portworx by Pure Storage

Red Hat

TECHNICAL WHITE PAPER

Red Hat OpenShift on AWS with Portworx

Architecting a secure, highly-available Kubernetes data services platform for Red Hat OpenShift.

Contents

Executive Summary

As part of digital transformation efforts, organizations are modernizing their applications and the infrastructure they run on by adopting containers and Kubernetes for their applications and leveraging a solution like RedHat OpenShift for their infrastructure. Many enterprises already use Red Hat OpenShift in their Kubernetes architectures. Red Hat OpenShift allows organizations to take advantage of full-stack automated operations, a consistent experience—across all environments—and self-service provisioning for developers that lets teams work together to move ideas from development to production.

Modern applications built using containers and orchestrated by Kubernetes, still need a layer of persistence. To run stateful applications on Red Hat OpenShift, organizations need a robust data services platform like Portworx®. Portworx by Pure Storage® provides enterprise-level features like replication and high availability, security and encryption, capacity management, disaster recovery, and data protection to OpenShift deployments. Instead of spending resources architecting and managing a custom Kubernetes storage layer, organizations can accelerate their modernization journeys by adopting a solution like OpenShift with Portworx.

Pure Storage has integrated and validated their leading-edge storage technology for container workloads with an industry leading application solution platform to simplify deployment, to reduce risk, and free up IT resources for business-critical tasks. This process validates a solution as well as provides design consideration and deployment best practices to accelerate deployment. This document provides design considerations and deployment best practices for Red Hat OpenShift and Portworx to provide a modern infrastructure platform to run Kubernetes.

Introduction

In this paper, we discuss the benefits of using Portworx with Red Hat OpenShift on AWS (ROSA) to run stateful containerized applications. It is a vetted and tested design that includes different use cases, design considerations, deployment specifics, and configuration best practices for a developer-ready environment.

This white paper covers five use cases around Red Hat OpenShift described in the Solution Overview section below. To follow along with the deployment steps listed in this document, an administrator will need to first deploy a production-ready Red Hat OpenShift on AWS (ROSA) environment. The environment must include an installation of Red Hat OpenShift on AWS (ROSA) with a minimum of three master nodes and three worker nodes. The environment also must have internet access to pull resources for installation and configuration. For this validation we used the Red Hat OpenShift on AWS installer[1] for ROSA and Amazon Elastic Block Storage EBS storage with Portworx automated Cloud Drives provisioning to create Portworx storage pools for OpenShift.

[1]Note: this document only covers Red Hat OpenShift on AWS. For all other deployment methods, please refer to the Red Hat OpenShift and Portworx PVD.

Solution Overview

This solution covers four use cases that should help administrators deploy and operate a robust Kubernetes stack for their developers based on Red Hat OpenShift on AWS and Portworx Enterprise.

Use Cases

- **Stateful application monitoring**: In this use case, learn how to effectively monitor stateful applications on Red Hat OpenShift with Portworx using a combination of the OpenShift monitoring stack coupled with the Portworx monitoring stack. This use case will focus on using PX-Central with PX-Monitor which includes Prometheus and Grafana.
- **Automated storage pool and PVC capacity management with PX-Autopilot**: In this use case, learn how to install and use PX-Autopilot for Capacity Management with Red Hat OpenShift. PX-Autopilot allows OpenShift teams to automatically resize PVCs when they are running out of capacity and scale backend Portworx storage pools to accommodate increasing usage including rebalance volumes across Portworx storage pools when they come unbalanced. This use case uses Amazon EBS for the backend storage managed by PX-Autopilot.
- **Secure backup and restore with PX-Backup**: In this use case, learn how to use PX-Backup with secure self-service backup and restore for stateful and stateless applications on Red Hat OpenShift. This use case will set up PX-Backup to use clusters with role-based access controls and user management with enterprise LDAP integrations.
- **Highly available OpenShift container registry with Portworx read-write-many volumes**: In this use case, learn how the ability of Portworx to present RWX volumes to Kubernetes can be leveraged to provide developers with a highly available OpenShift Container Registry.
- **RBAC, security audit and encryption with PX-Security**: In this use case, learn how to effectively install, configure, and use Portworx PX-Secure to secure application volumes, volume requests, volume access as well as monitor and audit data management on Red Hat OpenShift.

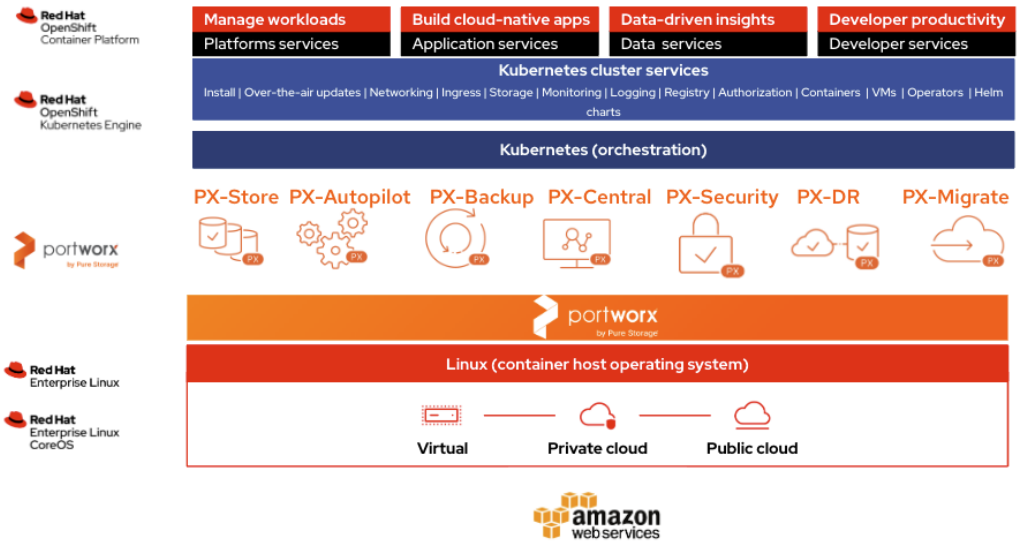


Figure 1: Portworx, RedHat OpenShift, and AWS

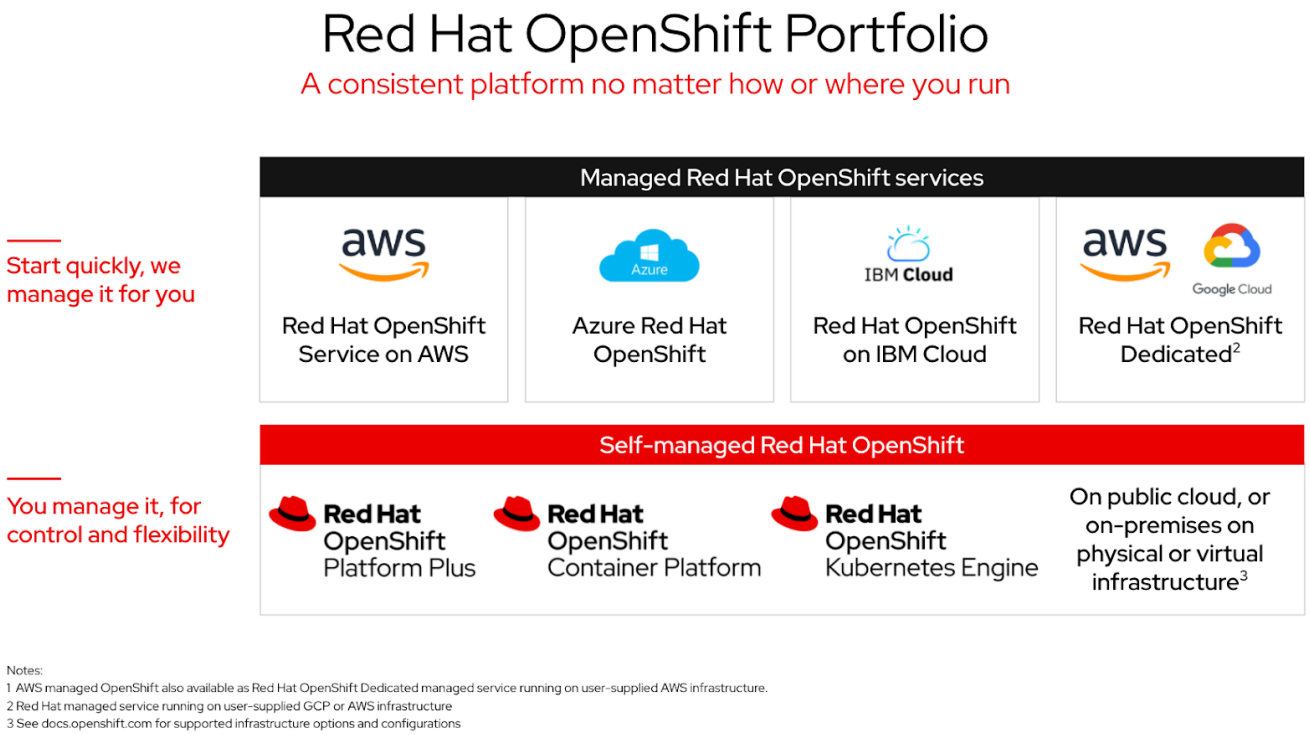
Solution Benefits

This solution enables organizations to accelerate their adoption of modern applications and Kubernetes by combining the best-in-class Kubernetes offering from Red Hat, the leading cloud provider with Amazon Web Services, with the gold standard of Kubernetes data services platform from Portworx. Organizations that are getting started with their Kubernetes adoption journey will find this solution valuable, as it walks them through the basic use case of providing persistent storage for their containers and advanced use cases like data protection and disaster recovery. Organizations that are already running applications on Kubernetes will also find this solution valuable, as it helps them take the next step and implement a platform that helps them ensure business continuity, while also ensuring that they get the best performance and reliability from their Kubernetes storage layer.

In addition to the consistency provided by Red Hat OpenShift across different environments, Portworx also provides consistency when it comes to Kubernetes storage. Organizations can choose to run their OpenShift clusters on-prem or managed offerings, like ROSA, in the AWS cloud, and still rely on Portworx to provide the same set of Kubernetes data services for their applications.

Red Hat OpenShift

Red Hat OpenShift Container Platform on AWS is a managed solution provided by Red Hat and AWS designed to help organizations just starting their container journey, or those wishing to focus on the applications and looking to leverage Red Hat's expertise for general maintenance and management tasks.



Notes:
1 AWS managed OpenShift also available as Red Hat OpenShift Dedicated managed service running on user-supplied AWS infrastructure.
2 Red Hat managed service running on user-supplied GCP or AWS infrastructure
3 See docs.openshift.com for supported infrastructure options and configurations

Figure 2: Red Hat OpenShift portfolio

Red Hat OpenShift comes with a streamlined, automatic install so you can get up and running with Kubernetes as quickly as possible. Once installed, Red Hat OpenShift uses Kubernetes Operators for push-button, automatic platform updates for the container host, Kubernetes cluster, and application services running on the cluster.

Red Hat OpenShift Container Platform delivers a single, consistent Kubernetes platform anywhere that Red Hat Enterprise Linux runs. The platform ships with a user-friendly console to view and manage all your clusters so you have enhanced visibility across multiple deployments.

Red Hat OpenShift comes with developer-friendly workflows including built-in CI/CD pipelines and our source-to-image capability that enables you to go straight from application code to container. Extend to new technologies—including serverless applications with Knative, cloud services, and streamlined service communications with Istio and service mesh.

Red Hat OpenShift offers:

- **Trusted platform**: Red Hat OpenShift builds security checks into your container stack—starting with Red Hat Enterprise Linux and continuing throughout the application life cycle.
- **Built-in monitoring**: Red Hat OpenShift includes Prometheus, the standard for cloud-native cluster and application monitoring. Use Grafana dashboards for visualization.
- **On-demand environments**: Self-service for application teams to access approved services and infrastructure, with centralized management and administration.
- **Ecosystem integration**: Red Hat has worked with hundreds of partners to validate technology integrations with Red Hat OpenShift, so organizations can make the most of their existing investments.
- **Centralized policy management**: Red Hat OpenShift gives administrators a single place to implement and enforce policies across multiple teams, with a unified console across all Red Hat OpenShift clusters.
- **Certified Kubernetes**: Red Hat OpenShift is part of the Cloud Native Computing Foundation (CNCF) Certified Kubernetes program, ensuring compatibility and interoperability between your container workloads.

Portworx

Portworx is a data management solution that serves applications and deployments in Kubernetes clusters. Portworx is deployed natively within Kubernetes and extends the automation capabilities down into the infrastructure to eliminate all the complexities of managing data. Portworx provides simple and easy-to-consume StorageClasses that are usable by stateful applications in a Kubernetes cluster.

Figure 3: Portworx family of products.

At the core of Portworx is PX-Store, a software-defined storage platform that works on practically any infrastructure, regardless of whether it is in a public cloud or on-premises. PX-Store is complemented by these modules:

- **PX-Migrate**: Allows applications to be easily migrated across clusters, racks, and clouds
- **PX-Secure**: Provides access controls and enables data encryption at a cluster, namespace, or persistent volume level
- **PX-DR**: A service that allows applications to have a zero RPO failover across data centers in a metro area as well as continuous backups across the WAN for even greater protection
- **PX-Backup**: A solution that allows enterprises to back up and restore the entire Kubernetes application, including data, app configuration, and Kubernetes objects, to any backup location—including S3, Azure Blob, etc.—with the click of a button.
- **PX-Autopilot**: A service that provides rules-based auto-scaling for persistent volumes and storage pools

PX-Store

PX-Store is a 100% software-defined storage solution that provides high levels of persistent volume density per block device per worker node. The key features of PX-Store include

- **Storage virtualization**: The storage made available to each worker node is effectively virtualized such that each worker node can host pods that use up to hundreds of thousands of persistent volumes per Kubernetes cluster. This benefits Kubernetes clusters deployed to the cloud, in that larger volumes or disks are often conducive to better performance.
- **Storage-aware scheduling**: Stork, a storage-aware scheduler, co-locates pods on worker nodes that host the persistent volume replicas associated with the same pods, resulting in reduced storage access latency.
- **Storage pooling for performance-based quality-of-service**: PX-Store segregates storage into three distinct pools of storage based on performance: low, medium, and high. Applications can select storage based on performance by specifying one of these pools at the StorageClass level.
- **Persistent volume replicas**: You can specify a persistent volume replication factor at the StorageClass level. This enables the state to be highly available across the cluster, cloud regions, and Kubernetes-as-a-service platforms.
- **Cloud volumes**: Cloud volumes enable storage to be provisioned from the underlying platform without the need to present storage to worker nodes. PX-Store running on most public cloud providers and VMware Tanzu have cloud volume capability.
- **Automatic I/O path tuning**: Portworx provides different I/O profiles for storage optimization based on the I/O traffic pattern. By default, Portworx automatically applies the most appropriate I/O profile for the data patterns it sees. It does this by continuously analyzing the I/O pattern of traffic in the background.
- **Metadata caching**: High-performance devices can be assigned the role of journal devices to lower I/O latency when accessing metadata.
- **Read- and write-through caching**: PX-Cache-enabled high-performance devices can be used for read- and write-through caching to enhance performance.

PX-Backup

Backup is essential for enterprise applications, serving as a core requirement for mission-critical production workloads. The risk to the enterprise is magnified for applications on Kubernetes where traditional, virtual machine (VM)-optimized data protection solutions simply don't work. Protecting stateful applications like databases in highly dynamic environments calls for a purpose-built, Kubernetes-native backup solution.

Portworx PX-Backup solves these problems and protects your applications' data, configuration, and Kubernetes objects with a single click at the Kubernetes pod, namespace, or cluster level. Enabling application-aware backup and fast recovery for even complex distributed applications, PX-Backup delivers true multi-cloud availability with key features, including

- **App-consistent backup and restore**: Easily protect and recover applications regardless of how they are initially deployed on or rescheduled by Kubernetes.
- **Seamless migration**: Move a single Kubernetes application or an entire namespace between clusters.
- **Compliance management**: Manage and enforce compliance and governance responsibilities with a single pane of glass for all your containerized applications.
- **Streamlined storage integration**: Back up and recover cloud volumes with storage providers including Amazon EBS, Google Persistent Disk, Azure Managed Disks, and CSI-enabled storage.

PX-DR

PX-DR extends the data protection included in PX-Store with zero RPO disaster recovery for data centers in a metropolitan area as well as continuous backups across the WAN for an even greater level of protection. PX-DR provides both synchronous and asynchronous replication, delivering key benefits like:

- **Zero data loss disaster recovery**: PX-DR delivers zero RPO failover across data centers in metropolitan areas in addition to HA within a single data center. You can deploy applications between clouds in the same region and ensure application survivability.
- **Continuous global backup**: For applications that span a country—or the entire world—PX-DR also offers constant incremental backups to protect your mission-critical applications.

PX-Autopilot

PX-Autopilot allows enterprises to automate storage management to intelligently provision cloud storage only when needed and eliminate the problem of paying for storage when over-provisioned. PX-Autopilot delivers several benefits:

- **Grow storage capacity on-demand**: Automate your applications' growing storage demands while also minimizing disruptions. Set growth policies to automate cloud drive and Kubernetes integration to ensure each application's storage needs are met without performance or availability degradations.
- **Slash storage costs in half**: Intelligently provision cloud storage only when needed and eliminate the problem of paying for storage when over-provisioned instead of consumed. Scale at the individual volume or entire cluster level to save money and avoid application outages.
- **Integrate with all major clouds and VMware**: PX-Autopilot natively integrates with AWS, Azure, and Google as well as Red Hat OpenShift, enabling you to achieve savings and increase automated agility across all your clouds.

Deployment Options

When creating a specification to deploy Portworx with, you have several options to consider:

- **Existing KVDB**: For most deployments, you can create a deployment specification with the option of storing Portworx metadata in a separate etcd cluster. There are two exceptions to this:
  - The first is when the PX-DR is used for Kubernetes clusters that are not within the same metro area, meaning the network round-trip latency between the primary and disaster recovery sites is greater than 10ms.
  - The second is when a dedicated etcd cluster should be used is for large-scale deployment, with 10 or more worker nodes, in which a heavy dynamic provisioning activity takes place.
- **Dedicated journal device**: A dedicated journal device can be specified to buffer metadata writes.
- **Dedicated cache device**: A dedicated cache device can be specified to improve performance by acting as a read/write-through cache.
- **Container storage interface (CSI) API compatibility**: You can choose the option to deploy Portworx with CSI enabled if PX-Security is to be used.
- **Stork**: Stork is a storage-aware scheduler that attempts to co-locate application pods onto the same nodes as the persistent volumes and persistent volume replicas that it uses. Use Stork if your underlying infrastructure uses either servers with dedicated internal storage or servers with dedicated network-attached storage appliances.
- **Dedicated network**: Consider using a dedicated network for storage cluster traffic if the existing network infrastructure does not support quality-of-service.

Planning, Design, and Prework

This section of the document covers the detailed setup used for Portworx deployment and testing on Red Hat OpenShift on AWS. The following chart will help in navigating the use case to the environment.

- This document focuses on Portworx 2.8.x, 2.9.x and OpenShift 4.7, 4.8, 4.9
- PX-Backup installation will be covered in the backup specific use case.
- This document does not cover the full breadth of the Portworx Platform for OpenShift but rather focuses on the subset of use cases defined. Reference the Additional Resources to find out more.

Use Case	Clusters	Environment
Stateful application monitoring	1	Amazon Web Services with EBS
RBAC, security audit and encryption with PX-Security	1	Amazon Web Services with EBS
Secure backup and restore with PX-Backup	2	Amazon Web Services with EBS and S3 Backup Location
Automated storage pool and PVC capacity management with PX-Autopilot	1	Amazon Web Services with EBS

Table 1: Use cases and environments

The specific setup, configurations, and requirements for AWS are shown below; however, there is also a common set of requirements for running a Portworx Data Management and ROSA together.

- A minimum of three master nodes for control plane high availability
- A minimum of three worker nodes dedicated to running Portworx. Portworx needs three (or more) worker nodes for quorum. Three is minimum, five is recommended for higher availability of quorum.
- Each worker node needs at least four CPUs and 8GB of RAM for Portworx + OpenShift, which means that each worker node will need slightly more than this to run application workloads.
- All Portworx worker nodes should be reachable from one another over local network.
- To provide backup and restore, it is common for enterprises to deploy a dedicated cluster used to host PX-Backup for central control and access to backup workflows. This is often a centralized Kubernetes or OpenShift cluster with connectivity to all clusters it will protect.

Below are specific requirements or configurations for AWS used in this validation.

Amazon Web Services

Red Hat OpenShift on AWS (ROSA) solution provides a managed deployment of OpenShift on AWS with billing for both resources and OpenShift licensing delivered through AWS. It has:

- Dedicated VPC per OpenShift Cluster
- Elastic Block Store (EBS) used for Portworx Cloud Drives storage pool provisioning
- Portworx IAM Policy allowing Portworx to control necessary AWS components
- NTP configuration must be the same across all hosts
- Installed using https://console.redhat.com/openshift/create/rosa/welcome for ROSA
- Worker nodes with M5.4xlarge 16vCPU, 64GB Ram to host Portworx, OpenShift and Application workloads

Portworx AWS IAM Policy Example:

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "pxnoderole",
            "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": [
                "*"
            ]
        }
    ]
}
```

The ROSA-CLI is used to deploy Red Hat OpenShift on AWS to the AWS environment. The installation configuration used for this validation was generated through a wizard-driven process provided by the ROSA-CLI. Responses to the prompts are provided below.

```
$ rosa create cluster -sts
? Cluster name: ck-rosa-aws01
? OpenShift version: 4.9.18
? External ID (optional):
? Operator roles prefix: ck-rosa-aws1-a4h5
? ? Multiple availability zones (optional): Yes
? AWS region: us-east-1
? PrivateLink cluster (optional): No
? Install into an existing VPC (optional): No
? Enable Customer Managed key (optional): No
? Compute nodes instance type (optional): m5.4xlarge
? Enable autoscaling (optional): No
? Min replicas: 3
? Max replicas: 3
? Machine CIDR: 10.123.0.0/16
? Service CIDR: 172.25.0.0/16
? Pod CIDR: 10.128.0.0/14
? Host prefix: 23
? Encrypt etcd data (optional): No
? Disable Workload monitoring (optional): No
```

Once the prompts are satisfied, the installer will proceed with deploying the cluster. The following is an example of the output from the ROSA installer. It is necessary to follow the final two steps listed at the end of this output block to be able to access the newly deployed ROSA cluster.

```
Name:                       ck-rosa-aws01
ID:                         1qc21p6ii2m502g8emc45d0nv9cmil4g
External ID:
OpenShift Version:
```

```
Channel Group:              stable
DNS:                        ck-rosa-aws01.h2fl.p1.openshiftapps.com
AWS Account:                803113055342
API URL:
Console URL:
Region:                     us-east-1
Multi-AZ:                   true
Nodes:
 - Control plane:           3
 - Infra:                   3
 - Compute:                 3
Network:
 - Service CIDR:            172.25.0.0/16
 - Machine CIDR:            10.123.0.0/16
 - Pod CIDR:                10.128.0.0/14
 - Host Prefix:             /23
STS Role ARN:               arn:aws:iam::803113055342:role/ManagedOpenShift-Installer-Role
Support Role ARN:           arn:aws:iam::803113055342:role/ManagedOpenShift-Support-Role
Instance IAM Roles:
 - Control plane:           arn:aws:iam::803113055342:role/ManagedOpenShift-ControlPlane-Role
 - Worker:                  arn:aws:iam::803113055342:role/ManagedOpenShift-Worker-Role
Operator IAM Roles:
 - arn:aws:iam::803113055342:role/ck-rosa-aws01-a4h5-openshift-ingress-operator-cloud-credentials
 - arn:aws:iam::803113055342:role/ck-rosa-aws01-a4h5-openshift-cluster-csi-drivers-ebs-cloud-crede
 - arn:aws:iam::803113055342:role/ck-rosa-aws01-a4h5-openshift-machine-api-aws-cloud-credentials
 - arn:aws:iam::803113055342:role/ck-rosa-aws01-a4h5-openshift-cloud-credential-operator-cloud-cre
 - arn:aws:iam::803113055342:role/ck-rosa-aws01-a4h5-openshift-image-registry-installer-cloud-cred
State:                      waiting (Waiting for OIDC configuration)
Private:                    No
Created:                    Feb 14 2022 15:01:59 UTC
Details Page:
https://console.redhat.com/openshift/details/s/256cLLwSKnqKPwclD1DZsoyVNWo
OIDC Endpoint URL:          https://rh-oidc.s3.us-east-
1.amazonaws.com/1qc21p6ii2m502g8emc45d0nv9cmil4g

I: Run the following commands to continue the cluster creation:

        rosa create operator-roles --cluster ck-rosa-aws01
        rosa create oidc-provider --cluster ck-rosa-aws01
```

Once provided the above information, proceed with issuing the two final commands, accepting the defaults. To access your ROSA cluster, you must first create a cluster-admin user. Issue the command 'rosa create admin –cluster=<cluster-name>' to create the admin account and obtain login credentials and instructions.

