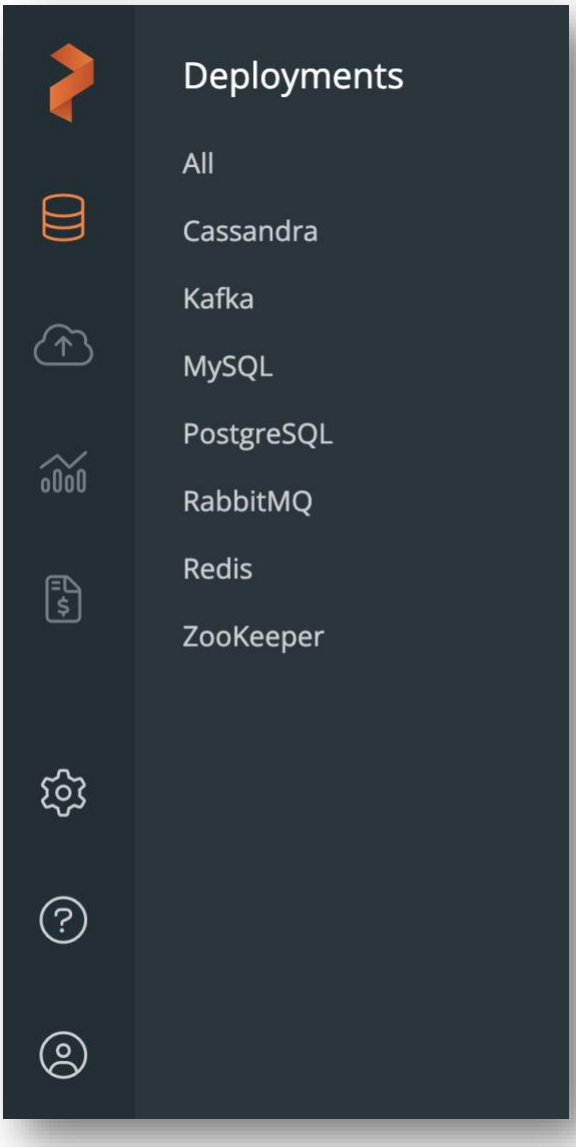
This is how easily you can deploy a highly available MySQL instance using Portworx Data Services.

Administrator workflow

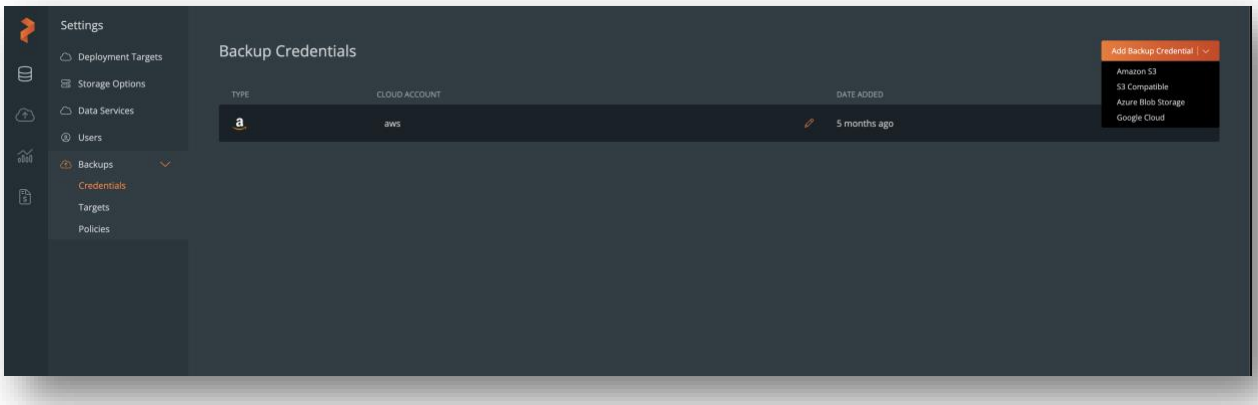
Configuring Data Protection for PDS

Portworx Data Services allows administrators to configure backup targets, policies, and credentials. These backup settings can be used by developers to enable Backups during Day 0 deployment of the data services or take manual ad-hoc backups on Day 2 for already deployed data services.

1. Navigate to the PDS dashboard and go to the Settings section from the left pane.



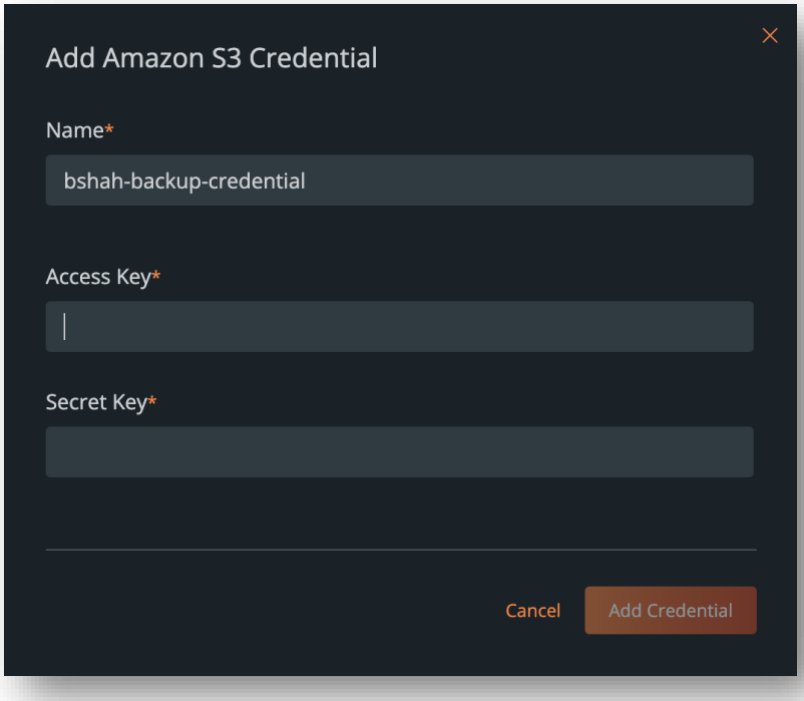
2. Select the Backups → Credentials tab on the left and click “Add Backup Credential” on top right and select “Amazon S3”.



- Name: <<yourname-backup-credential>>
- Access Key: workshop user access key
- Secret Key: workshop user secret key

Note: If you don't remember your AWS credentials, you can fetch IAM creds using the following command using AWS CloudShell session.

