PURESTORAGE®

TECHNICAL WHITE PAPER

Highly Available Jenkins at Scale with Portworx Enterprise and AWS EKS

Gain an enterprise-grade CI/CD platform in the cloud.

Contents

About Jenkins

Jenkins is an industry-standard tool used by the DevOps community to orchestrate and schedule software build processes and continuous integration and continuous delivery (CI/CD) pipelines. Jenkins' architecture of controller and agent nodes lends itself well to dynamic environments, like those available in the public cloud such as Amazon Web Services (AWS). With the accelerated adoption of containers and new automation and orchestration tools like Kubernetes, developers are turning more frequently to managed solutions like Amazon Elastic Kubernetes Service (Amazon EKS). This ability to automate software builds, testing, quality checks, and continuous deployment brings great flexibility to how Jenkins can be leveraged. The flexibility of the cloud-native technologies has brought dramatic improvements in "time to value," as well as providing both the development and deployment environments for the enterprise.

Introduction

Running production-grade and enterprise-ready Jenkins in the cloud requires an elastic, resilient, and fault-tolerant architecture. Jenkins agents are very memory- and IO-intensive workloads and the on-demand nature of the cloud is ideal for this. However, the need for high-performance access to data and automating processes around both provisioning and protecting that data has driven many organizations to choose between high availability (HA) or performance. Amazon Elastic Block Store (Amazon EBS)volumes are an ideal option to meet the performance requirements of a CI/CD pipeline, but they are bound by the borders of the availability zone (AZ) where they were created. Customers trying to implement HA workloads in the cloud can benefit from Portworx® solutions to enable cross-AZ replication, business continuity, and data protection with a cloud-native approach. Portworx by Pure Storage® provides an automation and orchestration layer for data management that allows enterprises to maintain the high availability they need while running Jenkins on EBS.

Portworx is a data management solution that serves applications and deployments in Kubernetes clusters. Portworx itself is deployed natively within Kubernetes and extends the automation capabilities down into the infrastructure to remove the complexities of managing data. Portworx provides simple and easy-to-consume storage classes that are usable by stateful applications in a Kubernetes cluster. In AWS, Portworx does this by claiming EBS volumes that are attached to the worker nodes of a Kubernetes cluster (EC2 instances). These volumes are then abstracted by the Portworx data management control plane to deliver a storage pool that offers and automatically provisions container granular volumes from these available resources. When an application like Jenkins creates a persistent volume claim (PVC) with a Portworx storage class, Portworx will automatically provision container volumes and address the capacity, level of performance, data protection, security, and availability required for the application. Portworx is topology-aware and ensures that as part of the provisioning process, a replica of the data is maintained on another node in another availability zone. In the event of a failure, Portworx influences the Kubernetes scheduler to restart Jenkins in another AZ alongside the replicated data in a matter of seconds. With Portworx, clusters can be *highly dense*, meaning that you can greatly scale the number of containers per host. While the recommendation from Kubernetes is 100 pods per VM, Portworx has customers routinely running 200-300 pods per host.

Data protection is critical for any IT endeavor, but traditional, virtual-machine-based data protection solutions have proven inadequate for providing recoverable backups of Kubernetes workloads. Built exclusively for containerized applications, Portworx PX-Backup solves these shortfalls and protects your applications, including data, application configuration, and Kubernetes objects. It does this with a single click at the Kubernetes pod, namespace, or cluster level. Enabling application-aware, zero-data-loss backup and fast recovery for even complex distributed applications, PX-Backup delivers true multi-cloud availability.

Reference Architecture Diagram

Below is the reference architecture for AWS EKS and Portworx.

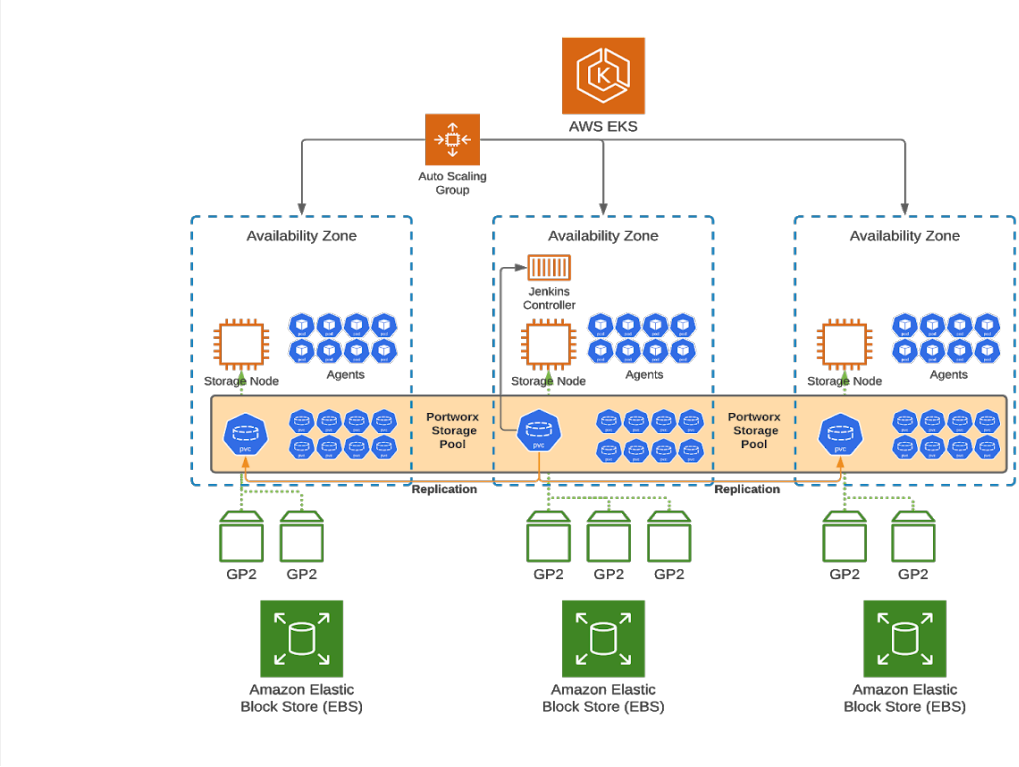


Figure 1. Portworx reference architecture.

Prerequisites

AWS EC2 Instance Sizing

The instances used with Portworx must be sized to handle both the storage operations as well as the applications you will be deploying to the cluster. For Portworx, we recommend four vCPUs, 4GB of RAM, and 10Gbit networking at a minimum. We recommend using these as a baseline, and then factor in your application needs. In this example, we chose M5.xlarge instances (four vCPUs, 16GB RAM, 10G network) for Portworx, Jenkins, and agent pods to use. Your workload needs may vary.

Utilities

To complete the steps outlined in this guide, you will need to install and configure the following utilities:

AWS CLI v2: This is the primary component for interacting with AWS Cloud resources from the command line, and it will be needed for additional utilities to handle authentication from your management workstation. Additional information and installation instructions are available at https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html.

EKSCTL: This utility was developed for managing EKS clusters, and it is used here to deploy EKS resources. To install the latest release of eksctl, follow the instructions at https://eksctl.io/introduction/#installation.

KUBECTL: This is the primary CLI utility for Kubernetes, and it will be used extensively in this guide. You can install it using the instructions here: https://kubernetes.io/docs/tasks/tools/.

HELM v3: Helm is a popular package manager and deployment tool for Kubernetes, allowing application developers to package applications in a templated format enabling consistent deployments with all the necessary components included. Helm v3 should be installed based on the instructions at https://helm.sh/docs/intro/install/.

Configuring AWS EKS and Portworx

Documentation on creating a Portworx-enabled EKS cluster can be found on the Portworx website at https://docs.portworx.com/portworx-install-with-kubernetes/cloud/aws/. Portworx can also be deployed easily via the AWS Marketplace located here: https://portworx.com/awsmarketplace.

The recommended method to deploy a Portworx-enabled EKS environment is to use the eksctl utility and provide a configuration file. The reference guide can be found here: https://docs.portworx.com/portworx-install-with-kubernetes/cloud/aws/aws-eks/eksctl/eksctl-operator/.

Depending on the performance needs and the number of simultaneous agent tasks an organization might decide to run, implementing Portworx on EKS provides the flexibility to grow your development environment as your needs change. This unique functionality allows optimal resource use and provides true flexibility in how the deployment operates and scales. The number of storage nodes is entirely configurable. It is defined in the eksctl configuration file and can be adjusted to suit your needs at the time of deployment or modified later as utilization might change.

Here is a link to the available deployment architectures to help guide your decision: https://docs.portworx.com/cloud-references/deployment-arch/.

The first step involves granting Portworx permissions to create and attach EBS volumes. It is a critical step. To keep this process simple, we suggest implementing this at the instance level with an identity and access management (IAM) policy attached through the eksctl configuration file.

In the example `config.yaml` file below, the name of the segment identifier (SID) is set to "EKSPortworxEC2mgmt". The Amazon resource name (ARN) is then included as part of the configuration for your node groups so the provisioned hosts can perform the required tasks.

The IAM policy can be created using the AWS CLI or IAM console. In the console, select the option to create a new IAM policy. Navigate to the JSON tab and define your policy. Here is an example IAM policy that includes all the needed permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": " EKSPortworxEC2mgmt",
            "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": [
                "*"
            ]
        }
    ]
}
```

If you are also deploying Portworx PX-Backup, then also create the following IAM policy using the JSON tab. This policy is named "EKS_PXBackup_Permissions"' for this example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {

            "Sid": "EKS_PXBackup_Permissions",
            "Effect": "Allow",
            "Action": [
                "ec2:DeleteSnapshot",
                "ec2:DescribeInstances",
                "ec2:CreateTags",
```

```
                "ec2:CreateSnapshots",
                "ec2:DescribeVolumes",
                "ec2:CreateSnapshot",
                "ec2:DescribeRegions",
                "ec2:DescribeSnapshots",
                "ec2:CreateVolume"
            ],
            "Resource": "*"
        }
    ]
}
```

Once the policies have been created, create your cluster. Note that the resulting IAM policies are attached to the node group in the eksctl configuration file below.

Deploy EKS cluster using a variation of the following config.yaml file:

> NOTE: Values in <brackets> should be changed to reflect your needs.

NOTE: For NodeGroup a minimum of three storage nodes are required. These can perform both workload and storage operations. More than three can be allocated based on a customer's need; however, it is recommended that the number of storage nodes per availability zone be changed in the Portworx spec generator or example file below to reflect any changes to the total number of storage nodes.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: <Cluster Name>
  region: <Region>
  version: 1.20
iam:
  withOIDC: true
addons:
- name: vpc-cni
  attachPolicyARNs:
  - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
managedNodeGroups:
  - name: storage-node
    instanceType: <m5.xlarge> # Select Instance type with minimum 4 vCPUs, 8Gi of Memory, and 10Gbit
networking for optimal performance
    minSize: 3 # Minimum configuration - Change to suite needs
    maxSize: 3 # Storage Nodes Min and Max must be set to equal values
    volumeSize: 30
    #ami: auto
```