```
❯ rosa create admin --cluster=ck-rosa-aws01
W: It is recommended to add an identity provider to login to this cluster. See 'rosa create idp --
help' for more information.
I: Admin account has been added to cluster 'ck-rosa-aws01'.
I: Please securely store this generated password. If you lose this password you can delete and
```

```
recreate the cluster admin user.
I: To login, run the following command:

   oc login https://api.ck-rosa-aws01.h2fl.p1.openshiftapps.com:6443 --username cluster-admin --
password <Auto-Generated Complex Password>

I: It may take up to a minute for the account to become active.
```

Install Portworx Enterprise on Red Hat OpenShift

Portworx Enterprise is installed on OpenShift using the Portworx Operator. To install Portworx, first install the Portworx Operator from the OperatorHub within the OpenShift console.

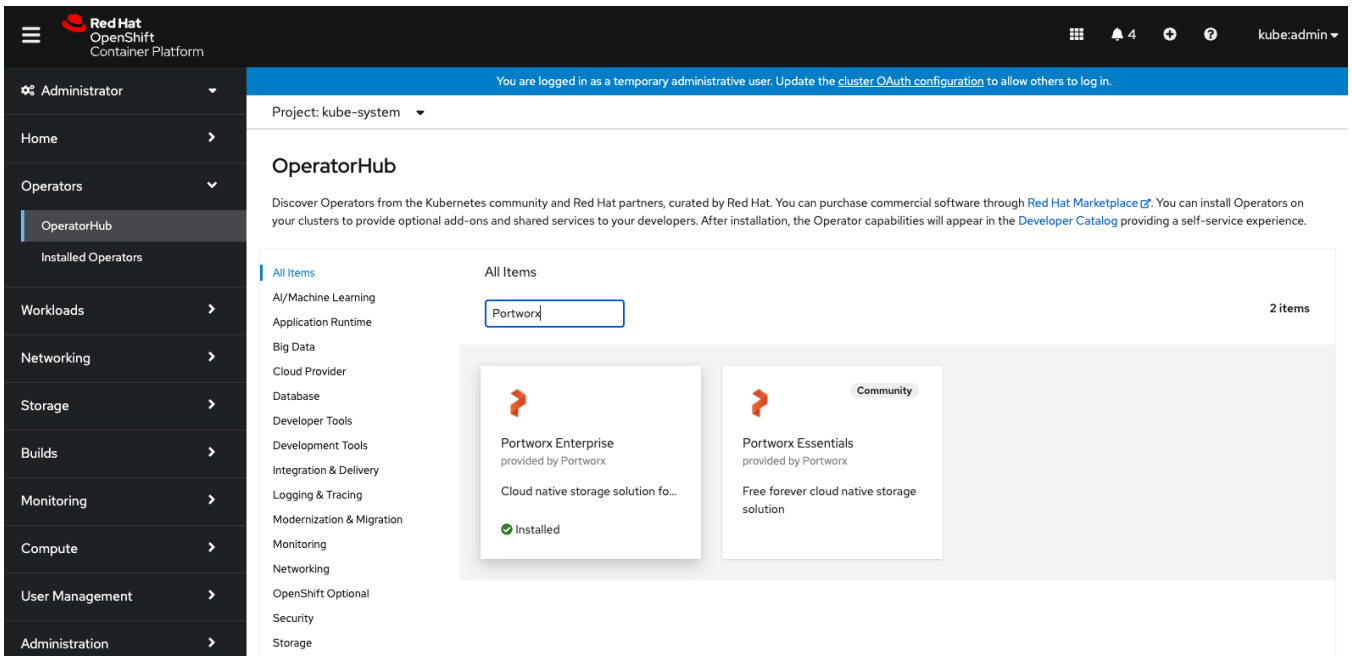


Figure 4: Installing the Portworx operator via the OperatorHub

Once the operator is deployed, the administrator can either use the form guided wizard in the OpenShift Console to create a StorageCluster. However, for a more streamlined configuration, navigate to https://central.portworx.com and choose “Portworx Enterprise” to start the process to produce a StorageCluster spec for the operator.

For AWS and ROSA

When configuring the Portworx StorageCluster for AWS to use EBS, select AWS in the “Storage” tab of the spec generation process. You can accept the defaults or modify the disk sizes and number of disks to use for each Portworx worker node. Available options include EBS volume types of GP2, GP3, or IO1. When selecting GP3 or IO1 you can also specify IOPS and Bandwidth settings.

When sizing the disks, it is recommended that you configure volumes with adequate capacity for any given workload to be deployed in the cluster. If an application requires 500GB of capacity, then you will want to configure more than 500GB per node using the configuration wizard. This could be a quantity of four 150GB EBS volumes, or one large 600GB volume.

Be sure to take time to plan your capacity needs. Additionally, it is recommended that you configure PX-Autopilot as shown later in this guide to protect applications from downtime related to filling the PVCs in use and the Portworx cluster as a whole.

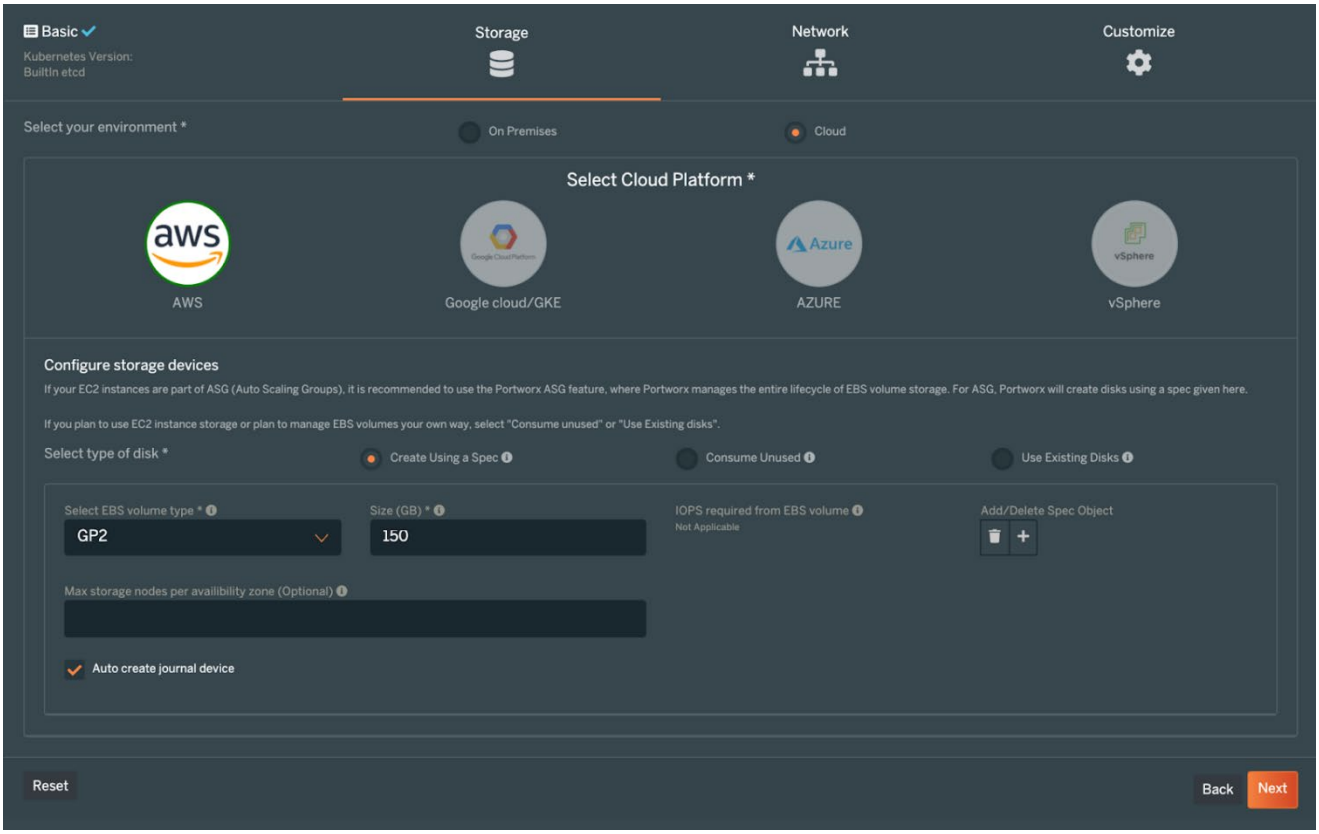


Figure 5: Configuring volumes

For any OpenShift environment, select **OpenShift 4+** on the **Customize** tab within the spec generator.

> **Note**: For PX-Security-enabled clusters select **Security Settings** and enable it. However, this guide will show how to enable security post-installation within a Portworx cluster on OpenShift.

> **Note**: For PX-Monitoring, select **Enable Monitoring** in the Advanced Settings drop down. This will be used in the monitoring validation use case within this document.

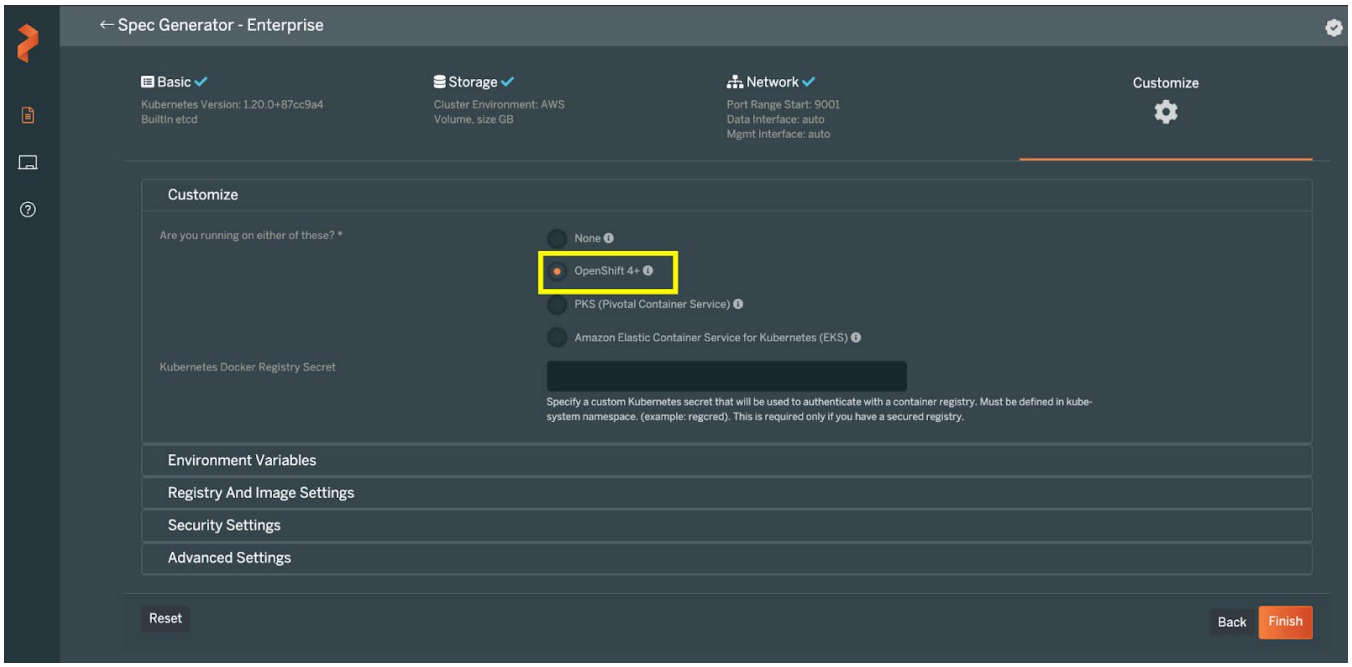


Figure 6: Selecting OpenShift.

Once you have completed the wizard at https://central.portworx.com, you will be presented with the option to copy a kubectl command that can be applied to deploy the StorageCluster. However, there is also another option to view the RAW manifest. We recommend viewing the RAW version and copy this into a YAML text file to modify it for use with ROSA. Below is an example manifest generated from Portworx Central. This will be the basis for the modifications described next.

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
  name: px-cluster-01bd27e1-fa3d-4324-8df3-3d20ee3a6a7a
  namespace: portworx
  annotations:
      portworx.io/is-openshift: "true"
spec:
  image: portworx/oci-monitor:2.8.0
  imagePullPolicy: Always
  kvdb:
    internal: true
  cloudStorage:
    deviceSpecs:
    - type=gp2,size=150
    kvdbDeviceSpec: type=gp2,size=150
  secretsProvider: k8s
  stork:
    enabled: true
    args:
      webhook-controller: "false"
  autopilot:
    enabled: true
```

```
    providers:
    - name: default
      type: prometheus
      params:
        url: http://prometheus:9090
  monitoring:
    telemetry:
      enabled: true
    prometheus:
      enabled: true
      exportMetrics: true
  featureGates:
    CSI: "true"
```

For Red Hat OpenShift on AWS (ROSA)

Modifications must be made to the storage cluster manifest when using Red Hat OpenShift on AWS to maintain compliance with the managed service agreement. It is necessary to make the following edits to the cluster spec file before applying it to create the Portworx Cluster.

1. Change the namespace Portworx is deployed to from the default ("kube-system") to a different namespace. In this validation process, we created and used a namespace called "Portworx".
2. Add an annotation to prevent the creation of Portworx related objects in the portworx namespace. The needed annotation is in bold below.

Before applying the manifests take the following steps in the AWS management console.

1. Log into your AWS Console, navigate to the IAM console and attach the Portworx IAM Policy to the Worker Node IAM Role.
3. Navigate to the VPC your ROSA Cluster is in and add 2 Inbound rules to the worker security group. They will be named based on your cluster name and include 'worker' in the security group name.
   a. Add a rule allowing inbound TCP traffic on ports 17001 - 17022 from your Machine CIDR, 10.123.0.0/16 in this example.
   b. Add a rule allowing inbound UDP traffic on port 17002 from your Machine CIDR, 10.123.0.0/16 for example.

Once these steps are completed, apply the StorageCluster configuration manifest in the OpenShift Portworx Operator Interface to create the Portworx cluster.

Here is an example of a storage Cluster specification for use with ROSA. The needed modifications are in bold along with the Portworx CloudDrive definition for AWS.

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
  name: px-cluster-01bd27e1-fa3d-4324-8df3-3d20ee3a6a7a
  namespace: portworx #ROSA requires using a different namespace
  annotations:
```

```
      portworx.io/is-openshift: "true"
      portworx.io/portworx-proxy: "false" #ROSA specific to prevent Proxies in kube-system
  spec:
    image: portworx/oci-monitor:2.9.1.1
    imagePullPolicy: Always
    kvdb:
      internal: true
    cloudStorage:
      provider: aws
      deviceSpecs:
      - type=gp2,size=150
      kvdbDeviceSpec: type=gp2,size=150
    secretsProvider: k8s
    stork:
      enabled: true
      args:
        webhook-controller: "false"
    autopilot:
      enabled: true
      providers:
      - name: default
        type: prometheus
        params:
          url: http://px-prometheus:9090
    monitoring:
      telemetry:
        enabled: true
      prometheus:
        enabled: true
        exportMetrics: true
    featureGates:
      CSI: "true"
```

When the StorageCluster is applied, Portworx should take a few minutes to completely become online. You can check the status of the Portworx StorageCluster within the installed operators table of your “portworx” namespace. Portworx should report “Phase: Online”.

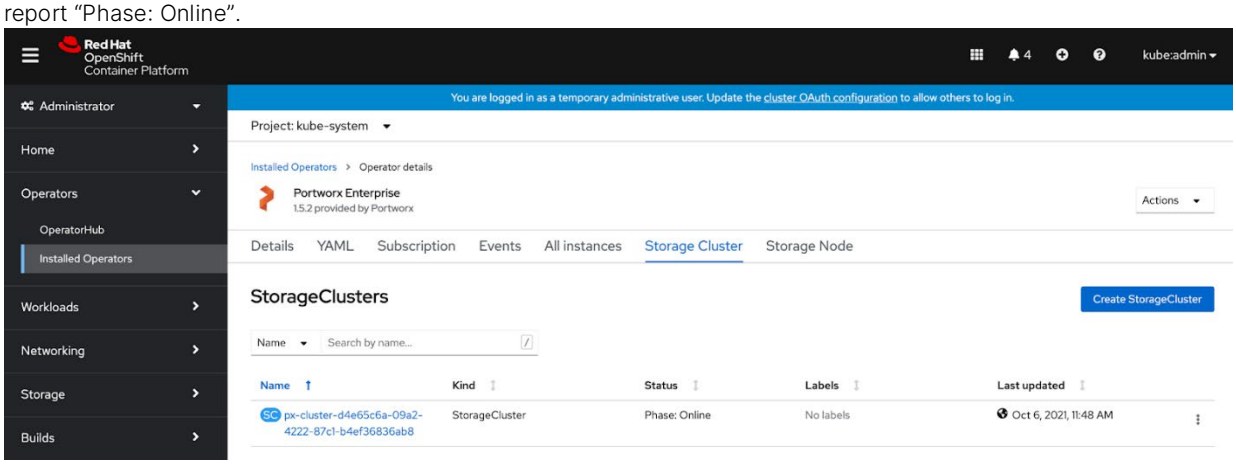


Figure 7: StorageCluster status showing as “Phase: Online”

We can monitor the installation status of Portworx by watching the pods in the portworx namespace and waiting until they are all ready by issuing the command `watch oc get pods -n portworx`:

```
Every 2.0s: oc get pods -n kube-system          tdarnell-ocp-adminws: Wed Oct 13 00:25:25 2021

NAME                                                READY   STATUS    RESTARTS   AGE
autopilot-6658db45c8-5pq9p                          1/1     Running   0          3m40s
portworx-api-2w2vw                                  1/1     Running   0          3m41s
portworx-api-4wbjl                                  1/1     Running   0          3m41s
portworx-api-777bl                                  1/1     Running   0          3m41s
portworx-kvdb-5vkc2                                 1/1     Running   0          39s
portworx-kvdb-7zx4m                                 1/1     Running   0          23s
portworx-kvdb-sd852                                 1/1     Running   0          13s
prometheus-px-prometheus-0                          3/3     Running   1          3m19s
px-csi-ext-86975b9df6-22ts9                         3/3     Running   0          3m41s
px-csi-ext-86975b9df6-k669s                         3/3     Running   0          3m41s
px-csi-ext-86975b9df6-mrcmw                         3/3     Running   0          3m41s
px-ocp-cluster-d5f9495e-54cb-4bdd-ba73-977f2f681fc9-2mbjk   3/3     Running   0          3m40s
px-ocp-cluster-d5f9495e-54cb-4bdd-ba73-977f2f681fc9-5j2m8   3/3     Running   0          3m40s
px-ocp-cluster-d5f9495e-54cb-4bdd-ba73-977f2f681fc9-xp4nr   3/3     Running   0          3m41s
px-prometheus-operator-cb5d646fc-q7n9z              1/1     Running   0          3m41s
stork-6bbc8fcc57-5467q                              1/1     Running   0          3m43s
stork-6bbc8fcc57-585hd                              1/1     Running   0          3m43s
stork-6bbc8fcc57-5w9zq                              1/1     Running   0          3m43s
stork-scheduler-6779686d96-7g698                    1/1     Running   0          3m43s
stork-scheduler-6779686d96-85wz5                    1/1     Running   0          3m43s
stork-scheduler-6779686d96-vm671                    1/1     Running   0          3m43s
```

We can also verify our Portworx cluster health by running `pxctl status` from within one of the OpenShift worker nodes where Portworx is installed. On AWS, your backing disks will be EBS and the type will match what was selected while creating the StorageCluster manifest.

```
Telemetry: Healthy
License: Trial (expires in 31 days)
Node ID: 0c9f9f01-e833-4835-97ea-3b4bba948aa4
        IP: 10.21.234.167
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE    REGION
        0       HIGH            raid0           97 GiB  5.9 GiB Online  default default
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/sdb2       STORAGE_MEDIUM_SSD      97 GiB          13 Oct 21 06:24 UTC
        total                   -                       97 GiB
        Cache Devices:
         * No cache devices
        Kvdb Device:
        Device Path     Size
        /dev/sdc        65 GiB
         * Internal kvdb on this node is using this dedicated kvdb device to store its data.
        Journal Device:
        1       /dev/sdb1       STORAGE_MEDIUM_SSD
Cluster Summary
        Cluster ID: px-ocp-cluster-d5f9495e-54cb-4bdd-ba73-977f2f681fc9
        Cluster UUID: 42976133-18bf-4292-ad2e-612dc6b3100b
        Scheduler: kubernetes
        Nodes: 3 node(s) with storage (3 online)
        IP              ID                                      SchedulerNodeName                               A
uth             StorageNode     Used    Capacity        Status  StorageStatus   Version         Kernel          O
S
        10.21.234.168   f18bbfca-df80-4241-bc3b-fd9bc7370361    tdarnell-ocp-worker-1.px-ocp-1.cluster.test     D
isabled Yes             5.9 GiB 97 GiB          Online  Up              2.8.0.0-1ef62f8 4.18.0-305.19.1.el8_4.x86
_64     Red Hat Enterprise Linux CoreOS 48.84.202109210859-0 (Ootpa)
        10.21.234.167   0c9f9f01-e833-4835-97ea-3b4bba948aa4    tdarnell-ocp-worker-0.px-ocp-1.cluster.test     D
isabled Yes             5.9 GiB 97 GiB          Online  Up (This node)  2.8.0.0-1ef62f8 4.18.0-305.19.1.el8_4.x86
_64     Red Hat Enterprise Linux CoreOS 48.84.202109210859-0 (Ootpa)
        10.21.234.169   031442b5-b0ea-455b-b87a-60e5baf9b9a7    tdarnell-ocp-worker-2.px-ocp-1.cluster.test     D
isabled Yes             5.9 GiB 97 GiB          Online  Up              2.8.0.0-1ef62f8 4.18.0-305.19.1.el8_4.x86
_64     Red Hat Enterprise Linux CoreOS 48.84.202109210859-0 (Ootpa)
Global Storage Pool
        Total Used      :  18 GiB
        Total Capacity  :  291 GiB
```

Monitoring Stateful Applications in Red Hat OpenShift

Using PX-Monitor to Monitor Stateful Applications on Red Hat OpenShift

The Red Hat OpenShift monitoring stack uses tools such as Prometheus, Alertmanager, Node Exporter, and Grafana, much like the Portworx monitoring stack with PX-Central and PX-Monitor. These industry standard tools allow for a deep level of monitoring capabilities along with the flexibility of configuration for Kubernetes environments.

To get started when you created the Portworx cluster spec on https://central.portworx.com. Make sure to select the Red Hat OpenShift box as well as "Enable Monitoring" under "Advanced Settings." This will make sure your Portworx cluster is set up with the Prometheus operator that enables PX-Monitor to connect.

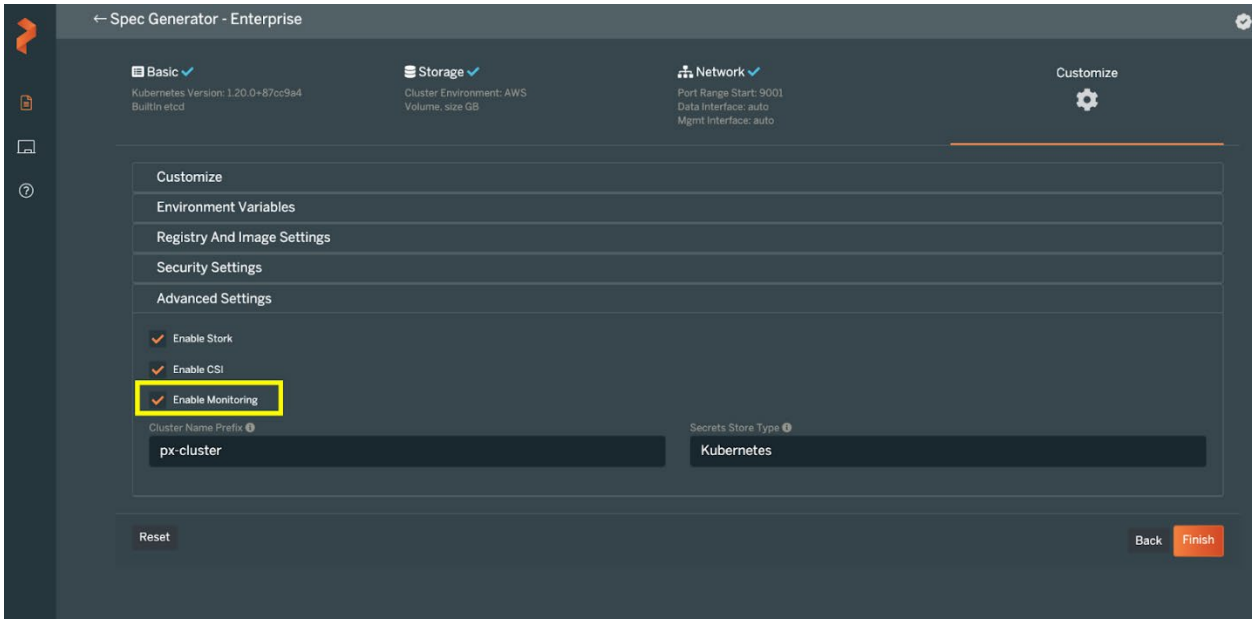


Figure 8: Enabling monitoring

You can verify that prometheus is installed with your Portworx installation by navigating to the portworx namespace and viewing the deployments. There should be a px-prometheus-operator installed.

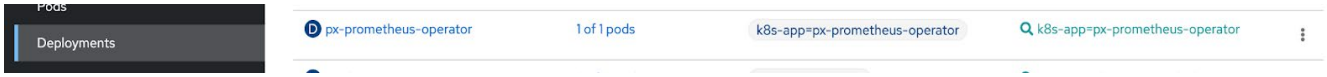


Figure 9: Checking px-prometheus-operator is installed

Once your Portworx cluster is installed, you will need to install two components: PX-Central UI and PX-Monitor.

PX-Central UI: You can install this on the same or different cluster by choosing "License Server and Monitoring" or "PX-Backup spec generation" from https://central.portworx.com. Both sets of instructions will enable the helm chart to install PX-Central.

First add the needed permissions to the central namespace.

```
$ oc adm policy add-scc-to-user restricted system:serviceaccount:central:default
$ oc adm policy add-scc-to-user restricted system:serviceaccount:central:pxcentral-apiserver
$ oc adm policy add-scc-to-user restricted system:serviceaccount:central:px-keycloak-account
$ oc adm policy add-scc-to-user restricted system:serviceaccount:central:px-backup-account
```

Then install PX-Central into your OpenShift cluster.