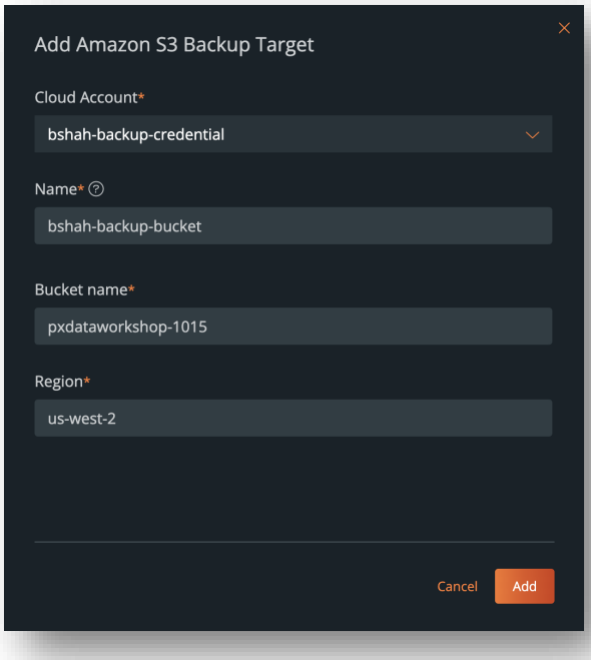
```
cat workshop-creds.txt
```



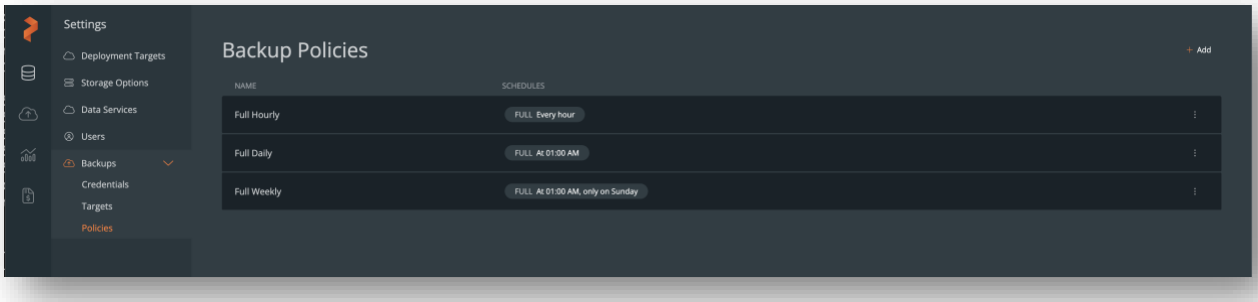
3. Next, to add a Backup Target, select Backups → Target → Add Backup Target.
    - Select the backup-credential you created in the previous step.
    - Gave the backup location a name: <<yourname-backup-bucket>>
    - Bucket name: <<name of the s3 bucket you created>>

- Region: us-west-2
- Click Add

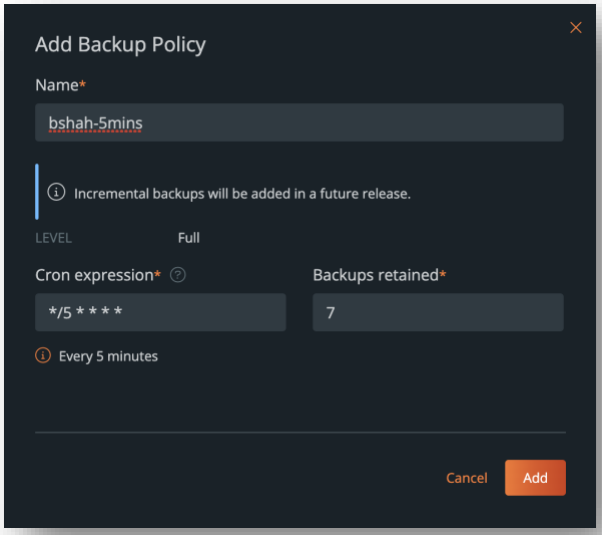
Note: To find the name of your S3 bucket, you can navigate back to the AWS CloudShell and use the command – “aws s3 ls”.



4. Next, add Backup Policy for every 5 mins: using Backups → Policies → Add.



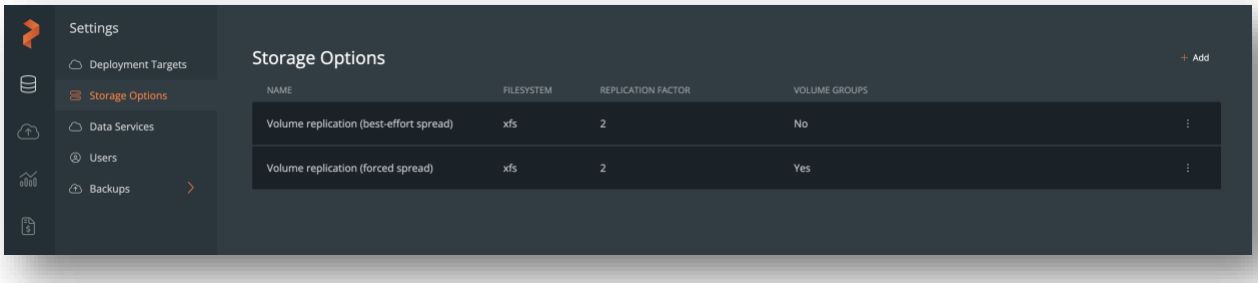
- Name: <<yourname-5mins>>
- Cron expression: */5 * * * *
- Backups retained: 7



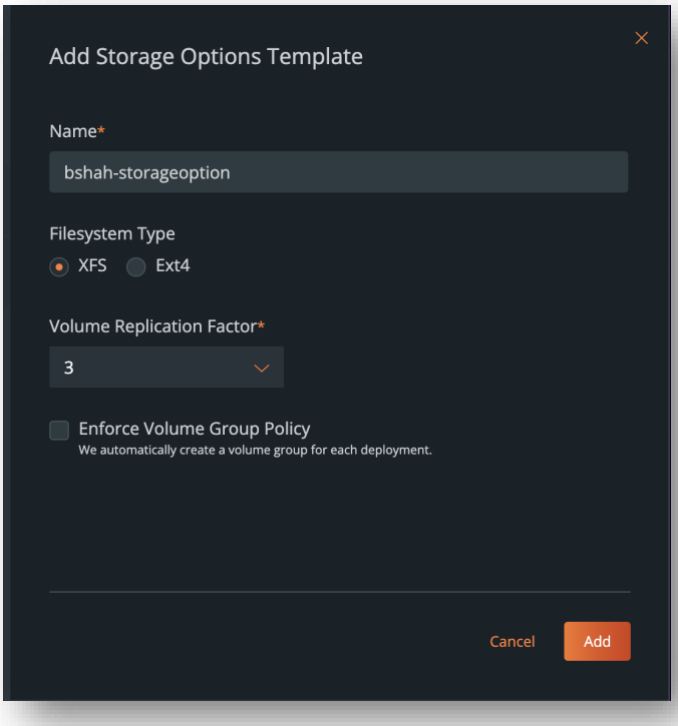
Configuring Storage Options for PDS

Portworx Data Services delivers a couple of default storage options that developers can use to deploy their data services. But, as administrators, you can still customize these storage options and select different parameters that get translated into Kubernetes Storage Class parameters on your Amazon EKS cluster.

1. Navigate to PDS Dashboard → Settings → Storage Options → Add



- Name: <<yourname-storageoption>>
- Filesystem Type: XFS
- Volume Replication Factor: 3
- Keep the box for “Enforce Volume Group Policy” unchecked.