```
iam:
  withOIDC: true
addons:
- name: vpc-cni
  attachPolicyARNs:
  - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
managedNodeGroups:
  - name: storage-node
    instanceType: <m5.xlarge> # Select Instance type with minimum 4 vCPUs, 8Gi of Memory, and 10Gbit
networking for optimal performance
    minSize: 3 # Minimum configuration - Change to suit needs
    maxSize: 3 # Storage Nodes Min and Max must be set to equal values
    volumeSize: 30
    #ami: auto
    amiFamily: AmazonLinux2
    labels: {role: worker, "portworx.io/node-type": "storage", "px/metadata-node":"true"}
    tags:
      nodegroup-role: worker
    ssh:
      allow: true
  # Set to the path for your key file. See eksctl documentation for more details.
      publicKeyPath: < $HOME/.ssh/id-rsa.pub >
    iam:
      attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess
      - arn:aws:iam::<AWSAccountNumber>:policy/EKSPortworxEC2mgmt
  # If you are also installing PX-Backup, also create the optional IAM Policy above and add it here
also.
     - arn:aws:iam::<AWSAccountNumber>:policy/EKS_PXBackup_Permissions
     withAddonPolicies:
       imageBuilder: true
       autoScaler: true
       ebs: true
       fsx: true
       efs: true
       albIngress: true
       cloudWatch: true
availabilityZones: ['<region-AZ1>', '<region-AZ2', 'region-AZ3']
```

To deploy a new EKS cluster using the above configuration, save the file with the name `cluster-config.yaml` and issue the following command: `eksctl create cluster -f cluster-config.yaml`.

Allow up to 30 minutes after issuing the command to create your EKS cluster for it to become available in your AWS account. There are additional components that may be needed for a production-ready EKS cluster, so please consult Amazon's documentation for items like the ALB/ELB Load Balancer Controller, CertManager, and ExternalDNS for Route53.

Installing and Configuring Portworx on Amazon EKS

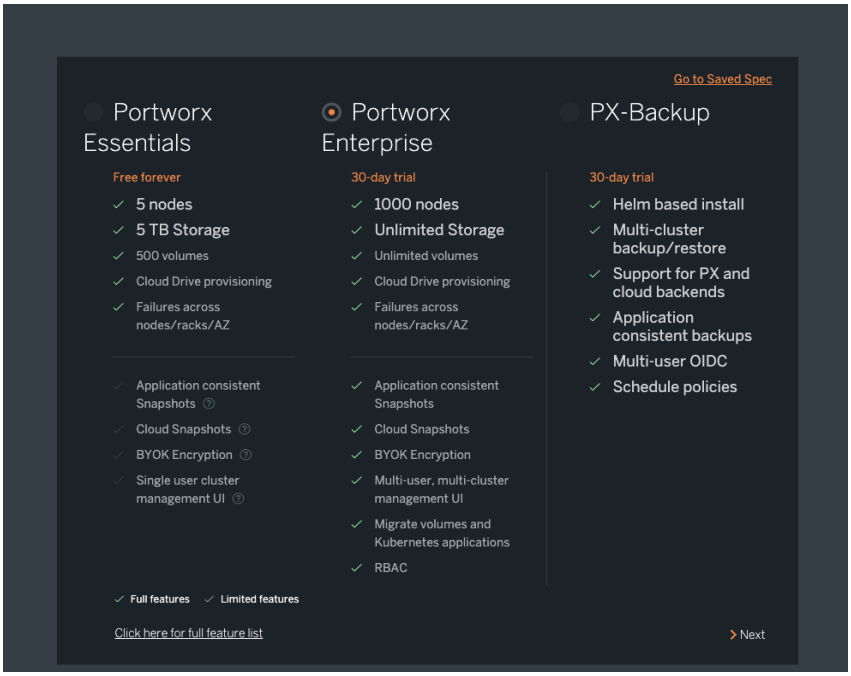
This section provides a walkthrough of the steps to install and configure Portworx on Amazon EKS; however, before proceeding, we recommend that you read the full documentation and instructions on the Portworx website https://docs.portworx.com/portworx-install-with-kubernetes/cloud/aws/aws-eks/. For additional information and options, the documentation website provides extensive resources for you to explore. Also, be sure to review all volume sizes and types in this document and adjust them to reflect your workloads. They are provided as examples and are likely to not align exactly with your needs.

Install the Portworx Operator

To install the Portworx Operator, run `kubectl create -f https://install.portworx.com/2.7?comp=pxoperator`.

Once the operator is installed, we need to provide a specification file to deploy the Portworx cluster. Navigate to https://central.portworx.com to generate the needed Kubernetes manifest that you will deploy to your cluster. We have also included an example specification that you can deploy, but using the Portworx spec generator is preferred.

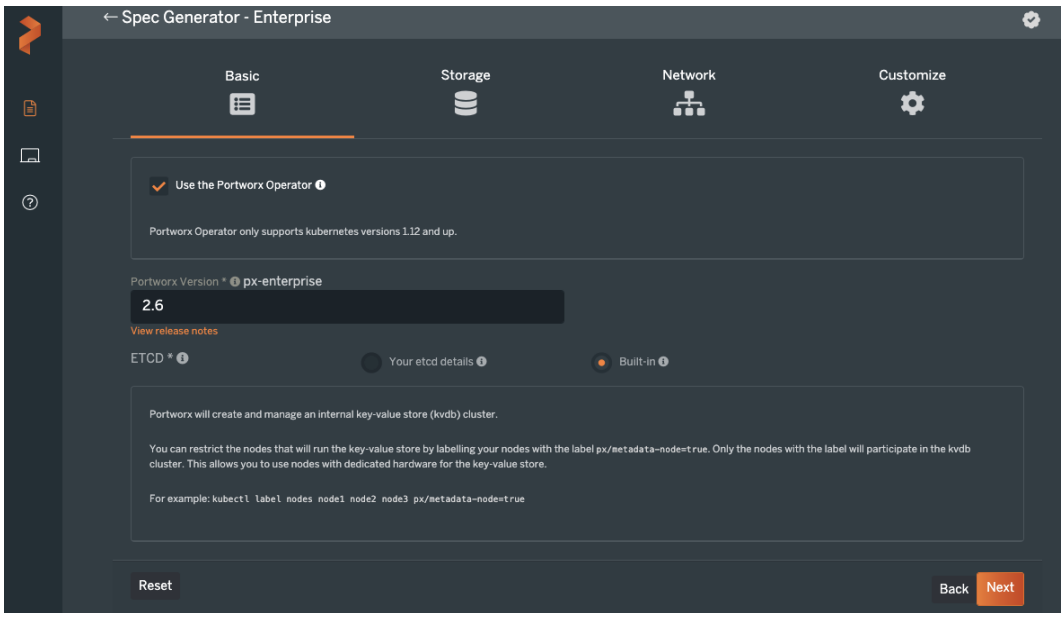
First, you will need to create an account on the Portworx website and log in. After logging in to the website, you are presented with the spec generator wizard. On the first tab, select **Portworx Enterprise** and click **> Next** in the bottom left corner.



On the next page , you'll find the basic configuration options.

On this page, make the following selections:

1. Select **Use the Portworx Operator**.
2. For Portworx Version, select the latest available version (presently 2.7).

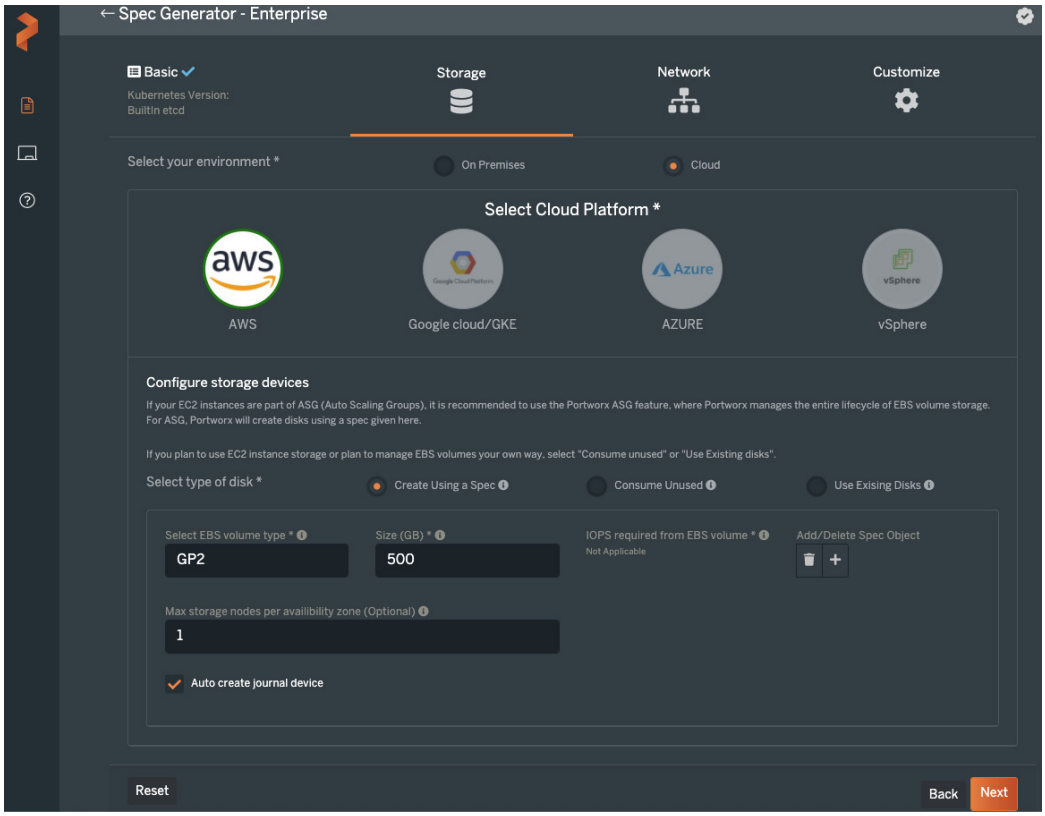


Click **Next**, and you are presented with the storage options. The following steps are for an example configuration for AWS: make sure that the volume type and size are appropriate for your deployment:

1. Select **Cloud**, and then select **AWS**.
2. Configure storage devices:
   a. Select Create Using a Spec.
   b. Under Select EBS Volume Type, select **GP2** or **IO1**.

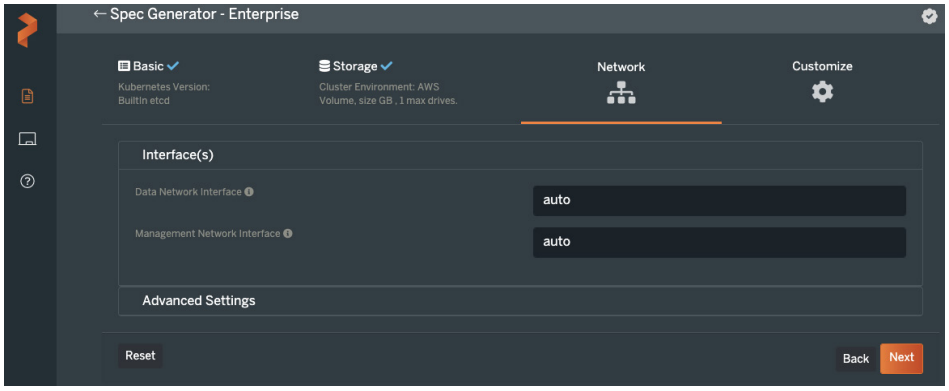
      NOTE: When selecting cloud volume configurations, consider your workloads and the characteristics of the available drive. GP2 provides a solution for balancing cost and performance. IO1 is recommended for production workloads that need a consistent input/output operations per second (IOPS) and throughput.

   c. Size: **500**
   d. Max storage nodes per availability zone: **1**
   e. Select Auto create journal device.



3. Click **Next** to proceed to the network setup.

On this page, unless you have configured multiple network interfaces on your EKS hosts, all values can be left at the default values.



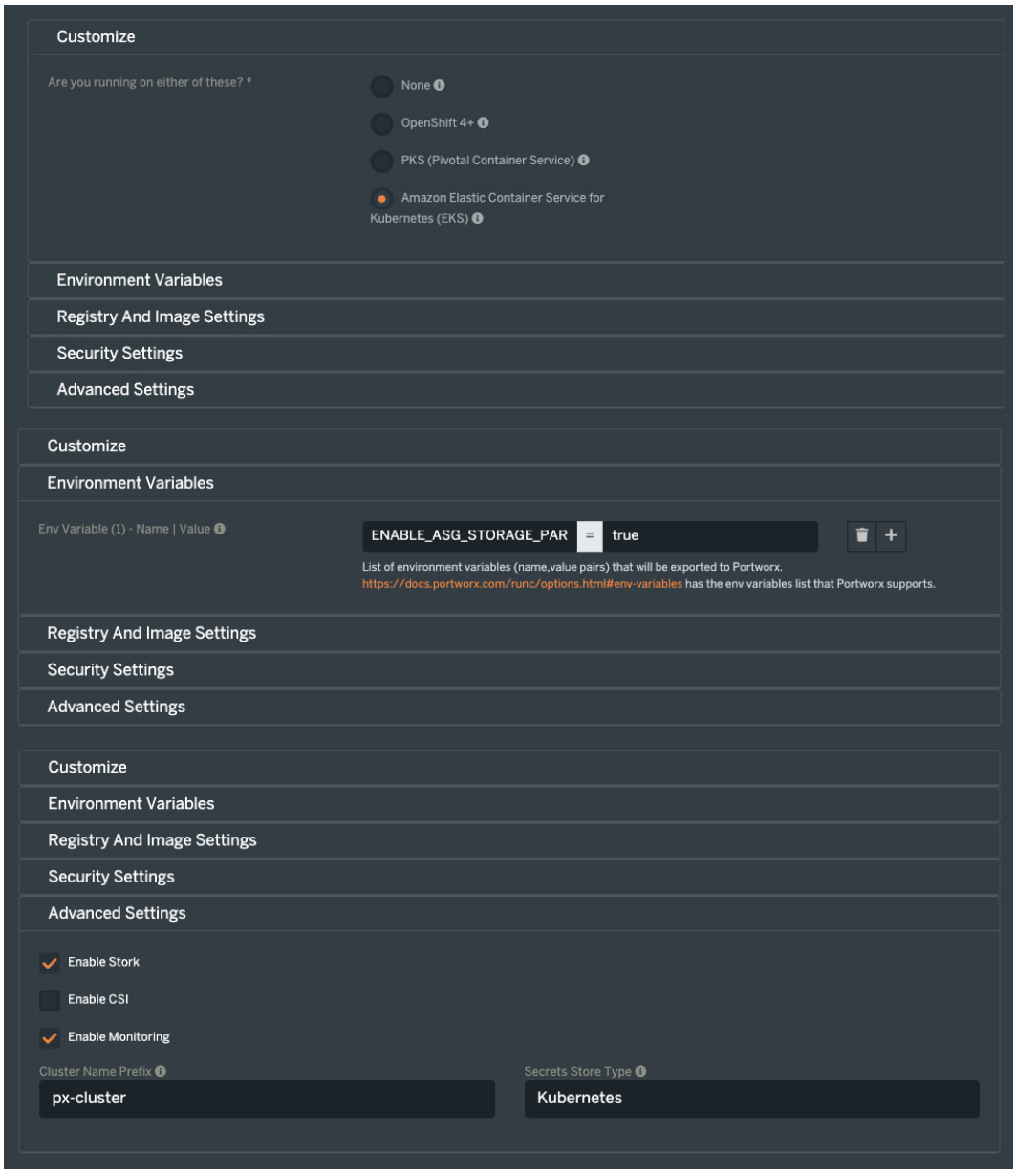
Click **Next** to continue to the Customize page:

1. At the top of the page, select **Amazon Elastic Container Service for Kubernetes (EKS)**.
2. For environment variables, add **ENABLE_ASG_STORAGE_PARTITIONING = true** if you are using Auto Scaling groups.
3. Under Advanced Settings, select the following:
   a. Select Enable Stork.
   b. If needed based on your use case, select **Enable CSI**.

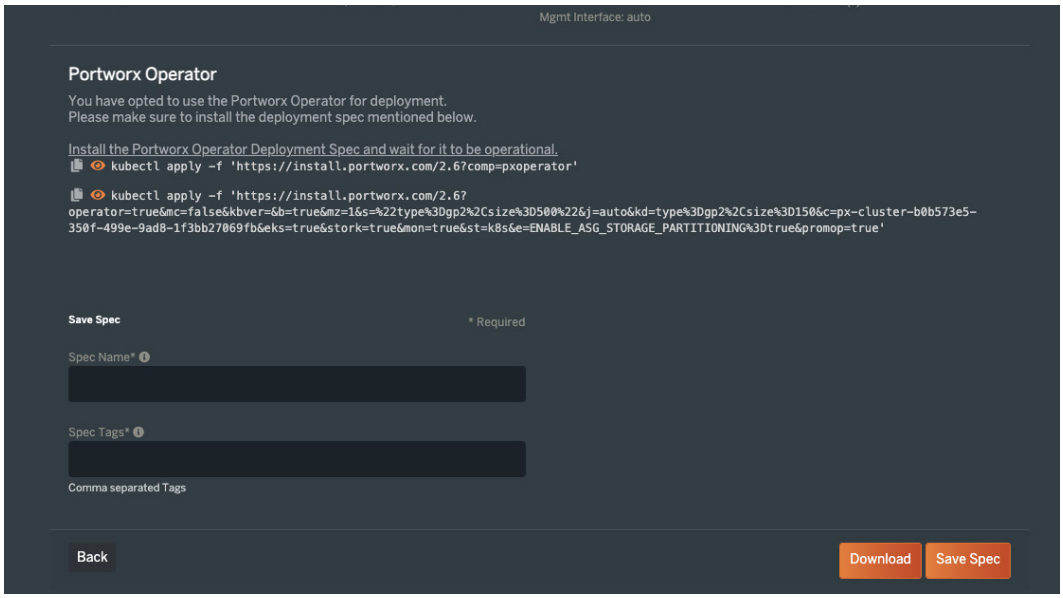
c. Select **Enable Monitoring**.

4. In Cluster Name Prefix, enter an appropriate value, such as the cluster name. .
5. In Secret Store Type, you can select your Kubernetes Secrets, but Portworx can also leverage AWS's KMS service, Hashicorp Vault, or other standard KMS solutions.

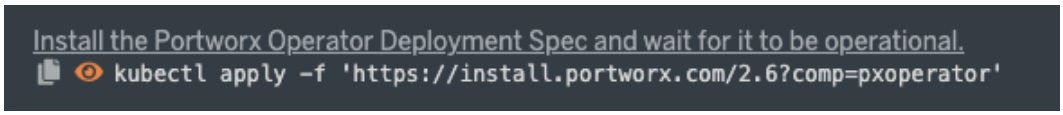
Click **Finish** and agree to the licensing agreement to generate the manifests and spec files.



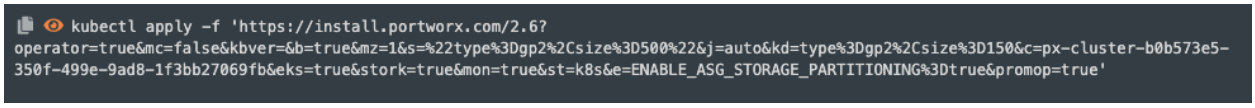
The next page provides the instructions to install Portworx into your EKS cluster.



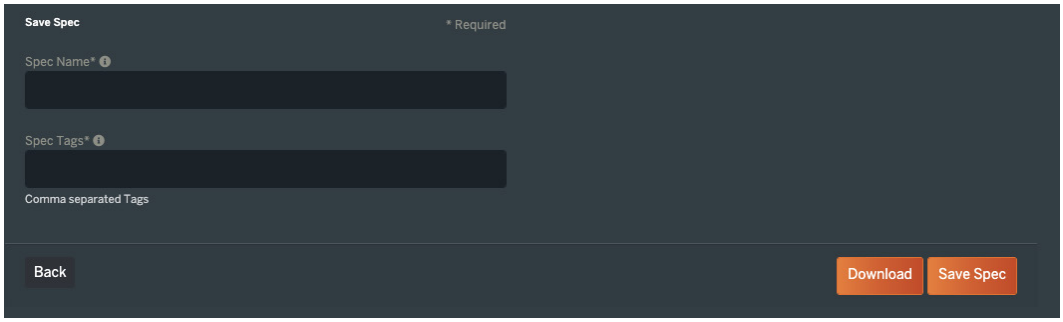
The first step on this page was covered previously, and it is not necessary to repeat the instructions.



Once the operator has been deployed, you can apply your StorageCluster specification by copying and pasting the command shown here on the page:



Finally, you are given the option to save the spec files in PX Central for later reference or reuse by entering a name and any relevant tags, and then clicking **Save Spec**.



Below is an example specification for StorageCluster that you can use. Simply save it in a file named storagecluster.yaml and apply it to the cluster. The Portworx Operator will handle the rest of the deployment. It takes approximately 5-10 minutes for Portworx to fully initialize the cluster.

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
name: px-cluster-jenkins
namespace: kube-system
annotations:
   portworx.io/is-eks: "true"
spec:
 image: portworx/oci-monitor:2.7.2
 imagePullPolicy: Always
 kvdb:
   internal: true
 cloudStorage:
   deviceSpecs:
   - type=gp2,size=500
 journalDeviceSpec: auto
 kvdbDeviceSpec: type=gp2,size=150
 maxStorageNodesPerZone: 3
 secretsProvider: k8s
 stork:
   enabled: true
   args:
     webhook-controller: "true"
 autopilot:
   enabled: true
   providers:
   - name: default
     type: prometheus
   params:
     url: http://px-prometheus:9090
 env:
 - name: "ENABLE_ASG_STORAGE_PARTITIONING"
 value: "true"
 monitoring:
   prometheus:
     enabled: true
     exportMetrics: true
```

Prepare Cluster for Installing Jenkins

To prepare the cluster to install Jenkins, first create a new namespace in your EKS cluster to run Jenkins: `kubectl create namespace Jenkins.`

Now that Portworx is installed and the StorageCluster object has been defined, create two Kubernetes StorageClass objects for Jenkins. The first will be used for the controller pods and the second for agent pods.