```
$ helm repo add portworx http://charts.portworx.io/ && helm repo update

$ helm install px-central portworx/px-central --namespace central --create-namespace --version 2.0.1
--set persistentStorage.enabled=true,persistentStorage.storageClassName="px-
replicated",pxbackup.enabled=true
```

PX-Monitor: This can be installed by navigating to https://central.portworx.com and selecting "License Server and Monitoring," then filling in the needed information. Make sure to select the **Monitoring on PX-Central** box.

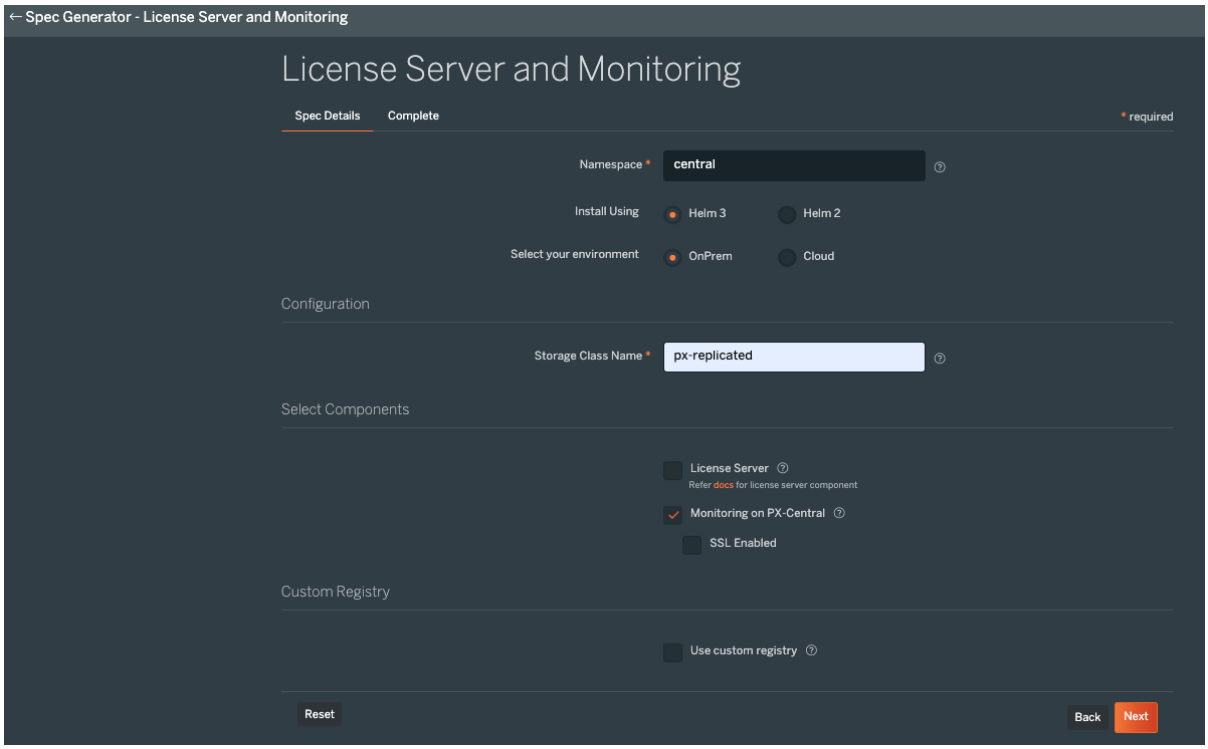


Figure 10: License Server and Monitoring spec details

Click **Next**. Make sure and provide the PX-Central UI installed in step one. This endpoint should be Ingress, Load Balancer, or IP:PORT. You can retrieve this service by using the following command:

```
$ oc get svc -n central px-central-ui
```

Once you provide the PX-Central UI Endpoint, follow the command prompts to update your central install to include PX-Monitor.

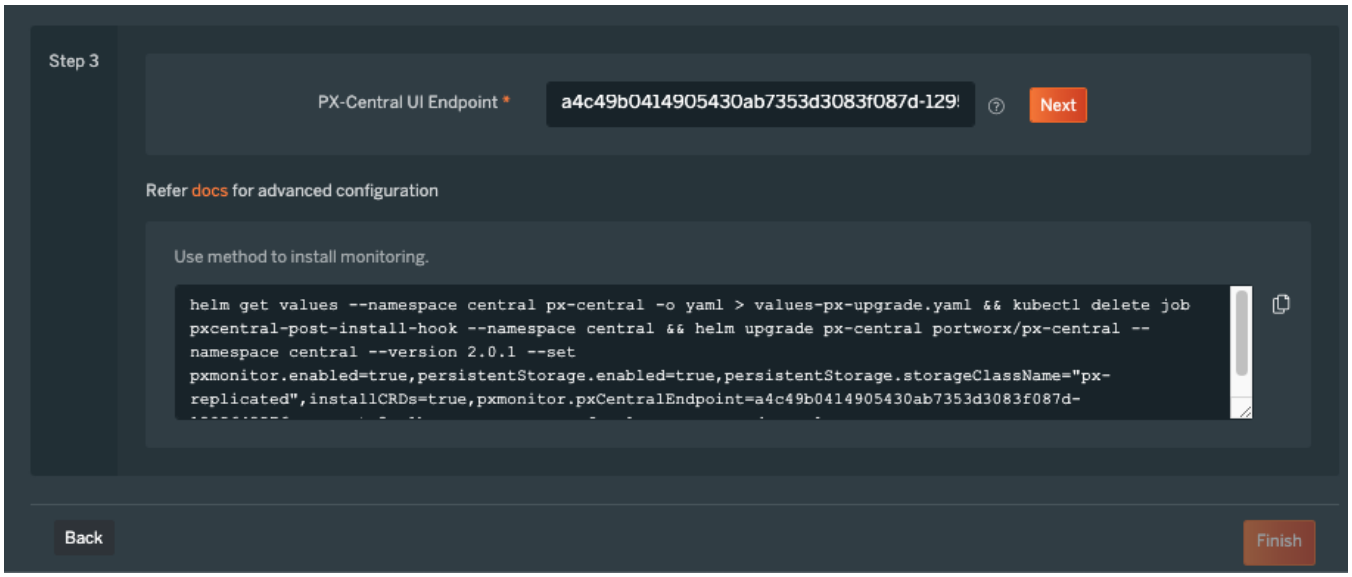


Figure 11: Updating the central install to include PX-Monitor

Once PX-Central and PX-Monitor are connected, head over to the Red Hat OpenShift dashboard and navigate to the **central** namespace where the **px-central-ui** service is available. This service will open the PX-Central interface for backup and monitoring.

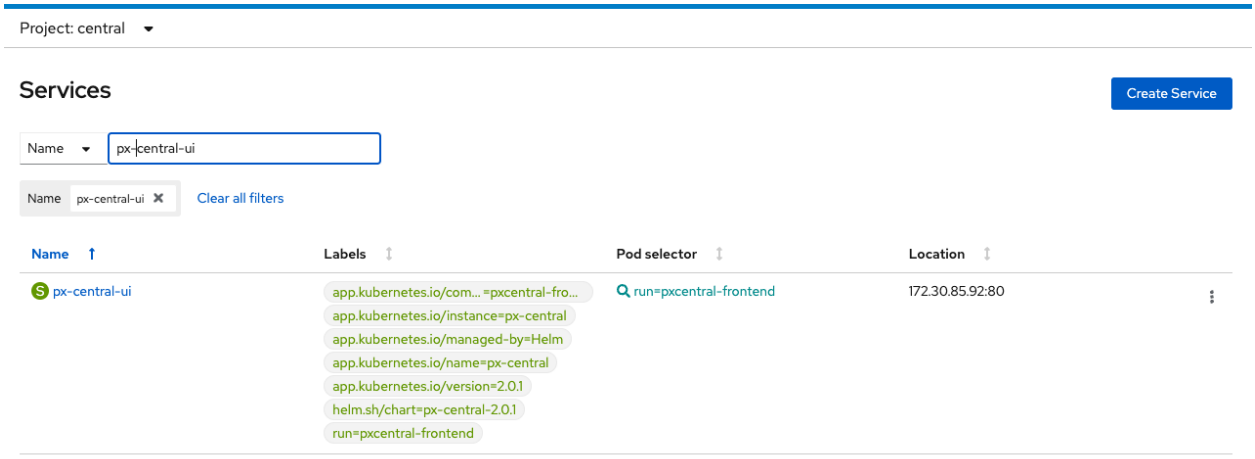


Figure 12: PX-Central interface

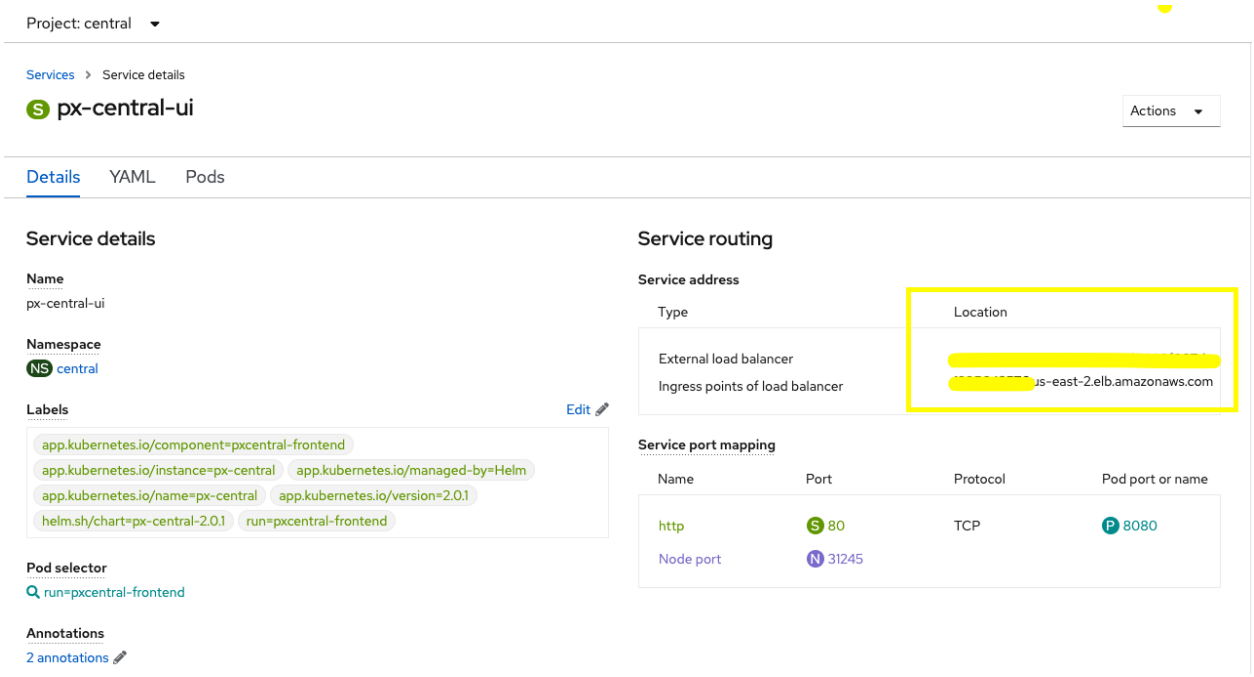


Figure 13: PX-Central service details

From here, log in with the admin credentials. Then, to connect your Portworx cluster to PX-Central click **Add PX Cluster**.

Figure 14: Adding a PX cluster

Then, fill out the cluster information with the StorageCluster name for the name and the portworx-service service as input.

```
oc get svc -n portworx portworx-service
NAME               TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                                AGE
portworx-service   ClusterIP   172.30.3.186   <none>        9001/TCP,9019/TCP,9020/TCP,9021/TCP    20h

oc get storagecluster -n portworx
NAME                                              CLUSTER
UUID                            STATUS   VERSION   AGE
px-cluster-01bd27e1-fa3d-4324-8df3-3d20ee3a6a7a   475ca5c3-c490-4123-a2be-
d9baa834583e   Online   2.8.0     20h
```

Provide the KubeConfig as well.

```
$ oc config view --flatten --minify
```

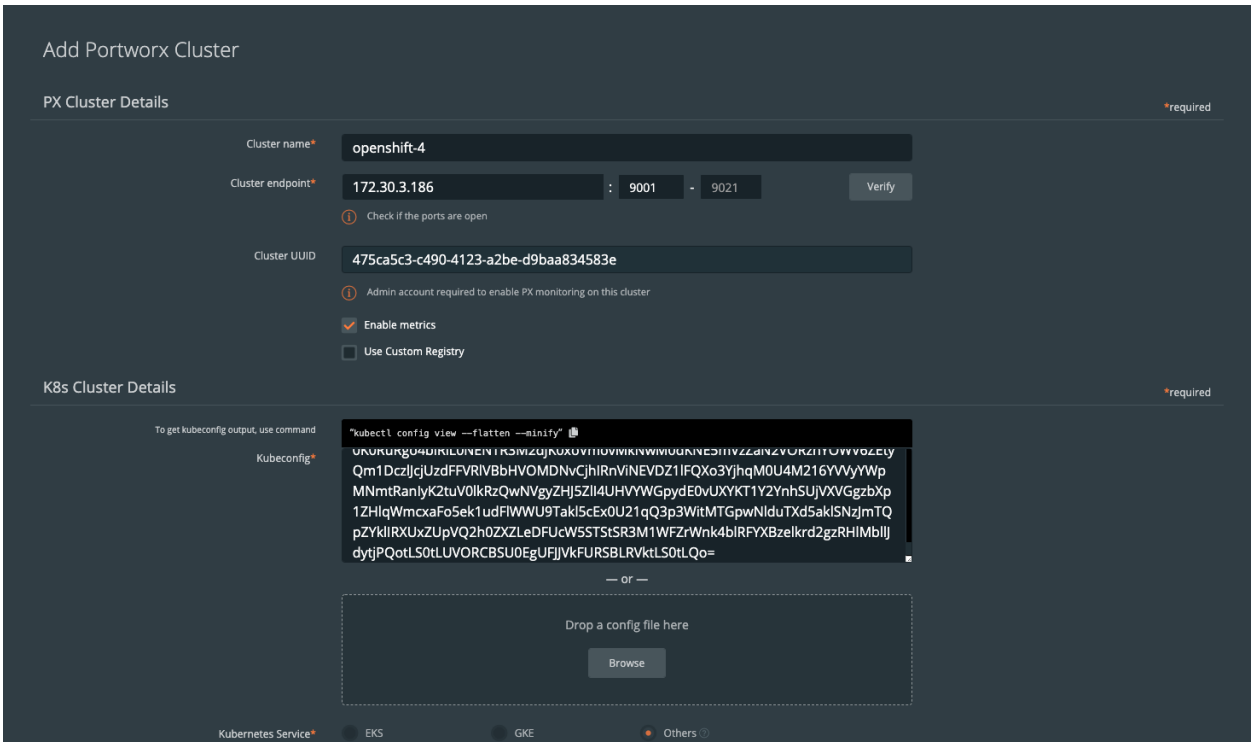


Figure 15: Adding PX cluster details

Once you provide this information, click **Submit.** From here, you can select "**Metrics**" to bring you the full Grafana monitoring dashboards.

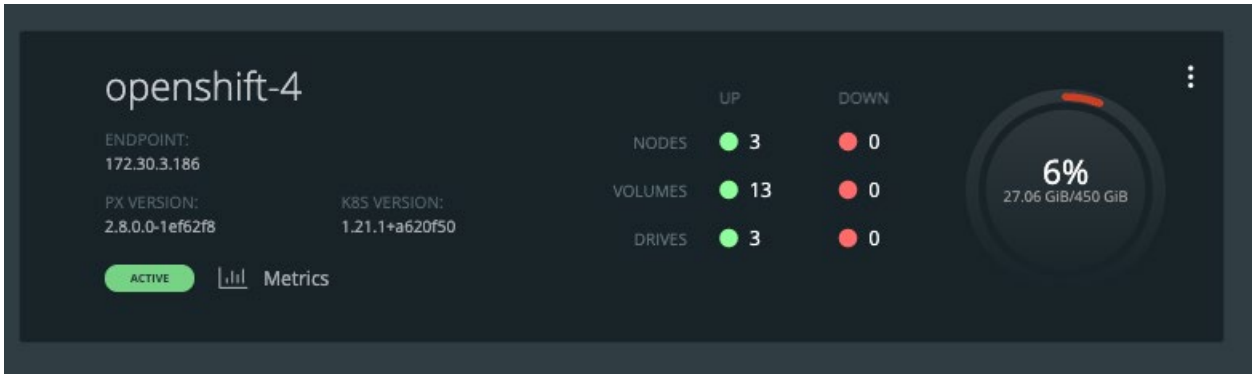


Figure 16: Grafana monitoring dashboard

From here, the Portworx cluster, nodes, backup, and volumes dashboards can allow you to monitor your data management components of the Red Hat OpenShift cluster.

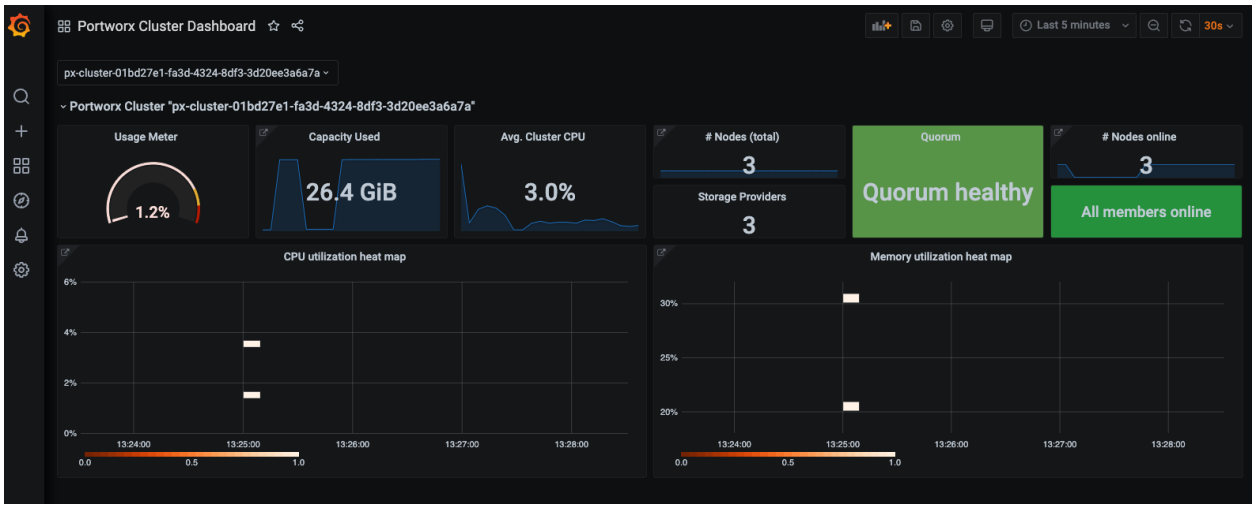


Figure 17: Monitoring data management components of the Red Hat OpenShift cluster.

Monitoring Postgres

To monitor a specific stateful application, navigate to your application, such as this Postgres pod seen below, which is using a StorageClass that uses the Portworx provisioner. From here, copy the PersistentVolumeClaim name that the database is using.

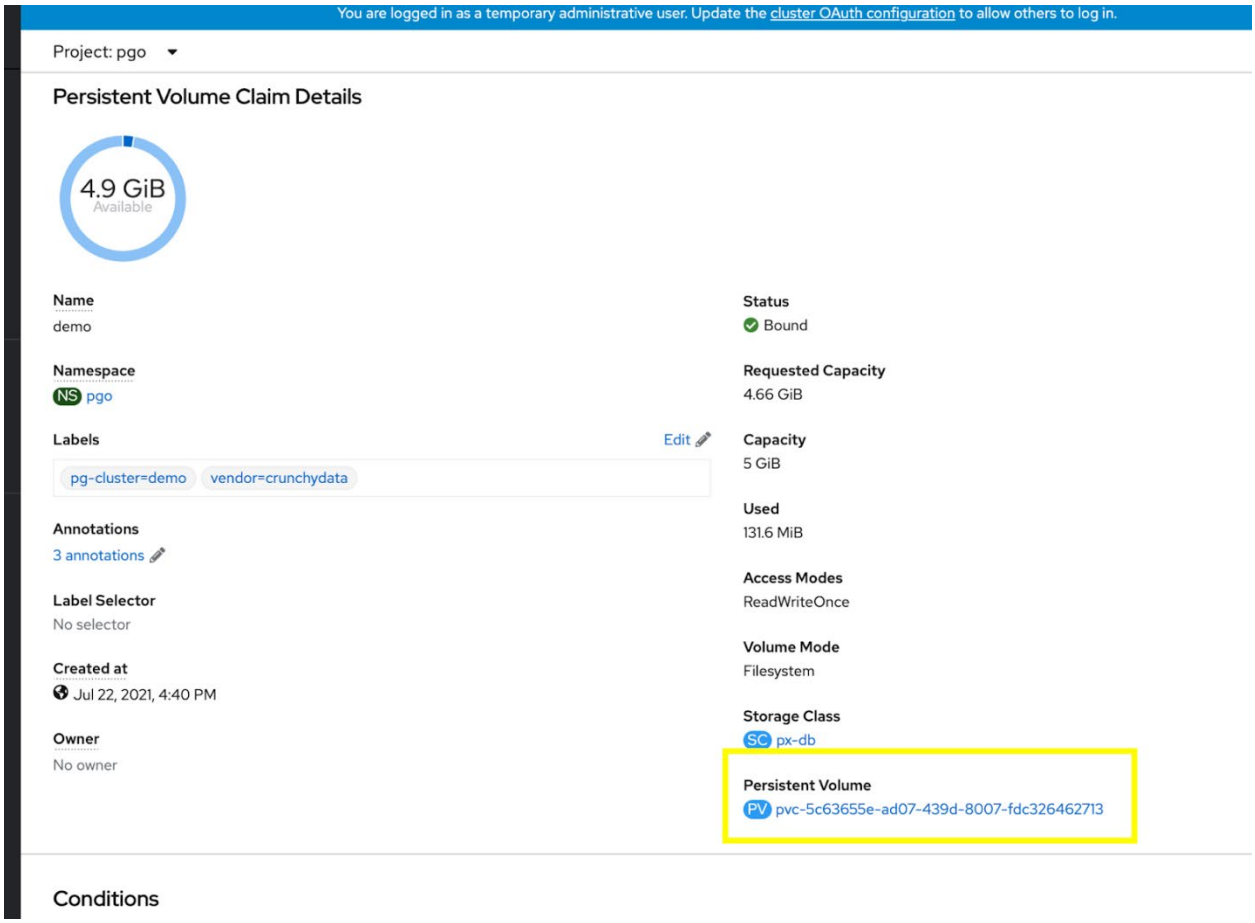


Figure 18: Monitoring a Postgres pod

Then navigate to the Portworx Volume Dashboard.

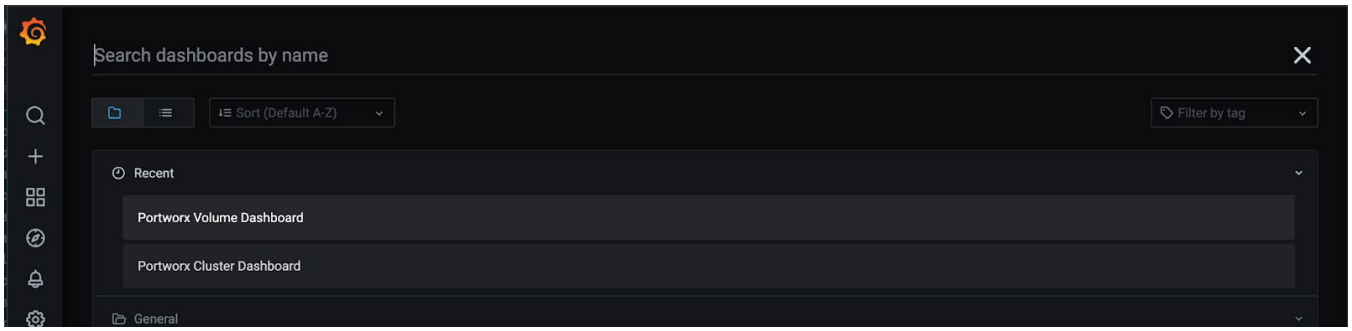


Figure 19: Navigating to the Portworx Volume dashboard.

Now you should be able to monitor Postgres volume metrics from the Grafana dashboard.

Figure 20: Monitoring Postgres volume metrics from the Grafana dashboard

PX-Monitor and PX-Central can simplify management, monitoring, and metadata services for one or more Portworx clusters on Red Hat OpenShift. Using this single pane of glass, enterprises can easily manage the state of their hybrid- and multi-cloud OpenShift applications with embedded monitoring and metrics directly in the Portworx user interface.

Automated Capacity Management on Red Hat OpenShift

Automated Portworx Storage Pool Expansion on AWS

When Portworx is deployed to Red Hat OpenShift on AWS, it is often configured with automated disk provisioning. This allows Kubernetes administrators to configure Portworx with automation of LUN creation and attachment for Portworx storage pools

available to OpenShift. This also allows administrators to provision only as much storage as they need, as PX-Autopilot will allow them to automatically scale when cluster usage rises.

As stated in the pre-work section, make sure you install the Portworx Enterprise operator prior to applying the StorageCluster spec above. Once the StorageCluster spec is applied, you will see the automatically provisioned 150GB cloud drives defined in the StorageCluster spec appear in your EC2 EBS Console page in AWS.

Volumes (36)

Filter volumes

Name	Volume ID	Type	Size	IOPS
PX-DO-NOT-D...	vol-007fee8fe2ec938b1	gp2	150 GiB	450
PX-DO-NOT-D...	vol-0cecc5eadfe9a78fe	gp2	150 GiB	450
PX-DO-NOT-D...	vol-0c80d7b90e14ce8d8	gp2	150 GiB	450
PX-DO-NOT-D...	vol-0d91bcf4225cc8051	gp2	150 GiB	450
PX-DO-NOT-D...	vol-0b2b824426ec4a837	gp2	150 GiB	450

Figure 21: Automatically provisioned 150 GB cloud drives

Portworx should also become healthy within the OpenShift cluster during this time. You may choose to check the status of Portworx by running `pxctl status` from within one of the OpenShift worker nodes where Portworx is installed.

Before continuing to the next step, it is advisable to check to make sure AutoPilot is running. To do so, run the following command. Autopilot should show Status as "Running" and Ready as "1/1."

```
# oc get po -n portworx -l name=autopilot
NAME                          READY   STATUS    RESTARTS   AGE
autopilot-6658db45c8-httfk    1/1     Running   0          18h
```

Autopilot Pool Expansion

For PX-Autopilot to be able to expand the backend storage pool once its usage crosses a threshold condition, we need to set up an Autopilot Rule first.

NOTE: PX-Autopilot pool expansion is a PX-Enterprise support feature only and is not available within PX-Essentials.

The below autopilot rule can be applied to the OpenShift cluster using `oc apply -f rule.yaml`. The rule below states the following conditions and actions:

- **Condition**: If the pool capacity on any given Portworx node is above 50%
- **Condition**: Pools on any given Portworx node should not exceed 1TB in size.

- **Action**: Scale the pool by 50% as long the pool will remain at or below 1TB. Scale by adding a disk.

```
# cat rule.yaml
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: pool-expand
spec:
  enforcement: required
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    expressions:
    # pool available capacity less than 50%
    - key: "100 * ( px_pool_stats_available_bytes/ px_pool_stats_total_bytes)"
      operator: Lt
      values:
        - "50"
    # pool total capacity should not exceed 1TB
    - key: "px_pool_stats_total_bytes/(1024*1024*1024)"
      operator: Lt
      values:
       - "1000"
  ##### action to perform when condition is true
  actions:
    - name: "openstorage.io.action.storagepool/expand"
      params:
        # resize pool by scalepercentage of current size
        scalepercentage: "50"
        # when scaling, add disks to the pool
        scaletype: "add-disk"
```

After applying the autopilot rule, you may look at the OpenShift events and search for the **AutopilotRule** object with the name **pool-expand**. This will show each of the pools being initialized to normal, as they are all within the threshold of 50%. Note that we see three events because there are three storage nodes within one storage pool each in this cluster.