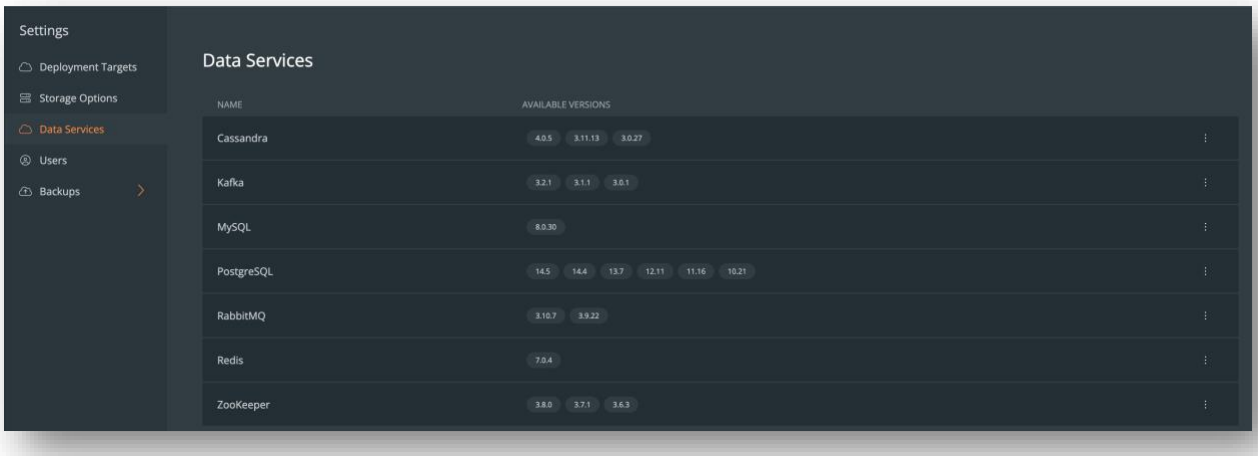


Configuring Data services templates for PDS

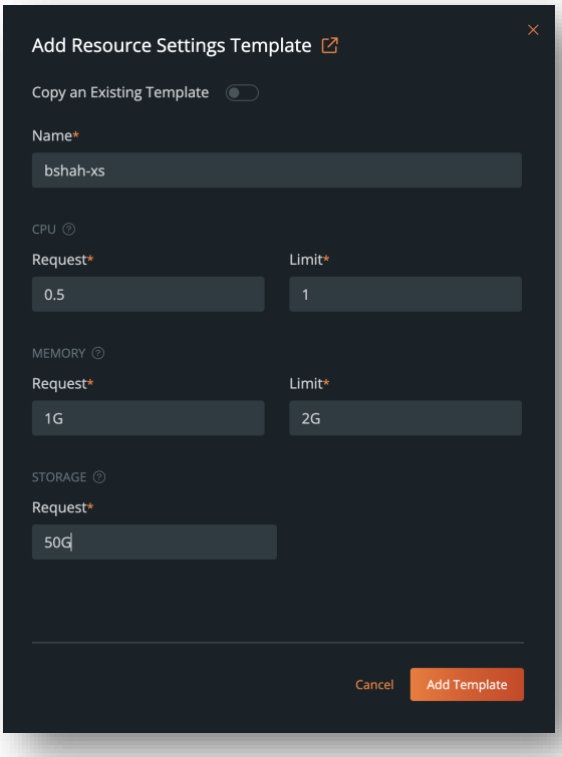
PDS allows administrators to create custom Application Configuration templates and Resource Setting templates for each data service.

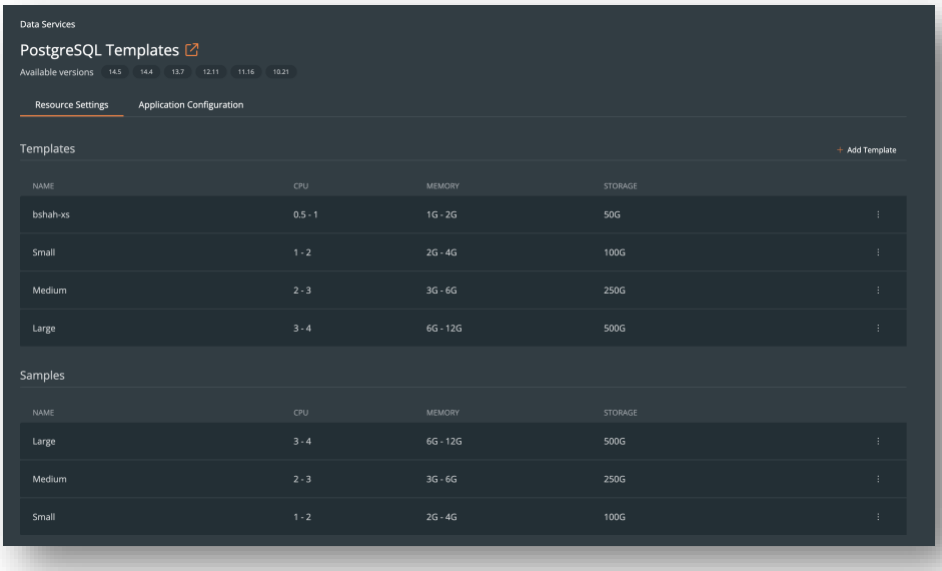
- Application Configuration Templates allows administrators to pass deploy-time key-value pairs to customize your database deployment.
- Resource Setting Templates allows administrators to create new sizing quotas and limits for CPU, Memory, and Storage for your database nodes.

1. To create a new Resource Setting template for PostgreSQL, navigate to PDS Dashboard → Settings → Data Services → PostgreSQL.



2. In the Resource Settings tab, click on “+Add Template”.
    a. Name: <<yourname-XS>>
    b. CPU Request – 0.5 | Limit - 1
    c. Memory Request – 1G | Limit – 2G
    d. Request – 50G
    e. Click Add

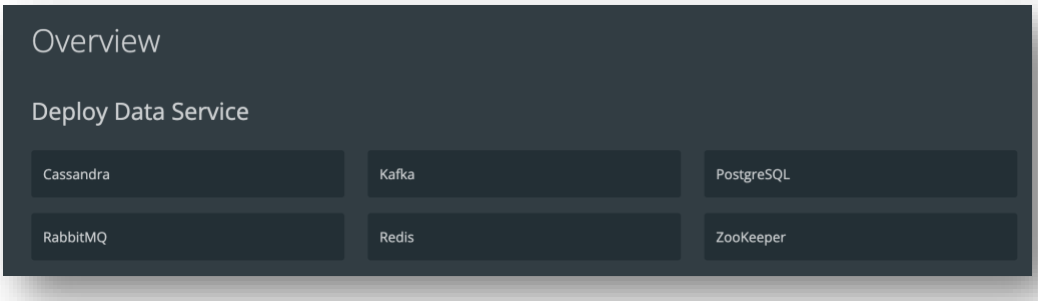




Now that we have created a new resource settings template, storage options and backup policies, targets, and credentials, let's deploy PostgreSQL in the next section using these new templates.

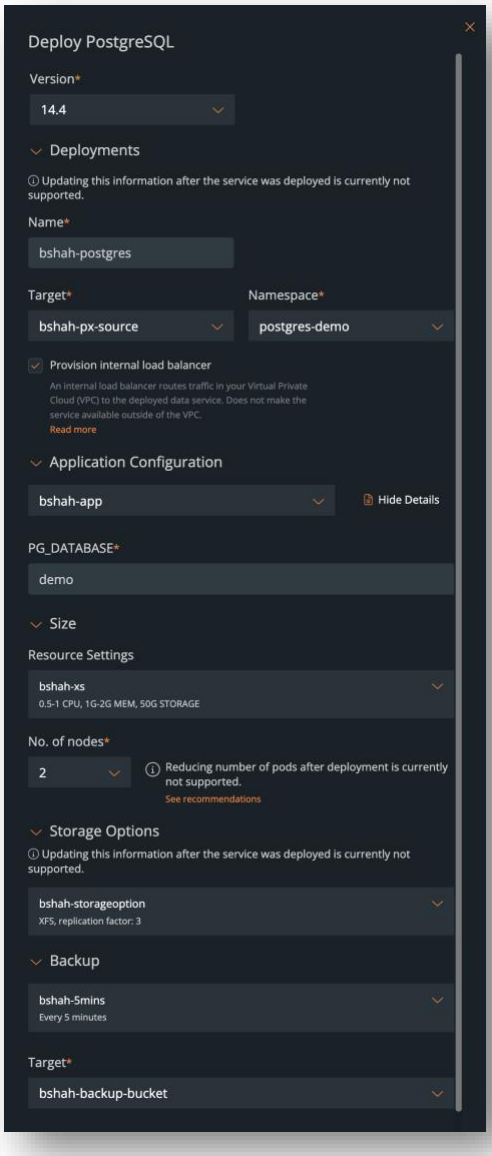
Deploying PostgreSQL using Portworx Data Services

1. Navigate to the PDS dashboard and select PostgreSQL. We will use this section to deploy PostgreSQL 14.4.



2. In the PostgreSQL deployment wizard, enter the following values:
    a. **Version: 14.4**
    b. Name: <<your-name-postgres>>
    c. Target: <<your-name-px-source>>
    d. Namespace: postgres-demo
    e. Keep the box for Provision Internal Load Balancers selected
    f. Application Configuration: Default
    g. Size:
        i. Resource Settings: <<yourname-xs>>
        ii. No. of Nodes: 2