Controller StorageClass

The first step to create a StorageClass object for the controller pods is to create and save a file named controller-sc.yaml and apply it using kubectl. The contents of this file should be:

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

Next, create a file named controller-pvc.yaml with the following contents:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: jenkins-home-pvc
  namespace: jenkins
  labels:
    app: jenkins
annotations:
  volume.beta.kubernetes.io/storage-class: px-sharedv4-sc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
```

This will create a ReadWriteMany volume, which can be used for both active and active-passive Jenkins controller deployments.

Agent StorageClass

To create a StorageClass object for agent pods, create and save a file named `jenkins-agent-sc.yaml` and apply it using kubectl. The contents of this file should be:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
  name: jenkins-agent-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "1"
  priority_io: high
allowVolumeExpansion: "true"
```

AutoPilot Rules (Optional)

To allow your EKS cluster to dynamically grow your persistent volume claims, you will need to define the AutoPilot rules. There are two rule types that you can define:

- **Storage pool capacity rule:** This rule allows Portworx to allocate additional EBS volumes automatically through monitoring and API calls. This rule is optional and should be used for workloads where there is not a clear understanding of the storage needs.[1]
- **Persistent volume claim rule:** This type of rule allows the PVC for your Jenkins controller to dynamically grow, allowing you to start small and expand it within the bounds of your Storage Pool based on utilization or a user-defined limit. It is highly recommended that you use this rule for Jenkins deployments.

Please note the thresholds and capacity controls to ensure you are optimizing your pool of resources for your workload.

Below is an example of an optional AutoPilot storage pool rule.

```
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 70%
  - key: "100 * (px_pool_stats_available_bytes / px_pool_stats_total_bytes)"
  operator: Lt
  values:
  - "70"
  # pool total capacity should not exceed 2TB
  - key: "px_pool_stats_total_bytes / (1024*1024*1024)"
 operator: Lt
 values:
 - "2000"
 ##### action to perform when condition is true
 actions:
```

[1]Pool-level capacity management with Autopilot requires additional licensing.

```
 - 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"
```

Recommended AutoPilot Rule

The following AutoPilot rule for resizing the Jenkins controller volume PVC is recommended:

```
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: jenkins
  ##### namespaceSelector selects the namespaces of the objects affected by this rule
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed conditions:
  # volume usage should be less than 70%
  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: "50"
  # volume capacity should not exceed 400GiB
    maxsize: "400Gi"
```

Deploy Jenkins to Your Portworx-enabled EKS Cluster

At this point, you are now ready to install Jenkins. Jenkins is packaged in a Helm chart for easy deployment.

Next, add the Jenkins repository to your Helm installation with the following commands:

```
helm repo add jenkinsci https://charts.jenkins.io
helm repo update
```

Then you will need to obtain the values.yaml file from the Helm chart with the following command:

```
curl -o jenkins-values.yaml \ https://raw.githubusercontent.com/jenkinsci/helm-charts/main/charts/jenkins/values.yaml
```

Review the contents of the values.yaml file for any customizations you may need. To complete the setup of Jenkins with Portworx, create an override-values.yaml file with the following contents. Other customizations may be needed to accommodate your CI/CD pipeline.

```
controller:
 initializeOnce: true
schedulerName: "stork"
agent:
 fsgroup: 1000
 volumes:
 - type: PVC
 claimName: jenkins-home-pvc
 mountPath: /var/jenkins_home
 readOnly: false
 workspaceVolume:
 - type: DynamicPVC
   storageClassName: jenkins-agent-sc
   requestSize: 20
   accessModes: ReadWriteOnce
persistence:
 enabled: true
 existingClaim: jenkins-home-pvc
```

To deploy using these settings, issue the following command: `helm install jenkins -f values.yaml -f override-values.yaml jenkinsci/jenkins -n jenkins`

This will deploy Jenkins to your cluster with Portworx orchestrating your storage operations. Follow the instructions provided at the end of the install to obtain the URL and username/password to access your Jenkins environment.

> IMPORTANT: Jenkins uses a Config Map to maintain the configuration during pod restarts. It is *highly* advised that you review the "Configuration as Code" section under "Managing Jenkins" in the Jenkins UI and use it as a reference to keep the Config Map consistent with your running configuration. If you chose not to do this, some configuration changes can be lost if the pod restarts.

Deploy and Configure a Disaster Recovery Site

Portworx provides a true disaster recovery (DR) and business continuity (BC) solution with the ability to replicate your applications between two different EKS clusters in different availability zones or regions, or other variations of Kubernetes that might live outside the Amazon ecosystem. Using this capability allows businesses to ensure that their most critical applications are always available.

Portworx Enterprise can provide both synchronous and asynchronous disaster recovery configurations, allowing you to build a DR/BC capability for almost any scenario. Metro-DR, or synchronous replication, requires a connection between the sites with adequate bandwidth and a maximum of 10ms latency between the EKS hosts. This is only achievable using a single region and multiple availability zones. This can offer cost savings, but it lacks true business continuity if the selected region suffers a catastrophic event.

For this white paper, region-to-region DR was selected as the preferred DR solution. This is configured as Async-DR using our 3D-Snapshot capabilities that include the Kubernetes objects and manifests that are needed, along with the data volumes to maintain an application-consistent DR site. Replication schedules are configurable by the cluster administrator and offer as low as a 10-minute recovery point objective (RPO) and the ability to restore service in minutes.

Prepare Your AWS Cloud for Disaster Recovery

Portworx Disaster Recovery–enabled clusters use an S3 bucket as an intermediary storage location that is accessible from both EKS clusters. As part of the configuration process, we use an AWS access key and a secret key. To ensure that you maintain a secure environment, it is best practice and strongly recommended that a separate IAM identity is created for this purpose. The identity should have the two IAM policies defined at the beginning of this guide attached to it, as well as policies allowing interaction with S3. Once you have created this identity and created access and secret keys, make sure to save them for future use in this white paper. When AWS account information is part of a step, please use this new IAM Identity to satisfy the requirements.

To configure Disaster Recovery, first select a second region and deploy a similar EKS cluster. The most important factor is that you need at least three storage nodes to accommodate the configuration used in the Jenkins home directory storage class. If look back, you will see that a replication factor of three is used to ensure the highest availability.

You can follow the above instructions to deploy a second cluster by changing the values necessary to reflect the use of a different region. Once that is done, it is necessary to create a VPC peering connection or transit gateway to allow connectivity between the clusters. See the links below to determine which will work best in your environment. Also, be sure to add rules to the security groups associated with both clusters to allow bidirectional communication. Once connectivity has been verified, you are ready to create a cluster pair and configure replication. For more information on VPC peering, see https://docs.aws.amazon.com/vpc/latest/peering/create-vpc-peering-connection.html. For more information on transit gateways, see https://docs.aws.amazon.com/vpc/latest/tgw/tgw-getting-started.html.

Create a Cluster Pair and Admin Namespace

To begin pairing the two clusters for replication, first establish an admin namespace. This is recommended so cluster administrators can replicate any namespace in the cluster without having to have full access to the kube-system namespace. Here are the instructions to configure and begin using an admin namespace with Portworx Enterprise:

1. First, obtain the storkctl command-line utility from either cluster using the platform-specific instructions found on the Portworx documentation site. There are instructions for Linux, macOS, and Windows available. More information is available at: https://docs.portworx.com/portworx-install-with-kubernetes/disaster-recovery/async-dr/.
2. Once you have the storkctl utility installed, you then need to provide credentials to Stork so it can access the needed resources for migrations. Use the IAM Identity's credentials in your AWS account, and ensure that this account has the same roles that were applied to the EKS clusters during deployment.

> **IMPORTANT:** Until otherwise instructed, the following steps *must* be performed on both the source and destination clusters.

3. Create a Kubernetes namespace to serve as the admin namespace: `kubectl create namespace migrations`
4. Next, create a secret in this new namespace using the AWS access key ID and secret access key created for the previously discussed IAM Identity. You should receive confirmation that the Kubernetes secret was created.

```
kubectl create secret generic --from-literal=aws_access_key_id=<AWS_ACCESS_KEY_ID> --from-
literal=aws_secret_access_key=<AWS-SECRET-ACCESS_KEY> -n kube-system aws_creds
```

5. Edit the StorageCluster object in the kube-system namespace and add a volumeMount for the newly created secret. To edit the StorageCluster object, run `kubectl edit stc -n kube-system.`
6. Move down through the manifest and find the `stork:` configuration and modify it to look like the following:

```
  stork:
    args:
      admin-namespace: migrations
      health-monitor-interval: "30"
      webhook-controller: "true"
    enabled: true
    volumes:
    - mountPath: /root/.aws/
      name: aws-creds
      readOnly: true
      secret:
        secretName: aws-creds
```

Upon saving the changes to the StorageCluster object, a new replica set will be created and the Stork pods will restart to implement the changes in a rolling update.

7. On the destination cluster obtain the Cluster ID by issuing the following command:

```
PX_POD=$(kubectl get pods -l name=portworx -n kube-system -o jsonpath='{.items[0].metadata.name}')
&&  kubectl exec $PX_POD -n kube-system -- /opt/pwx/bin/pxctl status | grep UUID | awk '{print $3}'
```

Copy the output to a notepad-type app for use in the following steps. It will be referenced as <destination_cluster_uuid>.

To use the Portworx CLI, you can either log into one of your worker nodes and run the command from /opt/pwx/bin/ where it is located, or more conveniently create a local alias to access the CLI tool from inside a Portworx pod. For Linux or macOS, add the following to your shell configuration file (bash, zsh, etc.)

```
export PX_POD="$(kubectl get pods -l name=portworx -n kube-system -o
jsonpath='{.items[0].metadata.name}')"
alias pxctl="kubectl exec $PX_POD -n kube-system -- /opt/pwx/bin/pxctl"
```

8. Now provide Portworx with the IAM Identity's AWS credentials. The command to use is:

```
pxctl credentials create --provider s3 --s3-access-key <AWS_ACCESS_KEY_ID> --s3-secret-key <AWS-SECRET-ACCESS_KEY> --s3-region <region of source cluster> --s3-endpoint s3.amazonaws.com --s3-storage-class STANDARD clusterPair_<destination_cluster_uuid>
```

As a reminder, all steps for creating a cluster pair to this point *must* be performed on both the source and destination clusters. The only exception is the command to obtain the Cluster ID.

IMPORTANT: The following steps will be performed on specific clusters.

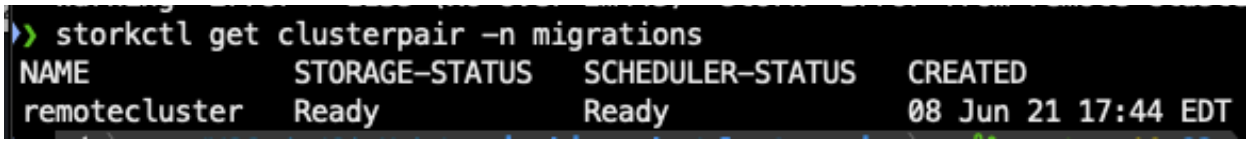
9. To create the cluster pair, on the *destination* cluster, first, run this command and pipe it to a file for editing: `storkctl generate clusterpair -n migrations remotecluster >> clusterpair.yaml`.

   Once the file has been created, edit the file and replace the contents of 'options' in the file with the following.

```
options:
  ip: "<IP of any Portworx node in DR Cluster>"
  port: "9001"
  token: "<cluster token from DR Cluster>"
  mode: DisasterRecovery[2]
```

10. To obtain the cluster token, run the following on the destination cluster: `pxctl cluster token show.` Use the output for the token requested above. Once the file has been modified, you will apply it to the source (production) cluster using: `kubectl apply -f clusterpair.yaml`

    You can check the status of the cluster pair with: `storkctl get clusterpair -n migrations.` The output should look like this:



Next, create a schedule policy for the replication. This is where you can configure how frequently the DR replication occurs. Currently, our shortest supported interval is every five minutes, resulting in an RPO of approximately 10 to 15 minutes.

Below is an example SchedulePolicy manifest that configures replication every 15 minutes, as well as a daily, weekly, and monthly example. Configure the replication interval based on your business needs and RPO policies. Multiple schedules can be created if different applications have different requirements.

```
apiVersion: stork.libopenstorage.org/v1alpha1
kind: SchedulePolicy
```

[2] Disaster recovery mode requires licensing additional features; please contact Portworx for more information. This line can be omitted, but every 7th replication interval will be a full copy versus forever incremental copies enabled by the DR license.

```
metadata:
  name: jenkins-dr-policy
  namespace: migrations
policy:
  interval:
    intervalMinutes: 15
  daily:
    time: "10:14PM"
  weekly:
    day: "Thursday"
    time: "10:13PM"
  monthly:
    date: 14
    time: "8:05PM"
```

Next, create a MigrationSchedule manifest that replicates the Jenkins namespace, where you have deployed Jenkins:

```
apiVersion: stork.libopenstorage.org/v1alpha1
kind: MigrationSchedule
metadata:
  name: jenkins-async-dr
  namespace: migrations
spec:
  template:
    spec:
      clusterPair: remotecluster
      includeResources: true
      includeVolumes: true
      startApplications: false
      namespaces:
      - jenkins
      adminClusterPair: remotecluster
      purgeDeletedResources: false
  schedulePolicyName: jenkins-dr-policy
```

Since you are configuring disaster recovery, take note that the options startApplications and purgeDeletedResources have been set to false. Migrations and migration schedules can be used for disaster recovery replication like you are doing here. But they can also be used for one-time application migrations between clusters or namespaces within a cluster by changing the kind to Migrations and omitting schedulePolicyName. This type of migration begins once the manifest is applied.

For disaster recovery, apply the manifest as displayed above to the source cluster; replication will begin shortly after. The first interval is a full copy of the selected namespaces. Every following replication interval uses incremental snapshots to minimize the amount of data sent. Also, on AWS, Portworx uses an S3 bucket to hold the migration data during the replication to help further reduce transport costs.

Each time the replication occurs, a migrations object is created in the migrations namespace. After 5-10 minutes, you can verify the status of the replication by issuing the following commands:

```
kubectl get migrations -n migrations
kubectl describe migrations <selected migration> -n kube-system
```

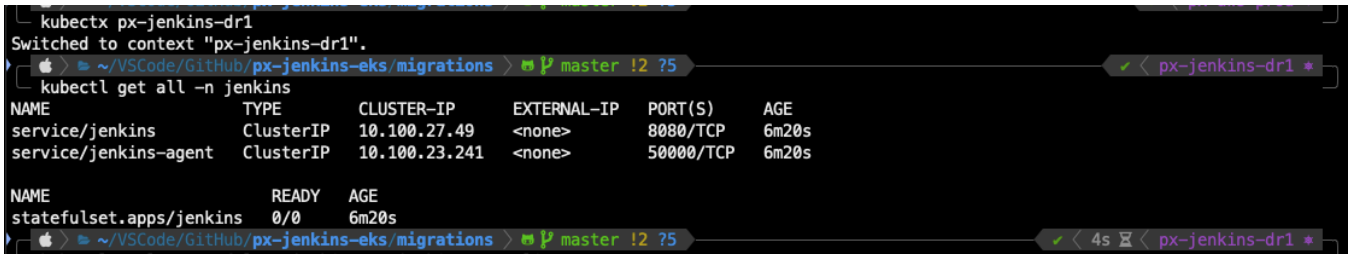
The output will list all objects that are part of the DR migration. For brevity, we have removed the bulk of the output; however, note that at the end of this block, "Stage: Final" has a status of successful.

```
# Some output was removed in the interest of brevity
Name:         jenkins-async-dr-interval-2021-06-09-200912
Namespace:    migrations
Labels:       <none>
Annotations:  <none>
API Version:  stork.libopenstorage.org/v1alpha1
Kind:         Migration
Status:
  Finish Timestamp:  2021-06-09T20:10:51Z
  Resources:
    Group:      core
    Kind:       PersistentVolume
    Name:       pvc-2f05d999-b391-46cc-a4a2-bdee199c98e9
    Namespace:
    Reason:     Resource migrated successfully
    Version:    v1
    Group:      rbac.authorization.k8s.io
    Kind:       Role
    Name:       jenkins-casc-reload
    Namespace:  jenkins
    Reason:     Resource migrated successfully
    Status:     Successful
    Kind:       RoleBinding
    Name:       jenkins-watch-configmaps
    Namespace:  jenkins
    Reason:     Resource migrated successfully
    Status:     Successful
    Version:    v1
  Stage:        Final
  Status:       Successful
  Volumes:
    Namespace:                jenkins
    Persistent Volume Claim:  jenkins
    Reason:                   Migration successful for volume
    Status:                   Successful
    Volume:                   pvc-2f05d999-b391-46cc-a4a2-bdee199c98e9
    Namespace:                jenkins
```

```
      Persistent Volume Claim:  jenkins-plugin-dir
      Reason:                   Migration successful for volume
      Status:                   Successful
      Volume:                   pvc-9a0d9983-ffab-4d7e-b11b-07cffb59c574
      Namespace:                jenkins
      Persistent Volume Claim:  jenkins-plugins
      Reason:                   Migration successful for volume
      Status:                   Successful
      Volume:                   pvc-72e65c45-f916-4f83-9c9a-ead8f96f443f
      Namespace:                jenkins
      Persistent Volume Claim:  sc-config-volume
      Reason:                   Migration successful for volume
      Status:                   Successful
      Volume:                   pvc-e4777535-2513-4cdf-98c5-8a1e9a3cb869
```

Let's look at a few important details. In the migration manifest, you specified that all objects from the jenkins namespace be replicated. This can be validated by looking at the output from querying the objects in the namespace on the target cluster.