```
oc get events --field-selector involvedObject.kind=AutopilotRule,involvedObject.name=pool-expand --
all-namespaces --sort-by .lastTimestamp

Every 2.0s: oc get events --field-selector
involvedObject.kind=AutopilotRule,involvedObject.name=pool-expand --all-namespaces --sort-by
.lastTimestamp                                              Thu Sep  9 11:07:23 2021

NAMESPACE   LAST SEEN   TYPE     REASON        OBJECT                       MESSAGE
default     7m51s       Normal   Transition   autopilotrule/pool-expand   rule: pool-
expand:a1b09e28-f06b-4b56-bea5-064a9b20aa97 transition from Initializing => Normal
default     7m51s       Normal   Transition   autopilotrule/pool-expand   rule: pool-
expand:afa5aec4-bcba-457d-8be0-6e2d9d2af9cb transition from Initializing => Normal
default     7m51s       Normal   Transition   autopilotrule/pool-expand   rule: pool-
expand:b6da8879-2956-4cf5-af0b-11cc009fe95e transition from Initializing => Normal
```

As the pool begins to fill due to usage, the events will show the state change from “Normal” to “Triggered.” This indicates that Autopilot has detected a rule condition within the inference engine. Once a pool is triggered, it will be placed into ActiveActionPending, then ActiveActionInProgress. During this time, Portworx will make sure to only expand a single storage pool and rebalance the storage pools one at a time so the cluster remains healthy and responsive.

Figure 22: Rebalancing storage pools

For a given Portworx node, you should see an additional disk (three instead of two) added to the host, indicating that Autopilot performed the expand operation by adding a disk.

To check the progress of the pool expansion, you can look at the nodes’ Portworx logs and make note of the “Expansion is already in progress for pool” log entries and the percentage that the expansion has left to rebalance.

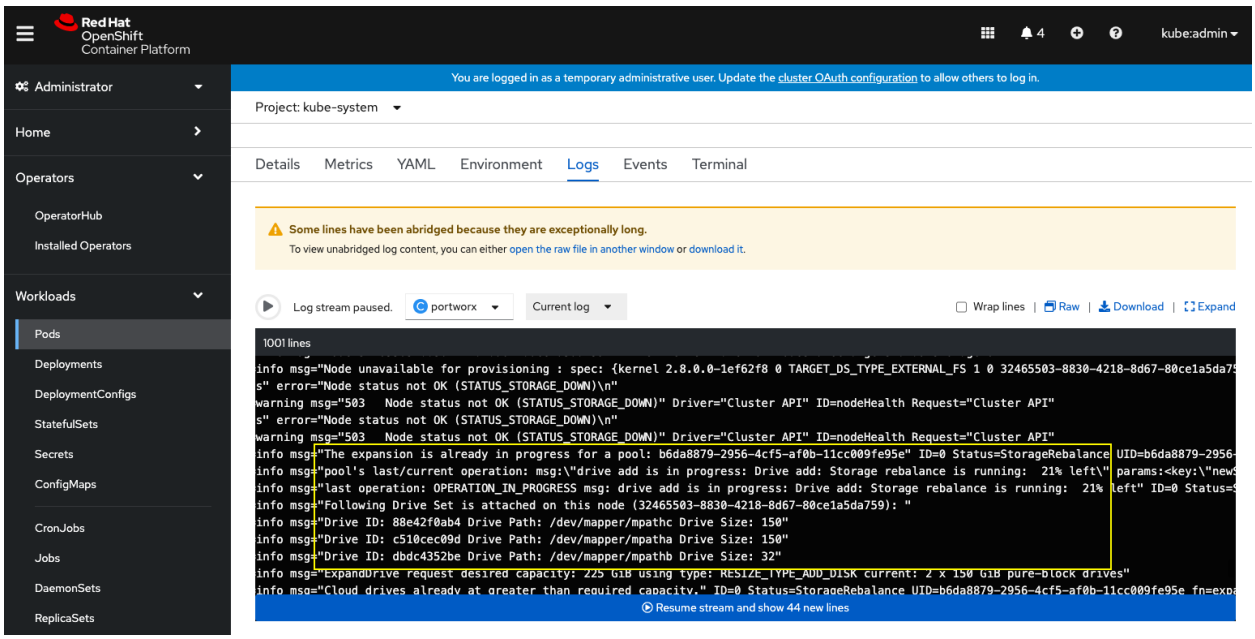


Figure 23: Checking the log for Expansion is already in progress for pool

Once this operation is complete, you may use the `pxctl service pool show` command to see the expand operation has occurred and that there is an additional disk based on the AutopilotRule.

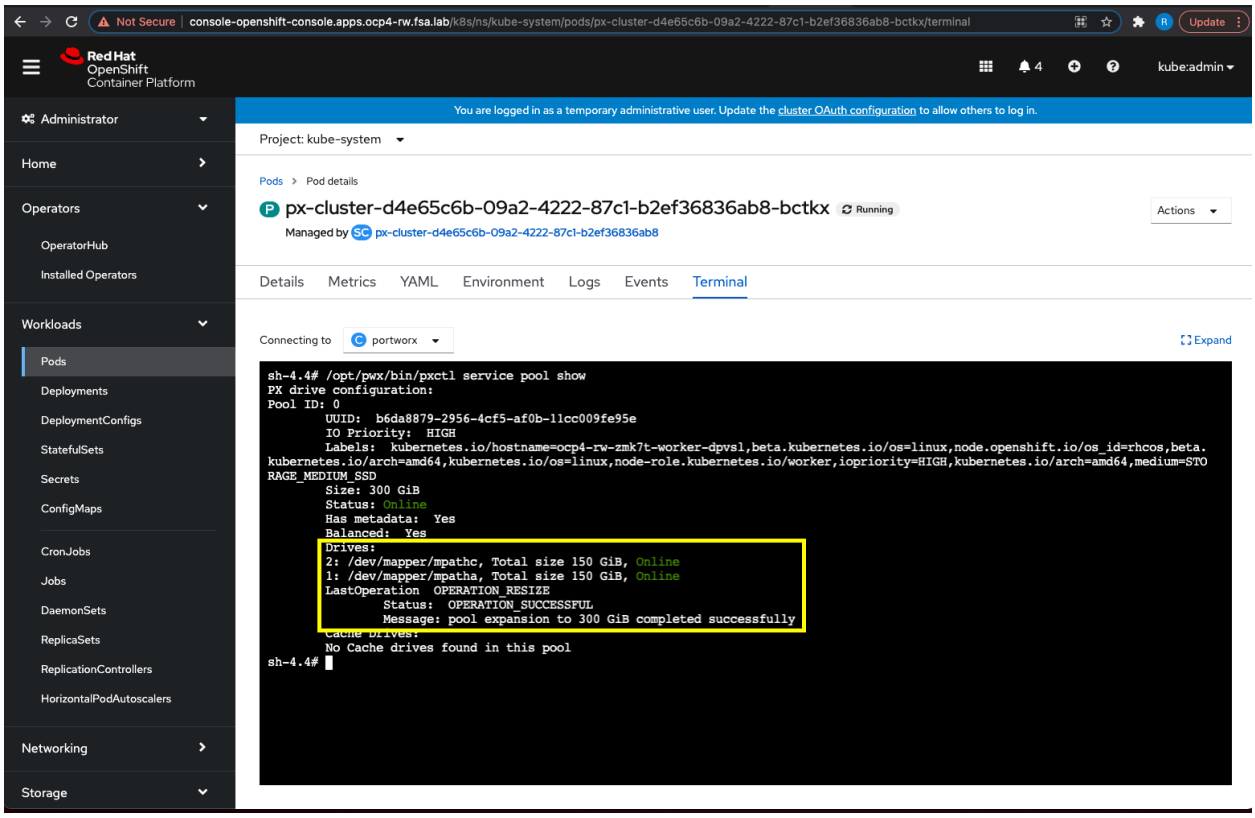


Figure 24: Using the pxctl service pool show command to see the expand operation has occurred

The events will also show the triggered condition as **ActiveActionsTaken** to indicate the "expand" operation is complete and that the action has been taken.

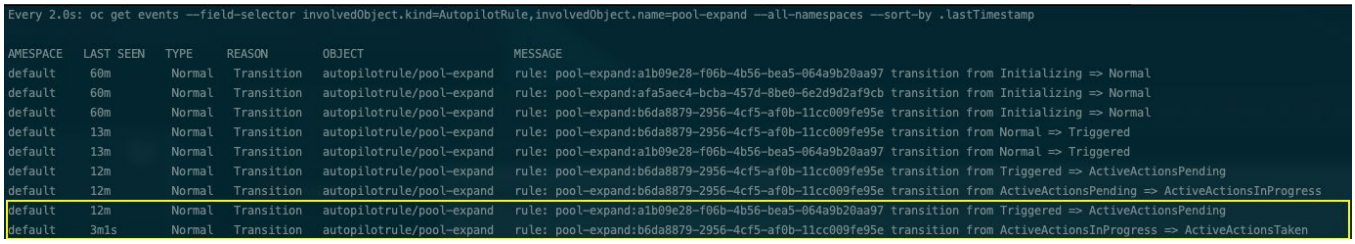


Figure 25: ActiveActionsTaken

Once this is done, Autopilot will resume watching for the condition to be true for the pools and their new sizes. Autopilot rules continue to work, even after they have been triggered, as long as the action will not meet the maximum pool size limit. If the maximum size limit of the pool has been met, the action will increase it to only meet this maximum size.

Automated PVC Expansion for OpenShift applications

PX-Autopilot can also be used to dynamically expand PersistentVolumeClaims on demand without any application downtime as well. The workflow is similar to setting up a storage pool rule; however, this rule will be targeting a specific PVC instead. To get started we need an application to work with.

First, create a namespace that will be used to host the application. In this case, we apply a label to the namespace because PX-Autopilot can target specific namespace labels for PVC rules.

```
oc create ns pg1
oc label ns pg1 type=db
```

To create an autopilot rule, define the AutoPilotRule YAML object and apply it to the OpenShift cluster. This rule targets the Postgres app in the “db” labeled namespaces and will resize PVCs when they reach 70% capacity.

```
# cat posgtres-autopilot-rule.yaml
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
 name: volume-resize
spec:
  ##### selector filters the objects affected by this rule given labels
  selector:
    matchLabels:
      app: postgres
namespaceSelector:
    matchLabels:
      type: db
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    # PVC usage should be less than 70% (30% remaining)
    expressions:
    - key: "100 * (px_volume_usage_bytes / px_volume_capacity_bytes)"
      operator: Gt
      values:
        - "70"
  ##### action to perform when condition is true
  actions:
  - name: openstorage.io.action.volume/resize
    params:
      # resize volume by scalepercentage of current size
      scalepercentage: "100"
      # volume capacity should not exceed 400GiB
      maxsize: "400Gi"

oc create -f postgres-autopilot-rule.yaml
```

A sample Postgres application can be used below. Note that the StorageClass used for the applications storage must set `allowVolumeExpansion: true` in order for expansion to occur.

```
# cat pgbench.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: postgres-pgbench-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "2"
allowVolumeExpansion: true
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pgbench-data
  labels:
    app: postgres
spec:
  storageClassName: postgres-pgbench-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pgbench-state
spec:
  storageClassName: postgres-pgbench-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pgbench
  labels:
    app: postgres
spec:
  selector:
    matchLabels:
      app: postgres
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```

```
    replicas: 1
    template:
      metadata:
        labels:
          app: postgres
      spec:
        securityContext:
          fsGroup: 1000649999
          runAsUser: 1000649999
        schedulerName: stork
        containers:
          - image: postgres:13.3
            name: postgres
            ports:
            - containerPort: 5432
            env:
            - name: POSTGRES_USER
              value: pgbench
            - name: POSTGRES_PASSWORD
              value: superpostgres
            - name: PGBENCH_PASSWORD
              value: superpostgres
            - name: PGDATA
              value: /var/lib/postgresql/data/pgdata
            securityContext:
              fsGroup: 1000649999
              runAsUser: 1000649999
            volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: pgbenchdb
          - name: pgbench
            image: portworx/torpedo-pgbench:latest
            imagePullPolicy: "Always"
            env:
              - name: PG_HOST
                value: 127.0.0.1
              - name: PG_USER
                value: pgbench
              - name: SIZE
                value: "70"
            securityContext:
              fsGroup: 1000649999
              runAsUser: 1000649999
            volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: pgbenchdb
            - mountPath: /pgbench
              name: pgbenchstate
        volumes:
        - name: pgbenchdb
          persistentVolumeClaim:
            claimName: pgbench-data
```

```
        - name: pgbenchstate
          persistentVolumeClaim:
            claimName: pgbench-state

oc create -f pgbench.yaml -n pg1
```

After applying the Postgres application to the cluster, verify app is creating data by navigating to the pg1 project in the OpenShift console and clicking on the “pgbench” pod and viewing the logs. You should see pgbench running in the background to fill up the PVC disk space.

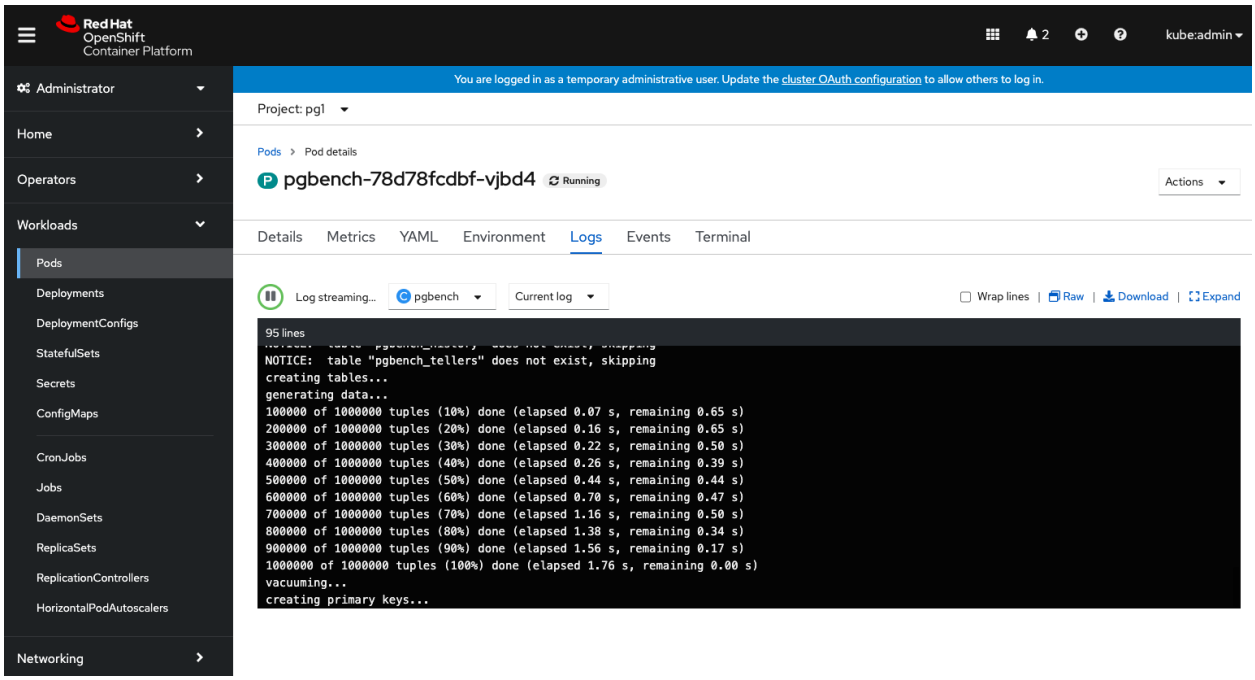


Figure 26: Verifying that Postgres is creating data

You may watch the PVCs within the OpenShift console to view their sizes increase as data is added.

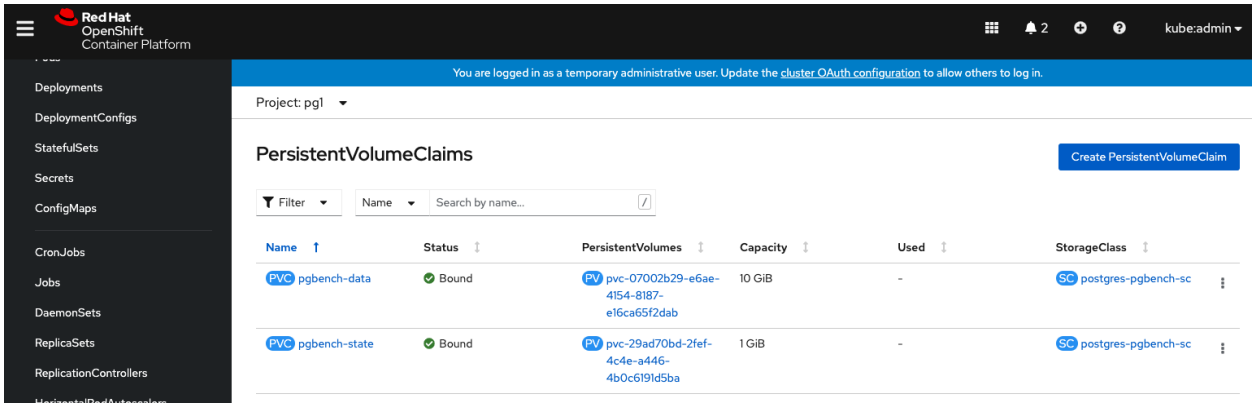


Figure 27: Viewing the PVCs

Next, watch for autopilot rule events by filtering by the specific volume-resize AutoPilotRule.

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

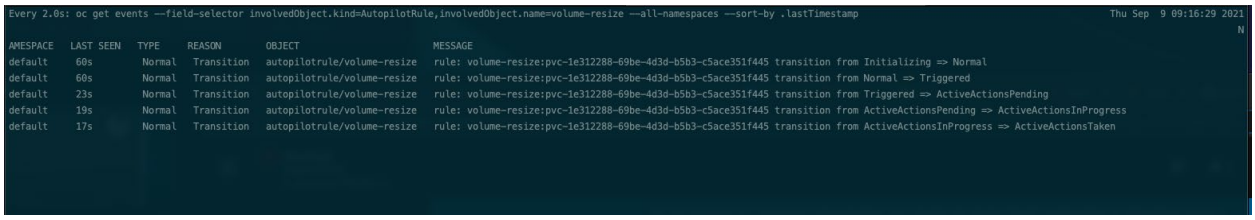


Figure 28: Filtering by the specific volume-resize AutoPilotRule

Verify the PVC expands from 10G to 20G when first triggered. Now your PVC has double the capacity without the need for editing YAML or for a storage admin to take manual action.

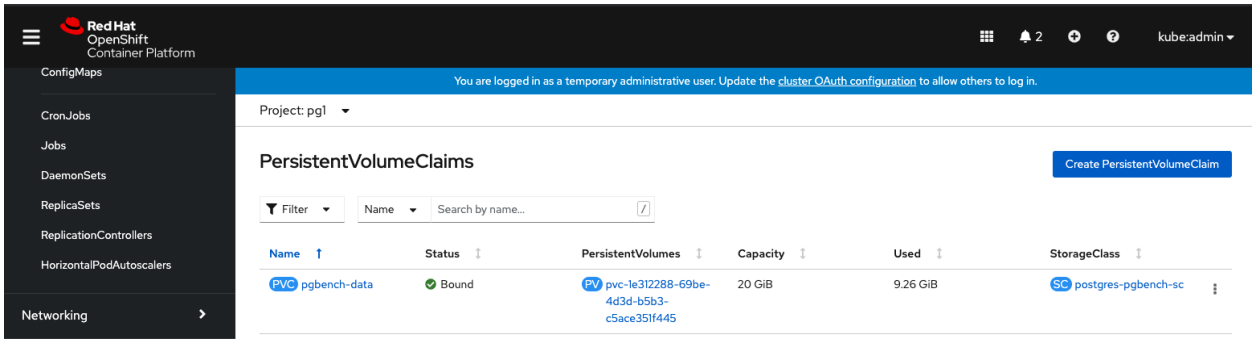


Figure 29: Checking PVC capacity

Secure Backup and Restore for Red Hat OpenShift

Portworx PX-Backup provides a modern Kubernetes-native backup and restore solution for OpenShift clusters. When it comes to modern applications, traditional backup solutions won't work for the following reasons:

- **Traditional backup is machine-focused**: Traditional backup solutions talk to the underlying machines (bare metal hosts or virtual machines) and protect them as the primary unit. But they don't consider the applications running on top. Containerized applications are distributed in nature; each machine might have containers that might belong to different applications running on top, and each application might have containers that are spread across multiple machines. If you are just protecting underlying machines without understanding how modern applications are deployed and run in production, you might not be able to restore your applications as expected when needed.
- **Traditional backup doesn't speak Kubernetes**: Traditional backup solutions are more focused on connecting directly with the physical servers or connecting to virtualization managers like vCenter server and inventorying all the different virtual machines running on top of it. A production Kubernetes cluster consists of multiple control plane nodes and multiple worker nodes that are responsible for running your application using constructs like Kubernetes pods, deployments, services, configmaps, etc. If your backup solution is not able to understand and identify these constructs, you might not be able to restore your applications.
- **Traditional backup is centrally managed**: Traditional backup solutions don't have self-service or role-based access control built in. They are more focused on enabling the backup administrator or infrastructure administrator to create

backup schedules and jobs and ensure that all the jobs are completed successfully. With modern applications, you need a more distributed approach, where the backup administrator will add the backup locations and create backup schedules. But individual application owners or developers might best know how to protect a particular application and would prefer self-service access and control over how their application is protected.

PX-Backup provides a modern data protection solution for Red Hat OpenShift that meets these specifications:

- **Container-granular**: PX-Backup runs on top of an OpenShift cluster. It can run on the same OpenShift cluster as your applications, or it can run on a dedicated OpenShift cluster. It helps you protect all the containers that are part of your application, not just the hosts that are running those containers.
- **Kubernetes namespace aware**: PX-Backup talks to the Kubernetes API server, and it can identify all the namespaces configured inside the Red Hat OpenShift clusters. It also identifies all the different Kubernetes objects from pods, deployments, services, config maps, secrets, etc. and helps you backup everything that constitutes your containerized application.
- **Application consistent**: Containerized stateful applications are distributed in nature, so it is essential to have a backup solution that can help take application consistent snapshot, and not just crash consistent snapshots. PX-Backup allows administrators to create pre- and post-backup rules that can be associated with backup jobs for your distributed applications.
- **Capable of backing up data and app config**: PX-Backup allows you to back up your entire application end to end. This includes all the Kubernetes objects, application configurations, and persistent volumes that store your application data.
- **Optimized for the multi-cloud world**: PX-Backup works with all Kubernetes distributions, so you can run your applications in Red Hat OpenShift clusters on-prem and restore them to an Amazon EKS or a Google Kubernetes Engine cluster or another Red Hat OpenShift cluster running in AWS or Azure.

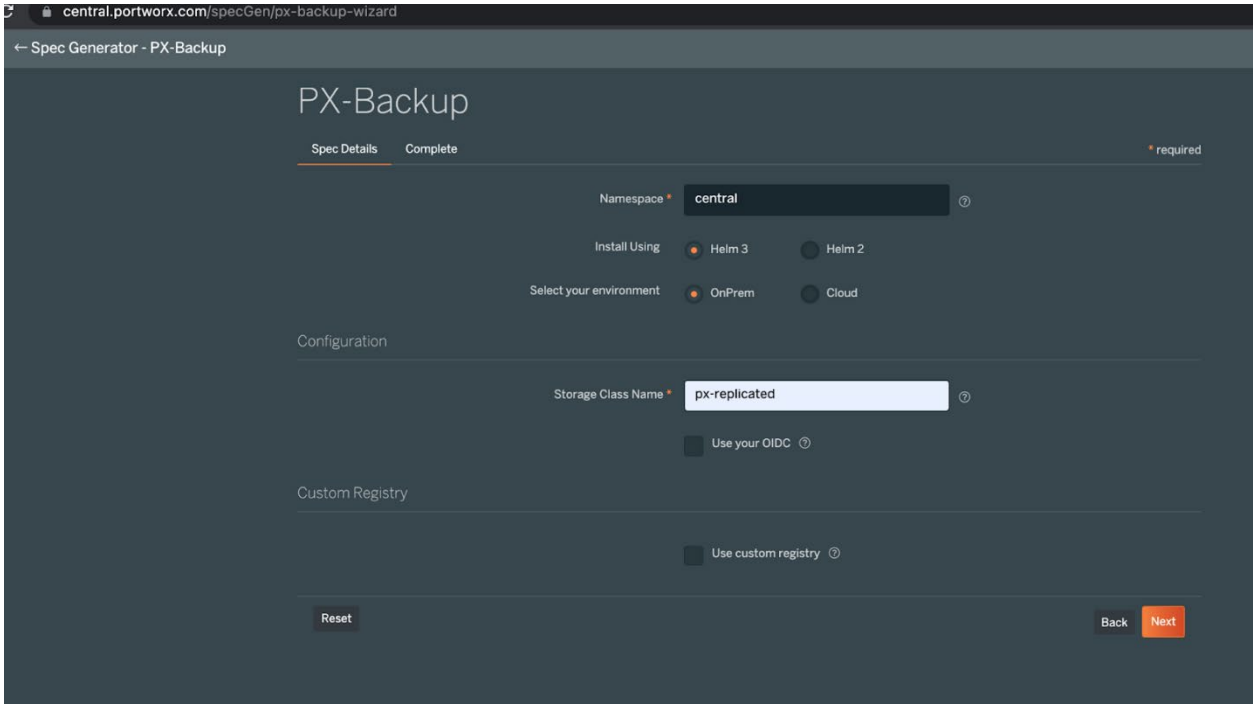
Deployment and Validation

To install PX-Backup on Red Hat OpenShift, use the following steps:

> Note: PX-backup is often installed in a separate “management” cluster to provide central access for backups on one or more clusters. You may install in an existing OpenShift cluster; however, we recommend installing in a central location for multi-user access.

1. Install Helm CLI wherever you have access to the oc command line for Openshift.
2. Navigate to PX-Central and create a New Spec. Select **PX-Backup** and click **Next**.

3. Enter a namespace that you want to install all the PX-Backup components. Select Helm3 and Cloud. Enter the name of the StorageClass that you want to use to install PX-Backup. In this example we installed PX-Backup on an OpenShift cluster which had Portworx installed, so the "px-replicated" StorageClass was available. Click **Next**.



4. Read through the License Agreement and click **Agree**.

5. Follow the two-step process to install PX-Backup on your Red Hat OpenShift cluster:

```
helm repo add portworx http://charts.portworx.io/ && helm repo update
helm install px-central portworx/px-central --namespace px-backup --create-namespace --version 2.0.1
--set persistentStorage.enabled=true ,persistentStorage.storageClassName="px-
replicated",pxbackup.enabled=true
```

6. You can monitor the PX-Backup deployment using the following commands:

```
kubectl get pods -n px-backup -w
kubectl get po --namespace px-backup -ljob-name=pxcentral-post-install-hook  -o wide | awk '{print
$1, $3}' | grep -iv error
kubectl get svc -n px-backup
```

7. To access the Backup UI from an OpenShift Route:

   a. Open the web console, go to **Networking** > **Routes**, and then click the **Create Route** button. On the Create Route page, configure your route by populating the following fields:

      i. Name: Enter a descriptive name

      ii. Hostname: Specify a public hostname. If you leave this field empty, OpenShift will generate a hostname.

      iii. Path: Leave this field unchanged.

iv. Service: Choose px-backup-up from the drop-down list.

v. Target Port: Choose 80 -> 8080

b. When you've finished configuring your route, select the **Create** button.

8. OpenShift now displays a link to the PX-Backup UI on the Routes page. To access PX-Backup, select that link.

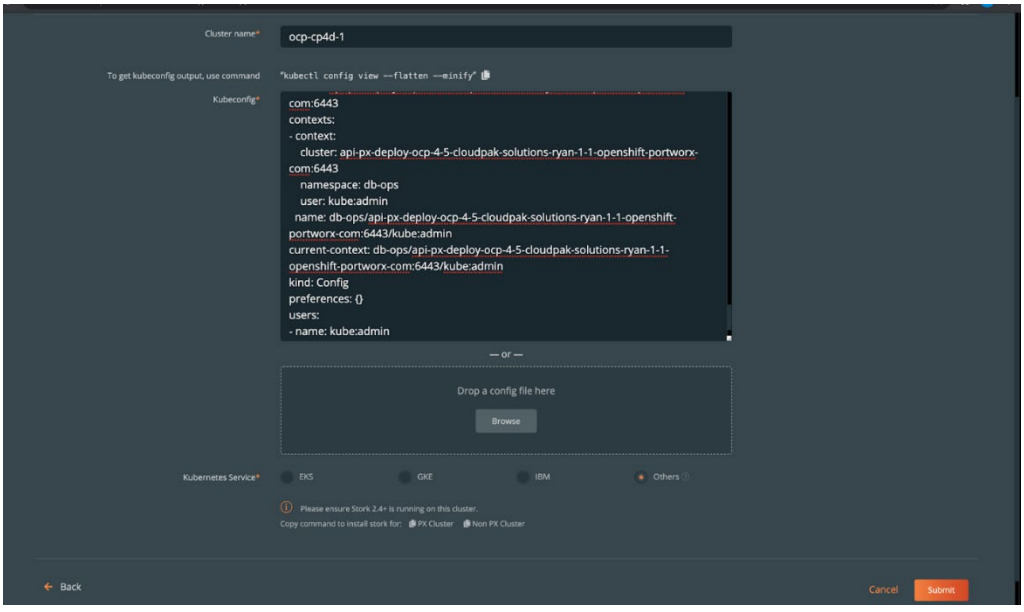
9. Log into the PX-Backup interface using the default credentials (admin/admin). You will be prompted to set a new password on your first login.