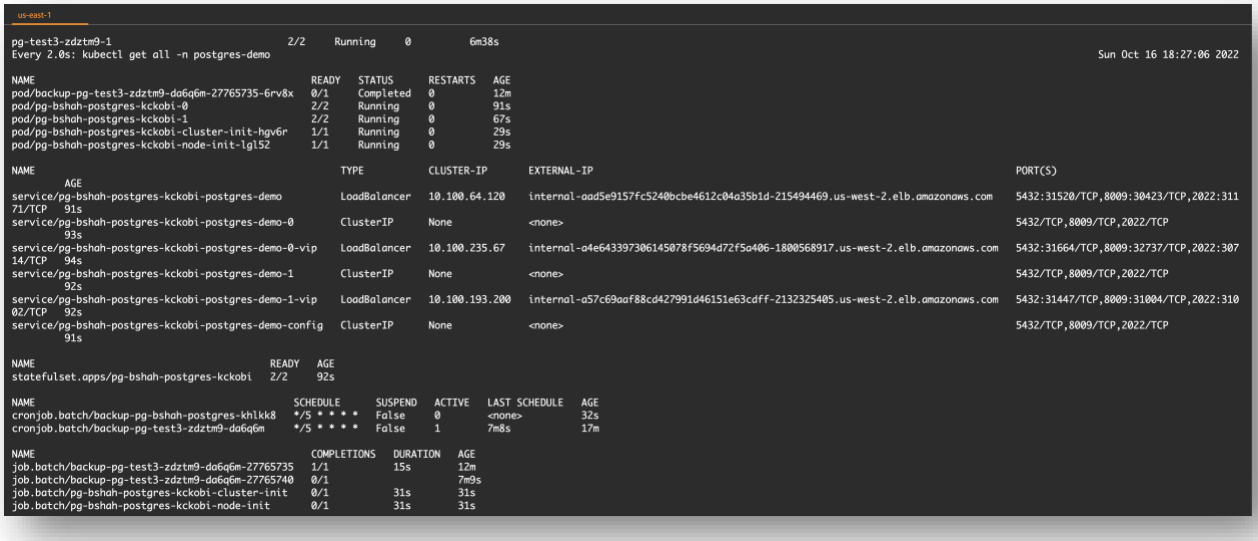
h. Storage Options: <<yourname-storageoption>>
i. Backups: <<yourname-5mins>>
j. Target: <<yourname-backup-bucket>>



7. Hit Deploy to create the PostgreSQL database.
8. This initiates a PostgreSQL deployment. To monitor the deployment, navigate back to the Amazon EKS cluster using AWS CloudShell and check the resources in the postgres-demo namespace.

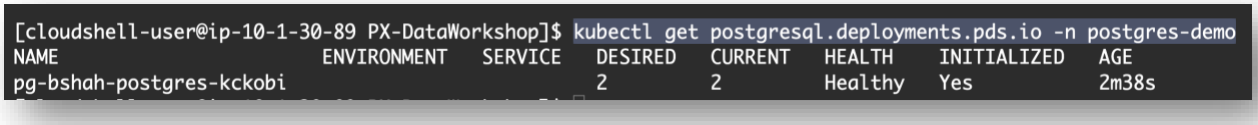
```
watch kubectl get all -n postgres-demo
```

Note: Ctrl + C to exit the Watch command



9. PDS deploys a custom resource using the Operator deployed during the cluster addition. Check the custom postgres resource using the following command:

```
kubectl get postgresql.deployments.pds.io -n postgres-demo
```

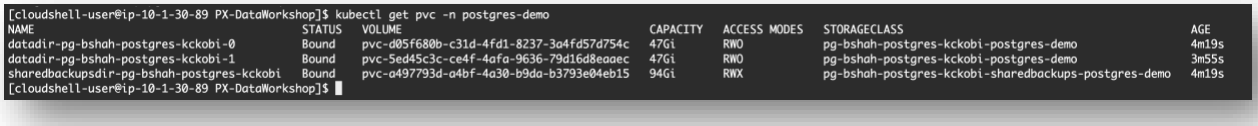


10. For each data service deployment, PDS will deploy a custom storage class. Check the Storage Class using the following command:

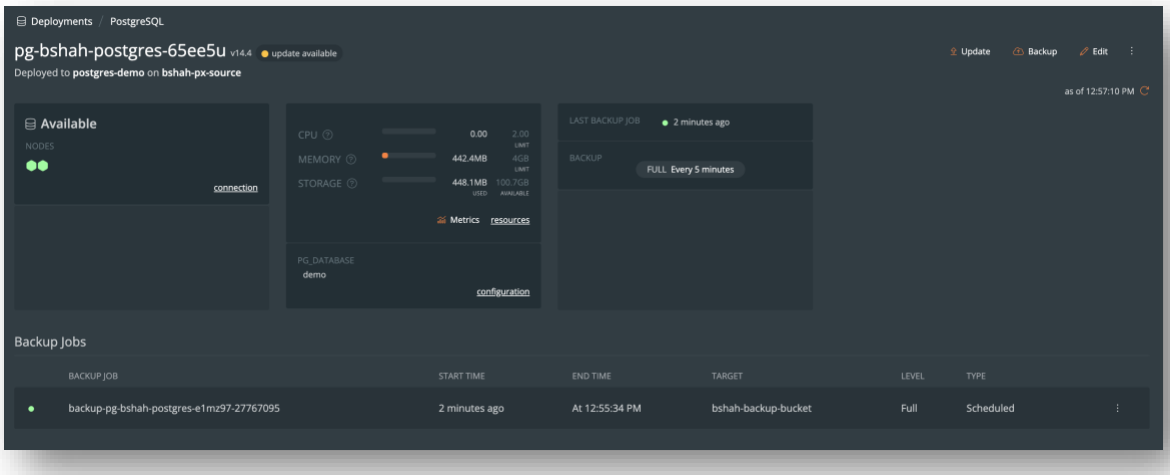
```
kubectl get sc
```

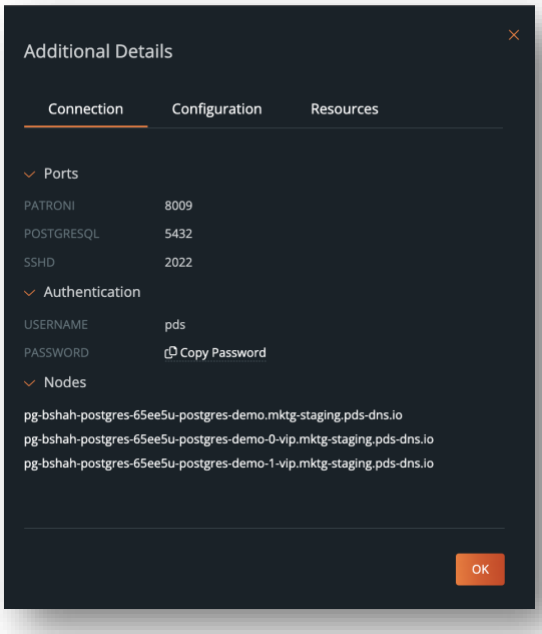
11. Using the above Storage Class, PDS dynamically deploys persistent volumes in the postgres-demo namespace. Check the PVCs deployed using the following command:

```
kubectl get pvc -n postgres-demo
```



12. You can also monitor your data service deployment using the PDS UI. Let's do that and copy the connection string, so we can start using this PostgreSQL cluster.