Switching to that cluster, you can issue `kubectl get all -n jenkins`



Notice that the Jenkins StatefulSet controller has no instances running. Everything is now replicated to the DR cluster, but it has not been started. Successive replications will keep these objects and the persistent volumes updated based on your replication interval. Also, at any time, you can suspend the replication and scale up the StatefulSet controller to test the viability of your DR site. See the Solution Testing and Validation section for more information.

Install and Configure PX-Backup

Installation

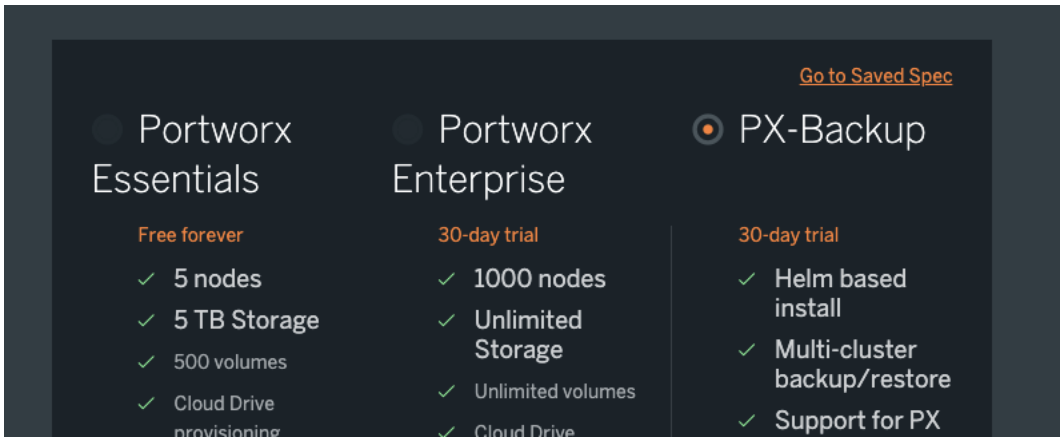
First, create a StorageClass object for the PX-Backup[3] database and other components using the following manifest. Copy this into an editor and save it as px-backup-sc.yaml and then applying it to your cluster.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

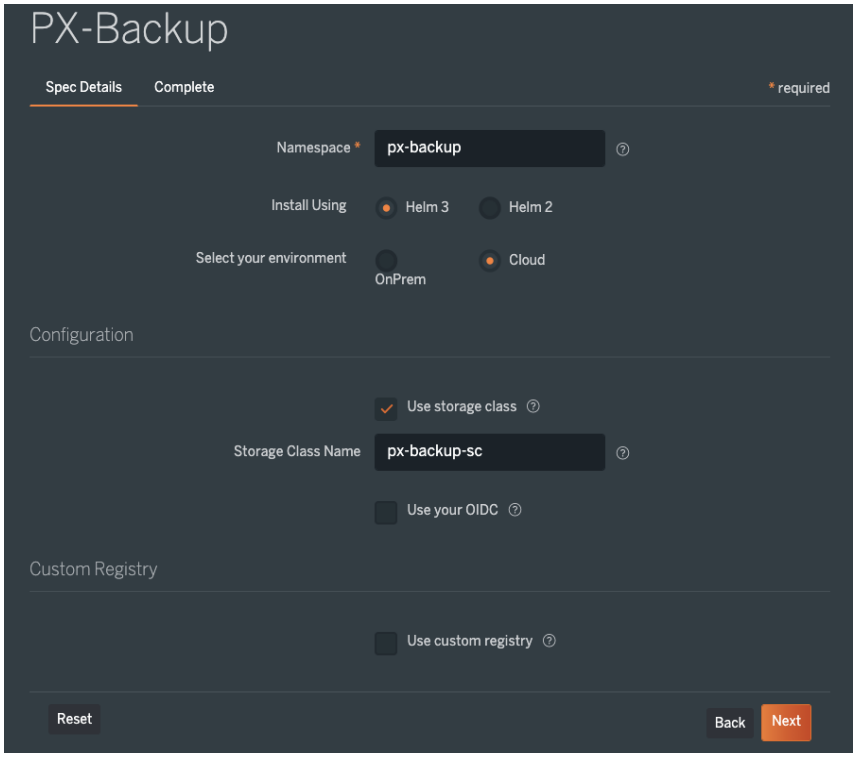
[3] Portworx PX-Backup is a separately licensed product. PX-Backup can be used with any CSI storage solution, including AWS's EBS CSI driver. Please consult with Portworx or AWS for more details.

```
 name: px-backup-sc
provisioner: kubernetes.io/portworx-volume
parameters:
 repl: "2"
 io_profile: auto
allowVolumeExpansion: true
```

To install PX-Backup to your environment, open https://central.portworx.com and log in to your account. In the Portworx spec generator, select **PX-Backup**, and then scroll to the bottom and click **Next**.



Then, on the Spec Details page, under **Select your environment,** choose **Cloud**. In the **Storage Class Name** field, enter "px-backup-sc" from the manifest above that was applied to the cluster. If you are using OpenID Connect (OIDC), you can configure the connection here, as well as if you have a custom registry with the PX-Backup images. Click **Next** to proceed.



On the next screen, you are provided with the Helm commands to install PX-Backup. Since you installed Helm before installing Jenkins, you are ready to issue the commands.

1. First, copy the contents of "Step 1" and apply it to your production cluster: `helm repo add portworx http://charts.portworx.io/ && helm repo update`
2. Now, copy the contents of the left box in "Step 2" and apply it to your production cluster. This will install PX-Backup in the px-backup namespace. The namespace will be created in the process.

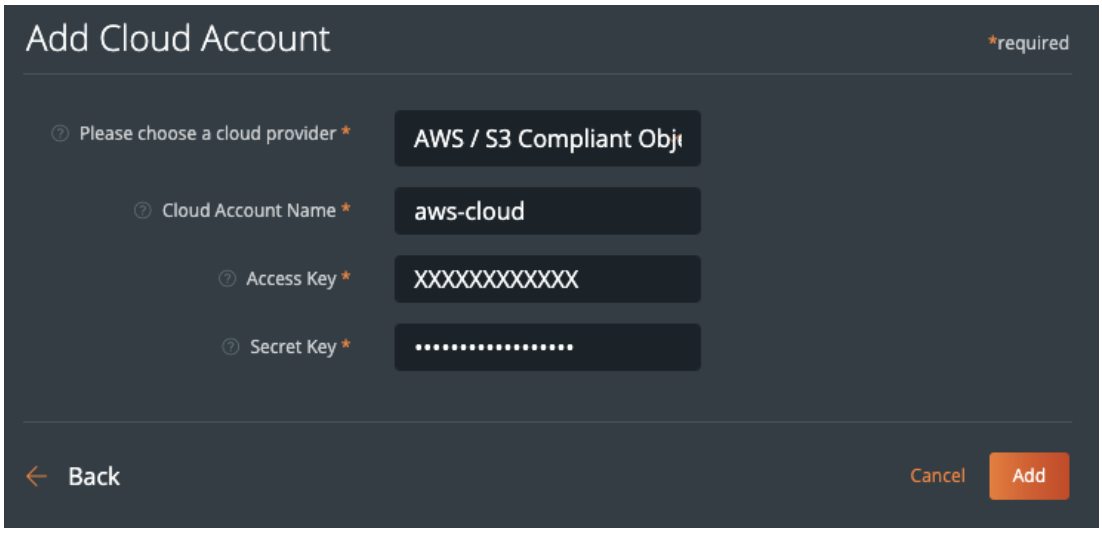
```
helm install px-backup portworx/px-backup --namespace px-backup --create-namespace --version 1.2.3 -
-set persistentStorage.enabled=true,persistentStorage.storageClassName="px-backup-sc"
```

You can monitor the progress of the installation by issuing `watch kubectl get pods -n px-backup` and wait until all pods have reached a running state. Once everything is up and running, you can port-forward to the UI service and access the GUI. The initial login is admin/admin. Change the password on the first login.

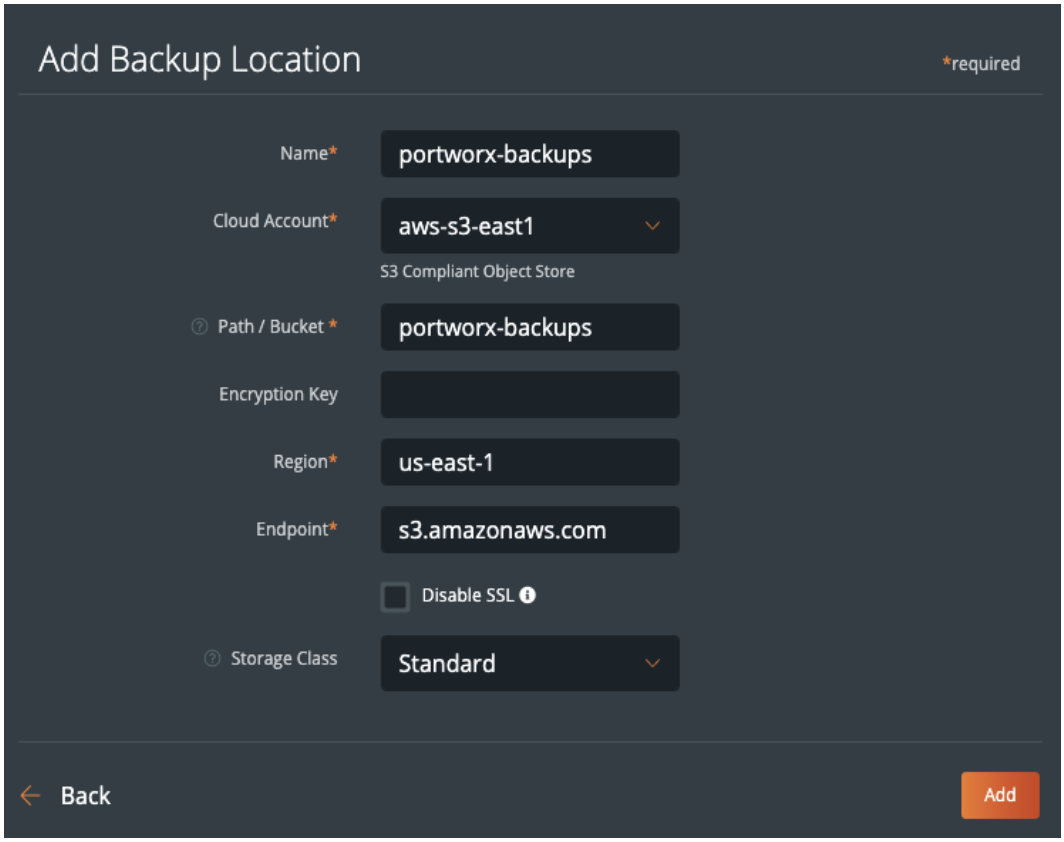
Configure PX-Backup

Once you are logged into the UI for PX-Backup, there are a couple of steps to take to be ready to start protecting your environment.

1. First, add your cloud credentials to PX-Backup so it can access both your cluster resources and the S3 bucket used to store the backup objects. To configure your credentials, click **Settings** in the upper-right corner and choose **Cloud Settings**.
2. On the Cloud Settings page, in the top section, add the IAM Identity's AWS Cloud account credentials used with the Disaster Recovery configuration. Click **+ Add** in the upper-right corner to enter your credentials. Click **Add** at the bottom once the page is complete.



3. Next, click **Backup Location** on the Cloud Settings page. Click **+ Add** in the lower-right corner.



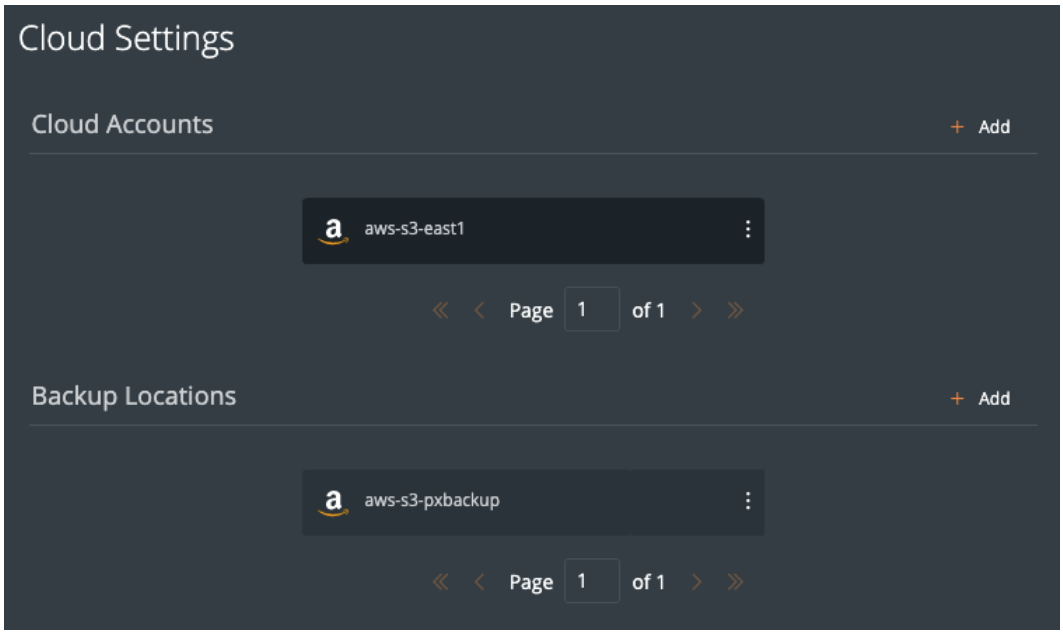
4. Fill out the form. The fields with asterisks (*) are required.
   a. Name: Provide a name for the backup location.
   b. Select your cloud account from the drop-down list.
   c. Provide a bucket name. If the bucket does not exist, it will be created with private access permissions.
   d. Enter an encryption key if you want to encrypt your backups.

      IMPORTANT: Be sure to save the encryption key for restores and backup validation tasks.

   e. Enter the region that your production cluster is in.
   f. The endpoint will be pre-populated.
   g. Leave SSL enabled (do not select **Disable SSL**).
   h. Select **Standard** for the storage class.
   i. Click **Add** to finish.

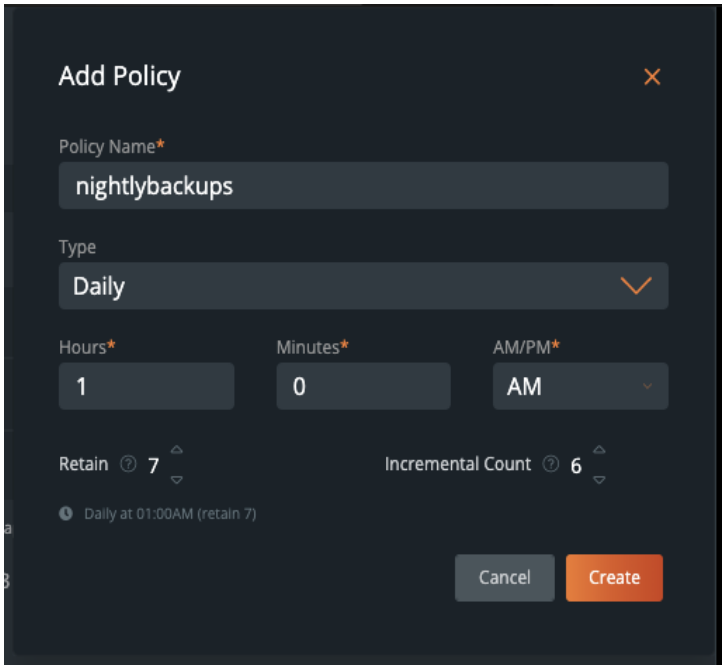
If you want to keep monthly or extended retention backups in Standard-IA, create a second backup location to use with the job definitions.