10. Once you log in, you can configure cloud accounts, backup locations, schedule policies and backup rules.

- **Cloud accounts**: These credentials allow PX-Backup to authenticate with clusters for the purpose of taking backups and restoring to them. They also add and manage backup locations where backup objects are stored.
- **Backup locations**: PX-Backup supports AWS S3, Azure Blob Storage, Google Cloud Object Storage and any S3 compliant object store as the backup repository to store your backup objects.
- **Schedule policies**: PX-Backup allows administrators to create periodic, hourly, daily, weekly, and monthly schedule policies that will be leveraged by the application owners to create their backup jobs.
- **Backup rules**: To ensure application consistency, PX-Backup allows administrators to create pre- and post-backup rules for their applications. Stateful and distributed applications like Cassandra, Elasticsearch, MongoDB, MySQL, PostgreSQL, and others need these backup rules to take application consistent snapshots.

11. You can add your Red Hat OpenShift clusters using the PX-Backup UI. Click on **Add Cluster** on the top right. Enter the name of the OpenShift cluster. Copy the kubectl command and run it against your OpenShift cluster. Select **Others** and click **Submit**. If you are connecting PX-Backup to a cluster not running Portworx, you will have to install Stork using this command:

```
curl -fsL -o stork-spec.yaml "https://install.portworx.com/2.9?comp=stork&storkNonPx=true"
kubectl apply -f stork-spec.yaml
```



Once your Red Hat OpenShift cluster(s) is added, you can start backing up your applications using the PX-Backup UI.

Secure Backup and Restore with Role-based Access Controls

As part of this document, we validated using PX-Backup with Red Hat OpenShift by enabling PX-Backup Security and Role Based Access Controls for users.

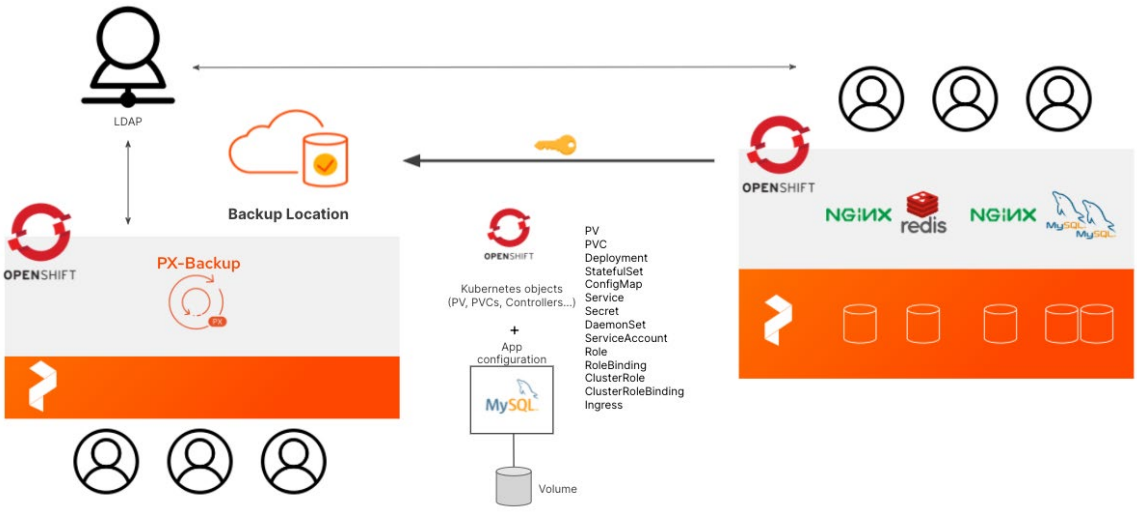


Figure 30: Role-based access control for users.

Manage Users

Once logged into PX-Backup as the administrator, PX-Backup Security is accessible from the bottom left corner menu.

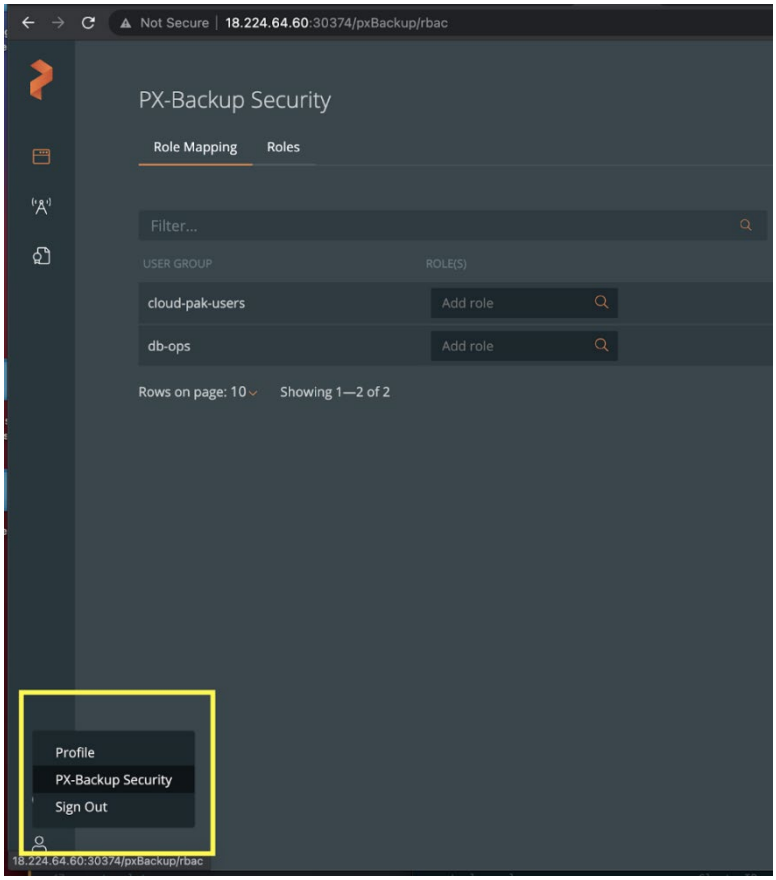


Figure 31: PX-Backup Security

From the PX-Backup Security view, you will be able to handle the following tasks:

- Manage imported or created backup users and groups
- Manage, create, or delete backup roles
- Apply role mappings to backup users and groups

In order to add users to a PX-Backup installation, you will need to access the Keycloak administration console deployed with PX-Backup. First, identify the Keycloak service by running the following commands on the Red Hat OpenShift cluster that PX-Backup is deployed to:

```
$ kubectl get svc pxcentral-keycloak-http -n central
NAME                      TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
pxcentral-keycloak-http   NodePort   10.104.180.137   <none>        80:30544/TCP,8443:32488/TCP
33d
```

In the above scenario, the Keycloak service will be available at the NodePort of 32488. You may also choose to use LoadBalancers or OpenShift routes to expose these services.

```
https://<URL>:32488/auth
```

Once you access the Keycloak administration console, click on **Administration Console**. This will prompt you for the PX-Backup admin user and password.

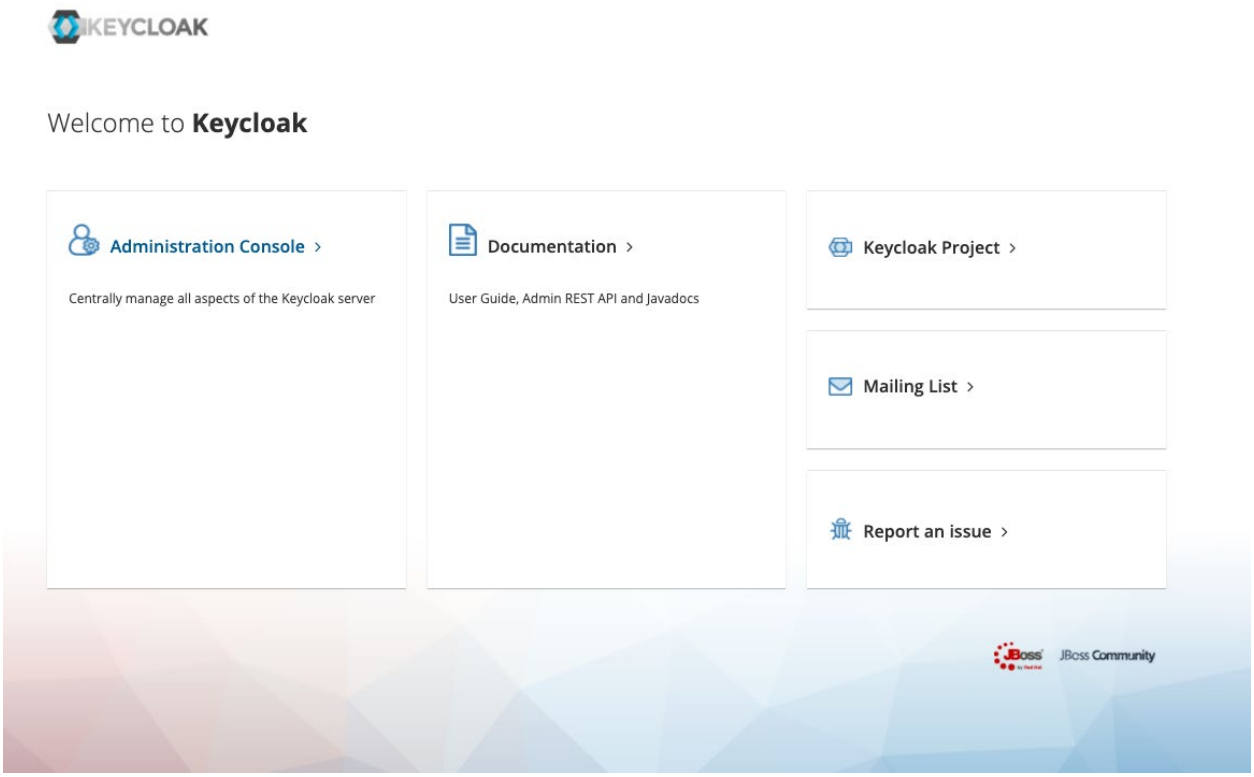


Figure 32: Keycloack interface

LDAP Integration

Once logged in, you may manage users manually by navigating to the **Users** section on the left. However, we will explain how to federate PX-Backup users with an LDAP provider.

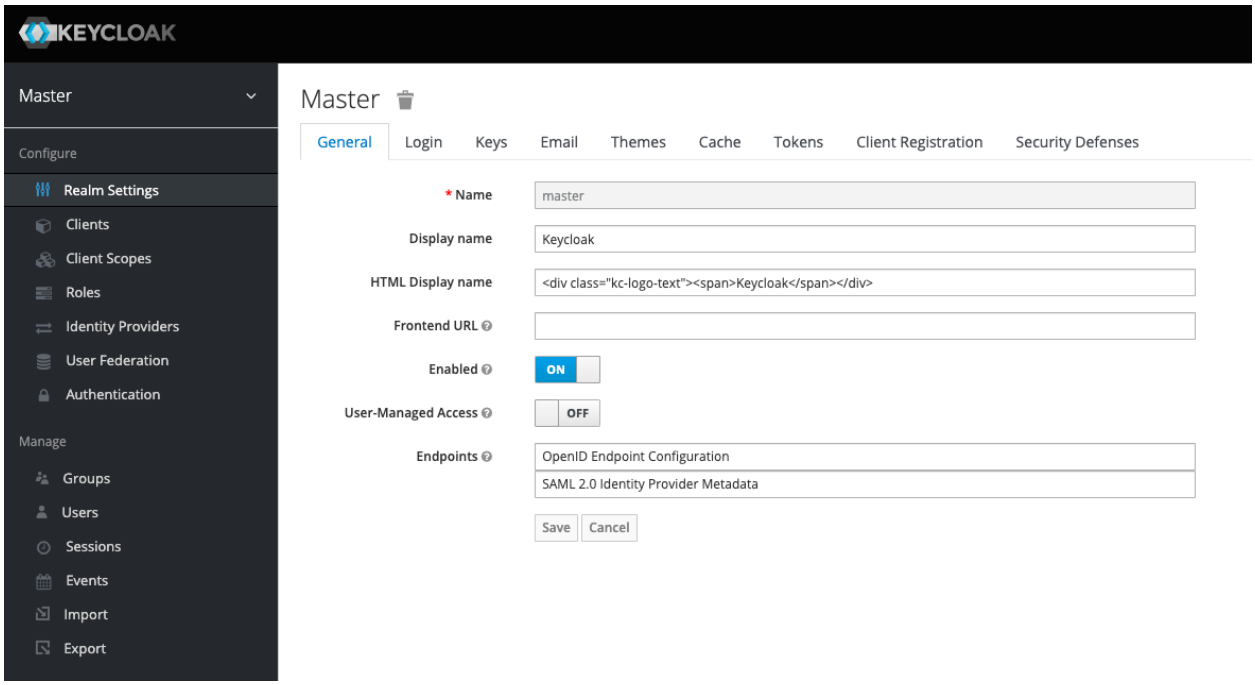


Figure 33: Keycloack Realm settings

Head to the **User Federation** section on the left-hand menu. From here, select **Add Provider** and choose **ldap**.

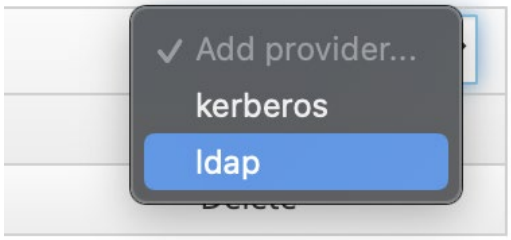


Figure 34: Adding ldap

Fill out the necessary LDAP settings and click **Save**.

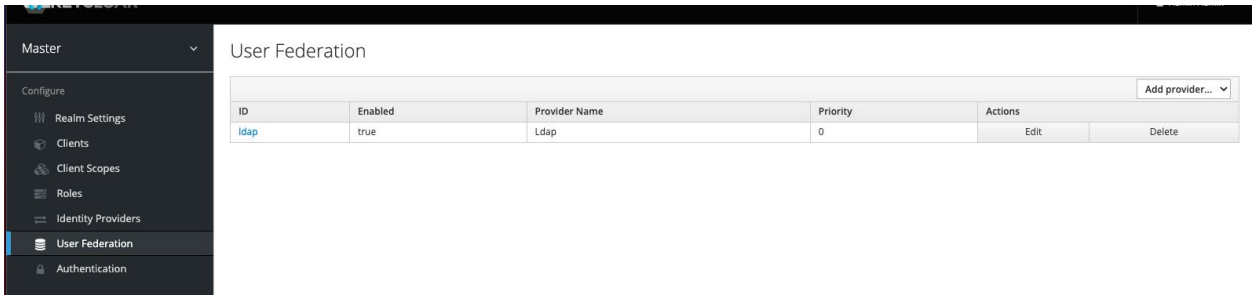


Figure 35: Saved ldap settings

You may also choose to set up automatic synchronization of new or updated users. This will keep your PX-Backup users up to date.

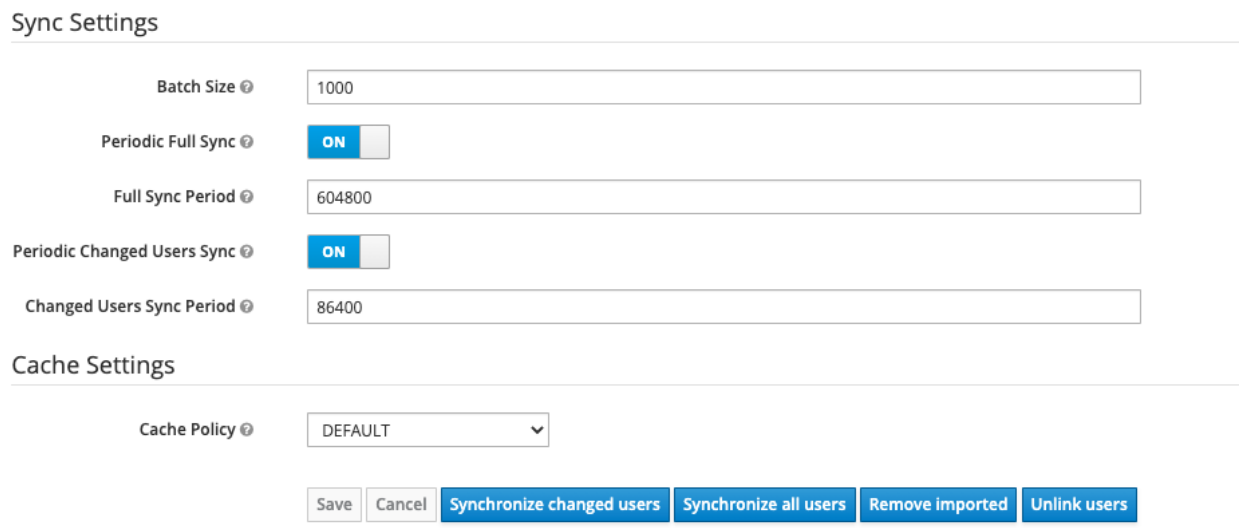


Figure 36: Automatic synchronization

After synchronizing all users with your PX-Backup installation, PX-Backup should report all users and give them the px-backup-app.user role by default.

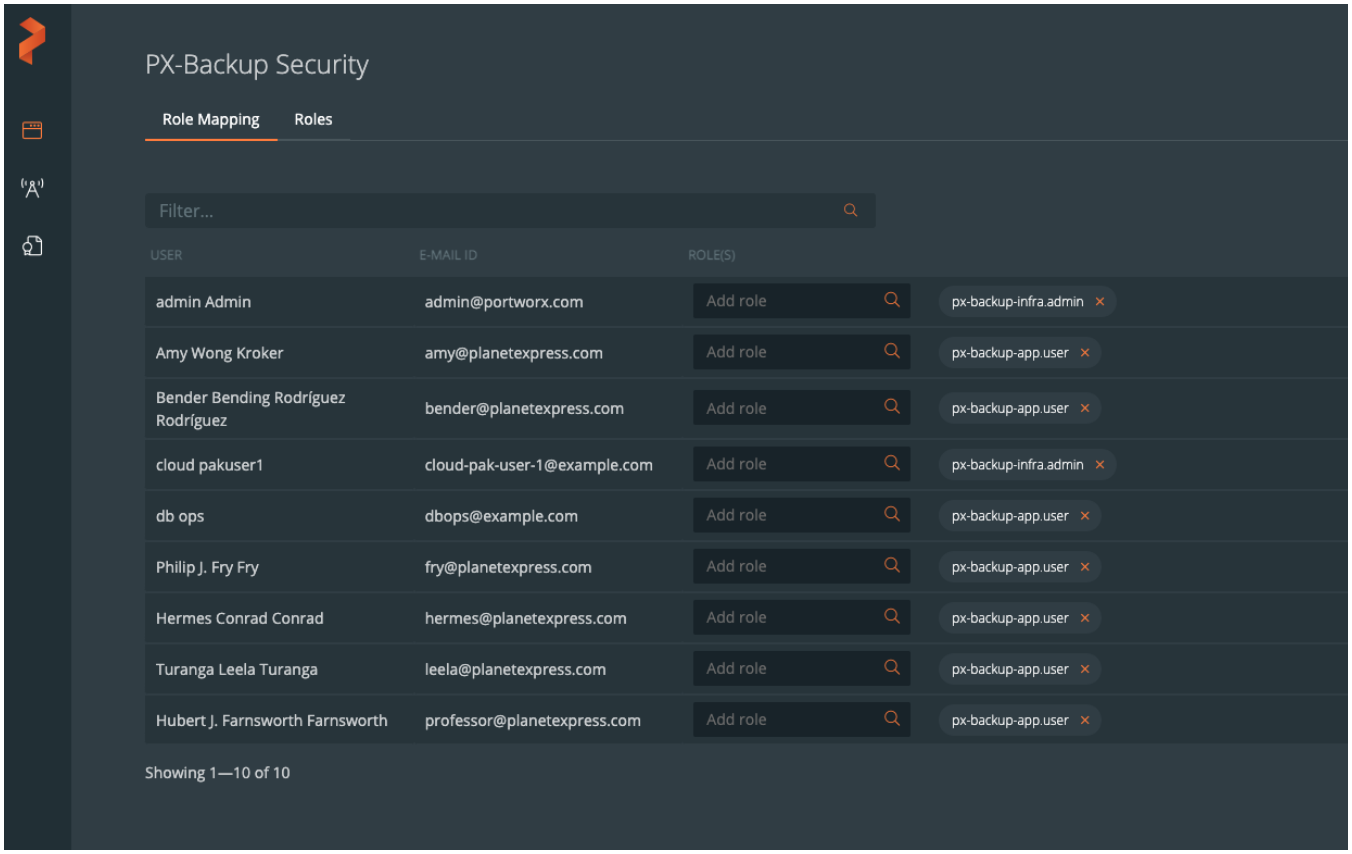


Figure 37: PX-Backup user roles

The default roles available within PX-Backup are as follows:

- **px-backup-infra.admin**: Infrastructure owner with admin privileges for all PX-Backup objects
- **px-backup-app.admin**: Application owner—manage the apps you own with admin privileges for Schedules and Rules. You can also use existing cloud accounts.
- **px-backup-app.user**: Application user—You can backup and restore your application but cannot create a schedule policy or rules.

To create groups within PX-Backup, navigate to **Groups** on the left-hand menu of the Keycloak administration console and click **New**. Enter a new group name and click **Save**.

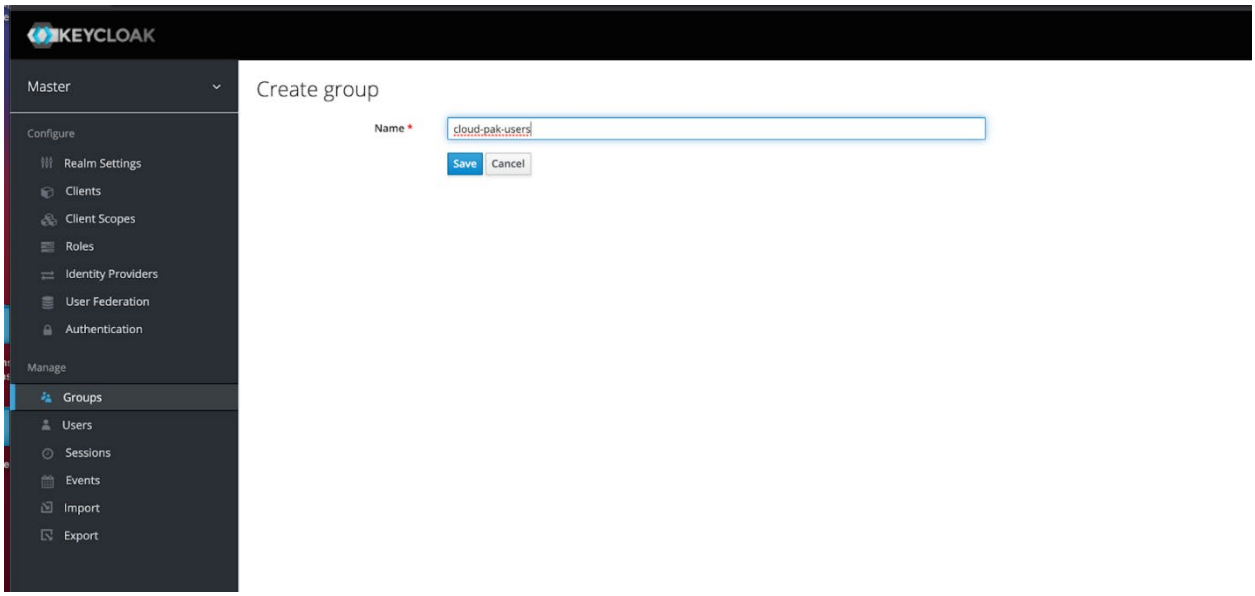


Figure 38: Creating a new group in Keycloak

You can then map PX-Backup roles to these new groups so any user within the group can participate using this role.

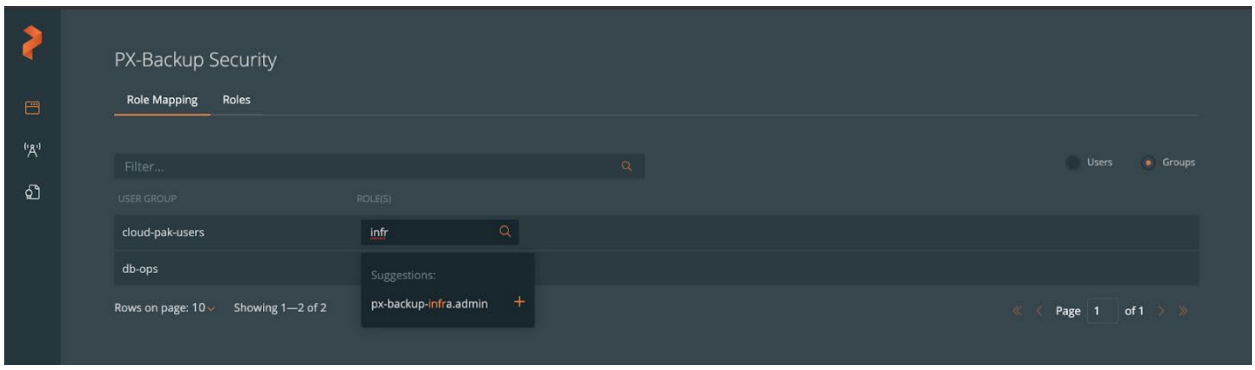


Figure 39: Mapping PX-Backup roles to Keycloack groups.

Then, from the Keycloak administration console, you can add users to these groups.

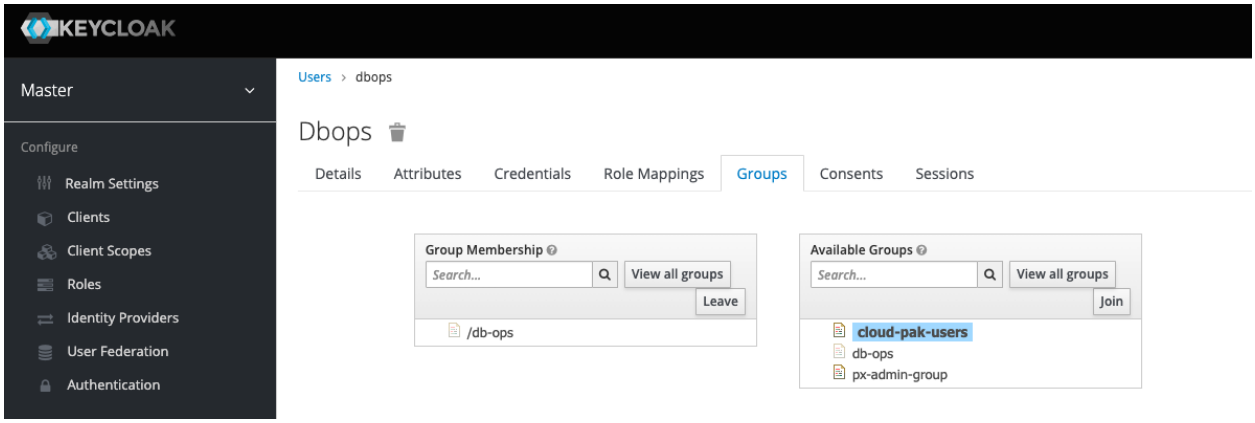


Figure 40: Adding users to groups via Keycloak

Creating a Backup and Restore as an Application User

Once the administrative tasks are complete, users—such as the db-ops user below—may sign into both the Red Hat console and the PX-Backup console.

Note: If Red Hat OpenShift imports the same users as PX-Backup, users can log in using the same credentials.