13. Deploy an app using the postgresql endpoint, go back to CloudShell and edit the pds-postgres-app.yaml file and update the USE_POSTGRES_HOST and POSTGRES_PASSWORD variables to match the connection details from the PDS UI.

```
vi pds-postgres-app.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-counter-deployment
  labels:
    app: k8s-counter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: k8s-counter
  template:
    metadata:
      labels:
        app: k8s-counter
    spec:
      containers:
      - name: k8s-counter
        image: wallnerryan/moby-counter:k8s-record-count
        imagePullPolicy: Always
        ports:
        - containerPort: 80
        env:
        - name: USE_POSTGRES_HOST
          value: "pg-bshah-postgres-8as41o-postgres-demo.mktg-prod.pds-dns.io"
        - name: USE_POSTGRES_PORT
          value: "5432"
        - name: POSTGRES_USER
          value: "pds"
        - name: POSTGRES_PASSWORD
          value: "0ZDhqp34iqfkUG5E4dA6DJ21MkyXGwnZnWXboa89"
        - name: POSTGRES_DB
          value: "pds"
```

14. Next, apply this pds-postgres-app.yaml file to the mysql-demo namespace

```
kubectl apply -f pds-postgres-app.yaml -n postgres-demo
```

This will deploy a simple application that you can use to interact with your PostgreSQL instance.

```
[cloudshell-user@ip-10-1-30-89 PX-DataWorkshop]$ kubectl apply -f pds-postgres-app.yaml -n postgres-demo
deployment.apps/k8s-counter-deployment created
service/k8s-counter-service created
```

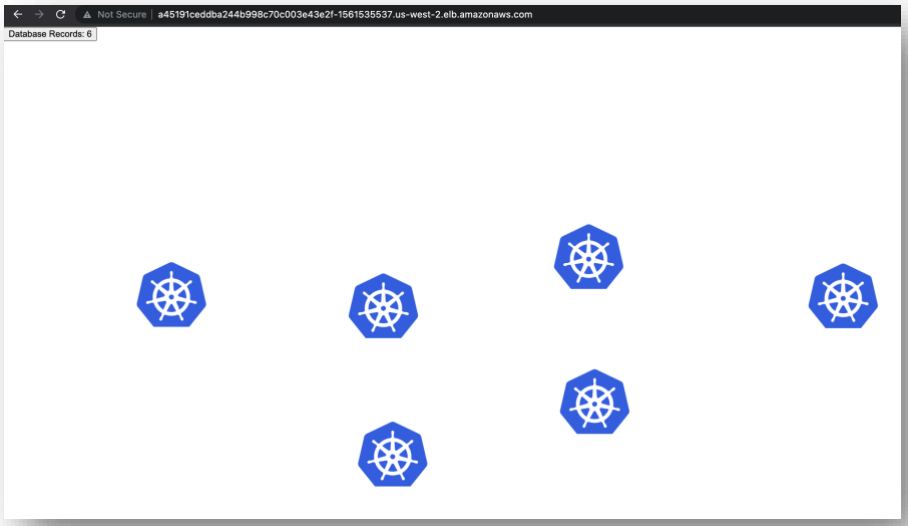
15. Use the following commands to look at the deployed resources and fetch the application endpoint and navigate to it using your browser session.

```
kubectl get all -n postgres-demo
```

```
kubectl get svc -n postgres-demo k8s-counter-service
```

```
[cloudshell-user@ip-10-1-30-89 PX-DataWorkshop]$ kubectl get svc -n postgres-demo k8s-counter-service
NAME                TYPE           CLUSTER-IP     EXTERNAL-IP                                                               PORT(S)        AGE
k8s-counter-service   LoadBalancer   10.100.165.9   a45191ceddba244b998c70c003e43e2f-1561535537.us-west-2.elb.amazonaws.com   80:30123/TCP   62s
```

16. Navigate to the application endpoint and generate some logos by clicking randomly on the screen.



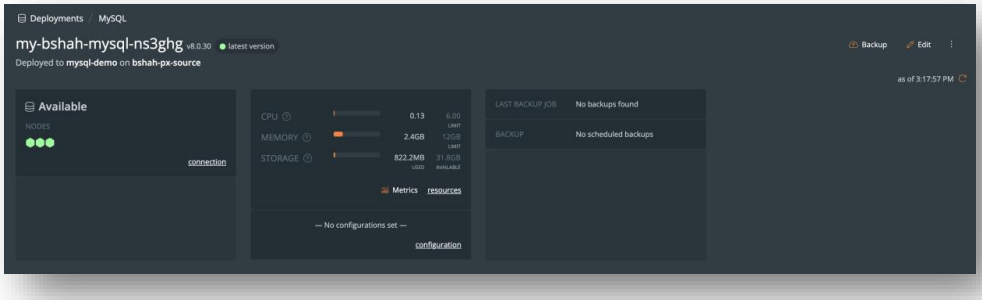
This is how easily you can deploy a highly available PostgreSQL instance using Portworx Data Services.

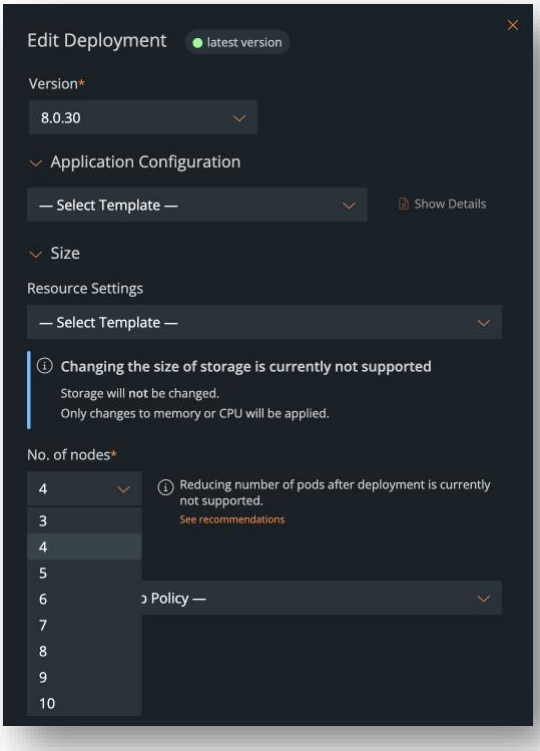
Simplifying Day 2 operations using Portworx Data Services

PDS allows you to scale up, scale out, create ad hoc backups, monitor, update versions, etc for your data services. In this section, we will cover four scenarios using our already deployed MySQL and PostgreSQL instances.

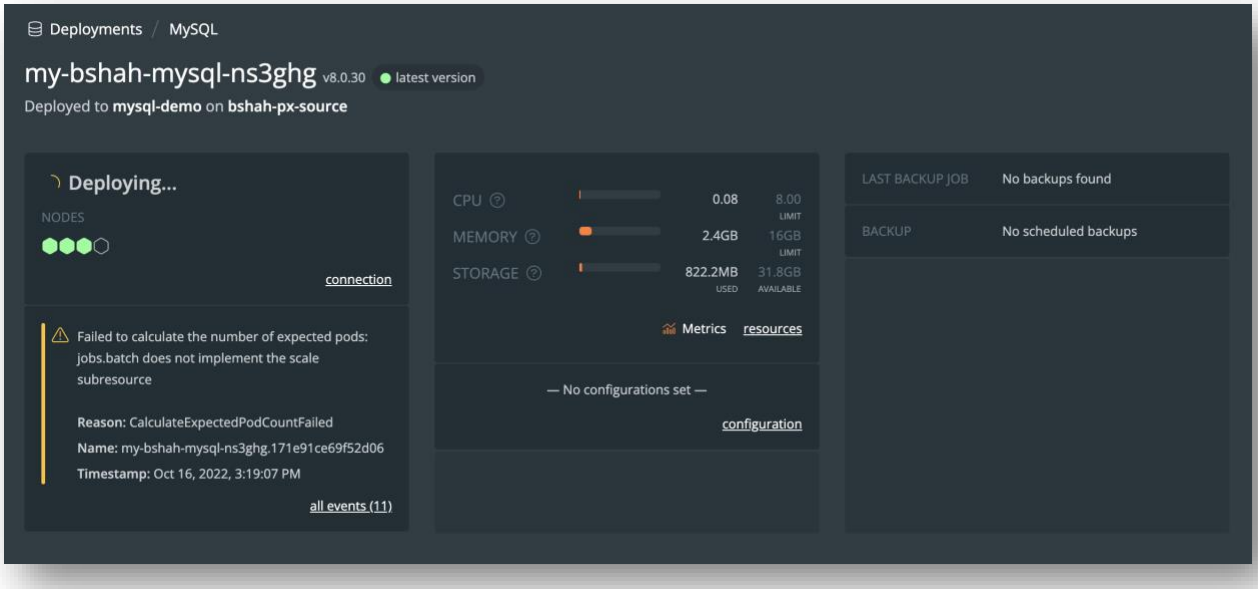
Scale Out Data Services

1. To perform a Scale Out operation, let's find the MySQL cluster you deployed earlier in the workshop.
2. Find the 'Edit' button on the top right and increase the number of nodes in the MySQL cluster from 3 to 4 and click Apply.

