AWS Immersion Day – Data Management

Portworx | AWS

Table of Contents

Author

Bhavin Shah

Sr. Technical Marketing Manager

Portworx by Pure Storage

Lab Overview

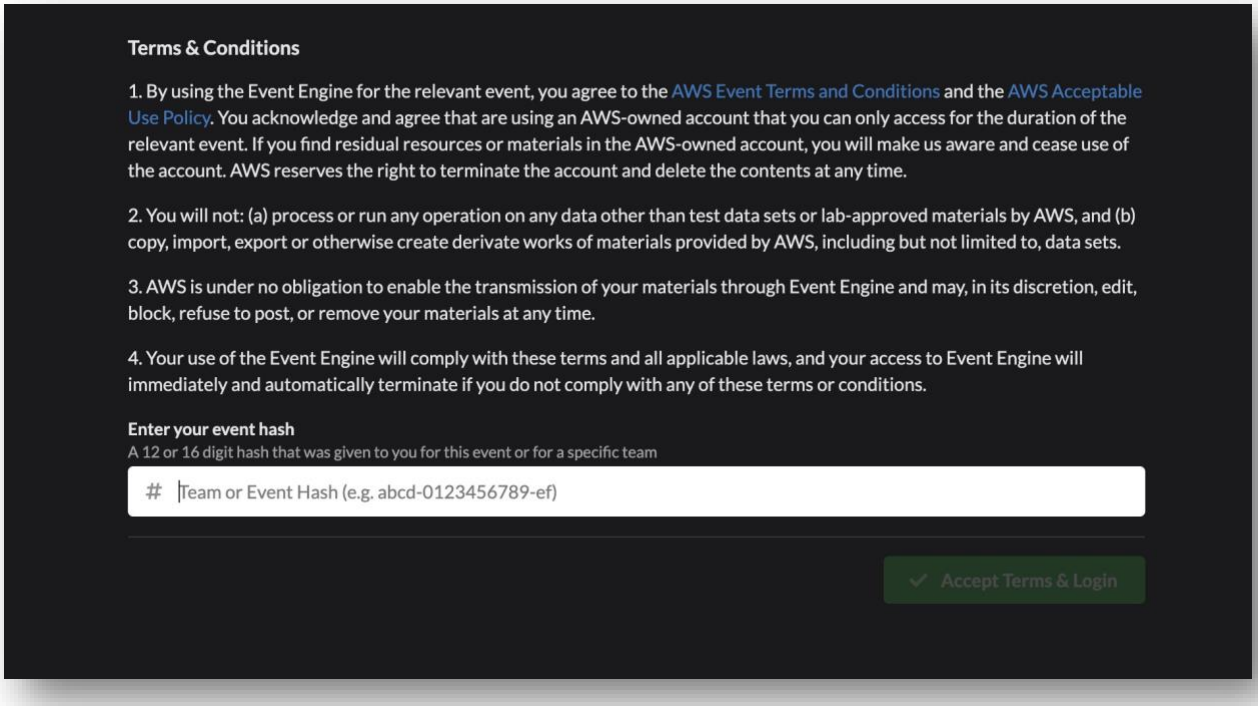
This lab guide is built for the Kubernetes Data Management Day ran by Portworx and AWS, to help users get hands-on experience with Portworx Enterprise - the #1 Kubernetes Data Platform solution in the ecosystem. As part of this lab, users will use an AWS Event Engine AWS account to deploy an Amazon EKS cluster, deploy highly available containerized applications on the Amazon EKS cluster and also use a free 30 day trial for Portworx Enterprise. The applications used in this lab are stateful applications based on PostgreSQL, Jenkins, etc. As part of the lab, we will simulate failure events like Availability Zone failures, Accidental database deletion, and see how Portworx Enterprise protects you from these scenarios. We will also look at how users can deploy block-based (ReadWriteOnce) and file-based (ReadWriteMany) applications using a unified Kubernetes storage layer from Portworx and leverage Portworx Autopilot to perform automated storage capacity management for their stateful applications.

Lab Setup

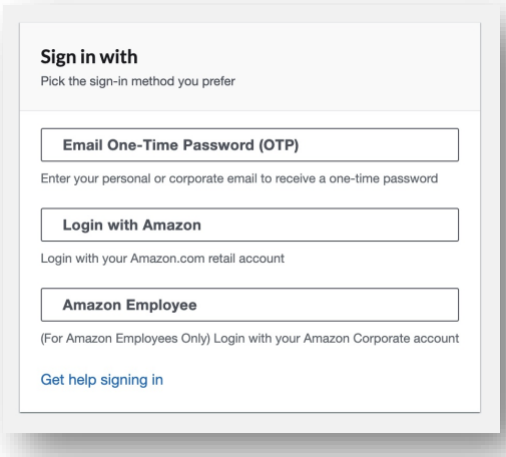
Accessing AWS Account

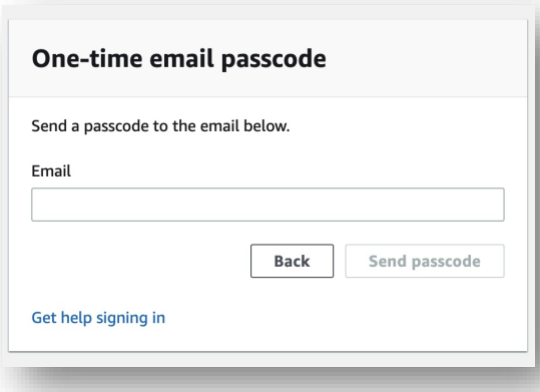
1. Navigate to the event engine platform using the link (https://dashboard.eventengine.run/login) and enter your 12- or 16-digit event hash.

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

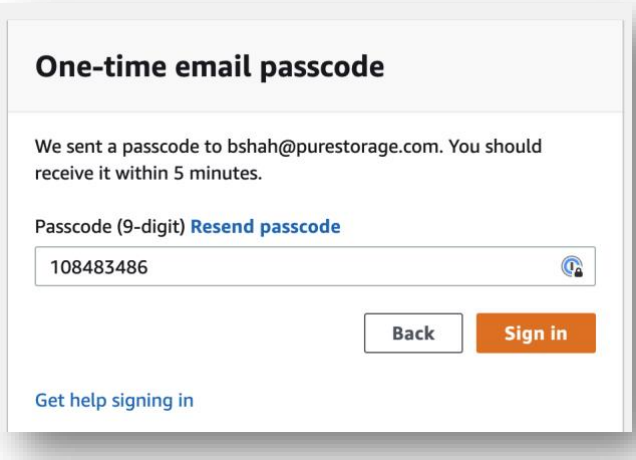


2. After entering your event hash, click *"Accept Terms & Login"*.
3. Next, click on *"Email One-Time Password (OTP)"* and enter your email address and hit *"Send Passcode"* to receive the OTP.

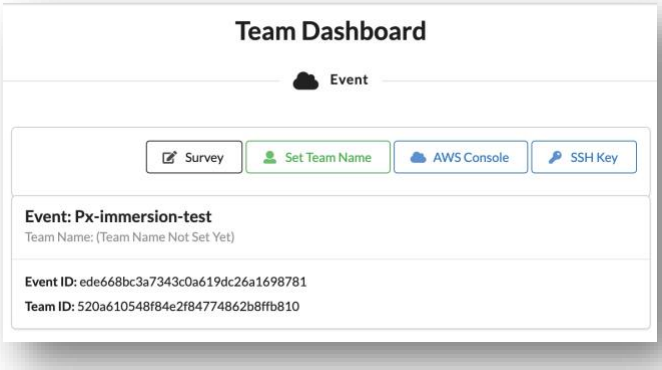




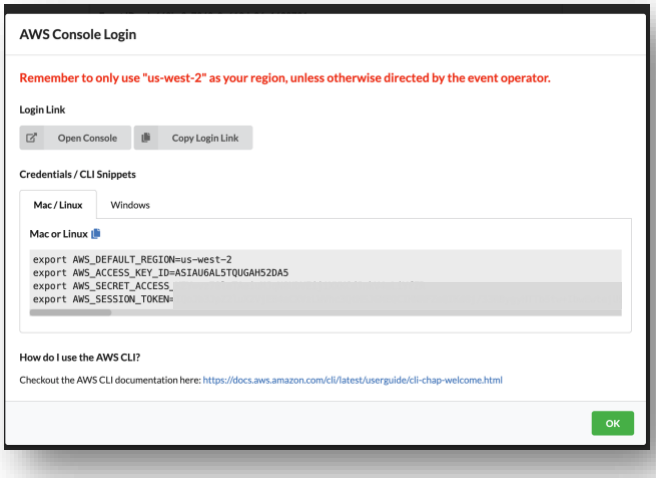
4. Check your email for the OTP. Copy the passcode and enter it in the Event Engine platform and hit *"Sign In"*.