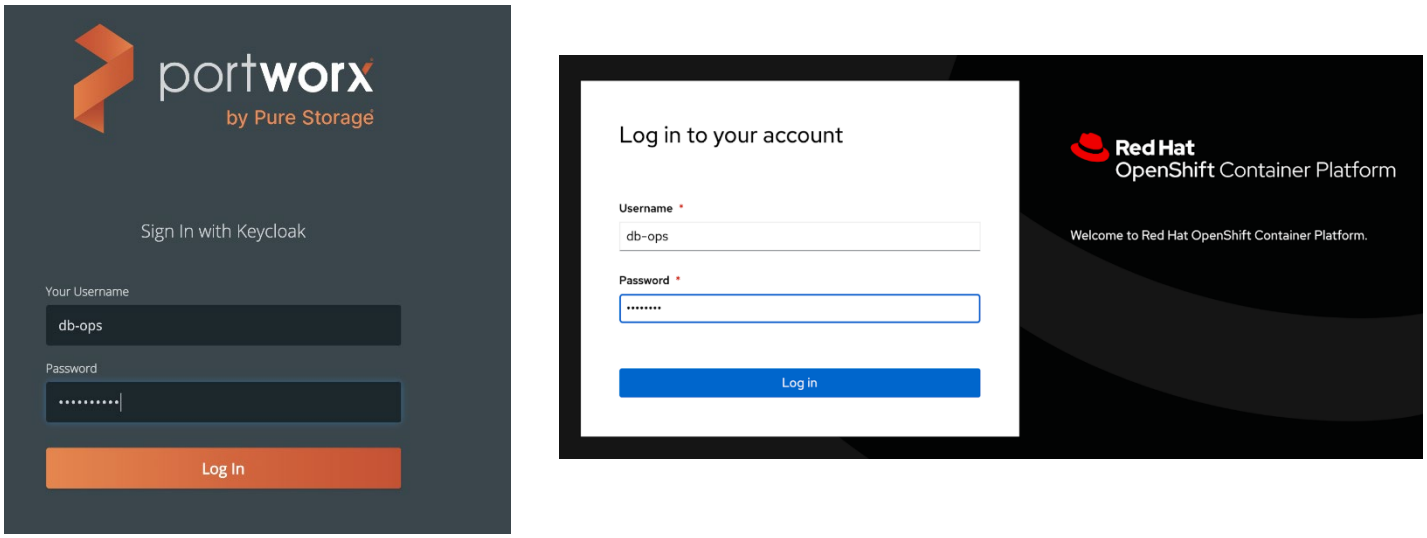


Figure 41: Logging in to Portworx and OpenShift

Red Hat OpenShift users will need some minimum RBAC permissions in order to use PX-Backup to successfully backup and restore applications within their namespaces. The below `Role`, `RoleBinding`, read-only `Cluster-Role`, and `ClusterRoleBinding` are examples for the db-ops user used in this document who only has access to the "db-ops" project (namespace) within OpenShift.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: team-users
rules:
```

```
- apiGroups:
    - "*"
  resources:
  - "*"
  verbs:
  - "*"

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: db-ops-team-user-bindings
  namespace: db-ops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: team-users
subjects:
- kind: User
  name: db-ops
  apiGroup: rbac.authorization.k8s.io

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: read-only
rules:
- apiGroups:
  - ""
  resources: ["*"]
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - extensions
  resources: ["*"]
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - apps
  resources: ["*"]
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - snapshot.storage.k8s.io
  resources: ["*"]
```

```
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - stork.libopenstorage.org
  resources: ["*"]
  verbs:
  - get
  - list
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: db-opsuser-cr-binding
subjects:
- kind: User
  name: db-ops
  namespace: db-ops
roleRef:
  kind: ClusterRole
  name: read-only
  apiGroup: rbac.authorization.k8s.io
```

Once a user is logged into PX-Backup and wants to start backups for an application, they must first add the application cluster to PX-Backup:

```
# oc login -u db-ops
Authentication required for https://api.px-ocp-b-1.openshift.portworx.com:6443 (openshift)
Username: db-ops
Password:
Login successful.

You have access to 65 projects, the list has been suppressed. You can list all projects with 'oc
projects'

Using project "".
```

Then the user needs to navigate to **Backups** and then **Add Cluster**, produce their specific kubeconfig, and enter it in the **Add Cluster** details screen along with an arbitrary name.

```
# kubectl config view --flatten --minify
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC…<snip>
```

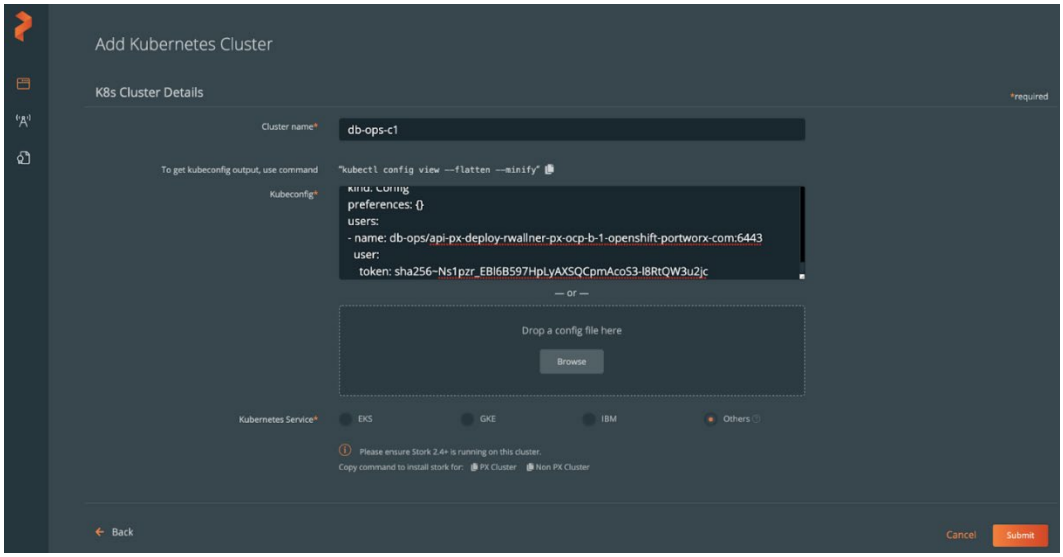


Figure 42: Adding cluster details

In this case, because of their OpenShift RBAC read-only permissions, they can only view the resources within the cluster. This user is also a px-backup-app.user, so they can back up and restore applications, but they can't create a schedule policy, cloud credentials, backup locations, or rules from within PX-Backup.

From the **Applications** tab, the db-ops user can filter by resource or tag to back up the entire db-ops namespace or specific resources within it.

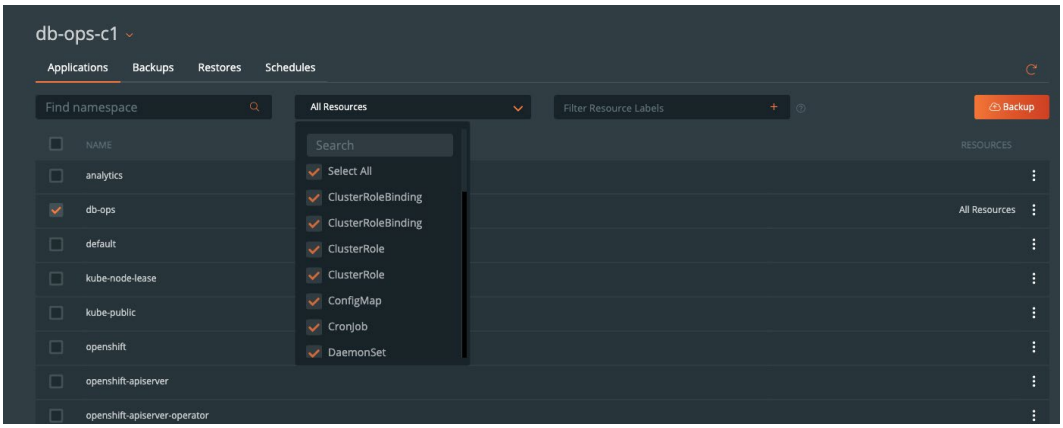


Figure 43: Filtering by resource

Since this user cannot create rules, schedules, or backup locations, they must use those that are provided by an admin user (aws in the example shown). The user should give their backup a name and click **Create**.

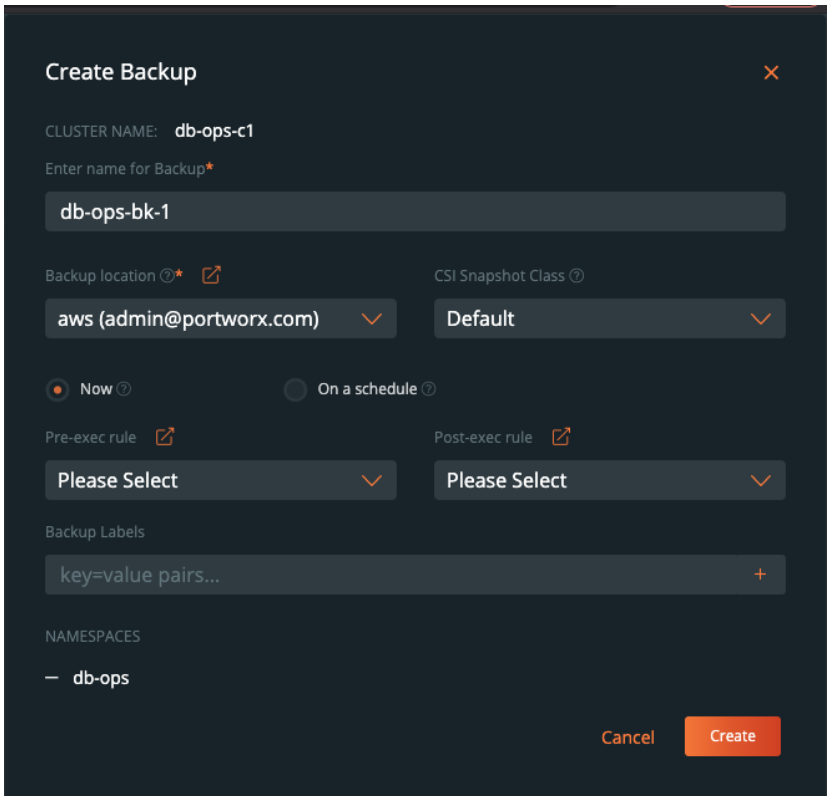


Figure 44: Creating a backup

The db-ops user will then be taken to a backup's timeline view, where they may see their backups, status, frequency, and details.

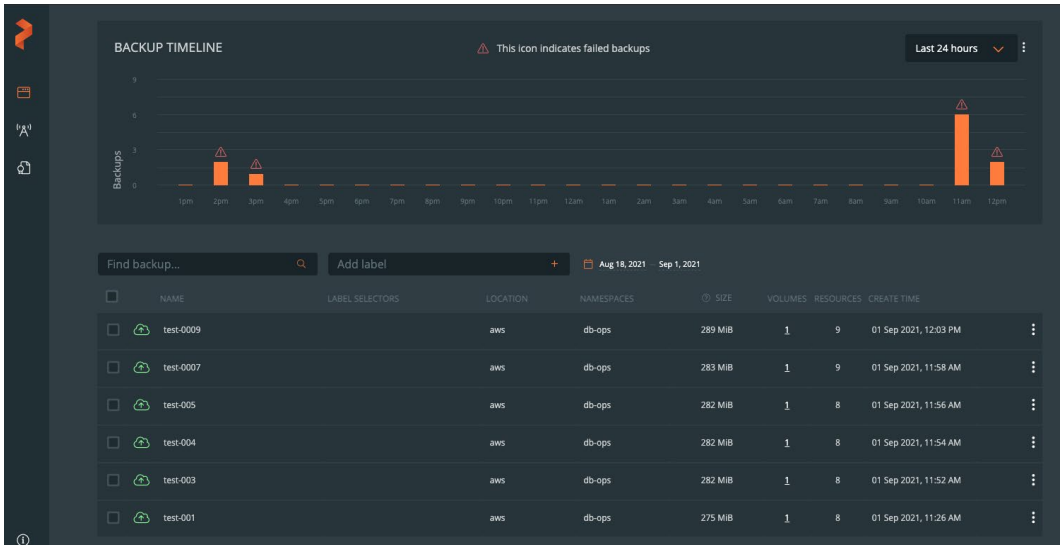


Figure 45: Timeline view showing backups, status, frequency, and other details

To restore from a backup, click on the backup you wish to restore and navigate to the **Restore** selection within the backup pop-out menu.

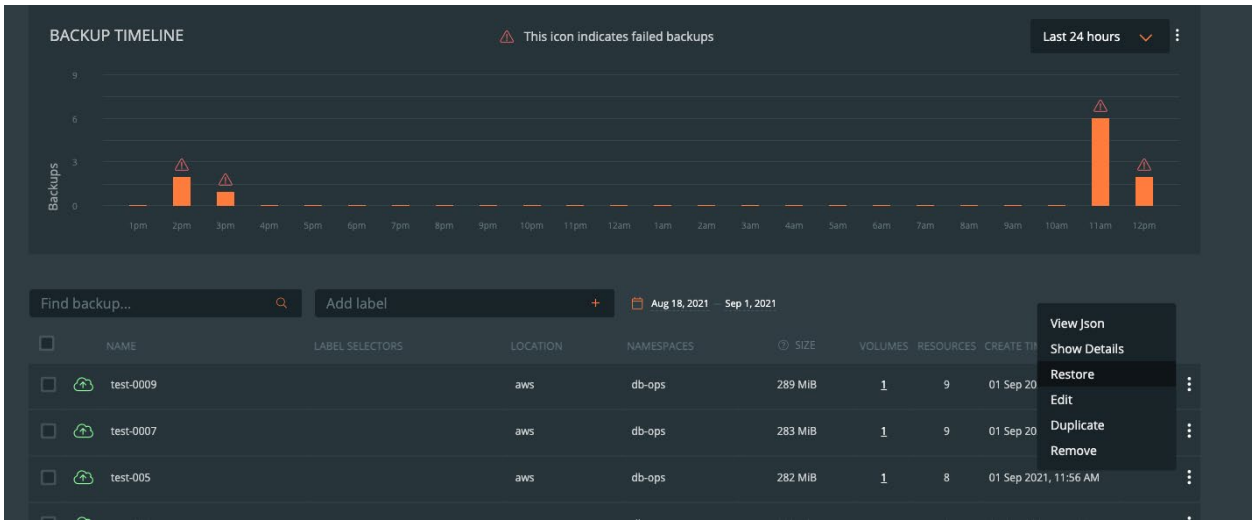


Figure 46: Restoring from a backup

Fill out the restore dialog with a restore name, a destination cluster (the same one in this case) and click **Replace existing resources** since we only have access to the one namespace and cannot restore to another with the db-ops user.

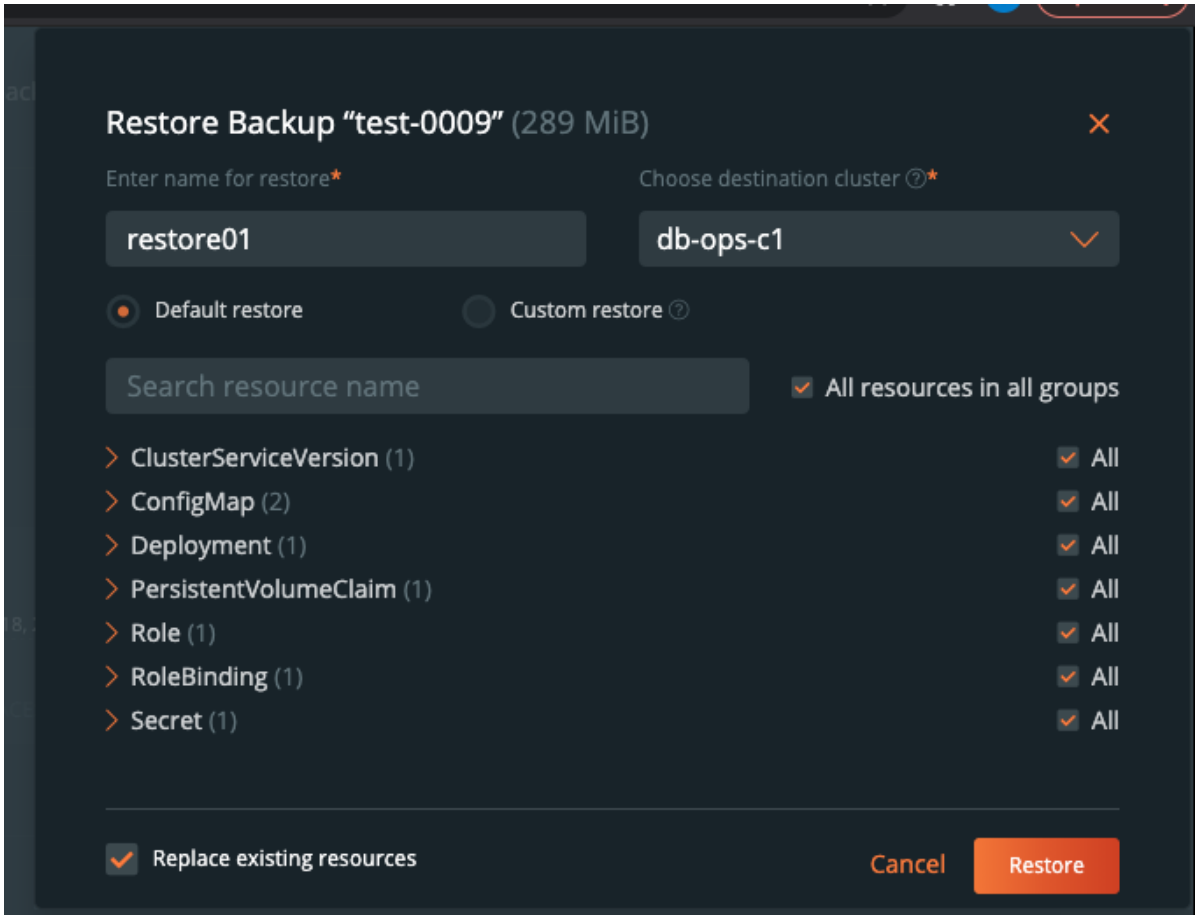


Figure 47: Restoring a backup

The restore should turn green when it is successfully restored.

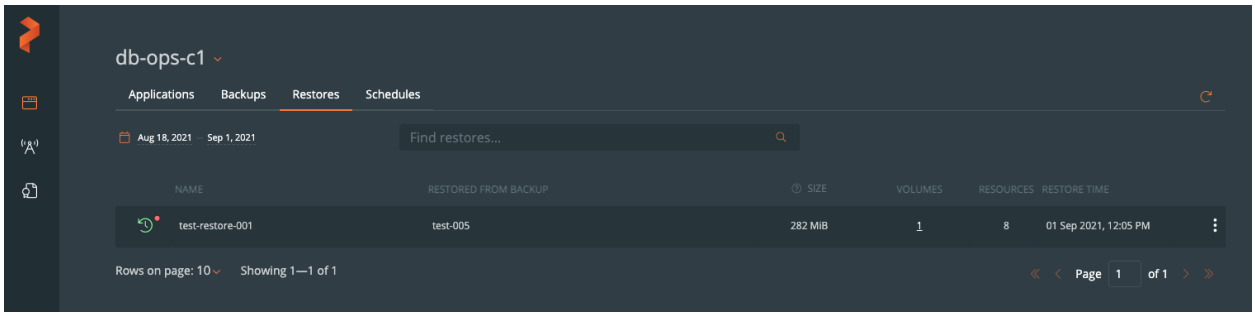


Figure 48: Restore turning green, showing a successful restore

PX-Backup users will not be able to create backups for other namespaces or resources within RedHat OpenShift clusters they do not have access to. The example below shows the error the “db-ops” user would see if they tried to backup resources from the “analytics” namespace, which they do not have access to.

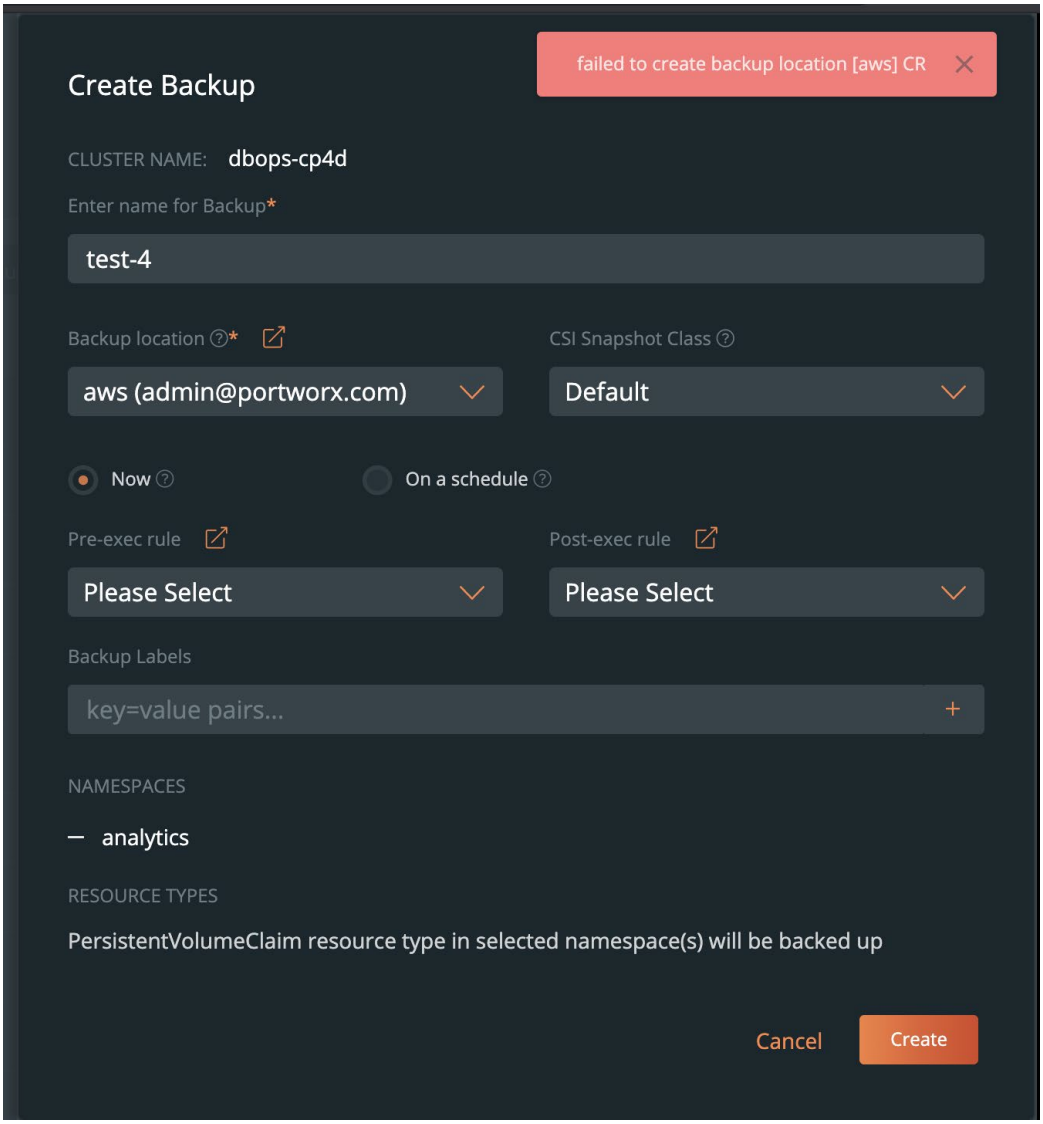


Figure 49: Error when creating a backup in a non-accessible namespace

Remember, PX-Backup provides RBAC for backup objects—such as backup locations, rules, schedules, backups, and restores—while OpenShift RBAC dictates the roles and access allowed for users within the OpenShift cluster. Users add their kubeconfig to PX-Backup and thus are limited only to the resources exposed by their OpenShift admins. This allows PX-Backup and Red Hat OpenShift to work together to provide the most secure backup and restore capabilities for applications running on Red Hat OpenShift.

Highly Available OpenShift Container Registry with Portworx ReadWriteMany (RWX) Storage

A common first step after deploying a fresh OpenShift cluster is to configure the internal private registry. This provides your DevOps teams with a local repository for the container images they will use for developing and testing applications. If these registries are backed by unreliable storage, are configured in ReadWriteOnce (RWO) mode in OpenShift, and require multiple personas to provision and configure the backing storage, your developer efficiency can be reduced or halted altogether.

Portworx ReadWriteMany storageClasses increases DevOps efficiency and lets you stop worrying about rouge NFS servers that can bring your pipelines to a grinding halt in case of a failure. Let's walk through how you might configure a highly available internal private registry for ROSA using Portworx.

Create a Sharedv4 Portworx StorageClass

Once Portworx is installed, it is simple to create a PV through Portworx for use in OpenShift. First, we will create a StorageClass that is configured for ReadWriteMany mode. This is a mode in Kubernetes allowing multiple pods or containers to access a single PVC, similar to using NFS.

Below is an example of the StorageClass we will create so we can provision our storage. Note that we are using the **pxd.portworx.com** provisioner. The **parameters** section contains the information that tells Portworx we want to use the Sharedv4 volume type and includes any NFS export rules we want Portworx to pass to the containers mounting the volumes.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: portworx-sharedv4-sc
provisioner: pxd.portworx.com
parameters:
 repl: "3"
  sharedv4: "true"
  sharedv4_mount_options: "vers=4.0"
  io_profile: auto
allowVolumeExpansion: true
```

Now that we have the StorageClass created, we can create a PVC and get our NFS PV provisioned and ready for use to back the storage for the OpenShift internal private registry. Let's create a 100Gi RWX PVC in the openshift-image-registry namespace, referencing the StorageClass we just created:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-registry-pvc
  namespace: openshift-image-registry
spec:
  accessModes:
  - ReadWriteMany
  resources:
 requests:
   storage: 100Gi
  storageClassName: "portworx-sharedv4-sc"
```

We can now issue the command `oc get pvc -n openshift-image-registry` to verify that we have a bound PVC that has been provisioned through Portworx:

```
[root@tdarnell-ocp-adminws ~]# oc get pvc -n openshift-image-registry
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
          AGE
nfs-registry-pvc   Bound    pvc-2f0cdaec-2c0d-41f4-a287-389c22cdfd4f   100Gi      RWX            flashblade-direc
taccess   13s
```

And we can also check the volume in Portworx by issuing the command `pxctl volume list` from one of the worker nodes:

```
[root@tdarnell-ocp-adminws ~]# pxctl volume list
Defaulted container "portworx" out of: portworx, csi-node-driver-registrar, telemetry
ID                  NAME                                    SIZE    HA      SHARED  ENCRYPTED       PROXY-VOLUME    IO_PRIO
RITY    STATUS          SNAP-ENABLED
261319295930206650  pvc-2f0cdaec-2c0d-41f4-a287-389c22cdfd4f        100 GiB 1       no      no              no              HIGH  u
p - detached    no
```

Now we have verified that we have provisioned a ReadWriteMany PV for the OpenShift internal private registry, so let's configure it.

Configure and Scale the OpenShift Internal Private Registry

Now that we have resilient and reliable backing storage presented to the OpenShift cluster, we can configure the internal private registry to use it and scale the registry to be highly available.

All we need to do to configure the registry is modify the operator config by issuing the command `oc edit configs.imageregistry.operator.openshift.io/cluster`, add our storage configuration using our PVC name, modify the number of replicas to three to ensure the registry is highly available, and set the state to Managed:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: "2021-10-04T22:43:00Z"
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 15
  name: cluster
  resourceVersion: "5489504"
  uid: cd8c358b-8df4-4622-a839-6826ae825077
spec:
  logLevel: Normal
  managementState: Managed
  observedConfig: null
  operatorLogLevel: Normal
  proxy: {}
  replicas: 3
  requests:
    read:
      maxWaitInQueue: 0s
    write:
      maxWaitInQueue: 0s
  rolloutStrategy: RollingUpdate
  storage:
    pvc:
      claim: nfs-registry-pvc
  unsupportedConfigOverrides: null
status:
```