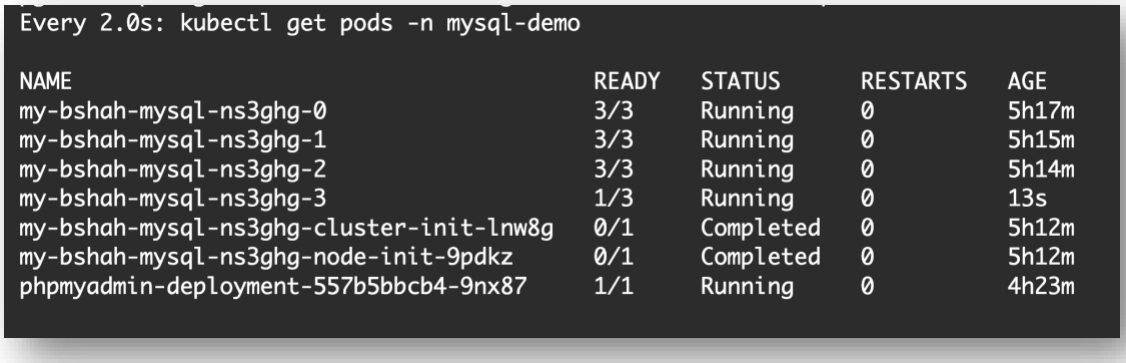




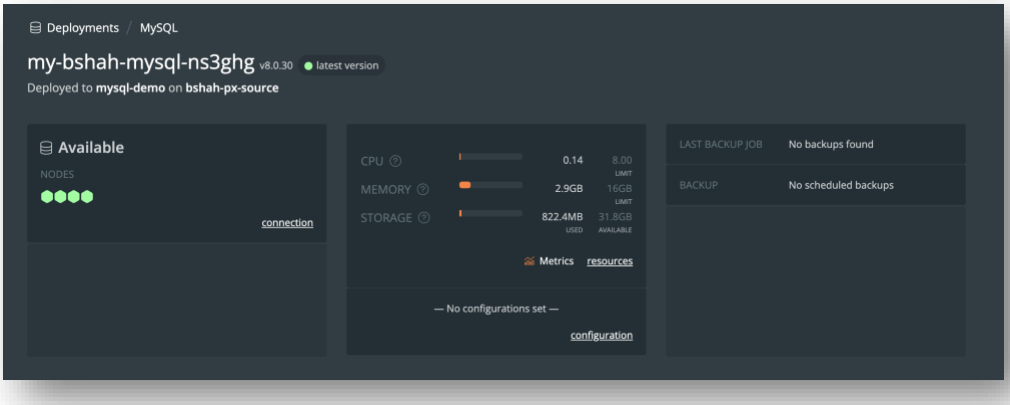
3. Monitor the scale out operation using the PDS UI or navigate to the AWS CloudShell and use the following command to watch the pods terminate and respawn.




```
watch kubectl get pods -n mysql-demo
```

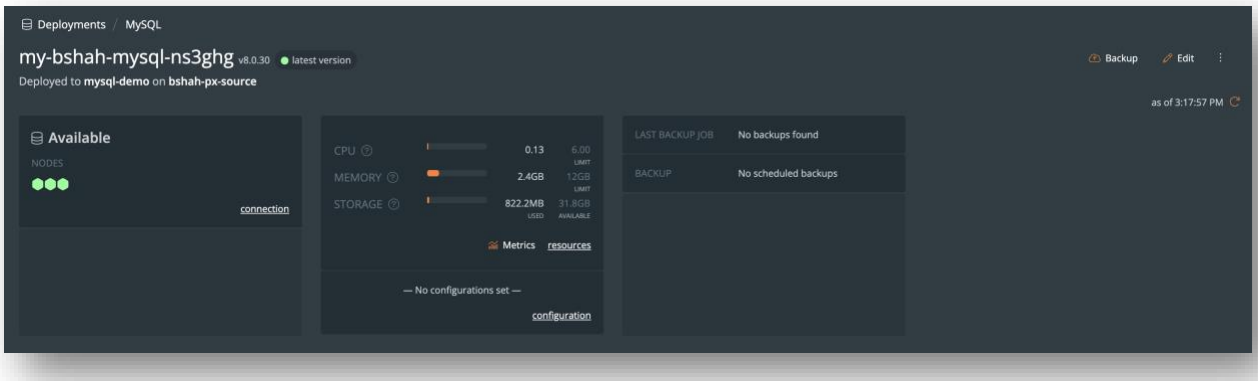


4. Once completed, you will also see the new nodes in the PDS UI.

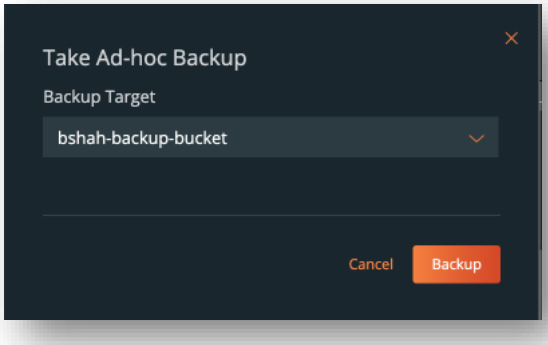


Ad-hoc backup

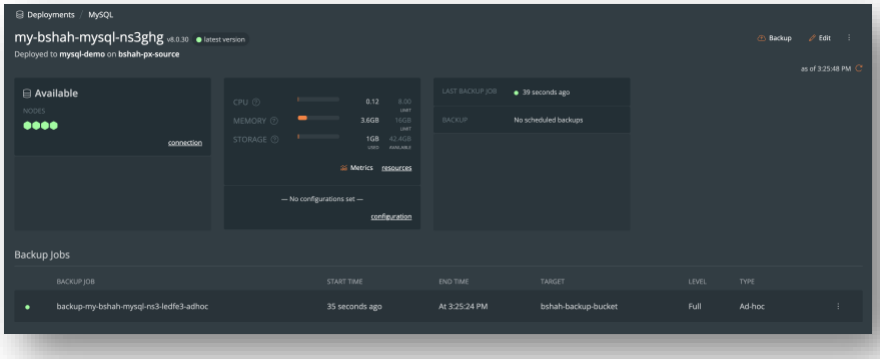
1. Once the scale out operation is completed successfully and all four nodes are available, we will perform an ad-hoc backup operation, let's find the MySQL cluster again and find the 'Backup' button on the top right.