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



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

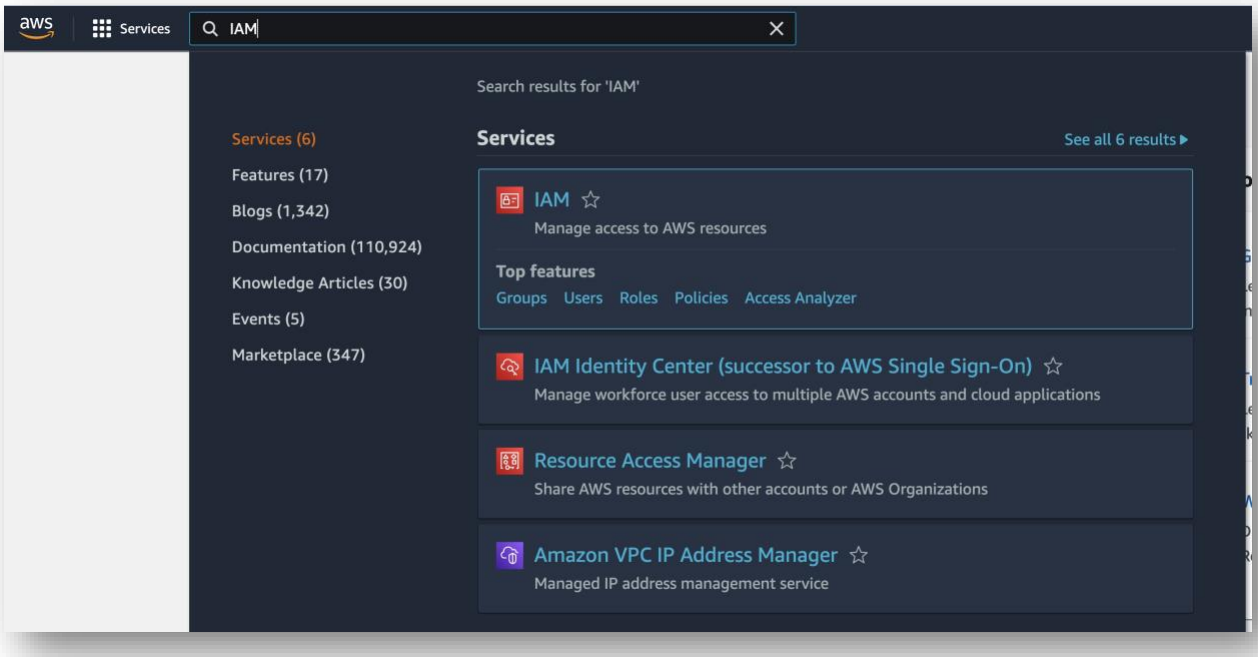


At this point, you have an AWS account that can be used for this Immersion Day workshop.

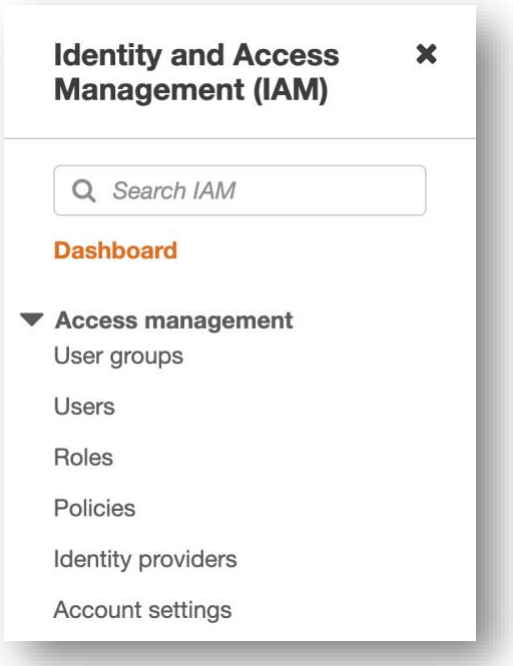
Create AWS IAM Policy for Portworx

Once you have access to your AWS account using the Event Engine platform, we will go ahead and create an IAM policy that will be used by Portworx to deploy Amazon EBS volumes and mount them on the Amazon EKS worker nodes.

1. From the AWS Console, navigate to Services and search for IAM.



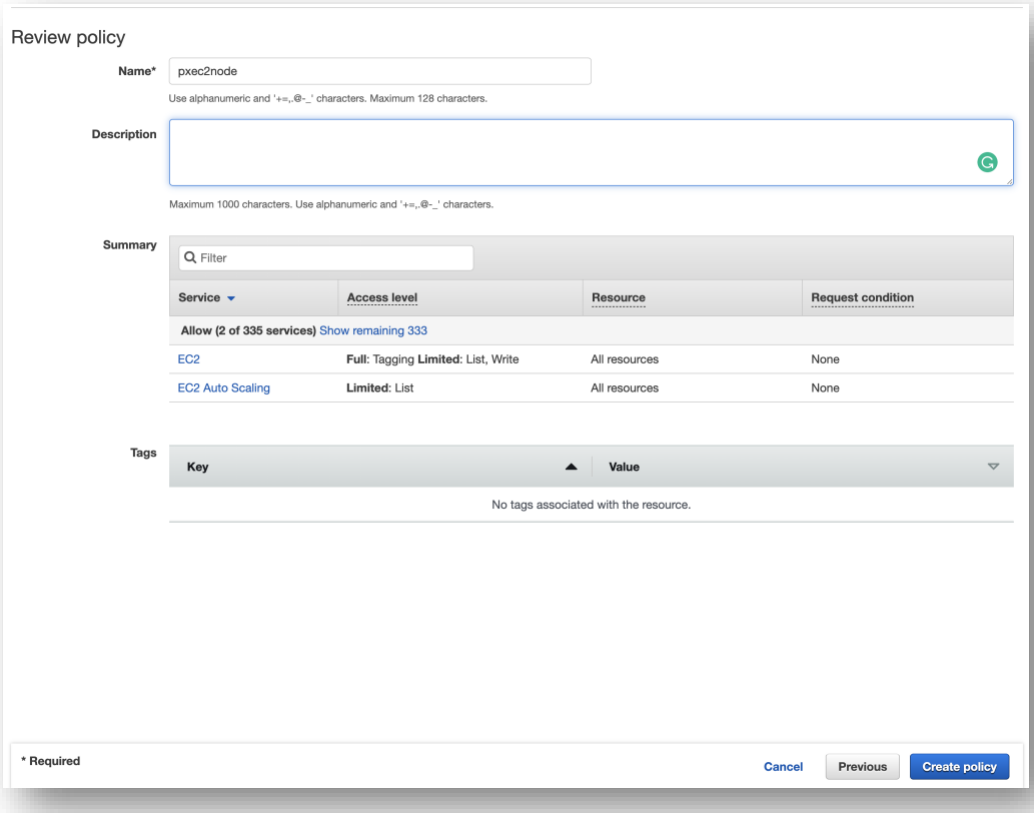
2. Click on “Policies” in the left Access Management pane.



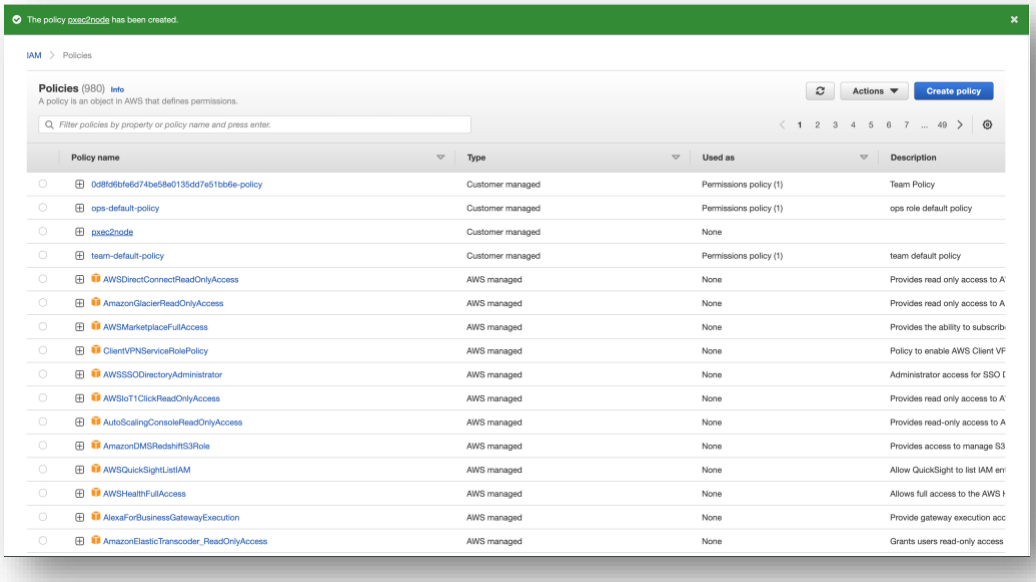
3. Click **"Create Policy"** on the right and select the JSON tab. Replace the text in the text editor with the following policy and click **Next: Tags**.