We can monitor the status of the image registry pods by issuing the command `watch oc get pods -n openshift-image-registry` and wait for all the pods to become ready:

```
Every 2.0s: oc get pods -n openshift-image-r...  tdarnell-ocp-adminws: Wed Oct 13 01:11:30 2021

NAME                                               READY   STATUS      RESTARTS   AGE
cluster-image-registry-operator-5f56b8d5b4-6fn7k   1/1     Running     0          7d9h
image-pruner-27231840-hsvb5                        0/1     Completed   0          2d7h
image-pruner-27233280-2w5f9                        0/1     Completed   0          31h
image-pruner-27234720-9lf7d                        0/1     Completed   0          7h11m
image-registry-7c859b446c-2j6c6                    1/1     Running     0          109s
image-registry-7c859b446c-nxt5z                    1/1     Running     0          109s
image-registry-7c859b446c-vbjlp                    1/1     Running     0          109s
node-ca-218jl                                      1/1     Running     1          7d9h
node-ca-9jbbt                                      1/1     Running     1          7d9h
node-ca-151x9                                      1/1     Running     1          7d9h
node-ca-nv48h                                      1/1     Running     1          7d9h
node-ca-sct4d                                      1/1     Running     1          7d9h
node-ca-wn7vg                                      1/1     Running     1          7d9h
```

To use the registry, we need to add a route to its service. We can do this again by editing the config of the image registry operator and adding the `defaultRoute: true` key-value pair to the spec:

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: "2021-10-04T22:43:00Z"
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 17
  name: cluster
  resourceVersion: "5492488"
  uid: cd8c358b-8df4-4622-a839-6826ae825077
spec:
  defaultRoute: true
  httpSecret: 0d9b34556f809f787e60a5dcf142e1f0b7459f8
e0cd8bf837668493513996 22b9d51e604c9c4c919f6ad9b
  logLevel: Normal
  managementState: Managed
  observedConfig: null
  operatorLogLevel: Normal
  proxy: {}
  replicas: 3
```

Now that we have our registry exposed, we can get the DNS name by issuing the command `oc get route default-route -n openshift-image-registry`, then use podman to pull, tag, and push a simple helloworld container image to it:

```
[root@tdarnell-ocp-adminws ~]# podman push --tls-verify=false docker.io/karthequian/helloworld $HOST/test/helloworld
Getting image source signatures
Copying blob 02473afd360b done
Copying blob dbf2c0f42a39 done
Copying blob 689fb57937fb done
Copying blob 89c0daa71499 done
Copying blob 2f60cf94f33d done
Copying blob 9f32931c9d28 done
Copying blob ebb24b834d91 done
Copying blob 6967353de304 done
Copying blob 63f60cac95f0 done
Copying blob 8865d8e83073 done
Copying blob d178243a0617 done
Copying blob 9988350afb63 done
Copying blob e790d2968402 done
WARN[0005] failed, retrying in 1s ... (1/3). Error: Error writing blob: Error uploading layer chunked: blob upload unknow
n: blob upload unknown to registry
Getting image source signatures
Copying blob dbf2c0f42a39 skipped: already exists
Copying blob 02473afd360b done
Copying blob 9f32931c9d28 skipped: already exists
Copying blob 689fb57937fb skipped: already exists
Copying blob 2f60cf94f33d skipped: already exists
Copying blob ebb24b834d91 skipped: already exists
Copying blob 6967353de304 skipped: already exists
Copying blob 63f60cac95f0 skipped: already exists
Copying blob d178243a0617 skipped: already exists
Copying blob 89c0daa71499 skipped: already exists
Copying blob 8865d8e83073 skipped: already exists
Copying blob 9988350afb63 skipped: already exists
Copying blob e790d2968402 [--------------------------------------] 0.0b / 0.0b
Copying config a0d8db65e6 done
Writing manifest to image destination
Storing signatures
```

Now we have verified that we have provisioned a ReadWriteMany PV for the OpenShift internal private registry, so let's configure it.

Data Security on Red Hat OpenShift

Portworx provides imperative data security pillars when it comes to securing data access and control within Kubernetes platforms such as Red Hat OpenShift. These pillars include the following components.

Encryption

The importance of encryption is paramount to keeping information confidential—whether it's at rest or in transit. Data within the PVCs of a Kubernetes application should be encrypted for those applications using sensitive information—such as those in the healthcare space using PII and PHI.

Encryption can happen in a few different places; the first is encrypting data at rest with encryption keys for cluster-wide encryption or per-pvc encryption with Portworx. To enable encryption for volumes with Portworx, a key must hold a passphrase used for encryption. This key can live in Kubernetes as a Kubernetes Secret or in external KMS systems such as Vault. Portworx supports a variety of secret providers, which can be used for Portworx encryption secrets.

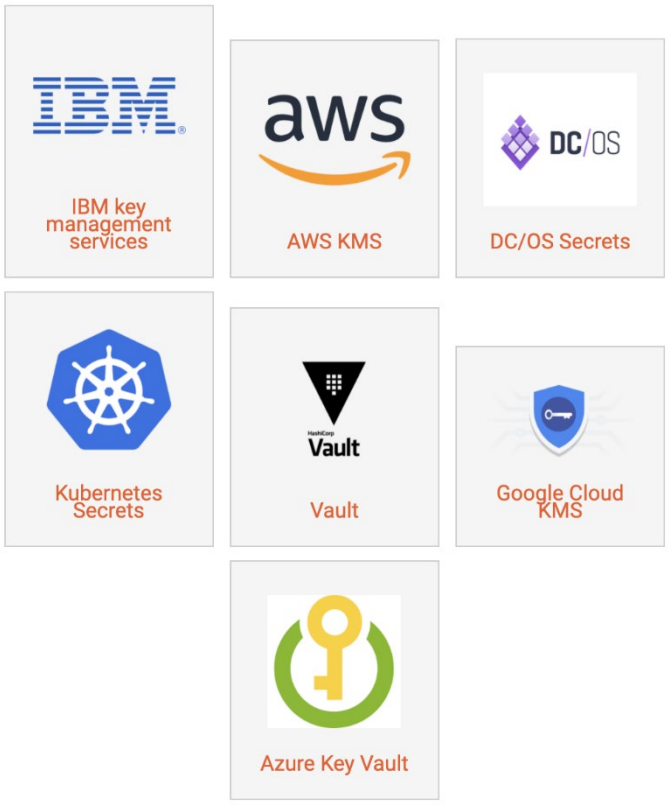


Figure 50: Portworx-supported secret providers.

Encrypted volumes come in two flavors on Portworx- enabled OpenShift clusters. One is a cluster-wide encrypted volume where all volumes share the same encryption key. The other is a per-volume encrypted volume where each volume has a unique secret. Per-volume encryption is great for multi-tenant environments where many teams may share a single OpenShift cluster.

Once encryption is enabled, a secrets provider needs to be configured; by default this will be the Kubernetes secrets provider and no extra configuration is needed. If you wish to use Vault, AWS KMS, or others, consult the documentation.

Next a cluster-wide secret needs to be enabled.

```
oc -n portworx create secret generic px-vol-encryption \
  --from-literal=cluster-wide-secret-key=<value>

PX_POD=$(oc get pods -l name=portworx -n portworx -o jsonpath='{.items[0].metadata.name}')
oc exec $PX_POD -n portworx -- /opt/pwx/bin/pxctl secrets set-cluster-key \
  --secret cluster-wide-secret-key
```

Enabling StorageClass based encryption now that the cluster-wide encryption key is configured is as easy as providing a StorageClass with `secure: “true”` with the key value pair secure: true in the parameters section.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-secure-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  secure: "true"
  repl: "3"
```

You may also enable per-pvc volume encryption by using a secret per volume as well as using a CSI StorageClass to inject the provisioner and publish secrets and namespaces. Take for instance a PVC named “mysql-pvc-1”.

First create the generic volumes secret.

```
oc create secret generic volume-secrets-1 -n portworx --from-literal=mysql-pvc-secret-key-
1=mysecret-passcode-for-encryption-1
```

Then, create a secret with the same name as the PVC, which maps to the above secret key.

```
oc create secret generic mysql-pvc-1 -n portworx --from-literal=SECRET_NAME=volume-secrets-1 --from-
literal=SECRET_KEY=mysql-pvc-secret-key-1 --from-literal=SECRET_CONTEXT=portworx
```

Then, a StorageClass can use generic parameters to allow per-pvc secrets with CSI.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: portworx-sc
provisioner: pxd.portworx.com
parameters:
  repl: "1"
```

```
    secure: "true"
    csi.storage.k8s.io/provisioner-secret-name: ${pvc.name}
    csi.storage.k8s.io/provisioner-secret-namespace: ${pvc.namespace}
    csi.storage.k8s.io/node-publish-secret-name: ${pvc.name}
    csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

Now, when a volume uses the following StorageClass, it will need to have the above associated per-pvc secrets. The below "mysql-pvc-1" PVC will automatically use the specific encryption key only for this PVC.

```
kind: PersistentVolumeClaim
  apiVersion: v1
  metadata:
    name: mysql-pvc-1
    namespace: portworx
  spec:
    storageClassName: portworx-sc
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 2Gi
```

Then, in either case, Portworx volumes will be encrypted and shown as **ENCRYPTED: yes** within Portworx.

```
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 volume list
```

```
ID                 NAME                                      ...  ENCRYPTED  ...
10852605918962284  pvc-5a885584-44ca-11e8-a17b-080027ee1df7  ...  yes        ...
```

Authentication and Authorization

Validating that the user is who they say they are and then subsequently verifying that the user has the right to access, manipulate, or create a resource is the art of authenticating a user and then authorizing one. Authentication and authorization are key to any Kubernetes platform, and the same is true with OpenShift. When interacting with OpenShift itself, you must be an authorized user or admin, so why should it be any different when interacting with Persistent Storage?

Authentication

To enable authentication and authorization for Portworx data management, make sure PX-Security is enabled in the Portworx StorageCluster on your OpenShift installation. To enable security, navigate to the Portworx project, select **Installed Operators**, then select **Portworx Enterprise**, and click on **Storage Clusters**.

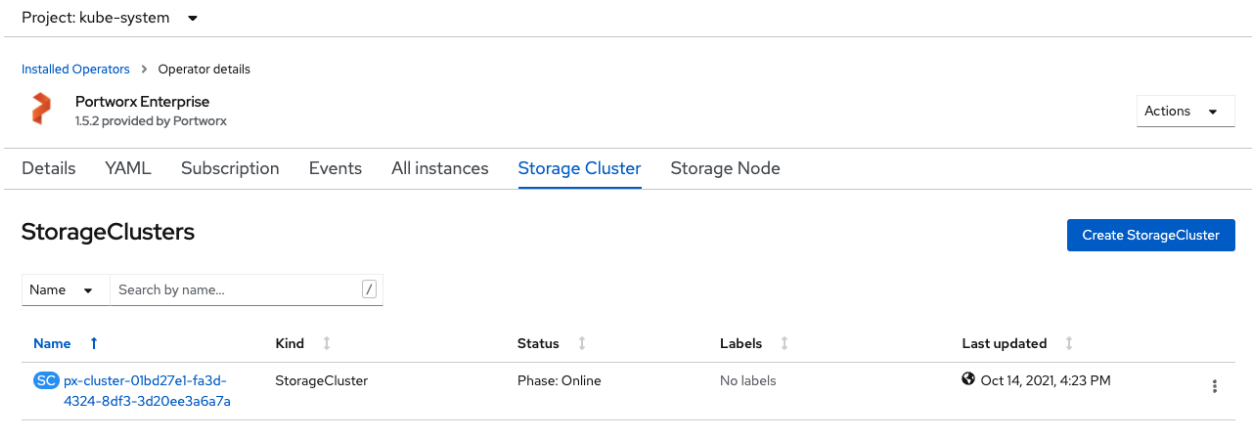


Figure 51: Navigating to the storage clusters.

Click on the Storage Cluster and select **YAML**. From here you can add the security section to the YAML and provide optional customizations such as configuring specific token issuers, OIDC provider information, and token lifetimes. Note that Portworx will restart each Portworx node one-by-one until security is enabled, so monitor the Portworx pods to make sure all pods are back online and healthy.

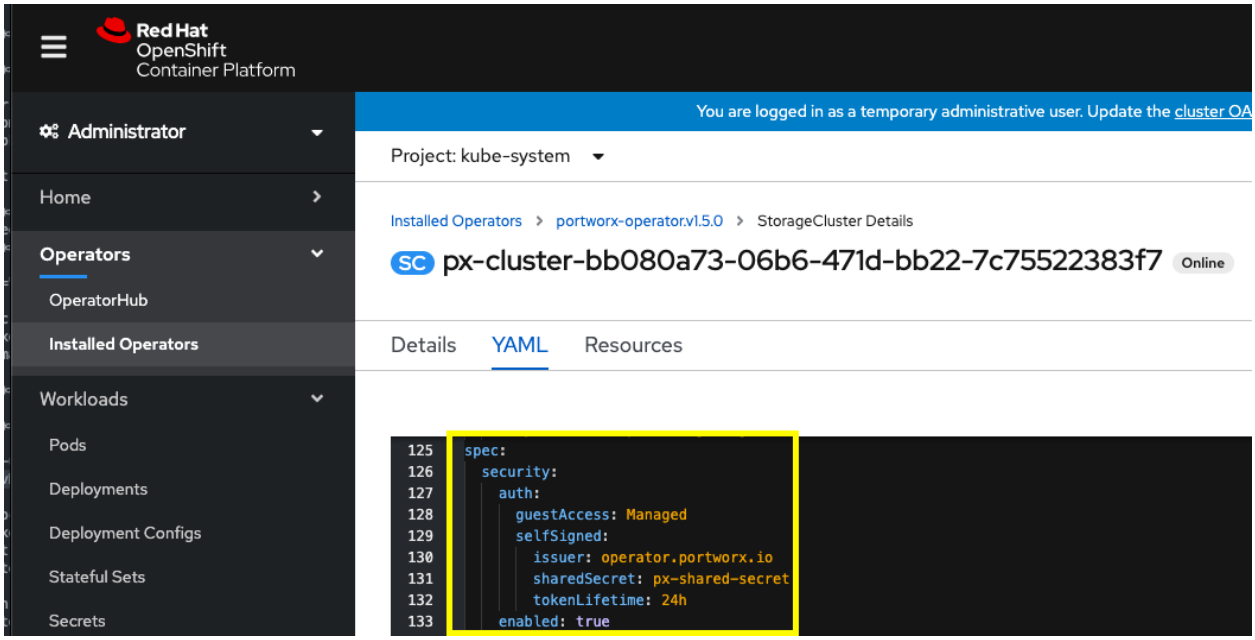


Figure 52: Configuring the YAML

Once the security enablement steps are complete, this will enable the Portworx-specific RBAC model. This RBAC model includes authentication and authorization for Portworx resources such as volumes, snapshots, cloud snapshots, and more.

To authenticate users in Portworx, PX-Security supports two types of token generation models: OIDC and self-generated tokens. OpenID Connect (or OIDC) is a standard model for user authentication and management and is a great solution for enterprise customers due to its integration with SAML 2.0, Active Directory, and/or LDAP. The second model is self-generated token validation. This guide will use self-generated tokens. Administrators would generate a token using their own token administration application and for convenience, Portworx provides a method of generating tokens using the Portworx CLI (**pxctl**).

For Portworx to verify the tokens are valid, they must be signed with one of the following:

- A shared secret
- An RSA private key
- An ECDSA private key

The token will be created by the token administrator and will contain information about the user in the claims section. When Portworx receives a request from the user, it will check the token validity by verifying its signature, using either a shared secret or public key provided during configuration.

An example profile for a user may be the following. The user below is an “analytics-team” user, who uses a “system.user” role within Portworx. They belong to the "analytics-users" and “dbs-analytics” groups, which can have specific access controls assigned to them.

```
name: analytics-team
sub: analytics@purestorage.com/analytics
email: analytics@purestorage.com
roles: ["system.user"]
groups: ["analytics-users", “dbs-analytics”]
```

To produce a token for this user, we can use pxctl. To produce a token, you will need the shared secret created by Portworx when security was enabled. To get this secret, run the following command:

```
$ oc -n portworx get secret px-shared-secret -o json | jq -r '.data."shared-secret"' | base64 -d
qbIsxGTEYP1/EBl0MCwn9e4uaHx3IbMM/7gwOWhNgElgUmK2qq4fAhwRwQiU+Cgn
```

Then, proceed to create a token for the analytics user with a one-year token duration.

```
$ pxctl auth token generate  --auth-config=analytics.yaml  --issuer operator.portworx.io --shared-
secret qbIsxGTEYP1/EBl0MCwn9e4uaHx3IbMM/7gwOWhNgElgUmK2qq4fAhwRwQiU+Cgn --token-duration=1y
```

The token can then be made available within Kubernetes by creating a secret that a StorageClass can reference.

```
oc -n db-ops create secret generic px-user-token --from-literal=auth-token=<auth-token>
```

A StorageClass can reference the token directly, or via CSI and then can be used by the specific tenant.

Direct:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-storage
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "1"
  openstorage.io/auth-secret-name: px-user-token
  openstorage.io/auth-secret-namespace: analytics
allowVolumeExpansion: true
```

via CSI:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-storage
provisioner: pxd.portworx.com
parameters:
  repl: "1"
  csi.storage.k8s.io/provisioner-secret-name: px-user-token
  csi.storage.k8s.io/provisioner-secret-namespace: analytics
  csi.storage.k8s.io/node-publish-secret-name: px-user-token
  csi.storage.k8s.io/node-publish-secret-namespace: analytics
  csi.storage.k8s.io/controller-expand-secret-name: px-user-token
  csi.storage.k8s.io/controller-expand-secret-namespace: analytics
allowVolumeExpansion: true
```

Authorization

The next step in this process is to verify that the token provided during a request (such as one to create a PVC) is valid.

Once the token has been determined to be valid, Portworx then checks if the user is authorized to make the request. The *roles* claim in the token must contain the name of an existing default or customer registered role in the Portworx system. A role is the name given to a set of RBAC rules that enable access to certain SDK calls. Custom roles can be created using pxctl or through the OpenStorage SDK.

As an example, if a token is not valid, users will see an “Access Denied” response with OpenShift, such as the one below when a PVC request is sent:

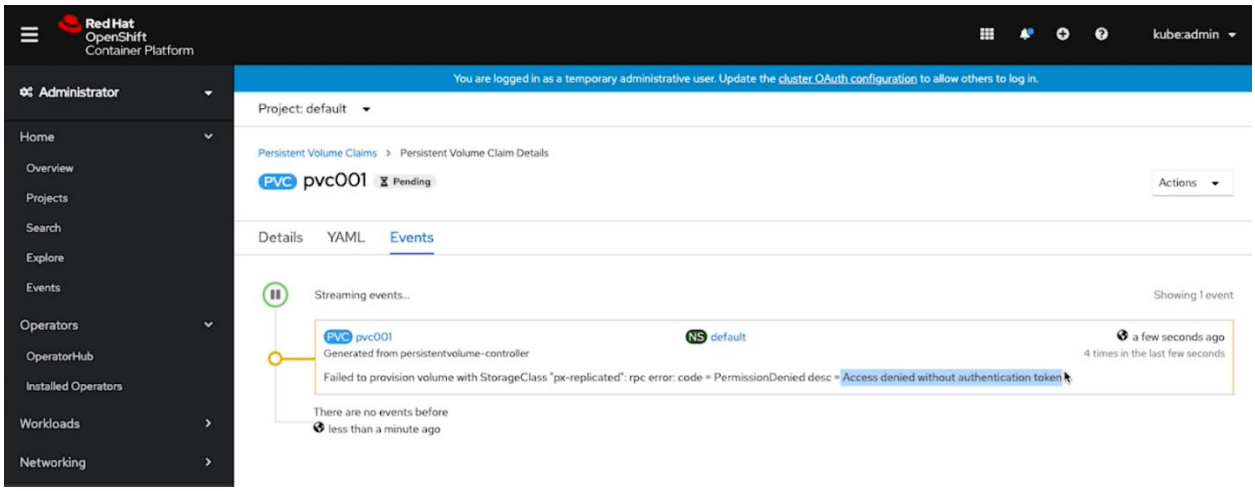


Figure 53: Access denied message in OpenShift

If a token was passed but is invalid or spoofed, a user will also be denied as seen in the below image.

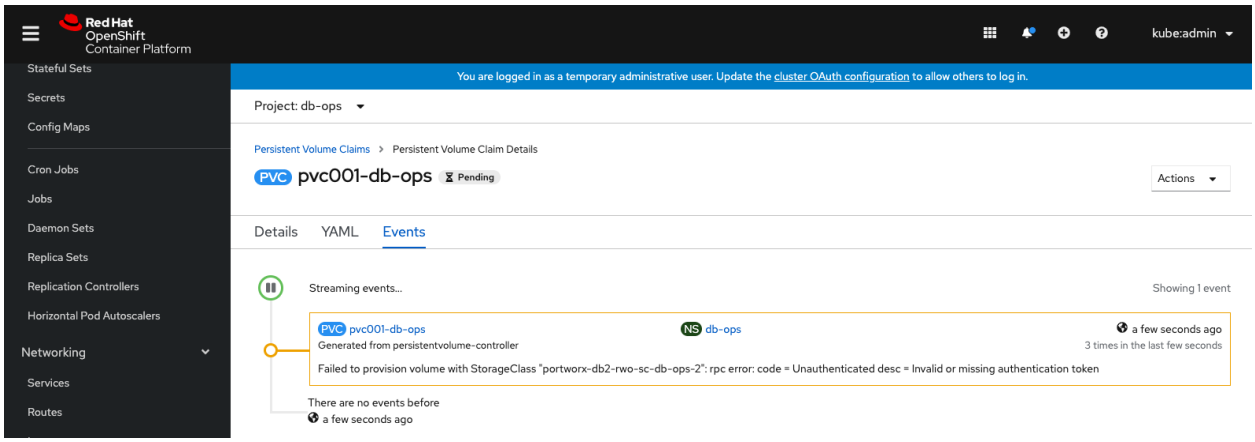


Figure 54: Error message when token was passed but is invalid or spoofed.

If a token was passed but is invalid or spoofed , a user will also be denied as seen in the below image.

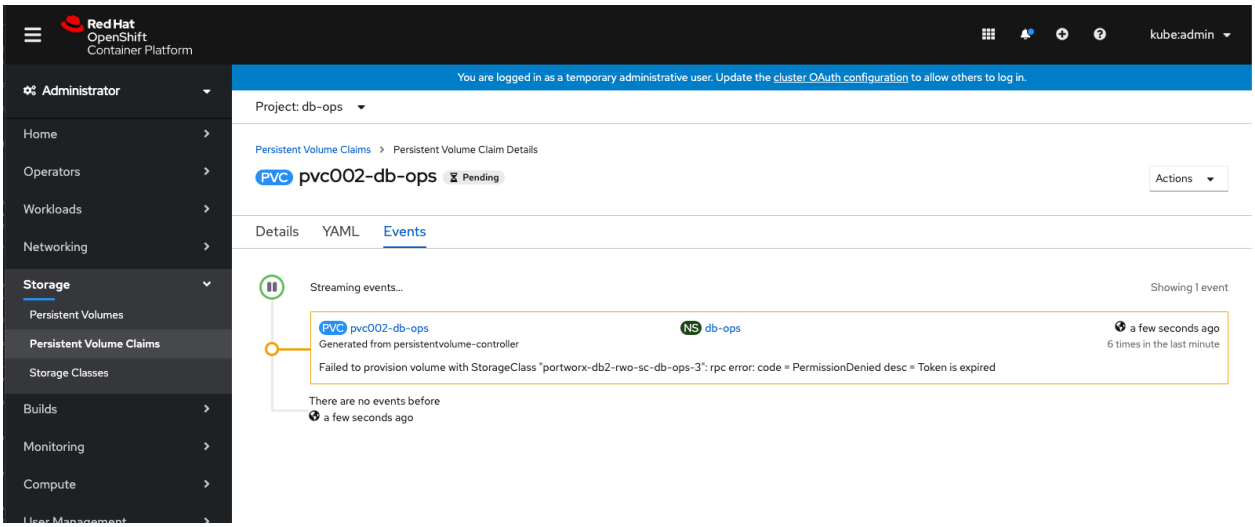


Figure 55: Error message if a token was passed but is invalid or spoofed

If a token is valid but has the wrong role for creating PVCs, a user will also be denied access to that resource as seen in the below image.

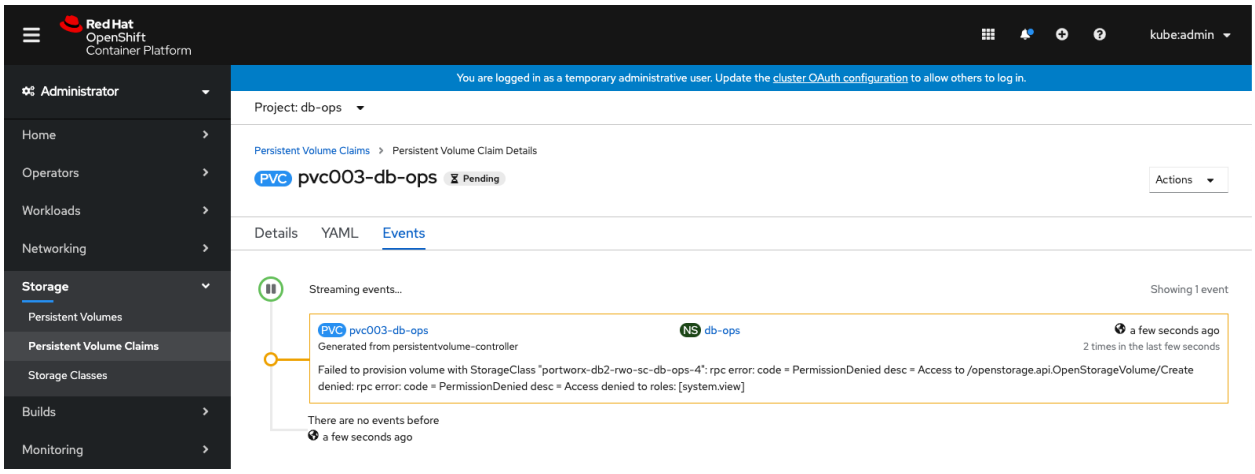


Figure 56: Error message if a token is valid but has the wrong role for creating PVCs

Ownership

Ownership is the model used for resource control. The model is composed of the owner and a list of groups and collaborators with access to the resource. Groups and collaborators can also have their access to a resource constrained by their access type. The following table defines the three access types supported:

Type	Description
Read	Has access to view or copy the resource. Cannot affect or mutate the resource.
Write	Has read access plus permission to change the resource.
Admin	Has write access plus the ability to delete the resource.