2. Select the backup target (<<yourname-backup-bucket>>) that we configured in the customized section and click Backup.

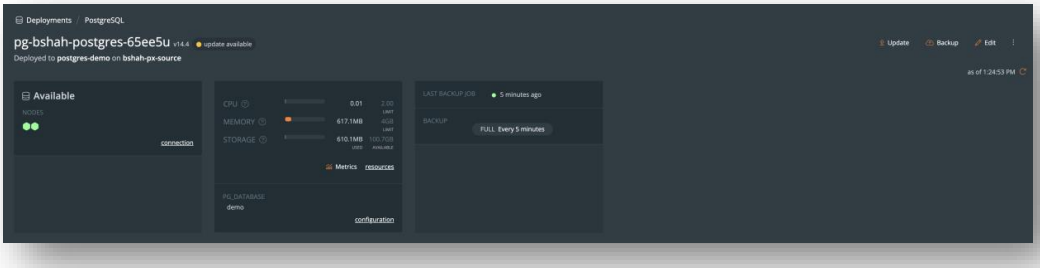


3. At this point, PDS will initiate a manual full backup for your MySQL cluster and store the application snapshot in your Amazon S3 bucket.

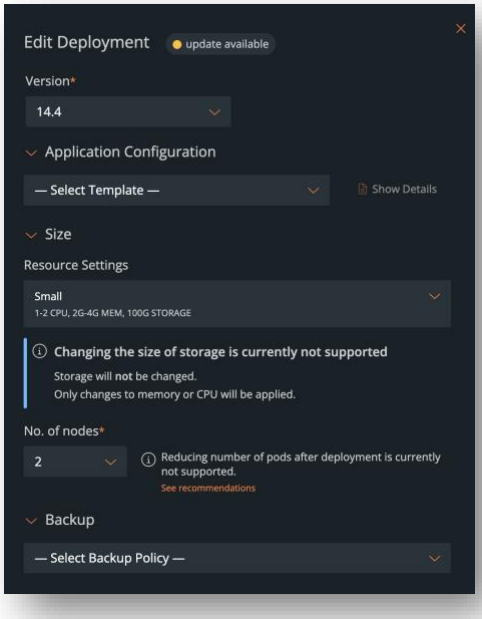


Scale Up

1. To perform a scale up operation, navigate back to the PDS dashboard and find your PostgreSQL cluster that you deployed in the last section.
2. Click on Edit in the Top Right section of the window:

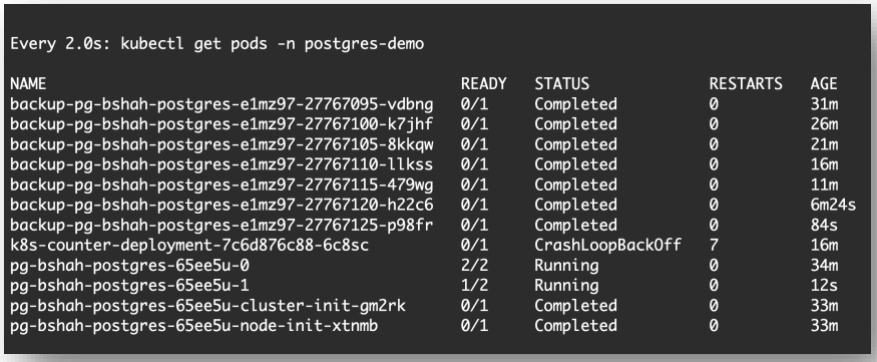


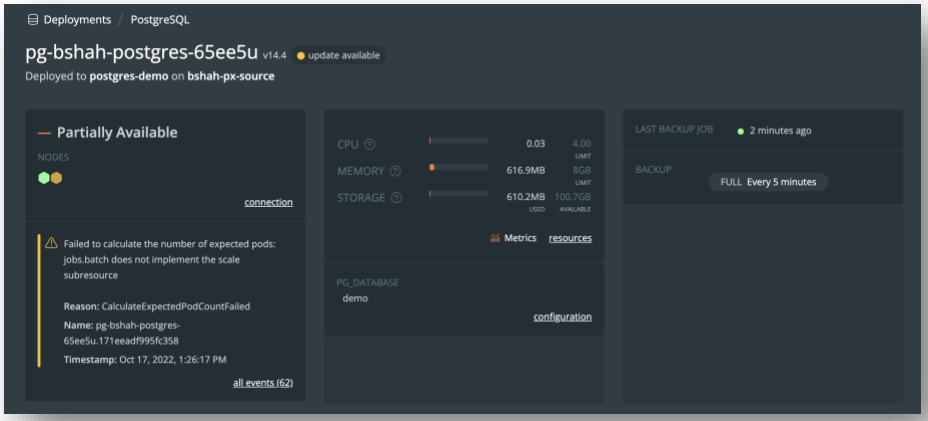
3. Change the Resource Settings and select Small template, to update the CPU and Memory resources limits and requests for individual PostgreSQL pods, click Apply.



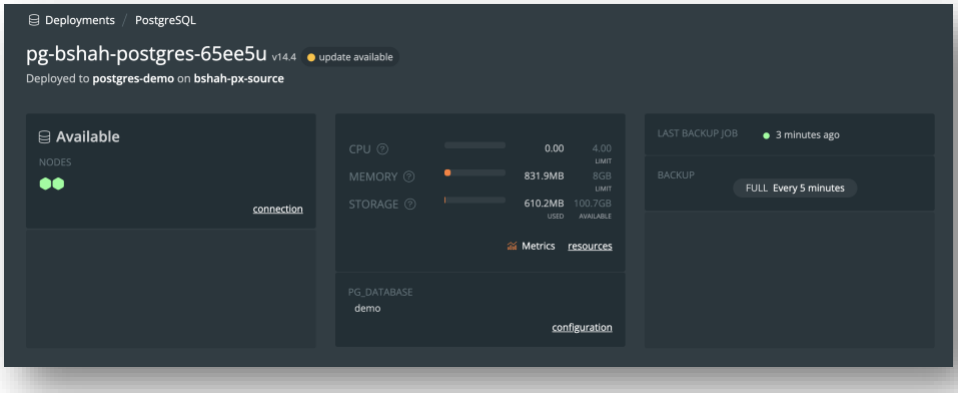
4. Monitor the scale up operation using the PDS UI or navigate to the AWS CloudShell and use the following command to watch the pods terminate and respawn.

```
watch kubectl get pods -n postgres-demo
```



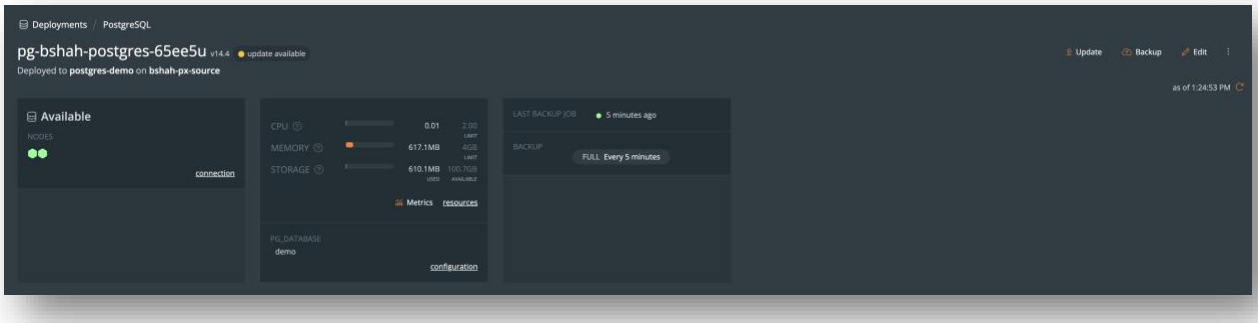


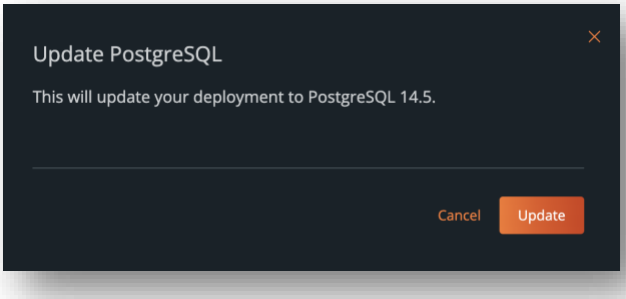
5. Once completed, you will also see the new limits set in the PDS UI.