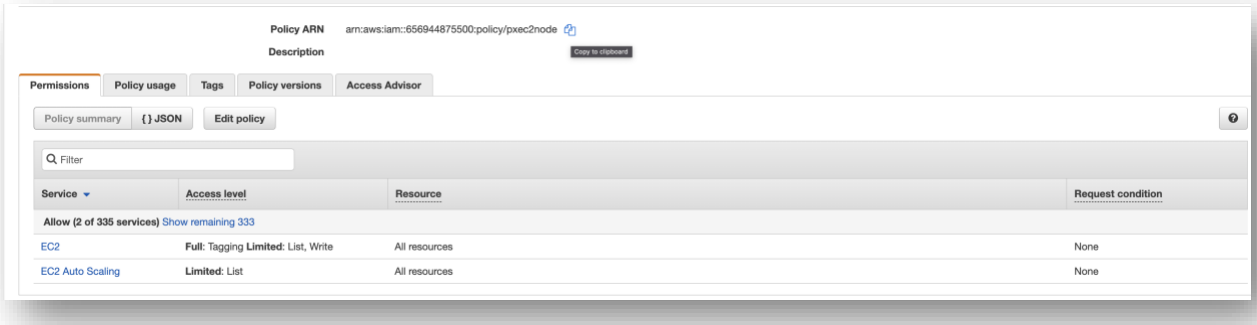
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "pxec2node",
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:ModifyVolume",
                "ec2:DetachVolume",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:DeleteTags",
                "ec2:DeleteVolume",
                "ec2:DescribeTags",
                "ec2:DescribeVolumeAttribute",
                "ec2:DescribeVolumesModifications",
                "ec2:DescribeVolumeStatus",
                "ec2:DescribeVolumes",
                "ec2:DescribeInstances",
                "autoscaling:DescribeAutoScalingGroups"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

4. Click **Next: Review** and enter “pxec2node” as the policy Name and click **“Create Policy”**.



5. Navigate to the pxec2node and copy the ARN for the policy and make a note of it for the next section.

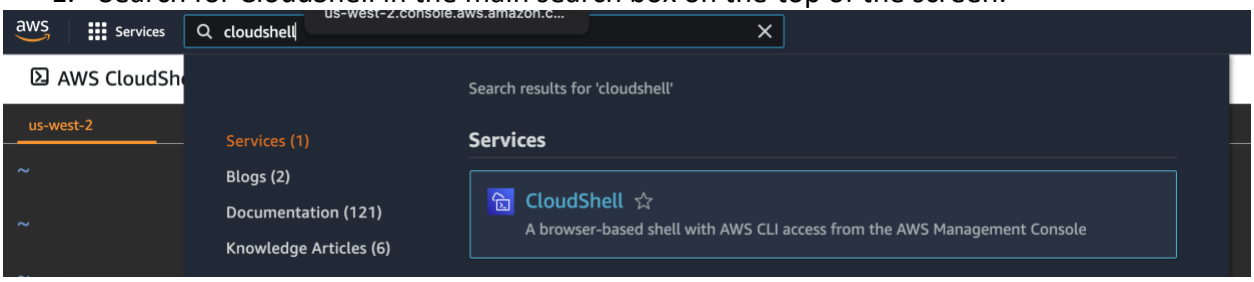




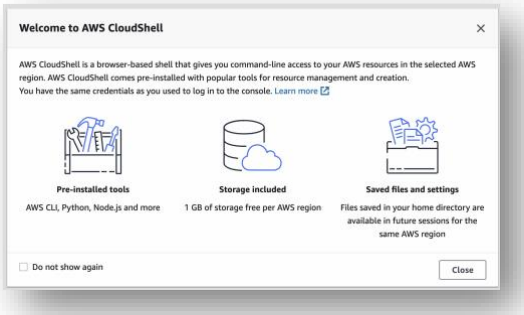
Deploying AWS Resources

Once you have created a new IAM policy (pxec2node), let's navigate to the AWS console and use the steps below to complete the pre-reqs for the lab:

1. Search for CloudShell in the main search box on the top of the screen.



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.

3. Clone the PX-DataManagement repository on to your CloudShell session using the following command:

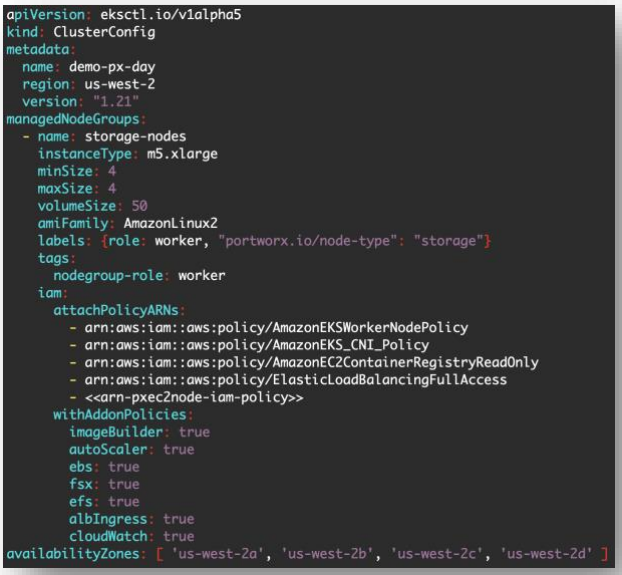
```
git clone https://github.com/bhavin04890/PX-DataMgmt.git
```

4. Once you have the repo cloned, we will go ahead and change directories.

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

5. Next, edit the "create-eks-cluster.yaml" file and update the ARN under the attachPolicyARNs section of the yaml file.

```
vi create-eks-cluster.yaml
```



6. Save the file using "esc → :wq"
7. Next, let's run the pre-req.sh file, which will deploy eksctl, kubectl and an Amazon EKS cluster.

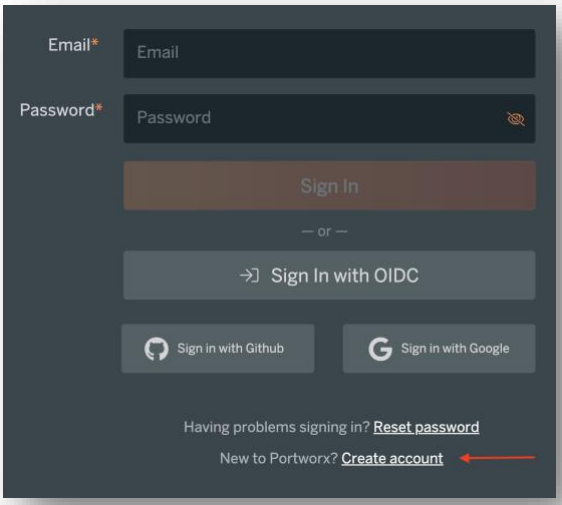
```
./pre-req.sh
```

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

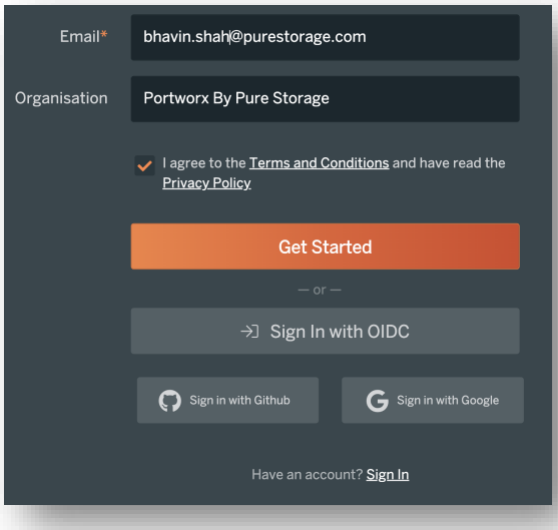
Create Portworx Central account

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

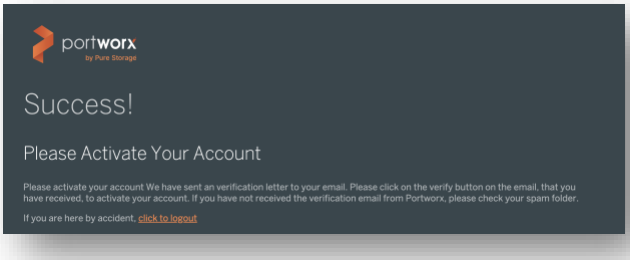
Note: Skip this section if you already have a Portworx Central account.



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



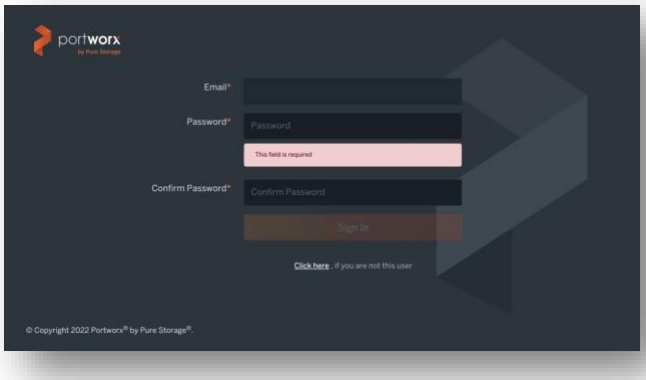
3. Activate your account by clicking the link in the email.



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.