Table 1: Supported access types

For example, user1 could create a volume and give Read access to group1. This means that only user1 can mount the volume. However, group1 can clone the volume. When a volume is cloned, it is owned by the user who made the request.

Volume access can be viewed by looking at the access metadata of a volume. This can be done using the `pxctl volume access show` command.

```
[root@master-1 demos]# kubectl pxc pxctl v access show  pvc-98343cb7-c79b-42d2-8b01-2454cf607e82
>> Running pxctl on ip-10-0-144-118
Volume:  707340059578410116
Ownership:
  Owner:  db-ops@purestorage.com/db-ops
```

Over-the-wire Backup Encryption

Portworx Enterprise volumes with cluster-wide or per-volume encryption are one way Portworx enables encryption techniques; another example is via PX-Backup. When backing up your OpenShift applications, PX-Backup admins can provide a “Encryption Key” to a PX-Backup backup location so that data is sent encrypted in transit.

> Note: This section does not cover installation and configuration of PX-backup. Please refer to the Secure Backup and Restore for Red Hat OpenShift section for more information.

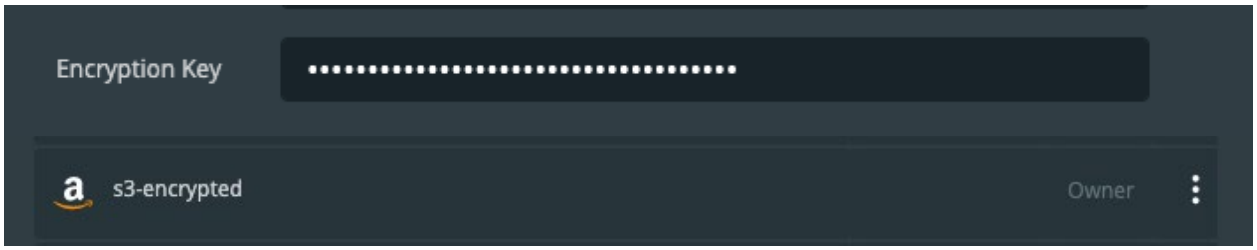


Figure 57: Encryption key

The example uses an Amazon S3 compatible backup location as a backup target. The examples below show the in-transit data when encrypted and unencrypted.

Encrypted:

```
20:57:12.705491 IP ip-10-0-123-45.us-east-2.compute.internal.42328 > s3.us-east-
2.amazonaws.com.https: Flags [P.], seq 1930:2511, ack 2829, win 1464, length 581
        0x0000:  4500 026d 423e 4000 3f06 8440 0a00 df57  E..mB>@.?..@...W
        0x0010:  34db 54da a558 01bb b978 b1b0 6872 bd02  4.T..X...x..hr..
        0x0020:  5018 05b8 756c 0000 1703 0302 4000 0000  P...ul......@...
        0x0030:  0000 0000 2aad 9c1a e84a ee27 d1ec c624  ....*....J.'...$
        0x0040:  023a da24 b206 36d8 3954 adec b894 a729  .:.$..6.9T.....)
        0x0050:  b7fe 731c 3f7f 1981 bb58 4fbe 1f11 2226  ..s.?....XO..."&
```

Unencrypted

```
20:53:17.639164 IP ip-10-0-123-45.us-east-2.compute.internal.57222 > s3.us-east-
2.amazonaws.com.http: Flags [P.], seq 689:1428, ack 678, win 226, length 739: HTTP: PUT /backup-
user-3/db-ops/test-unencrypted-01-3f8cc4b85224/namespaces.json HTTP/1.1
        0x0020:  5018 00e2 759a 0000 5055 5420 2f70 782d  P...u...PUT./px-
        0x0030:  6261 636b 7570 2d72 7761 6c6c 6e65 722d  backup-user-
        0x0040:  332f 6462 2d6f 7073 2f74 6573 742d 756e  3/db-ops/un
        0x0050:  656e 6372 7970 7465 642d 3031 2d32 3832  encrypted-01-282
        0x0080:  6363 3462 3835 3232 342f 6e61 6d65 7370  cc4b85224/namesp
        0x0090:  6163 6573 2e6a 736f 6e20 4854 5450 2f31  aces.json.HTTP/1
              0x0250:  3d0d 0a43 6f6e 7465 6e74 2d54 7970 653a  =..Content-Type:
        0x0260:  2074 6578 742f 706c 6169 6e3b 2063 6861  .text/plain;.cha
        0x0270:  7273 6574 3d75 7466 2d38 0d0a 582d 416d  rset=utf-8..X-Am
```

Data Security Audit on Red Hat OpenShift

Data Security Audit

Lastly, even with purpose-built security for Kubernetes, administrators should audit security by providing security audit logs. Portworx provides security audit and access logs so that organizations can help protect critical data, identify security loopholes, create new security policies, and track the effectiveness of security strategies.

The logs are available on each Portworx node at the following locations:

```
/var/lib/osd/log/security/openstorage-audit.log
 /var/lib/osd/log/security/openstorage-access.log
```

Using Elasticsearch, Kibana, and Filebeat, these audit and access logs can be captured and loaded into Kibana dashboards for data security audit monitoring on OpenShift.

First, install the Elasticsearch Operators:

```
oc create -f https://download.elastic.co/downloads/eck/1.8.0/crds.yaml
oc apply -f https://download.elastic.co/downloads/eck/1.8.0/operator.yaml
```

Then, install Elasticsearch and Kibana with Portworx.

Elasticsearch:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: elastic-pwx-storage-class
provisioner: kubernetes.io/portworx-volume
parameters:
   repl: "3"
   openstorage.io/auth-secret-name: px-user-token
   openstorage.io/auth-secret-namespace: portworx
allowVolumeExpansion: true
reclaimPolicy: Retain
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: elastic-shared-pwx-storage-class
provisioner: kubernetes.io/portworx-volume
parameters:
   openstorage.io/auth-secret-name: px-user-token
   openstorage.io/auth-secret-namespace: portworx
   repl: "3"
```

```
    shared: "true"
allowVolumeExpansion: true
reclaimPolicy: Retain
---
apiVersion: elasticsearch.k8s.elastic.co/v1
kind: Elasticsearch
metadata:
  name: elasticsearch
spec:
  version: 7.14.0
  nodeSets:
  - name: default
    count: 3
    podTemplate:
      metadata:
        labels:
          appname: "elastisearch-app"
    volumeClaimTemplates:
    - metadata:
        name: elasticsearch-data
      spec:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 5Gi
        storageClassName: elastic-pwx-storage-class
    config:
      node.master: true
      node.data: true
      node.ingest: true
      node.store.allow_mmap: false
```

Kibana:

```
  ---
apiVersion: kibana.k8s.elastic.co/v1
kind: Kibana
metadata:
  name: kibana
spec:
  version: 7.14.0
  count: 1
  elasticsearchRef:
    name: "elasticsearch"
  http:
    service:
      spec:
        type: LoadBalancer
        ports:
```

```
        - name: https
          protocol: TCP
          port: 443
          targetPort: 5601
  podTemplate:
    spec:
      containers:
      - name: kibana
        resources:
          limits:
            memory: 1Gi
            cpu: 1
```

Next, apply a filebeat ConfigMap and create a Filebeat daemonset that targets both the Portworx access and Portworx audit logs on the Portworx nodes:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-config
  namespace: portworx
  labels:
    k8s-app: filebeat
data:
  filebeat.yml: |-
    filebeat.inputs:
    - type: log
      enabled: true
      paths:
        - /var/lib/osd/log/security/openstorage-audit.log
       - /var/lib/osd/log/security/openstorage-access.log
     processors:
      - dissect:
          tokenizer: "%{time} %{level} %{msg}"
          field: "message"
          overwrite_keys: true
          target_prefix: "pxaudit"
      - add_cloud_metadata:
      - add_host_metadata:

    cloud.id: ${ELASTIC_CLOUD_ID}
    cloud.auth: ${ELASTIC_CLOUD_AUTH}

    output.elasticsearch:
      hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT:9200}']
      username: ${ELASTICSEARCH_USERNAME}
      password: ${ELASTICSEARCH_PASSWORD}
      ssl.certificate_authorities:
      - /etc/filebeat/certificates/ca.crt
```

```
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: filebeat
  namespace: portworx
  labels:
    k8s-app: filebeat
spec:
  selector:
    matchLabels:
      k8s-app: filebeat
  template:
    metadata:
      labels:
        k8s-app: filebeat
    spec:
      serviceAccountName: filebeat
      terminationGracePeriodSeconds: 30
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet
      containers:
      - name: filebeat
        image: docker.elastic.co/beats/filebeat:7.14.0
        args: [
          "-c", "/etc/filebeat.yml",
          "-e",
        ]
        env:
        - name: ELASTICSEARCH_HOST
          value: "https://elasticsearch-es-http"
        - name: ELASTICSEARCH_PORT
          value: "9200"
        - name: ELASTICSEARCH_USERNAME
          value: elastic
        - name: ELASTICSEARCH_PASSWORD
          valueFrom:
            secretKeyRef:
              name: elasticsearch-es-elastic-user
              key: elastic
        - name: ELASTIC_CLOUD_ID
          value:
        - name: ELASTIC_CLOUD_AUTH
          value:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        securityContext:
          runAsUser: 0
          # If using Red Hat OpenShift uncomment this:
```

```
            privileged: true
          resources:
            limits:
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 100Mi
          volumeMounts:
          - name: cert-ca
            mountPath: /etc/filebeat/certificates
            readOnly: true
          - name: config
            mountPath: /etc/filebeat.yml
            readOnly: true
            subPath: filebeat.yml
          - name: data
            mountPath: /usr/share/filebeat/data
          - name: varlib
            mountPath: /var/lib/osd
            readOnly: true
          - name: varlog
            mountPath: /var/log
            readOnly: true
        volumes:
        - name: cert-ca
          secret:
            secretName: elasticsearch-es-http-certs-public
        - name: config
          configMap:
            defaultMode: 0640
            name: filebeat-config
        - name: varlib
          hostPath:
            path: /var/lib/osd
        - name: varlog
          hostPath:
            path: /var/log
        # data folder stores a registry of read status for all files, so we don't send everything
again on a Filebeat pod restart
        - name: data
          hostPath:
            # When filebeat runs as non-root user, this directory needs to be writable by group (g+w).
            path: /var/lib/filebeat-data
            type: DirectoryOrCreate
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: filebeat
subjects:
- kind: ServiceAccount
  name: filebeat
```

```
    namespace: portworx
roleRef:
  kind: ClusterRole
  name: filebeat
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: filebeat
  namespace: portworx
subjects:
  - kind: ServiceAccount
    name: filebeat
    namespace: portworx
roleRef:
  kind: Role
  name: filebeat
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: filebeat-kubeadm-config
  namespace: portworx
subjects:
  - kind: ServiceAccount
    name: filebeat
    namespace: portworx
roleRef:
  kind: Role
  name: filebeat-kubeadm-config
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: filebeat
  labels:
    k8s-app: filebeat
rules:
- apiGroups: [""] # "" indicates the core API group
  resources:
  - namespaces
  - pods
  - nodes
  verbs:
  - get
  - watch
  - list
- apiGroups: ["apps"]
  resources:
```

```
      - replicasets
    verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: filebeat
  # should be the namespace where filebeat is running
  namespace: portworx
  labels:
    k8s-app: filebeat
rules:
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs: ["get", "create", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: filebeat-kubeadm-config
  namespace: portworx
  labels:
    k8s-app: filebeat
rules:
  - apiGroups: [""]
    resources:
      - configmaps
    resourceNames:
      - kubeadm-config
    verbs: ["get"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: filebeat
  namespace: portworx
  labels:
    k8s-app: filebeat
---
```

Verify the following items after deployment.

- The filebeat configuration looks correct
- Filebeat daemonset is healthy and running on all worker nodes
- Elasticsearch is running and healthy
- Kibana is running and healthy

The Filebeat configuration is below.

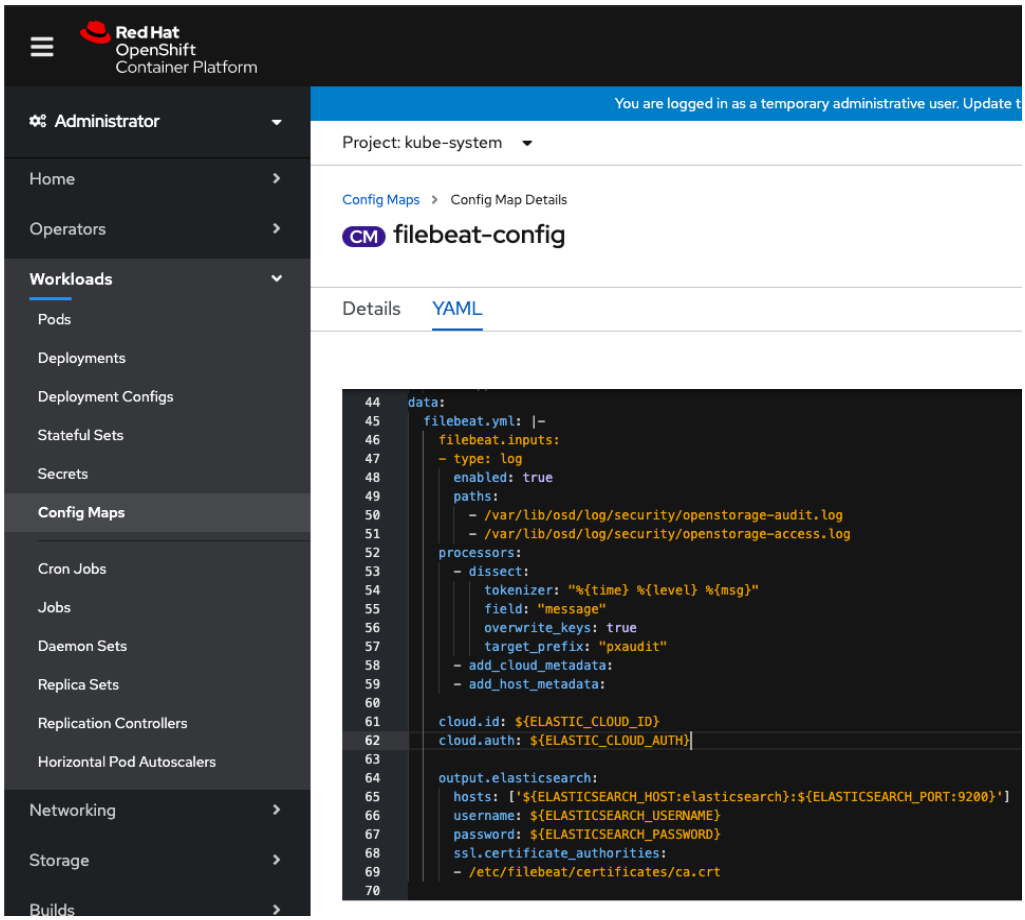


Figure 58: Filebeat configuration

Filebeat is running.

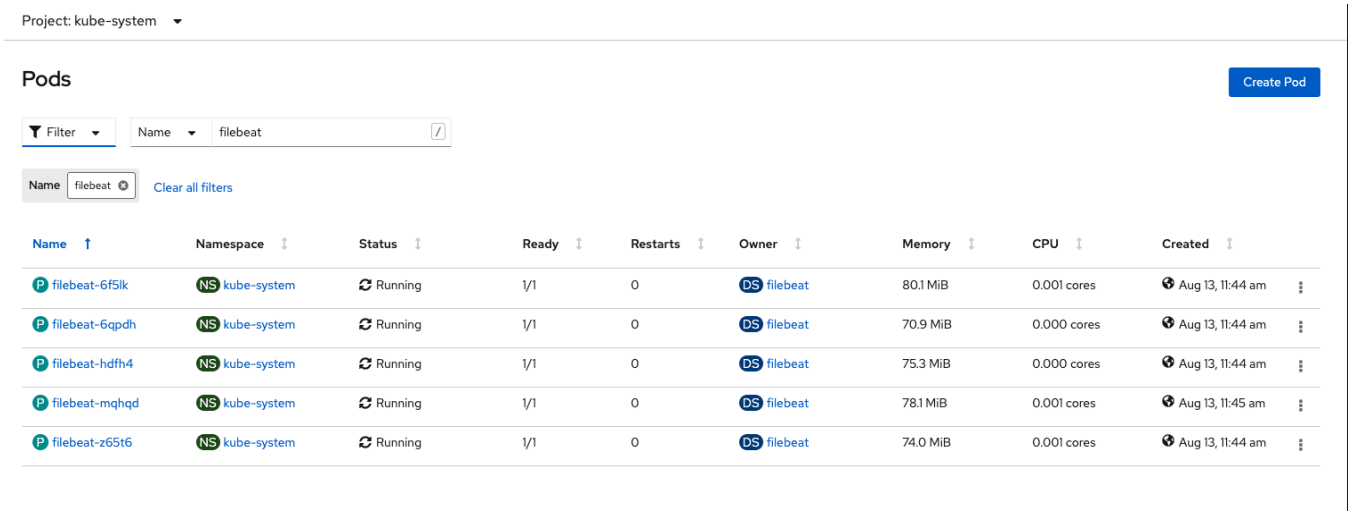


Figure 59: Filebeat status

Elasticsearch is using Portworx volumes and is running.

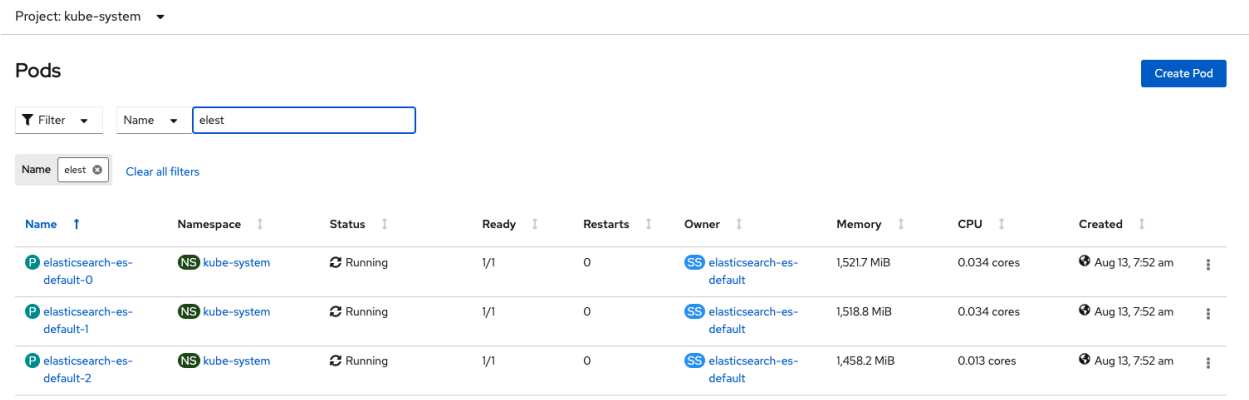


Figure 60: Elasticsearch pods using Portworx volumes

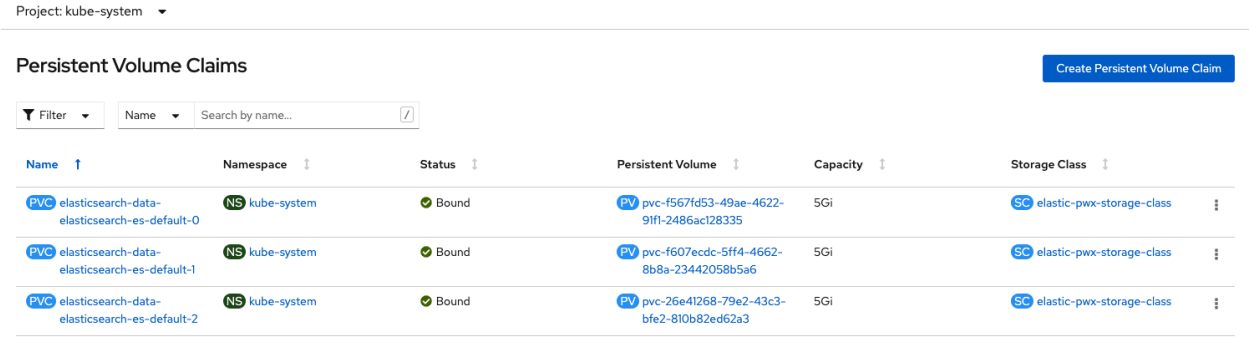


Figure 61: Elasticsearch PVCs using Portworx volumes

Kibana is up and running.

Figure 62: Kibana running

From here, you will be able to monitor your PX-Security audit logs. Connect to the Kibana UI by finding the Kibana load balancer.

```
oc get svc -n portworx kibana-kb-http
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)          AGE
kibana-kb-http    LoadBalancer   172.30.156.232   aafe42-12345.us-east-2.elb.amazonaws.com
443:30561/TCP    63d
```

This guide showed some examples of Access Denied errors working with a few PVC create requests that we made earlier. Those requests used invalid, expired, and read-only tokens. Those are some examples of audit messages that will show up within your dashboards. See below dashboard examples from the data collected by Filebeat.

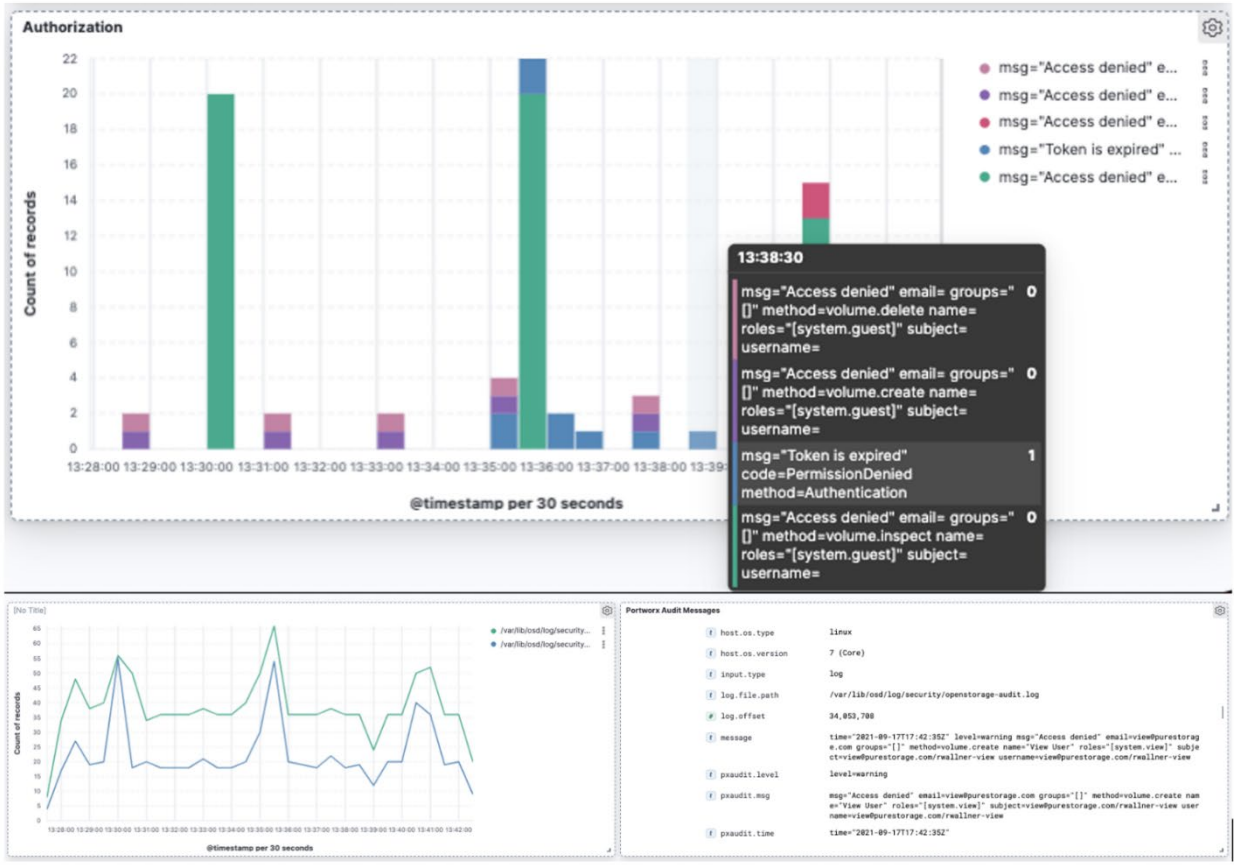


Figure 63: Error messages in Filebeat dashboard

Conclusion

Portworx provides best-in-class enterprise-grade data services for any application running on Red Hat OpenShift on AWS clusters at any scale. Solving for data protection, security, speed, density, and scale, Portworx not only enables efficient, automatic provisioning on top of your Red Hat OpenShift clusters, but also provides advanced features like high availability and replication, automated capacity management, and dynamic provisioning using application-specific StorageClasses (IO_profiles, IO_priority, etc.). Portworx also provides a complete disaster recovery and business continuity solution with PX-Backup and PX-DR. PX-DR allows customers to build synchronous and asynchronous DR solutions for their Red Hat OpenShift clusters. In addition to DR, PX-Backup completes your data protection solution with a Kubernetes-native backup and restore solution that can be leveraged to build architectures for local or remote backup and restore activities. Portworx from Pure Storage is the gold standard when it comes to Kubernetes Data Services, and it brings all its capabilities to Red Hat OpenShift clusters.

Additional Resources

- Portworx Blogs
- Portworx Demos
- How to achieve Disaster Recovery for Red Hat OpenShift
- Seamless Disaster Recovery for Red Hat OpenShift
- Portworx Enterprise OpenShift Documentation
- Portworx Backup OpenShift Documentation

About the Authors

Ryan Wallner is Head of Technical Marketing within the cloud native business unit at Pure Storage, which is responsible for defining solutions around PX-Enterprise, PX-Backup, and PX-Disaster Recovery for Kubernetes applications. Ryan has worked in the data management field for 10 years both as a practitioner in the field of healthcare and as a vendor developing products for emerging technologies. Ryan joined Pure Storage in October 2020 with Pure's acquisition of Portworx Inc.

Chris Kennedy is a Senior Cloud Native Architect within the cloud native business unit at Pure Storage. Chris serves in several roles, ranging from working directly with customers to solve their toughest problems with stateful workloads in Kubernetes to working alongside the technical marketing and the alliances teams to bring valuable field perspectives to the teams. Prior to joining Pure Storage in 2018, Chris was the founder, CEO, and CTO of a managed service provider located in eastern Tennessee. Having started the company from a room in his home, Chris grew the business into a $5 million per year operation and brought on two business partners, allowing him to step away to join Pure.

Tim Darnell is a Senior Technical Marketer within the cloud native business unit at Pure Storage. Tim has held a variety of roles in the two decades spanning his technology career, most recently as a product owner and master solutions Architect for converged and hyper-converged infrastructure targeted for virtualization and container-based workloads. Tim joined Pure Storage in October 2021.

Mayur Shetty is the Global Solution Architect lead for Red Hat's alliance with AWS, driving new joint solutions and working across the entirety of Red Hat's product portfolio and both company's sales organizations worldwide. He has been with Red Hat for over five years, where he was also part of the OpenStack Tiger Team. Prior to Red Hat, he worked as a Senior Solution Architect driving solutions with the OpenStack Swift, Ceph, and other Object Storage software. Mayur also led ISV Engineering at IBM, creating solutions around Oracle database and IBM Systems and Storage. He has been in the industry for almost 20 years and has worked on the Sun Cluster software at Sun Microsystems.

Pure Storage, Inc.
650 Castro Street, #400
Mountain View, CA 94041

PS2274-02 08/2022