Once complete, your cloud settings should resemble this:



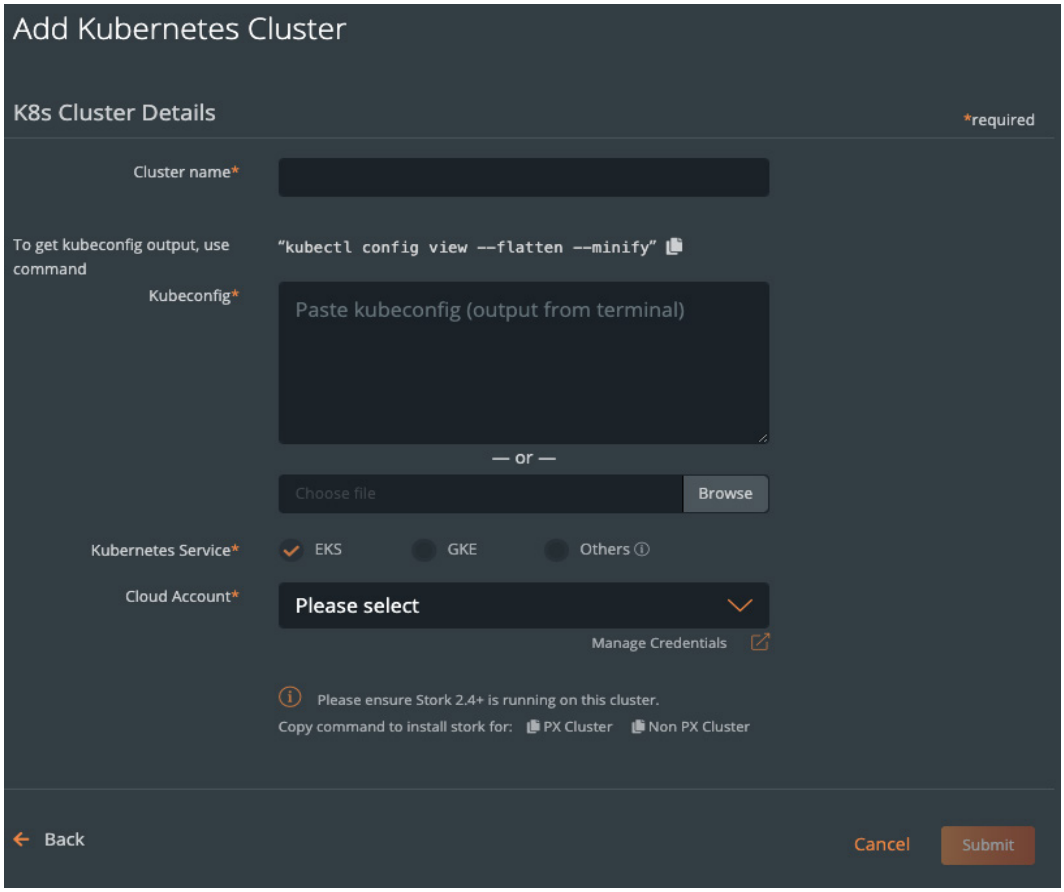
Next, configure your backup schedules.

1. Return to the main screen and click **Settings** again, but select **Schedule Policies**. Then, on the Schedule Policies page, click the plus (**+**) sign to add a new schedule policy.

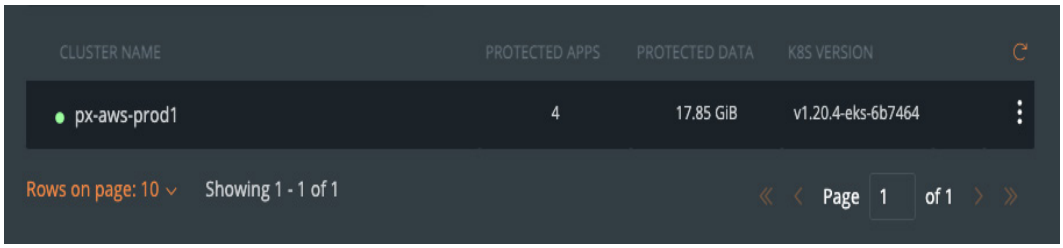


2. Provide a policy name and select the type of policy from the **Type** drop-down list. Above is an example of a nightly backup. You can configure a full GFS Backup schedule or a schedule that meets your organization's backup requirements. It might be necessary to create multiple schedules to implement long-term retention for compliance or regulatory reasons.

3. Next, configure any pre-backup and post-backup rules that you want to use. Example rules are available at https://backup.docs.portworx.com/use-px-backup/backup-stateful-applications/, including rules for Jenkins. If you have jobs that might be running, add the pre- and post-backup rules for Jenkins based on the reference found here: https://backup.docs.portworx.com/use-px-backup/backup-stateful-applications/jenkins/.
4. Finally, add your production cluster to PX-Backup to begin protecting your Jenkins environment. To do this, click **Add Cluster** in the upper-right corner. On the resulting screen, fill in the information as described.

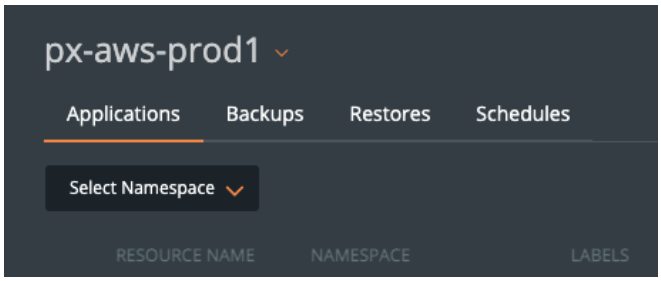


Once you complete this form with the information for your production cluster and click **Submit**, you will be returned to the PX-Backup main page and your cluster will appear with a green dot next to its name.

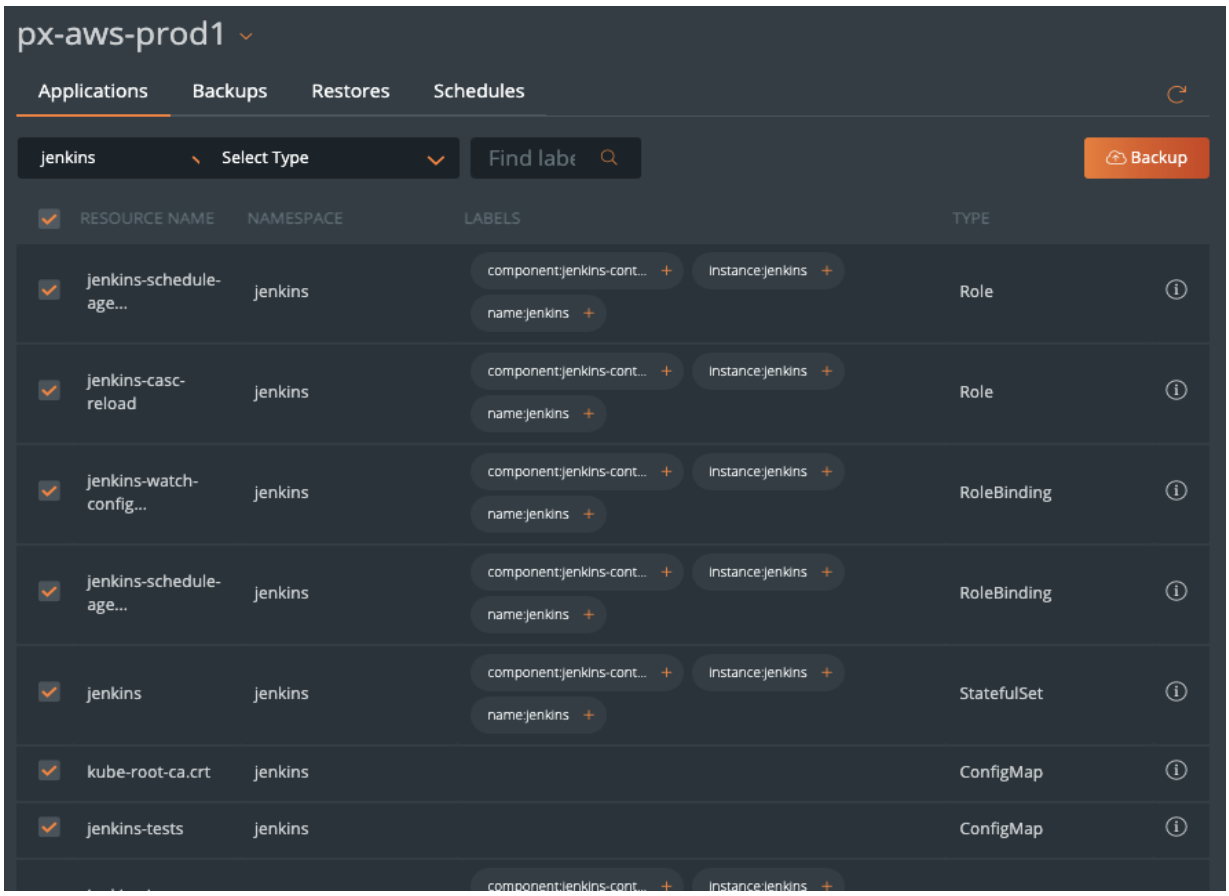


Configure Your First Backup Job

To configure your first backup job, click the name of the cluster and you will be taken to the job configuration screen.



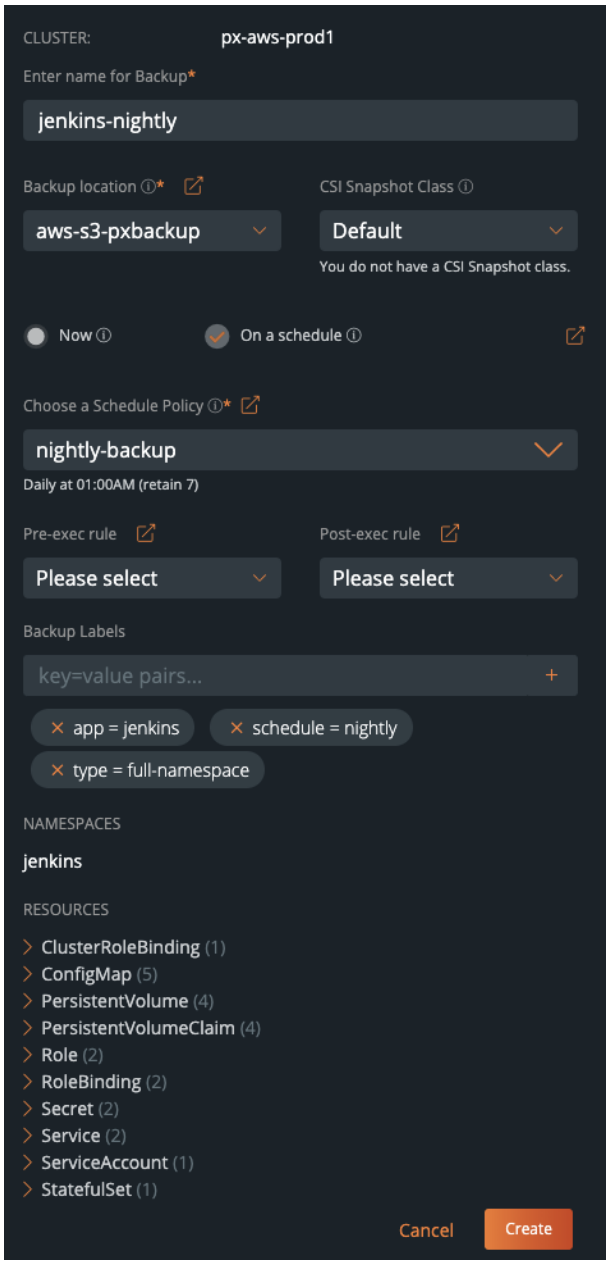
Here, you can select the namespace(s) to back up from. Once a namespace has been selected (you can select all namespaces if you want), then you can also filter by labels or resource types. However, for Jenkins, it is best to back up everything in the namespace. So, select the Jenkins namespace.



Scroll down the page and you will see that all the various types of Kubernetes objects in the namespace are listed. The PVCs and PVs are typically near the bottom. Be sure everything is selected, and then click **Backup**.

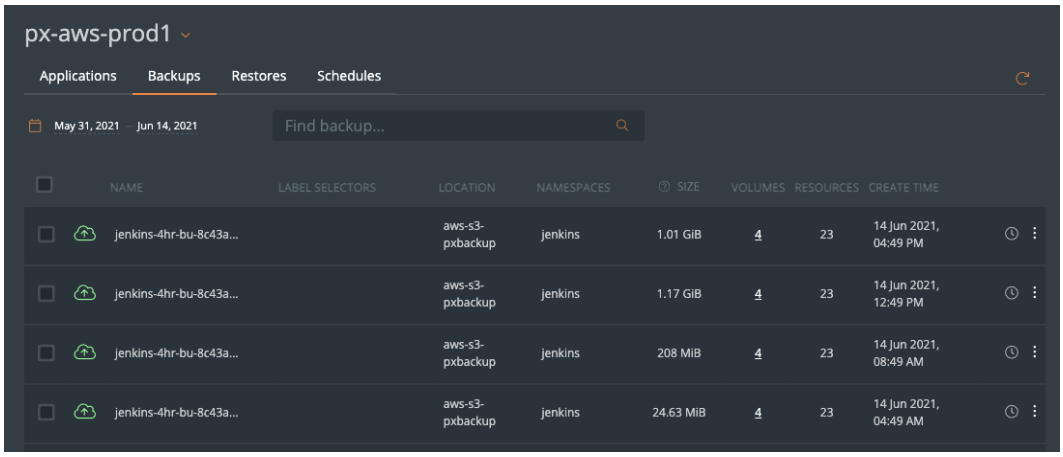
In the Backup window, you will provide a name, and then you can select the necessary options based on your desired backup schedule. Once you have the desired backup settings, click **Create** to finish setting up the scheduled backups.

Here is a sample configuration that includes a couple of labels added to the backup for later searches and filters:

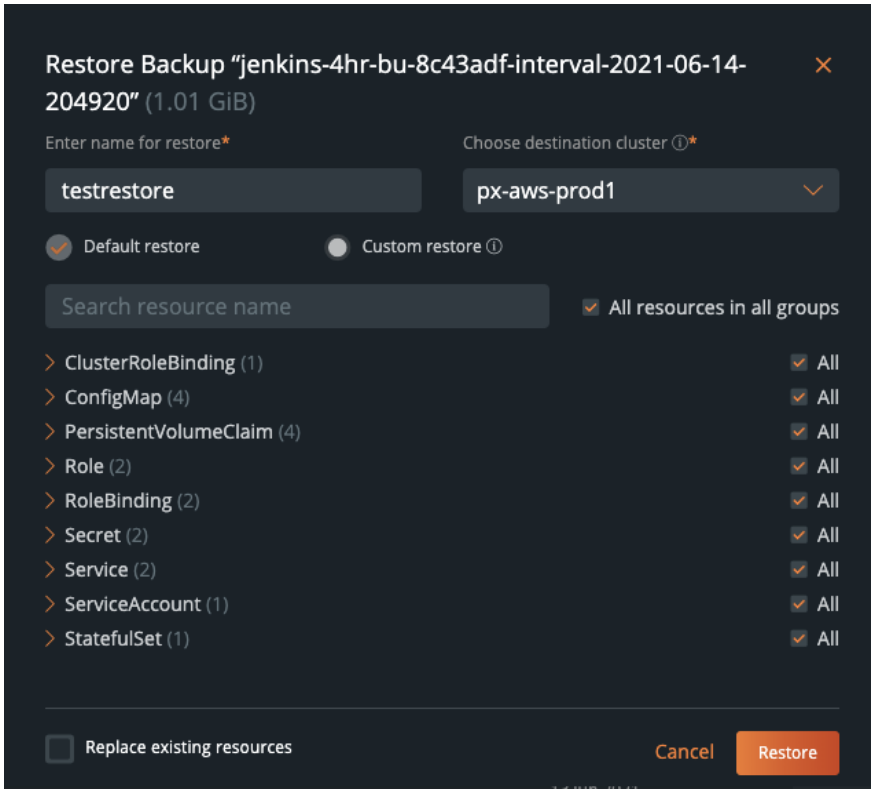


Once you complete the schedule, it will start. Once complete, it will appear with a green icon on the All Backups page or on the Backups page after clicking into the cluster from the main page.

Here you can see an example of a schedule that runs every four hours, with the most recent backup on top.



In the event you want to test your backups, or restore your environment for whatever reason, simply click the three stacked dots at the end of the line for the backup you want to use, and then select **Restore**.



Enter a name and select your production cluster. You can also add your DR cluster and restore to it as well. If you are simply testing your backups, select **Custom restore** and restore to a different namespace. If you need to recover with the backup, use the default restore option, but click the **Replace existing resources** checkbox at the bottom. Once you have the restore configured, click **Restore** to start.

The restore process can be monitored from the Restores tab. Once it finishes, you would find everything in place in the new namespace, if you did a test restore. You can port forward to the Jenkins service and log in to verify that your restore was successful.