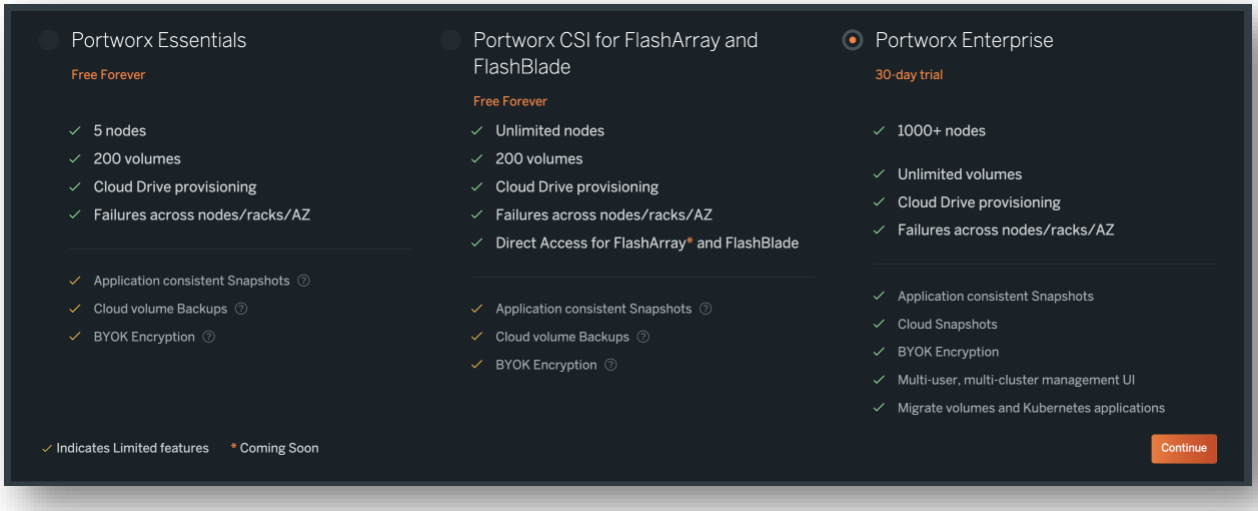


At this point, the pre-requisites for the AWS Immersion Day are done, and we will proceed to the slides portion and then come back for the hands-on lab again.

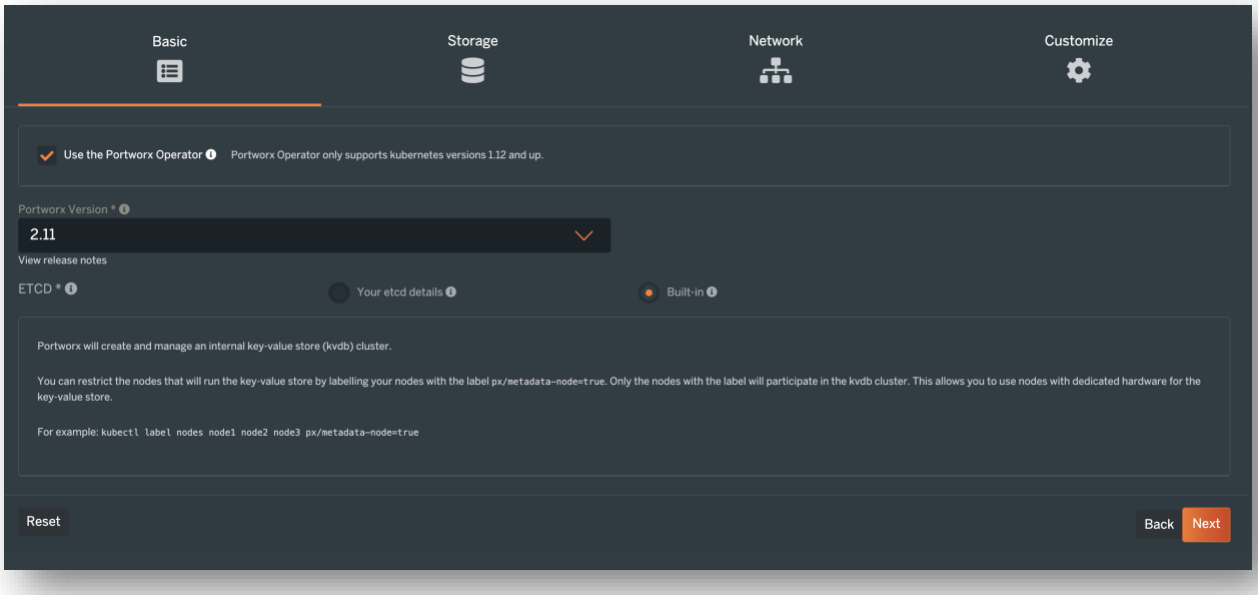
Deploying Portworx Enterprise on Amazon EKS

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

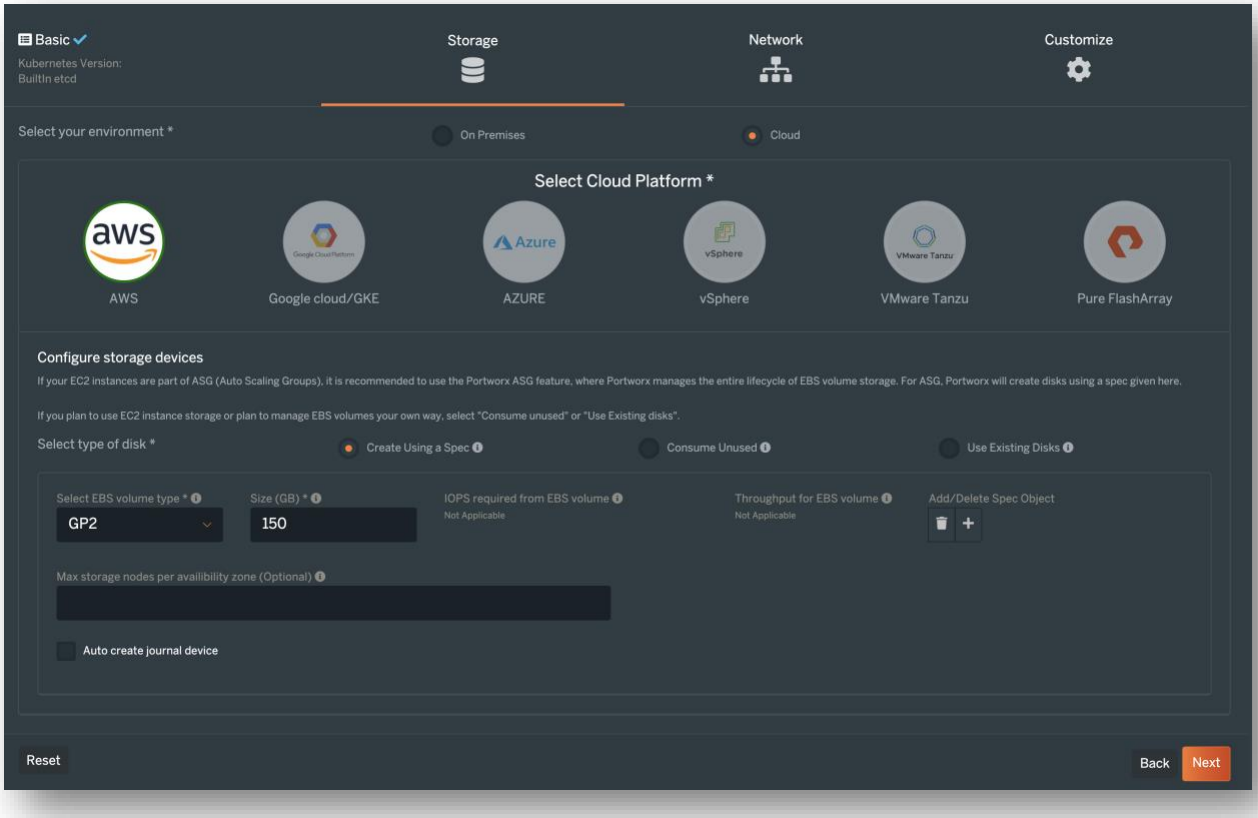
2. Select the "Portworx Enterprise – 30-day trial" click Continue.



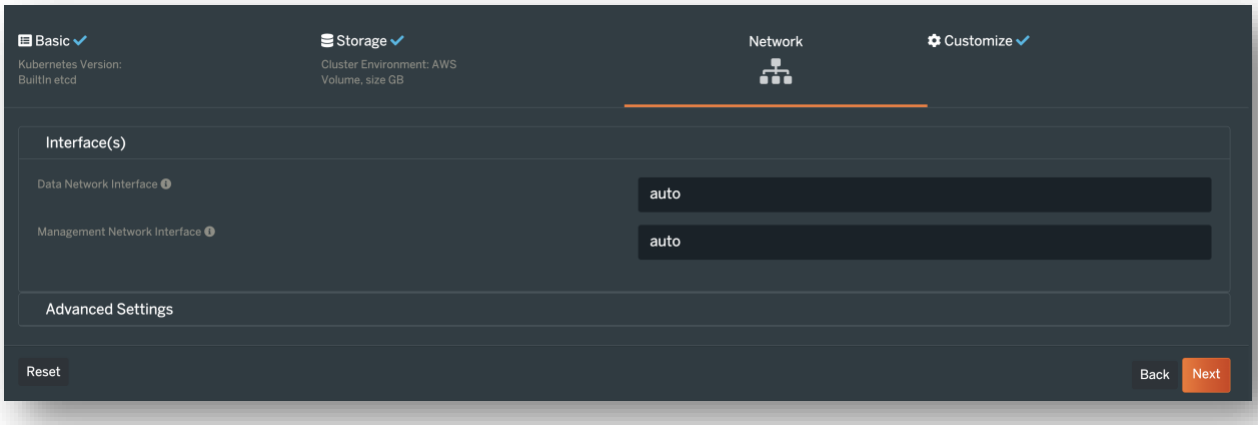
3. Check the box for "Portworx Operator" (If it's not checked already) and select 2.11 as the "Portworx Version" from the drop-down box. We will use the "Built-in" ETCD instance to use as the Key Value Database (KVDB) for Portworx. Click Next.