Database Upgrades

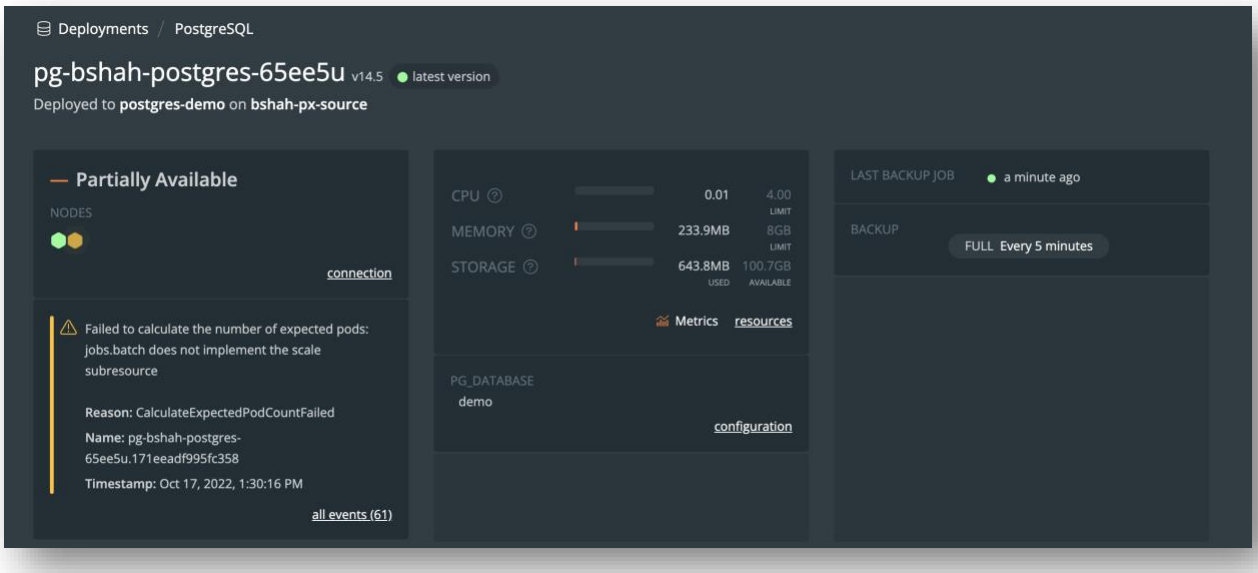
1. To perform a database upgrade operation, navigate back to the PostgreSQL cluster.
2. As you can see in the UI, there is an update available for your Postgres cluster. Click on "Update" button in the Top Right section of the window:



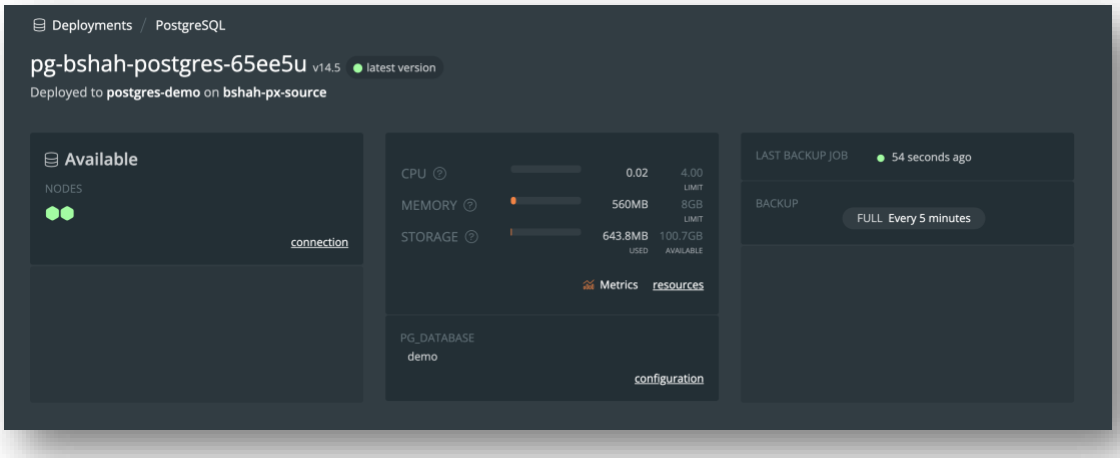


To perform the update – click "Update.

3. PDS performs a rolling update across your Postgres cluster, so that the current state/version matches the desired state/version that you have configured.



4. Once completed, you will also see the both the nodes Available and running the latest version.



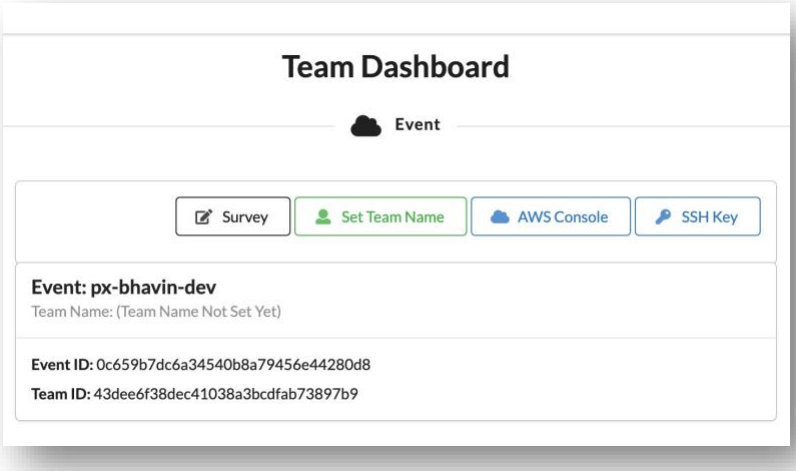
This is how easy it is to perform Day 0 and Day 2 operations using Portworx Data Services. Portworx Data Services truly allows developers to be agile and deploy data services they need for their applications using UI / REST APIs based workflow, while at the same time allowing administrators to customize things as needed and add Kubernetes target clusters to meet their availability and compliance requirements.

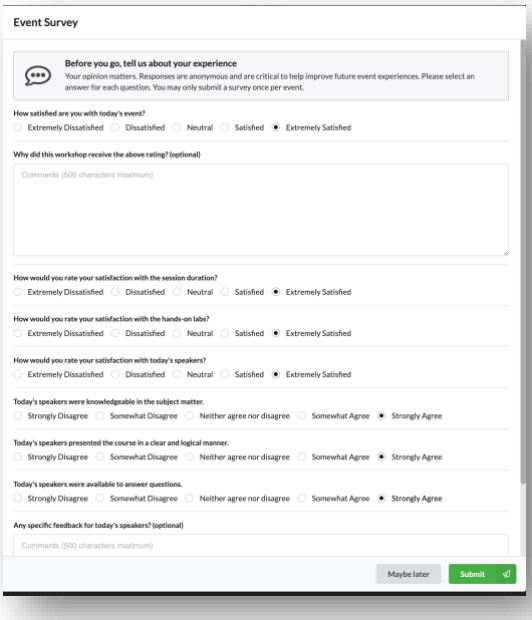
Clean Up

1. Once you are done with the workshop, navigate back to AWS CloudShell and use the following commands to clean up the resources deployed.

```
./cleanup.sh
```

2. Go back to the AWS Event Engine dashboard and click *"Survey"* to complete the survey.





3. Once you have filled out the survey, click *"Exit Event"* from the top right of the dashboard.

Additional Resources

- Portworx Blogs
- Portworx Demos
- Portworx Backup As A Service Documentation
- Portworx PX-Backup Documentation
- Portworx Enterprise Documentation
- Portworx Data Services Documentation