Solution Testing and Validation

Functional Testing: Jenkins Controller HA Failover Test

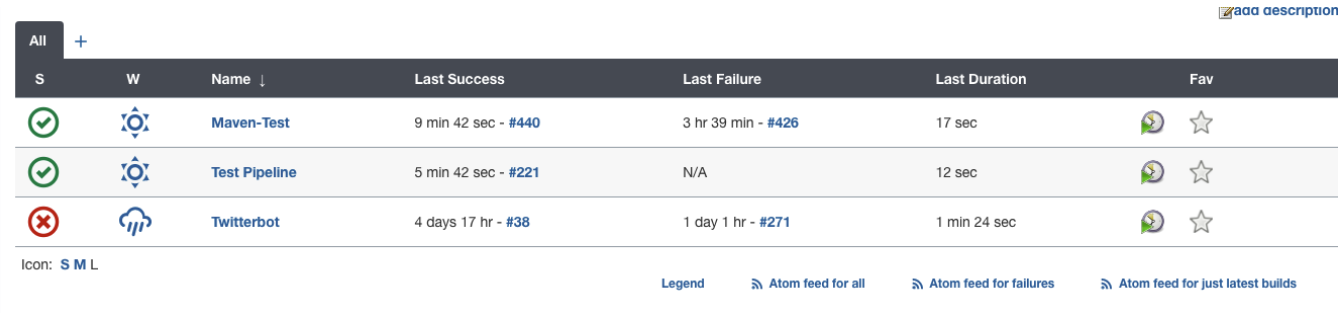
The current state of Jenkins deployment is as follows:

```
❯ kubectl get pods -n jenkins -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP            NODE                           NOMINATED NODE   READINESS GATES
jenkins-0   2/2     Running   0          3d15h   10.28.24.51   ip-10-28-11-87.ec2.internal   <none>           <none>
❯ kubectl get pvc -n jenkins
NAME                 STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS      AGE
jenkins              Bound    pvc-2f05d999-b391-46cc-a4a2-bdee199c98e9   100Gi      RWX            px-sharedv4-sc    17d
jenkins-plugin-dir   Bound    pvc-9a0d9983-ffab-4d7e-b11b-07cffb59c574   25Gi       RWO            px-jenkins-sc     17d
jenkins-plugins      Bound    pvc-72e65c45-f916-4f83-9c9a-ead8f96f443f   25Gi       RWO            px-jenkins-sc     17d
sc-config-volume     Bound    pvc-e4777535-2513-4cdf-98c5-8a1e9a3cb869   25Gi       RWO            px-jenkins-sc     17d
```

Describe the Jenkins volume to see replica placement and status:

```
❯ kubectl pxc volume inspect pvc-2f05d999-b391-46cc-a4a2-bdee199c98e9
Volume:         662147163463782487
Name:           pvc-2f05d999-b391-46cc-a4a2-bdee199c98e9
Pvc Name:       jenkins
Namespace:      jenkins
Size:           100 GiB
Format:         EXT4
HA:             3
IO Priority:    HIGH
Creation Time:  May 21 18:47:45 UTC 2021
Shared:         v4
Status:         UP
State:          on ip-10-28-9-164.ec2.internal
Device Path:    185717ed-734f-464b-af38-0fa9288350cd
Labels:         sharedv4_mount_options=vers=4.0
                namespace=jenkins
                pvc=jenkins
                io_profile=auto
                component=jenkins-controller
                instance=jenkins
                repl=3
                name=jenkins
                sharedv4=true
Fastpath:
  Preference:  false
Stats:
  Reads:             36694
  Reads MS:          15679
  Bytes Read:        247021568
  Writes:            723783
  Writes MS:         676970
  Bytes Written:     7128477696
  IOs in progress:   0
  Bytes used:        979 MiB
Replication Status:  UP
Replica sets on nodes:
  Set:     0
    Node:  ip-10-28-49-141.ec2.internal (Pool 0)
           ip-10-28-81-112.ec2.internal (Pool 0)
           ip-10-28-9-164.ec2.internal (Pool 0)
Pods:
  - Name:           jenkins-0 (113611b2-1295-4cbd-913f-33206a7b53b6)
    Namespace:      jenkins
    Running on:     ip-10-28-11-87.ec2.internal
    Controlled by:  jenkins (StatefulSet)
```

To view Jenkins jobs:



Jenkins HA Failover

In the above output, Jenkins pods are running on host: IP-10.28-11-87.ec2.internal

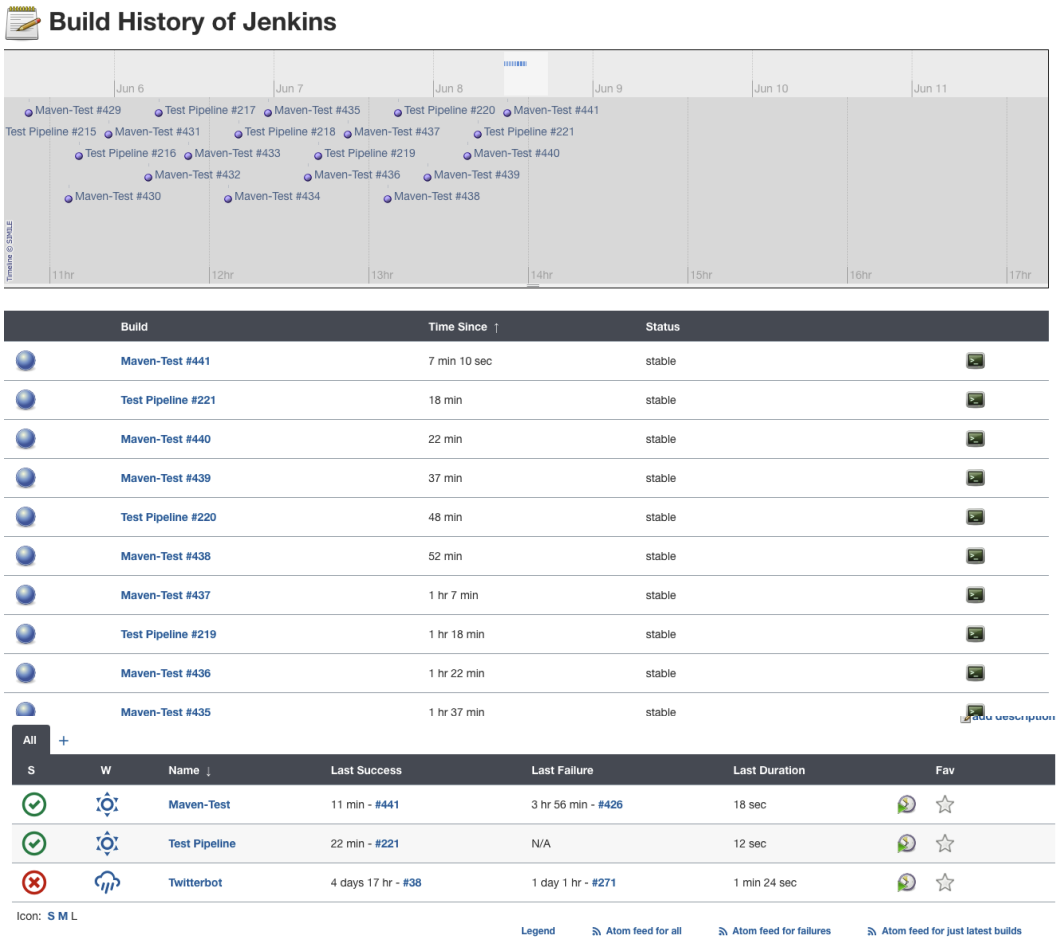
To cordon the cordon node:

```
❯ kubectl cordon ip-10-28-11-87.ec2.internal
node/ip-10-28-11-87.ec2.internal cordoned
```

To delete a Jenkins pod to force failover:

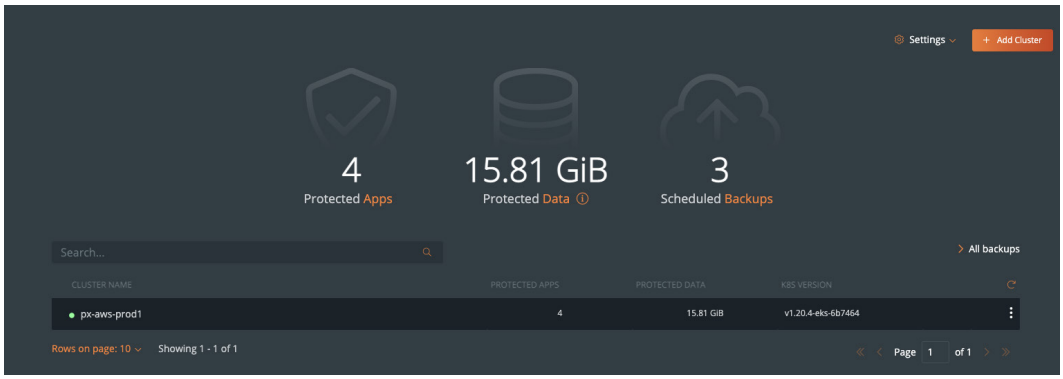
```
❯ kubectl delete po/jenkins-0 -n jenkins
pod "jenkins-0" deleted
❯ kubectl get pods -n jenkins
NAME        READY   STATUS     RESTARTS   AGE
jenkins-0   0/2     Init:0/1   0          10s
❯ kubectl get pods -n jenkins
NAME        READY   STATUS    RESTARTS   AGE
jenkins-0   2/2     Running   0          2m53s
❯ kubectl get pods -n jenkins -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP             NODE                              NOMINATED NODE   READINESS GATES
jenkins-0   2/2     Running   0          3m43s   10.28.75.180   ip-10-28-89-113.ec2.internal   <none>           <none>
```

Note that Jenkins is now running on host: IP-10-28-89-113.ec2.internal, and that failover time is based on pod start-up configuration. Once the pod is available again, the build history is intact and the jobs are still populated and executing on the set schedule as defined.

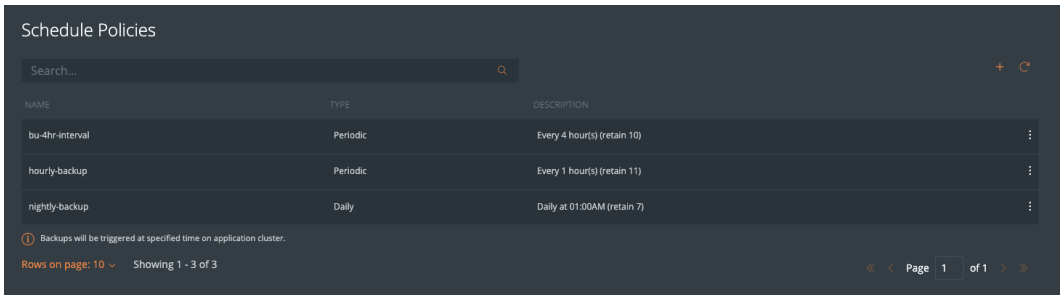


Portworx Backup and Recovery UI

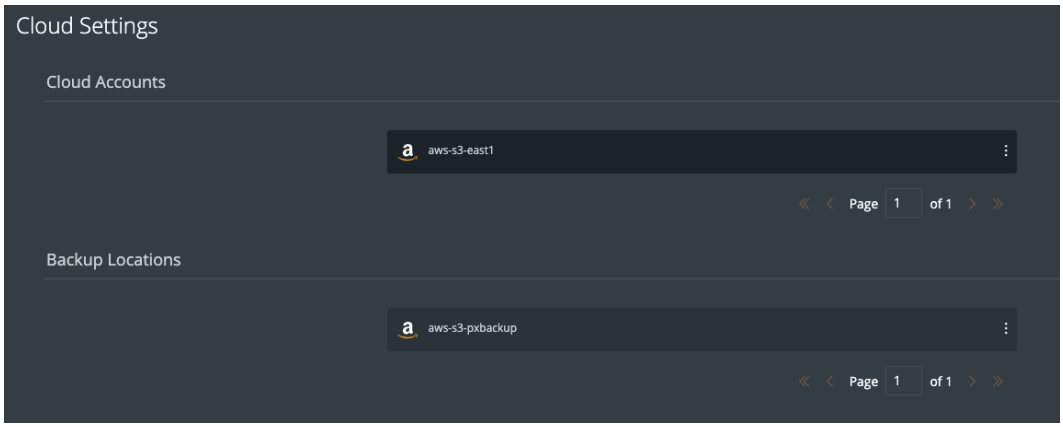
The PX-Backup UI looks like this:



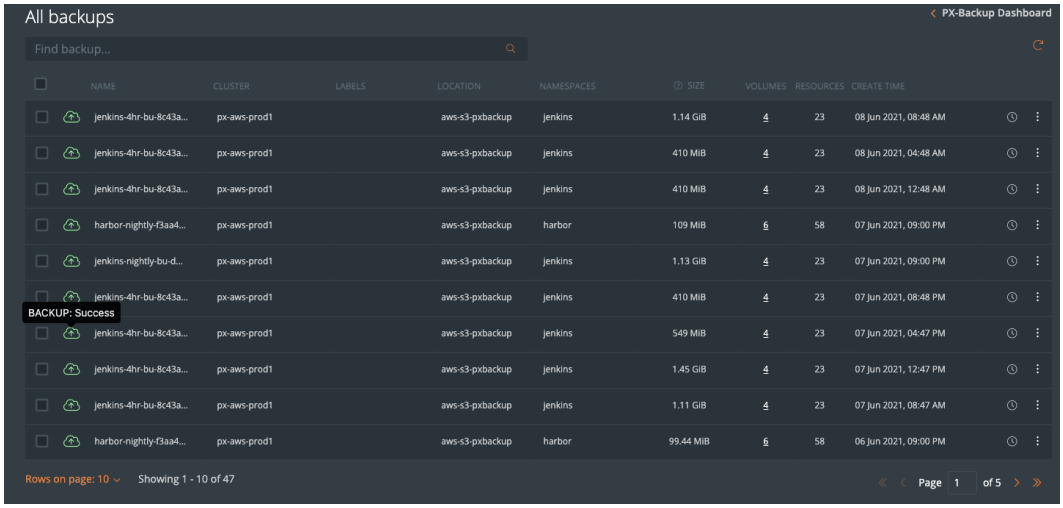
Here is the Schedule Policies interface.



Cloud accounts and backup location can be found in Cloud Settings:



The catalog of previous backups is displayed in All backups.



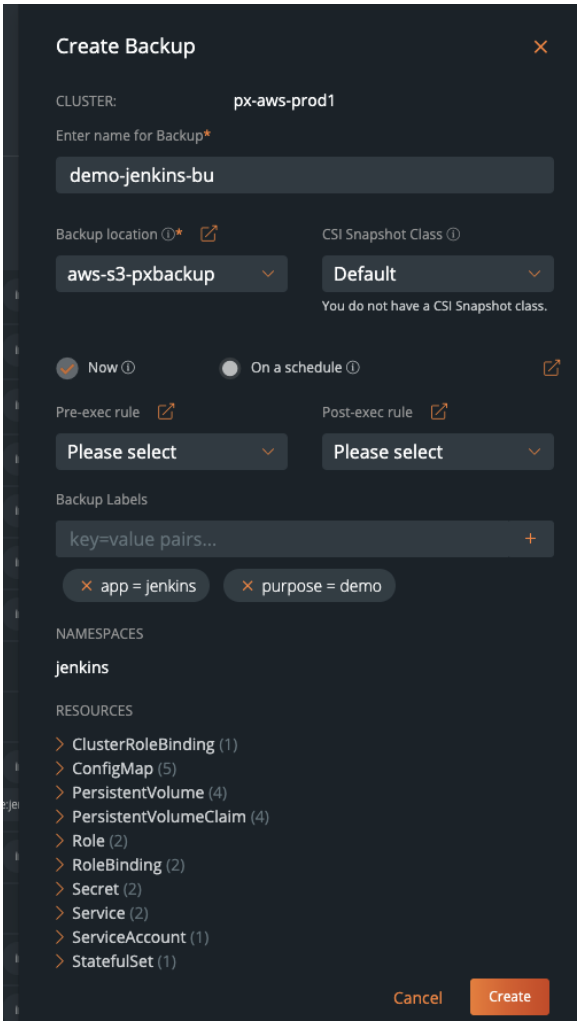
Backup and Restore Jenkins

To back up the app, back up the namespace. Note that the additional drop-down and label selector allows for more fine-grained backups.