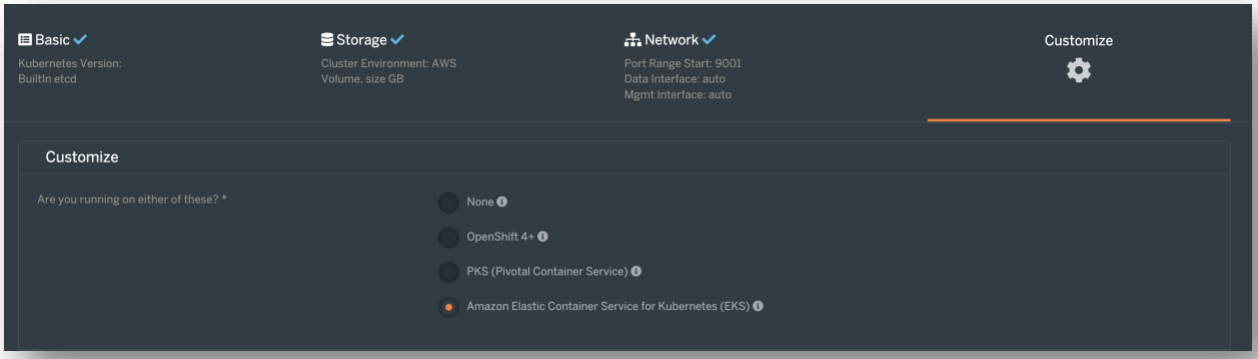
4. Select “Cloud” and select “AWS”. Click Next.

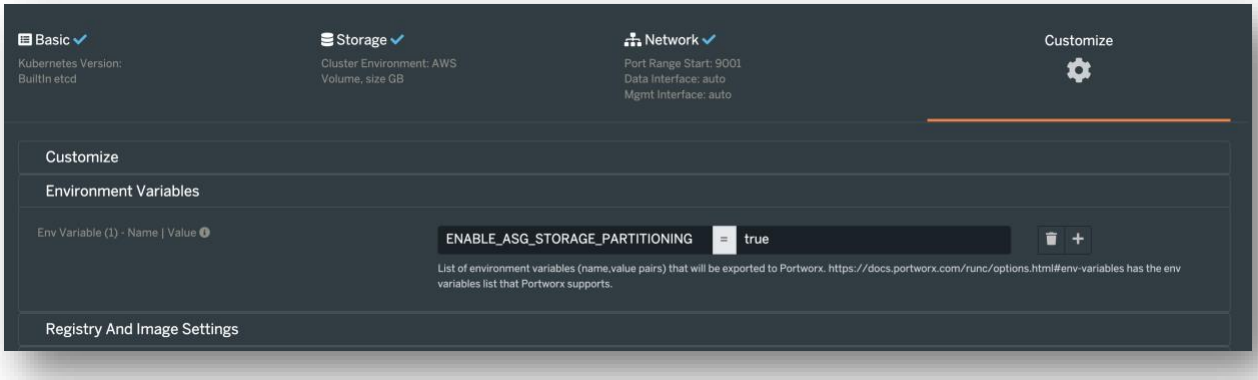


5. Leave the Network settings as default and click Next.



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.



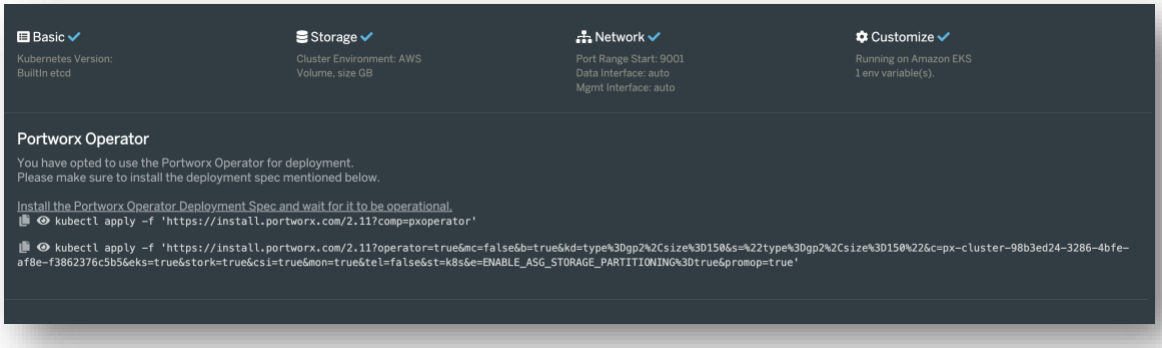


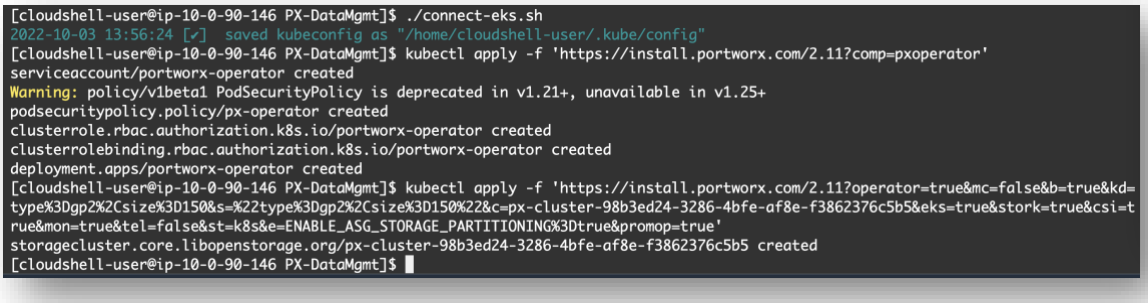
7. Read through the Portworx End User License Agreement and click Agree.
8. Copy both the ‘kubectl’ commands under the Portworx Operator section, we will use it to deploy Portworx Enterprise on our Amazon EKS cluster.
9. Next, navigate back to your AWS CloudShell session, and use the following command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





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
```

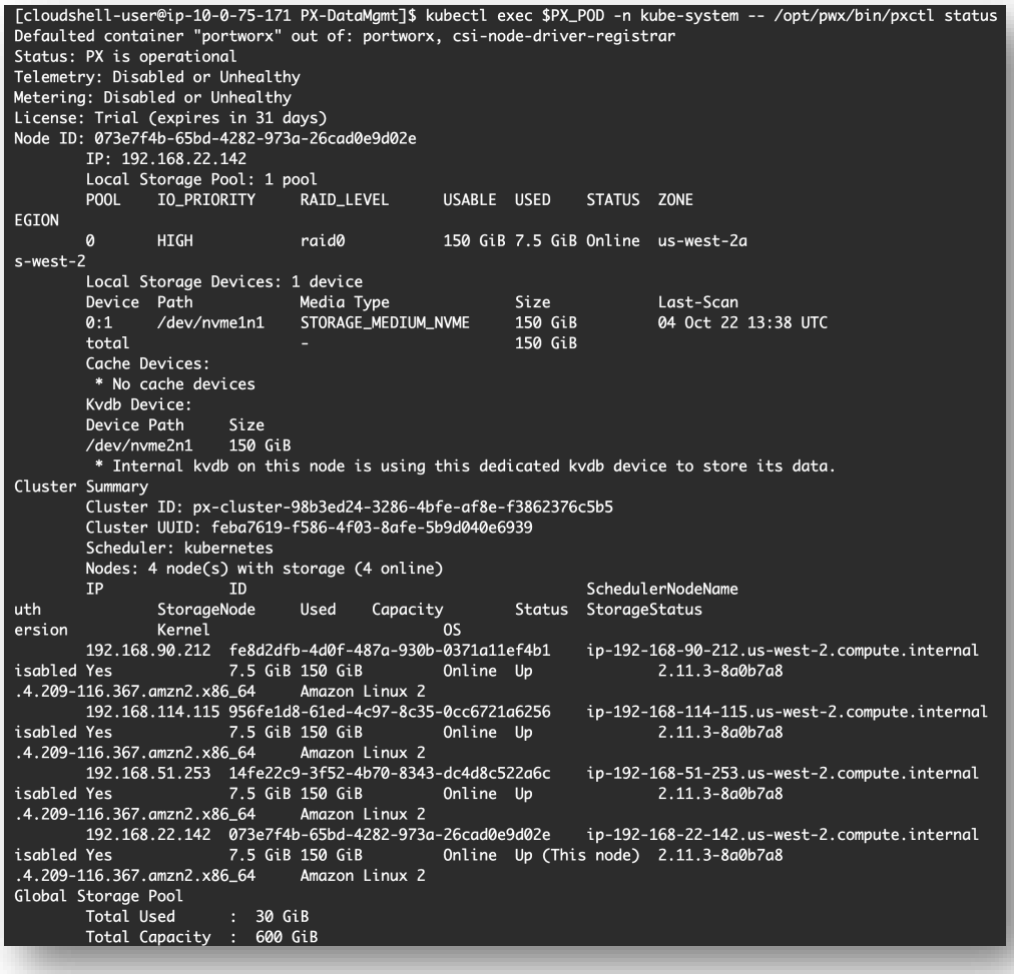
13. Use the following command to look at the Portworx StorageCluster status and the pre-created storageclasses.

```
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:

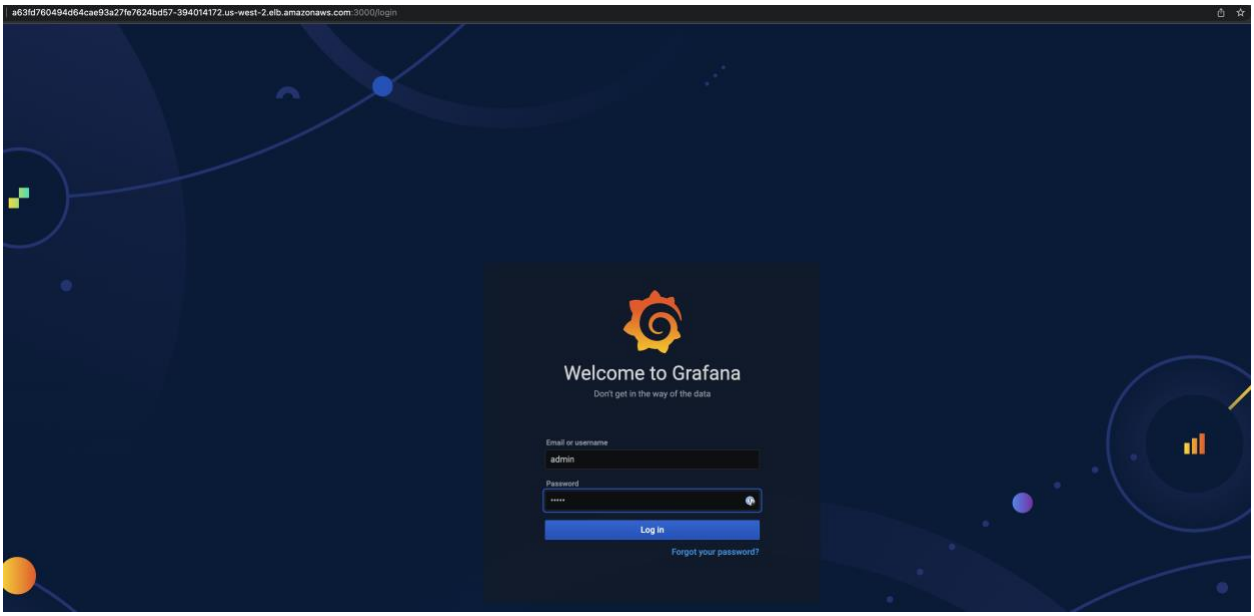
```
./install-grafana.sh
```

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

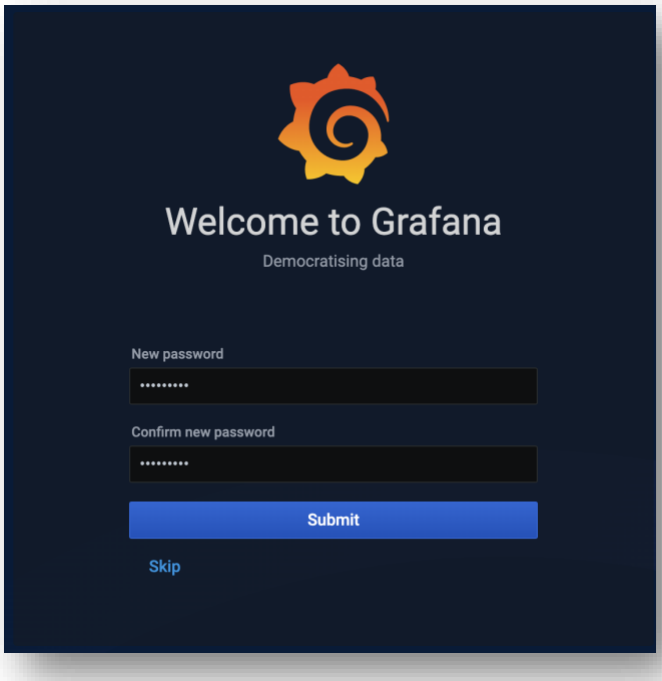
Note: Append port 3000 at the end of the load balancer URL (See example below) and navigate to the Grafana dashboard

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

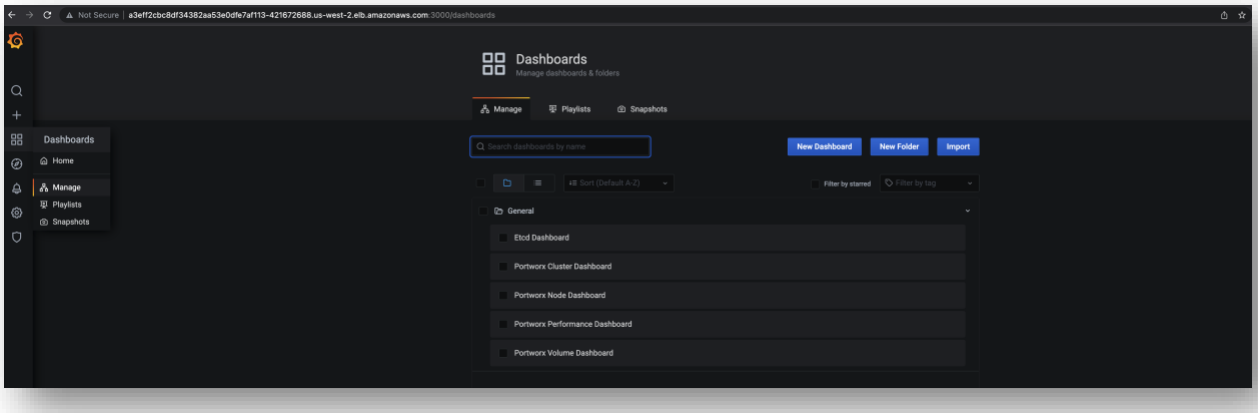
Note: If you are having issues connecting to the Amazon ELB endpoint, please disconnect from any VPNs you might be using. It might take a couple of minutes for the Grafana dashboard to be accessible.

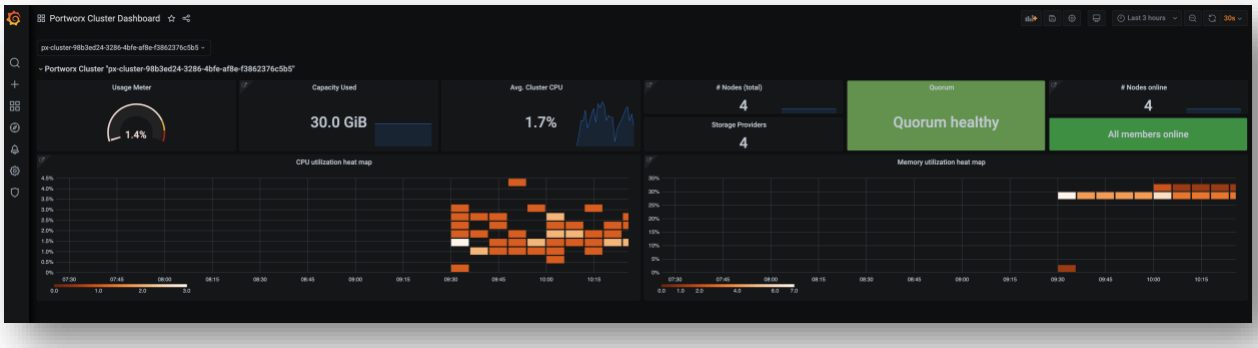


16. Set a new password for your Grafana instance on the next screen and then log into the Grafana instance.



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
```

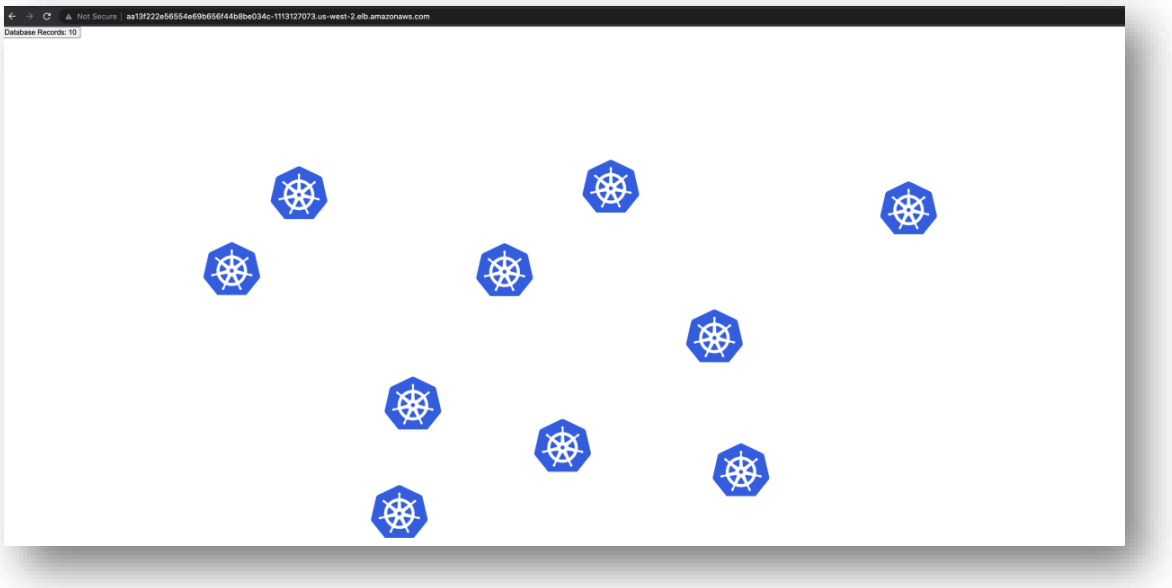
```
kubectl apply -f file-sc.yaml
```

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.
```