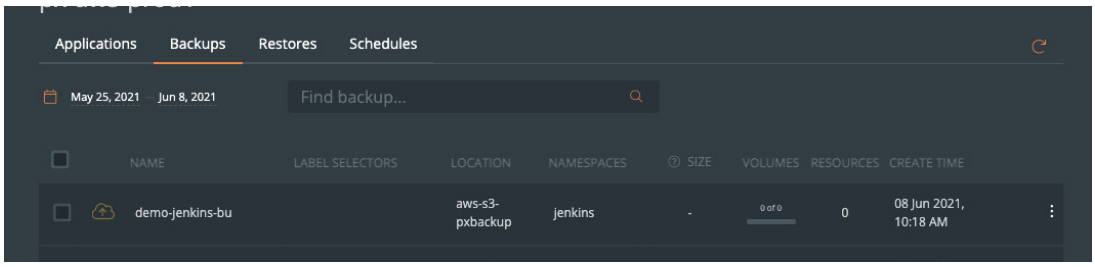
Backup

To back up the application, select the **Backup** button and the dialog box below will appear. Provide a name, select a location, and optionally select a schedule. You will also want to select the pre- and post-backup rules you created if needed. Then click **Create** to start the backup.

Notice that all Kubernetes Objects listed will be part of the backup. The target location is in an S3 bucket.



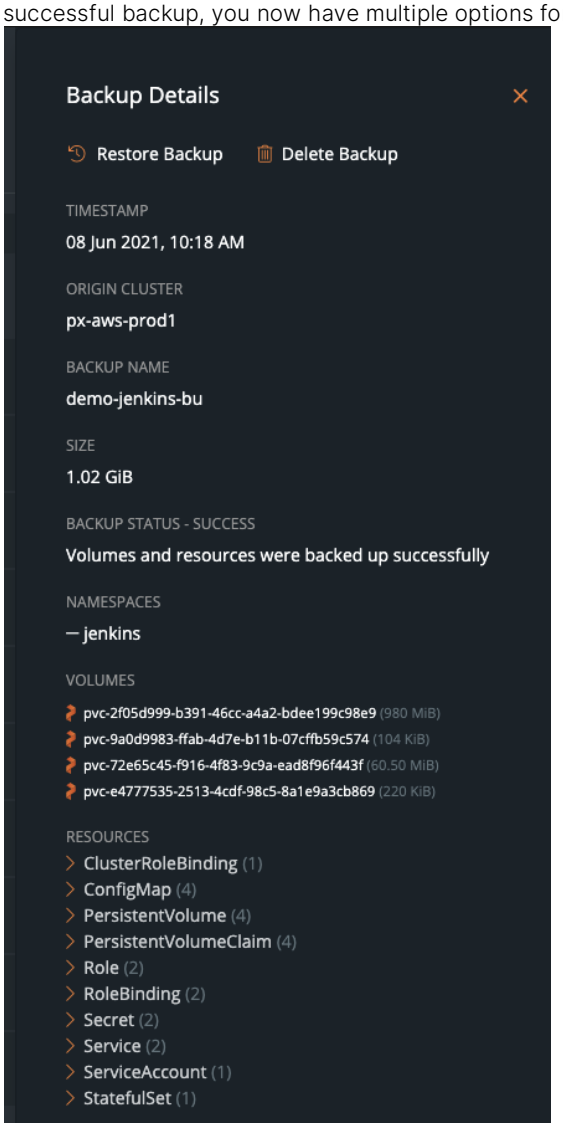
Once the backup is started, it will appear in the backup catalog.



Clicking the three stacked dots to the right shows the status and configuration of the backup job. You can also select to restore the backup from here.

Once the backup completes, you can view the results and see what was protected.

Note that all Kubernetes objects are listed, including the Persistent Volume and Persistent Volume Claim objects. With this successful backup, you now have multiple options for restoring the application.



If the cluster were to fail, or a mistake is made in applying manifests to the cluster that corrupts or breaks the Jenkins deployment, you can restore it to the current instance. But you can also restore to a different namespace or a different Portworx-enabled Kubernetes cluster.

In the following example, you will restore the Jenkins application to a second cluster located in a different region. The production cluster is in AWS US-East-1. The second cluster is in AWS US-West-1.

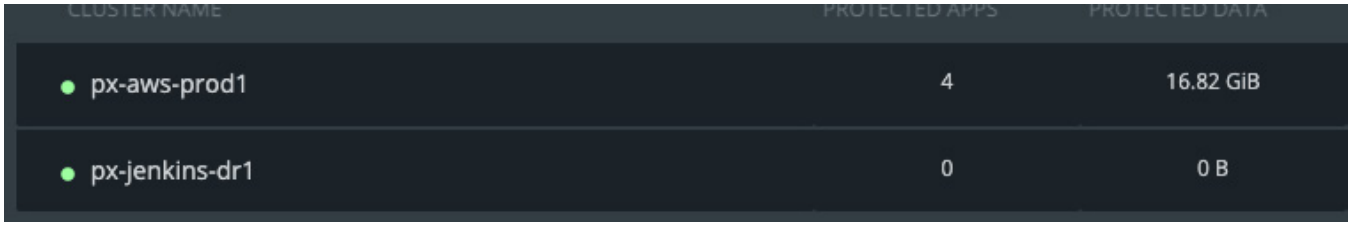
The US-West-1 Cluster - Portworx Status and Node List is below:

```
❯ kubectl pxc pxctl status
>> Running pxctl on ip-172-28-0-239.us-west-1.compute.internal
Status: PX is operational
License: Trial (expires in 31 days)
Node ID: 890971f9-5a9e-4131-a44a-86e70d866fc4
        IP: 172.28.0.239
        Local Storage Pool: 0 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE    REGION
        No storage pool
        Local Storage Devices: 0 device
        Device  Path    Media Type      Size            Last-Scan
        No storage device
        total           -       0 B
        Cache Devices:
         * No cache devices
Cluster Summary
        Cluster ID: px-jenkins-dr1-61192aae-444d-4782-b93b-dc5f7f6345f1
        Cluster UUID: 22189c9a-5cf6-4951-8d1c-66debb41e078
        Scheduler: kubernetes
        Nodes: 2 node(s) with storage (2 online), 3 node(s) without storage (3 online)
        IP              ID                                      SchedulerNodeName                       StorageNode     Used    Cap
acity   Status  StorageStatus           Version         Kernel                          OS
        172.28.62.78    c59407e6-889c-4b52-b6c3-8e7b63adaa60    ip-172-28-62-78.us-west-1.compute.internal      Yes             10 GiB  247
 GiB            Online  Up                      2.7.0.0-ccee71c 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        172.28.4.217    c2342107-37a5-4ae9-a84f-ae8942c31565    ip-172-28-4-217.us-west-1.compute.internal      Yes             10 GiB  247
 GiB            Online  Up                      2.7.0.0-ccee71c 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        172.28.0.239    890971f9-5a9e-4131-a44a-86e70d866fc4    ip-172-28-0-239.us-west-1.compute.internal      No              0 B     0 B
Online  No Storage (This node)  2.7.0.0-ccee71c 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        172.28.36.13    625ad823-68ef-4fc4-b486-270f7213f6a6    ip-172-28-36-13.us-west-1.compute.internal      No              0 B     0 B
Online  No Storage             2.7.0.0-ccee71c 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        172.28.25.237   2805d1a6-c107-4379-b6b9-4bdb0ddf6a8d    ip-172-28-25-237.us-west-1.compute.internal     No              0 B     0 B
Online  No Storage             2.7.0.0-ccee71c 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
        Total Used      :  20 GiB
        Total Capacity  :  494 GiB
❯ kubectl get nodes
NAME                                        STATUS   ROLES    AGE   VERSION
ip-172-28-0-239.us-west-1.compute.internal   Ready    <none>   89m   v1.19.6-eks-49a6c0
ip-172-28-25-237.us-west-1.compute.internal  Ready    <none>   89m   v1.19.6-eks-49a6c0
ip-172-28-36-13.us-west-1.compute.internal   Ready    <none>   89m   v1.19.6-eks-49a6c0
ip-172-28-4-217.us-west-1.compute.internal   Ready    <none>   89m   v1.19.6-eks-49a6c0
ip-172-28-62-78.us-west-1.compute.internal   Ready    <none>   89m   v1.19.6-eks-49a6c0
```

The current namespaces in US-West-1 Cluster:

```
❯ kubectl get namespaces
NAME                 STATUS   AGE
default              Active   100m
kube-node-lease      Active   100m
kube-public          Active   100m
kube-system          Active   100m
```

To add the US-West-1 Cluster to PX-Backup:



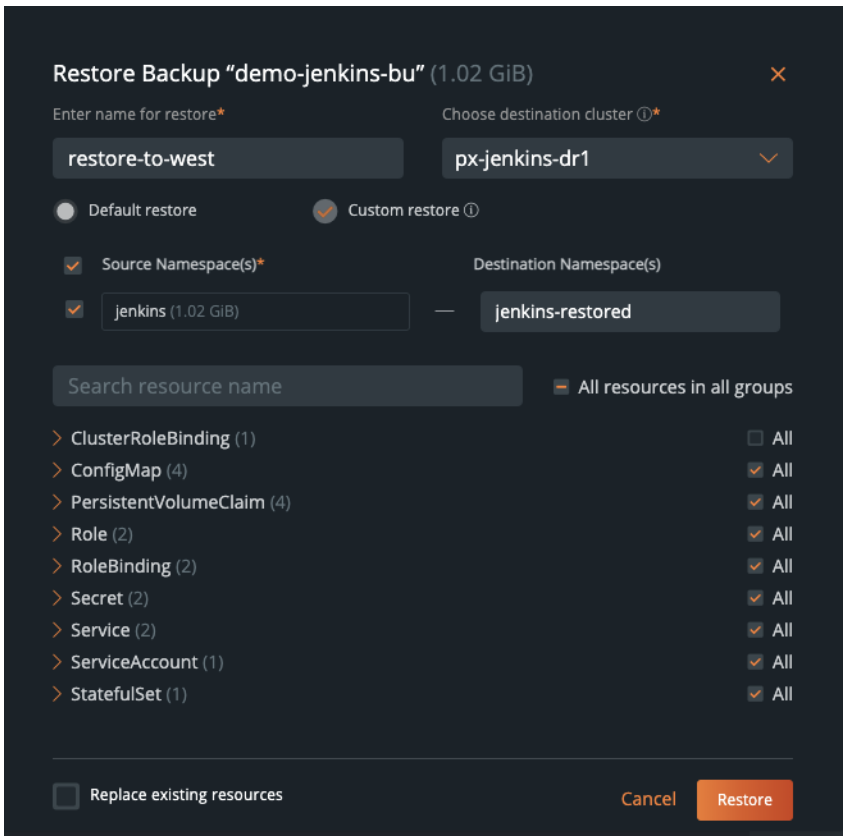
PX-Backup can provide backup and recovery services to all connected clusters.

Restore Jenkins Backup

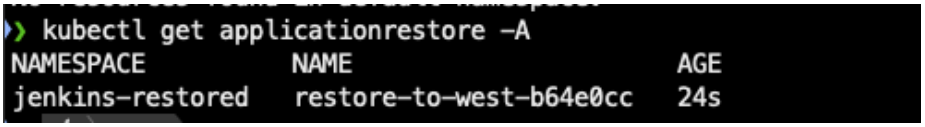
From the UI, click **All Backups >>**. This will present the backup catalog. Here, select the last backup and configure a restore to the remote cluster located in the US-West-1 region.

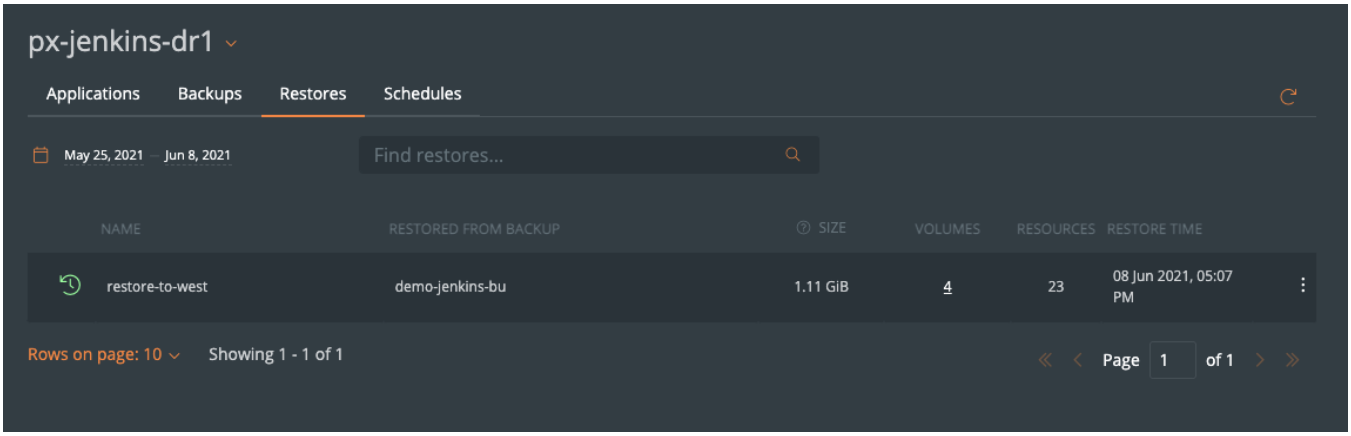
In the restore dialog box, provide a name for the restore job. In the destination cluster, both the source and remote clusters are available. The remote cluster is selected and **Custom Restore** is selected to place the restored copy of Jenkins into a different namespace. To proceed, ensure all items are selected and click the **Restore** button.

You can monitor the progress on the resulting page:



The backup was successfully restored to the cluster in the US-West-1 region, as seen below.





The output for `kubectl get all -n jenkins-restore` looks like this:

```
❯ kubectl get all -n jenkins-restored
NAME            READY   STATUS    RESTARTS   AGE
pod/jenkins-0   2/2     Running   0          4m39s

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
service/jenkins         ClusterIP   10.100.241.252   <none>        8080/TCP    4m41s
service/jenkins-agent   ClusterIP   10.100.132.21    <none>        50000/TCP   4m41s

NAME                       READY   AGE
statefulset.apps/jenkins   1/1     4m41s
❯ kubectl cluster-info
Kubernetes control plane is running at https://7DEF44B107F769CE27BF4B3A8DFE8D70.sk1.us-west-1.eks.amazonaws.com
CoreDNS is running at https://7DEF44B107F769CE27BF4B3A8DFE8D70.sk1.us-west-1.eks.amazonaws.com/api/v1/namespaces/kube-system/services/kube-
dns:dns/proxy
```

The application is running in the namespace “jenkins-restored” and we can see that the cluster is in US-West-1.