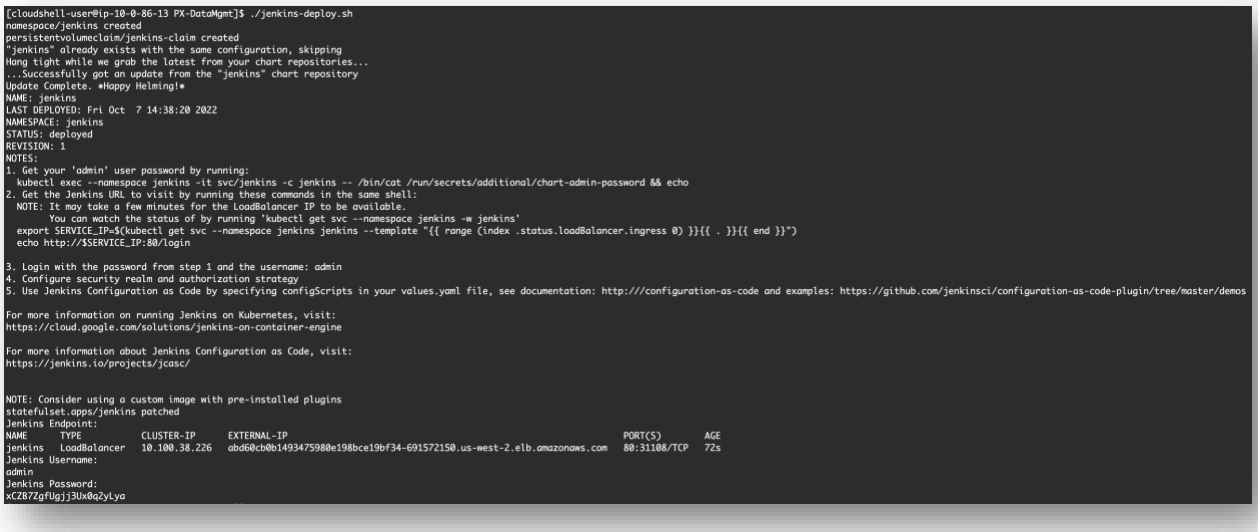
In this section, we will deploy a Highly available Jenkins deployment using a simple helm chart and a single ReadWriteMany persistent volume.

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

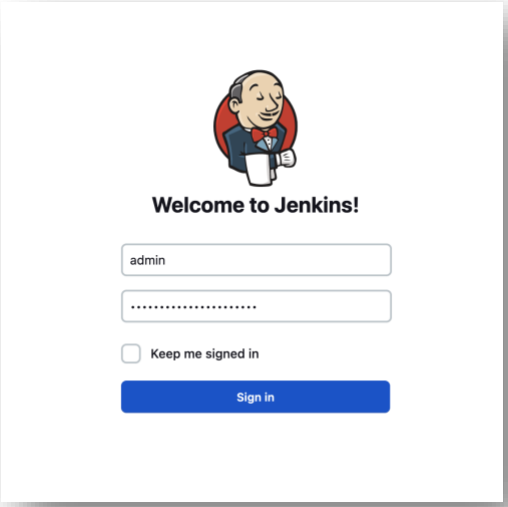
```
cat jenkins-pvc.yaml
```

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

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



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



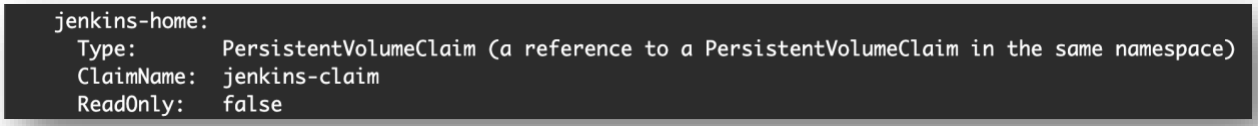
3. Next, go back to your CloudShell session and Inspect the persistent volume using the following script.

```
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
```



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
```

```
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           :  soft,timeo=600,vers=4.0,actimeo=60,port=2049,proto=tcp,retrans=4
Reads                     :  61
Reads MS                  :  77
Bytes Read                :  1146880
Writes                    :  16354
Writes MS                 :  23077
Bytes Written             :  584605696
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)
        - Name            : jenkins-3 (fc95b388-99d0-468c-85e3-bb80eed95fbf) (Pod)
          Namespace       : jenkins
          Running on      : ip-192-168-34-121.us-west-2.compute.internal
          Controlled by   : jenkins (StatefulSet)
        - Name            : jenkins-4 (0e94419e-46d0-43e7-b45c-7b1c1d43fe19) (Pod)
          Namespace       : jenkins
          Running on      : ip-192-168-34-121.us-west-2.compute.internal
          Controlled by   : jenkins (StatefulSet)
```

```
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
```

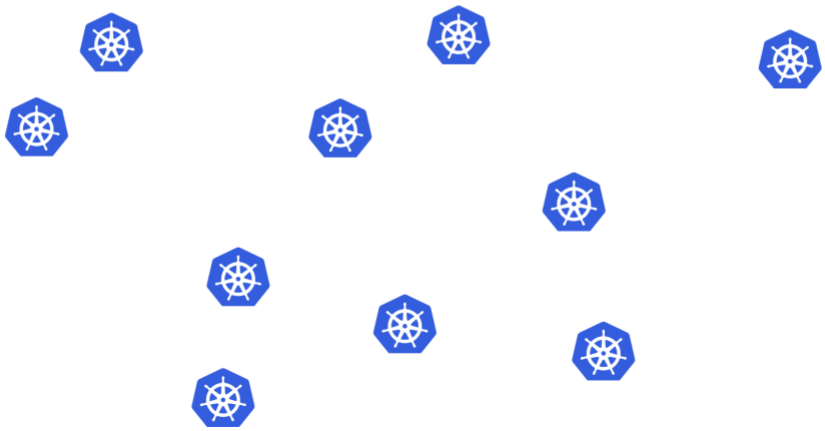
3. Once the entries have been deleted, we can go back to the UI and try to refresh the page. And, as you can see in the screenshot below, the UI is completely empty, as we deleted all the location information for the Kubernetes logos.

Click to add 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.

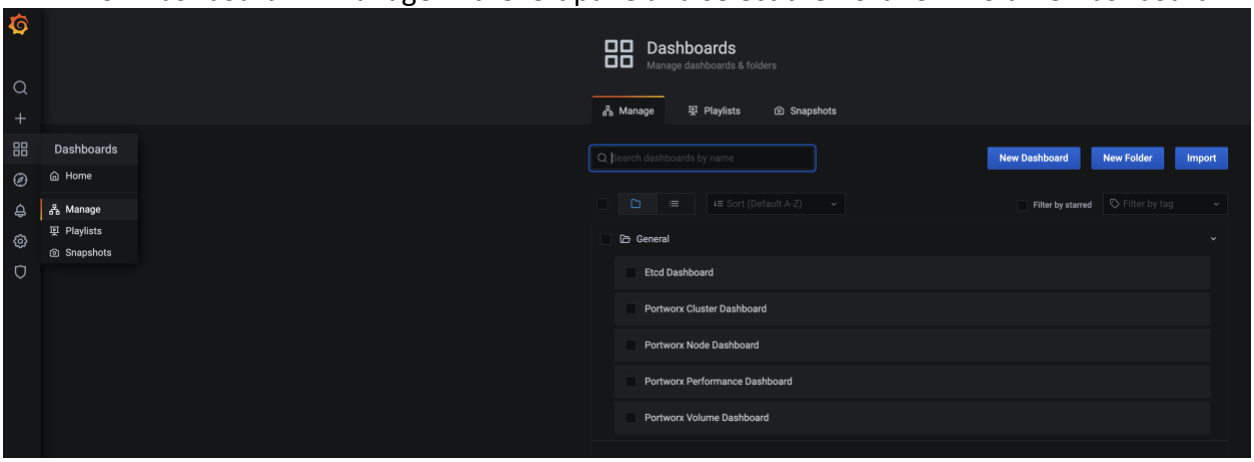
```
cat io-gen.sh

./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

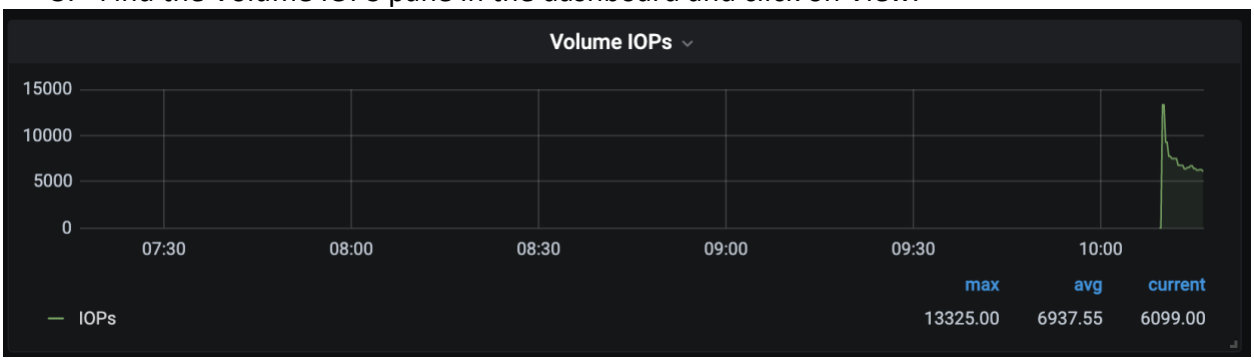
```
kubectl get pvc
```

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.

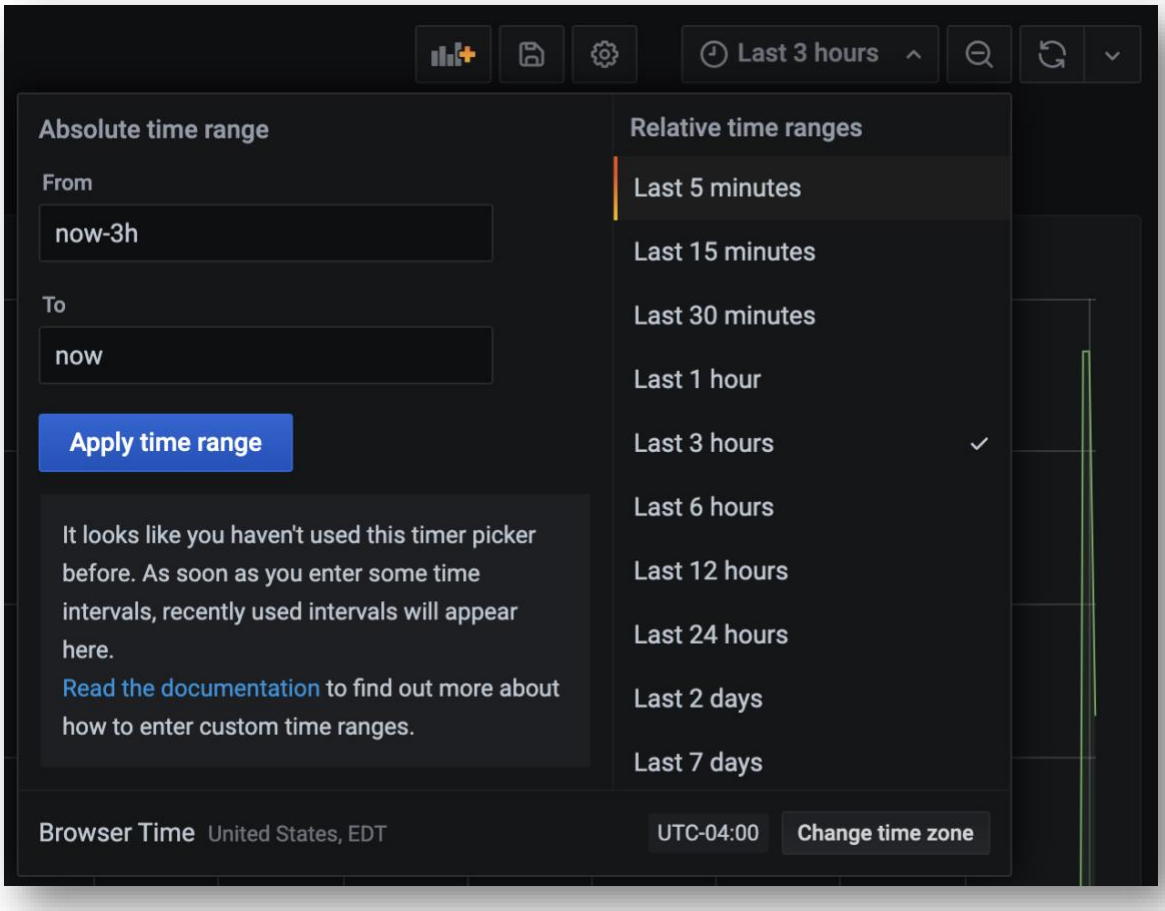


4. On the Portworx Volume Dashboard, select our FIO volume from the Volume Name drop down box.

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.

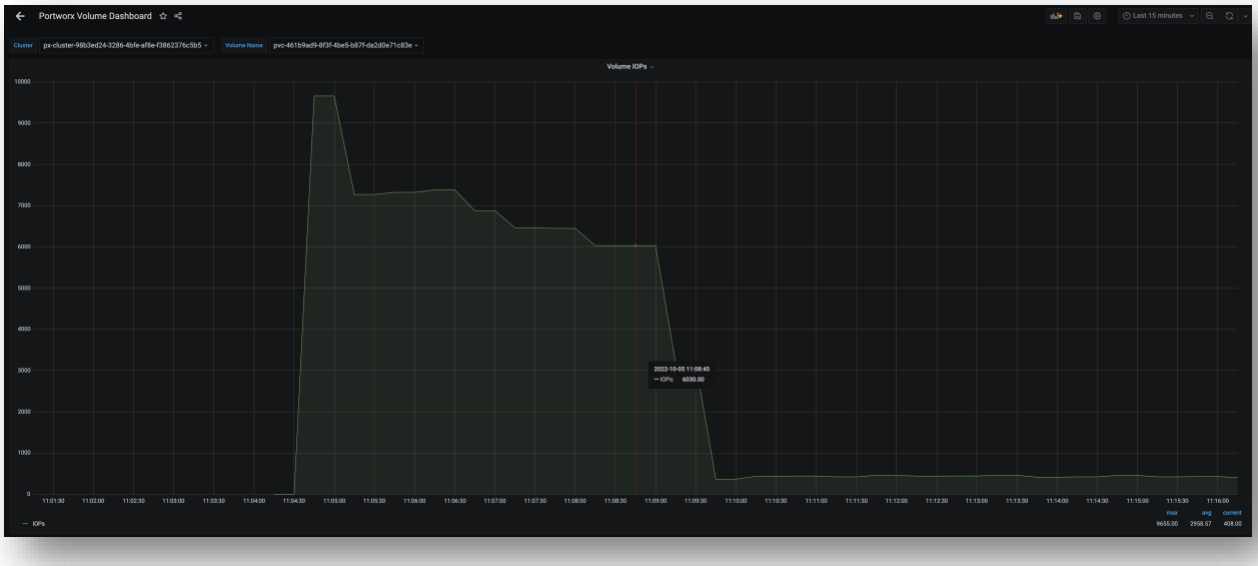
8. Navigate back to AWS CloudShell, and use a simple pxctl command to update the max read and write IOPS to 750 each. We will use a simple script called update-iops.sh to set these maximum limits for our persistent volume.

```
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
```

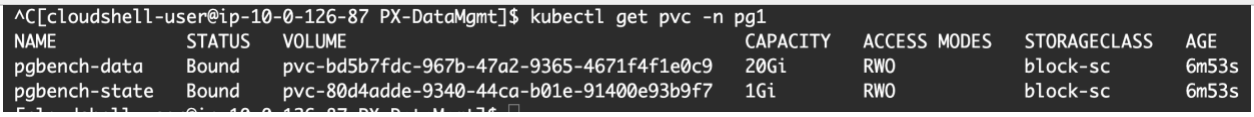
4. Next, let's use the watch command to monitor how Autopilot rule works:

```
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.

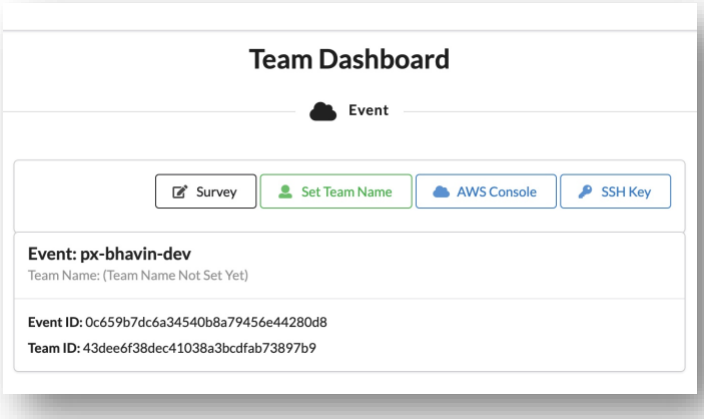
This is how Portworx allows you to reduce the operational burden when it comes to Day 2 operations for Amazon EKS clusters.

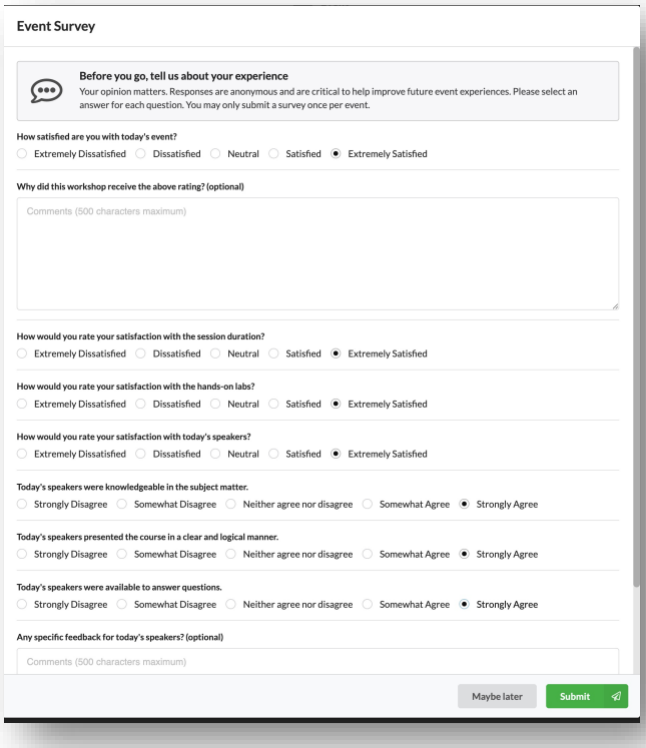
Clean Up

1. Once you are done with the lab, 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 Documentation