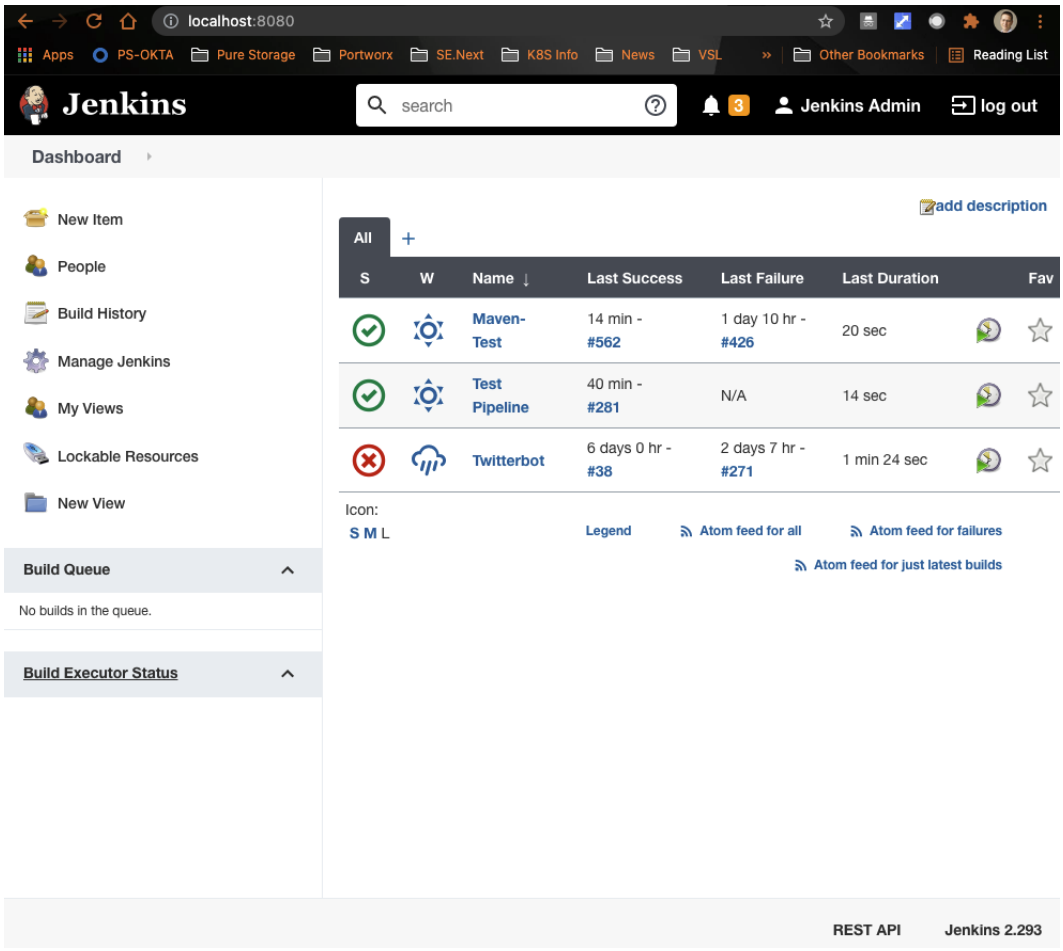
Validate DR Replication Failover Preparedness

Change to the DR Cluster and verify the DR site is usable:

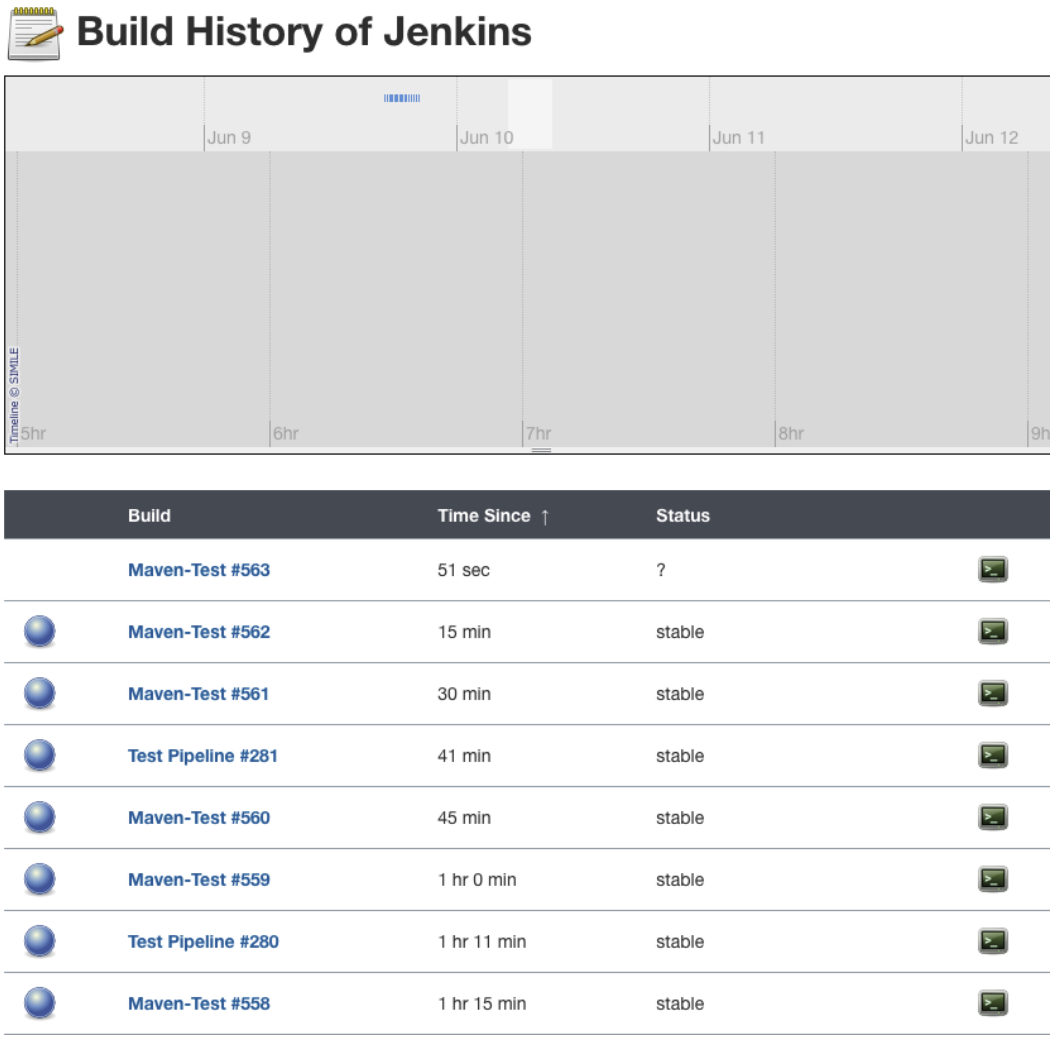
```
storkctl suspend migrationschedules jenkins-async-dr -n migrations
MigrationSchedule jenkins-async-dr suspended successfully
~/VSCode/GitHub/px-jenkins-eks/migrations  master !2 ?5                                                    px-aws-prod
kubectx px-jenkins-dr1
Switched to context "px-jenkins-dr1".
~/VSCode/GitHub/px-jenkins-eks/migrations  master !2 ?5                                                    px-jenkins-dr1
kubectl get all -n jenkins
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/jenkins         ClusterIP   10.100.27.49    <none>        8080/TCP    6m20s
service/jenkins-agent   ClusterIP   10.100.23.241   <none>        50000/TCP   6m20s

NAME                       READY   AGE
statefulset.apps/jenkins   0/0     6m20s
~/VSCode/GitHub/px-jenkins-eks/migrations  master !2 ?5                                          4s  px-jenkins-dr1
kubectl scale statefulset/jenkins -n jenkins --replicas=1
statefulset.apps/jenkins scaled
~/VSCode/GitHub/px-jenkins-eks/migrations  master !2 ?5                                                    px-jenkins-dr1
kubectl get pods -n jenkins
NAME        READY   STATUS     RESTARTS   AGE
jenkins-0   0/2     Init:0/1   0          11s
kubectl get pods -n jenkins
NAME        READY   STATUS    RESTARTS   AGE
jenkins-0   2/2     Running   0          95s
~/VSCode/GitHub/px-jenkins-eks/migrations  master !2 ?5                                                    px-jenkins-dr1
kubectl port-forward svc/jenkins -n jenkins 8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

After suspending the DR migration schedule and scaling up the StatefulSet controller in the DR Cluster, we can then port-forward to the Jenkins service and access the application running in the DR Cluster using `kubectl port-forward -n jenkins svc/jenkins 8080`.

Next, open your browser and navigate to http://localhost:8080. Your credentials are the same as the production deployment. Once logged in, you can verify that everything is in place and ready in case of a disaster recovery event.



You can see that the jobs are there and ready to run. When you check the build history, you'll see something like the screenshot below:



Not only is the job history there, but the Maven-Test has started on the DR Cluster.

Once done testing the DR site, scale the StatefulSet controller back to 0 and re-enable the migration schedule on the Production cluster:

```
~/VSCode/GitHub/px-jenkins-eks/migrations master !2 ?5 ✔ 4m 24s px-jenkins-dr1
kubectl scale statefulset/jenkins -n jenkins --replicas=0
statefulset.apps/jenkins scaled
~/VSCode/GitHub/px-jenkins-eks/migrations master !2 ?5 ✔ px-jenkins-dr1
kubectx px-aws-prod
Switched to context "px-aws-prod".
~/VSCode/GitHub/px-jenkins-eks/migrations master !2 ?5 ✔
storkctl resume migrationschedules jenkins-async-dr -n migrations
MigrationSchedule jenkins-async-dr resumed successfully
~/VSCode/GitHub/px-jenkins-eks/migrations master !2 ?5 ✔
```

You can then verify that the DR replication has resumed:

```
NAME                                    AGE
jenkins-async-dr-interval-2021-06-09-200912   17m
jenkins-async-dr-interval-2021-06-09-202546   38s
~/VSCode/GitHub/px-jenkins-eks/migrations master !2 ?5 ✔
```

Conclusion

Portworx by Pure Storage is the industry-leading container storage solution. By dynamically providing persistent storage to containers, Portworx enables your company to run your development operations at any scale. Portworx was specifically built from the ground up for containers. Like AWS EKS, which allows you to adapt your compute resources as your needs change, Portworx allows you to "right-size" your storage. No more guesswork or wasted capacity!

Portworx also enables HA across availability zones without the need to deploy multiple Jenkins controllers in an active-passive configuration. Portworx enables you to realize a zero RPO and a recovery time objective (RTO) measured in seconds rather than minutes or longer. By leveraging Portworx for your Jenkins deployments, you can start small and grow as your needs change. Portworx offers unique capabilities, including automated provisioning, dynamic volume expansion, storage pool expansions, data protection, and disaster recovery, and it works alongside EKS to ensure that your applications are always performant, highly available, and protected with the same level of enterprise-class data services as more traditional infrastructures.

Through testing of each component, we have shown that Portworx Enterprise can easily provide the enterprise-grade data services needed to run reliable production systems in the AWS Cloud ecosystem at a large scale. Solving for speed, density, and scale, Portworx not only enables efficient provisioning, cross-AZ high availability, and data that is as mobile as the containers it fuels. Portworx also provides a complete disaster recovery and business continuity solution. Simply add the Disaster Recovery option and enable Metro-DR for zero RPO, or Async-DR for longer distances with a low RPO of 10 minutes. If your business just can't be down, Portworx Enterprise paired with AWS Elastic Kubernetes Service are the tools for the job.

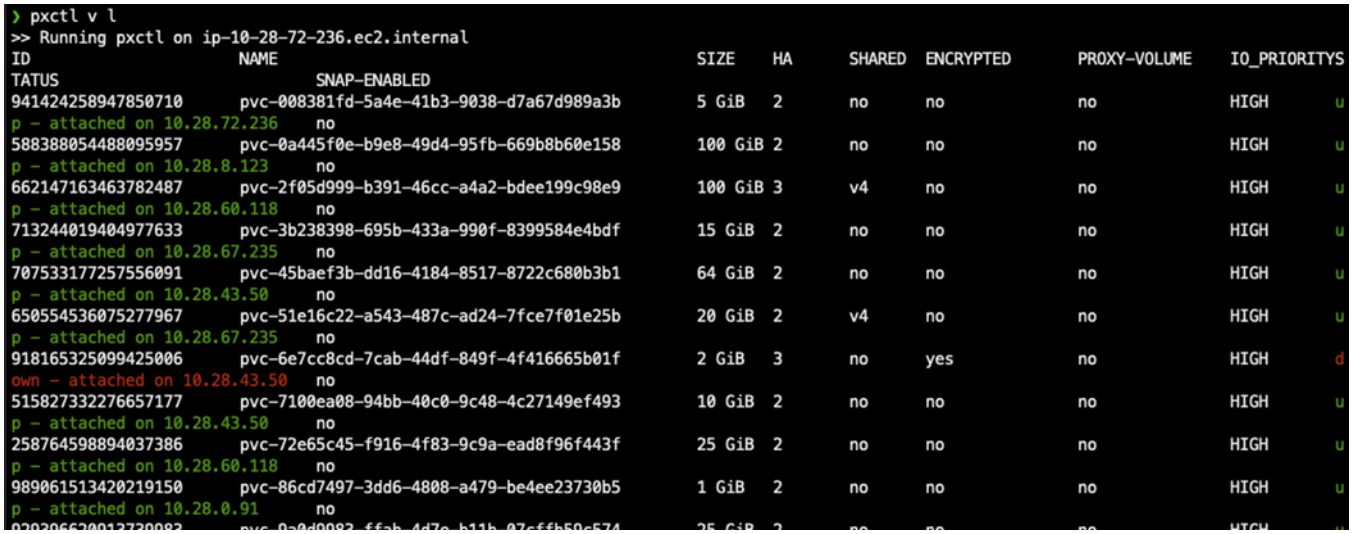
Complete the solution with PX Backup to provide Kubernetes-aware and application-consistent backups. PX-Backup enables enterprises to maintain their data-protection SLAs in this new world of modern applications. Capable of backing up everything from a single container app to distributed systems like Elastic, Cassandra, Kafka, and others, PX-Backup can even back up the entire cluster state to protect your business against the worst of days. Portworx Backup completes a solution delivering a new level of data protection, recoverability, and data mobility for containerized workloads.

Ultimately, AWS EKS and Portworx Enterprise provide a robust and reliable solution for Jenkins CI/CD pipelines. By ensuring that the data is performant, protected, and always available, Portworx Kubernetes Data Services enhances operations in the cloud to a level that no other currently available product can reach. With Portworx Enterprise and AWS EKS, your business can now build software in the cloud without compromise.

Appendix: Additional Testing

Volume Encryption: Cluster-Wide, Test 2.12

```
❯ kubectl -n kube-system create secret generic px-vol-encryption \
  --from-literal=cluster-wide-secret-key=Il0v3Portw0rX
secret/px-vol-encryption created
❯ pxctl secrets set-cluster-key --secret cluster-wide-secret-key
>> Running pxctl on ip-10-28-72-236.ec2.internal
Successfully set cluster secret key
❯ cat << EOF | kubectl apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-secure-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  secure: "true"
  repl: "3"
EOF
storageclass.storage.k8s.io/px-secure-sc created
❯ cat << EOF | kubectl apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: secure-pvc
spec:
  storageClassName: px-secure-sc
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
EOF
persistentvolumeclaim/secure-pvc created
❯ pxctl v l
```



```
❯ pxctl v l
>> Running pxctl on ip-10-28-72-236.ec2.internal
ID                      NAME                                    SIZE    HA      SHARED  ENCRYPTED       PROXY-VOLUME    IO_PRIORITYS
TATUS                           SNAP-ENABLED
941424258947850710      pvc-008381fd-5a4e-41b3-9038-d7a67d989a3b        5 GiB   2       no      no              no              HIGH        u
p - attached on 10.28.72.236    no
588388054488095957      pvc-0a445f0e-b9e8-49d4-95fb-669b8b60e158        100 GiB 2       no      no              no              HIGH        u
p - attached on 10.28.8.123     no
662147163463782487      pvc-2f05d999-b391-46cc-a4a2-bdee199c98e9        100 GiB 3       v4      no              no              HIGH        u
p - attached on 10.28.60.118    no
713244019404977633      pvc-3b238398-695b-433a-990f-8399584e4bdf        15 GiB  2       no      no              no              HIGH        u
p - attached on 10.28.67.235    no
707533177257556091      pvc-45baef3b-dd16-4184-8517-8722c680b3b1        64 GiB  2       no      no              no              HIGH        u
p - attached on 10.28.43.50     no
650554536075277967      pvc-51e16c22-a543-487c-ad24-7fce7f01e25b        20 GiB  2       v4      no              no              HIGH        u
p - attached on 10.28.67.235    no
918165325099425006      pvc-6e7cc8cd-7cab-44df-849f-4f416665b01f        2 GiB   3       no      yes             no              HIGH        d
own - attached on 10.28.43.50   no
515827332276657177      pvc-7100ea08-94bb-40c0-9c48-4c27149ef493        10 GiB  2       no      no              no              HIGH        u
p - attached on 10.28.43.50     no
258764598894037386      pvc-72e65c45-f916-4f83-9c9a-ead8f96f443f        25 GiB  2       no      no              no              HIGH        u
p - attached on 10.28.60.118    no
989061513420219150      pvc-86cd7497-3dd6-4808-a479-be4ee23730b5        1 GiB   2       no      no              no              HIGH        u
p - attached on 10.28.0.91      no
```

Volume Encryption: Volume Granular Encryption, Test 2.13

```
❯ kubectl -n jenkins-secure create secret generic jenkins-encryption-key --from-literal=secure-
pvc=SuperSecur3Key
secret/ jenkins-encryption-key created
❯ cat << EOF | kubectl apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-secure-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  secure: "true"
  repl: "3"
EOF
storageclass.storage.k8s.io/px-secure-sc created
❯ cat << EOF | kubectl apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-secure-rwx-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  sharedv4: "true"
  secure: "true"
  repl: "3"
EOF
storageclass.storage.k8s.io/px-secure-rwx-sc created
❯ cat << EOF | kubectl apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
```

```
metadata:
  name: jenkins-data-encrypted
  namespace: jenkins
  annotations:
    px/secret-name: jenkins-encryption-key
    px/secret-namespace: jenkins-encrypted
    px/secret-key: secure-pvc
spec:
  storageClassName: px-secure-rwx-sc
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
EOF
persistentvolumeclaim/jenkins-data-encrypted created
```

Deploy Jenkins Using Encryption

```
❯ kubectl apply -f yamls/Jenkins-controller-encrypted.yaml
kubectl get po -n jenkins -w
statefulsets.apps/jenkins-encrypted created
service/jenkins-encrypted created
NAME                                READY   STATUS              RESTARTS   AGE
jenkins-0             2/2     Running             0          32m
jenkins-encrypted-0   0/2     ContainerCreating   0          1s
jenkins-encrypted-0   2/2     Running             0          4s
❯ kubectl delete secret jenkins-encryption-key -n jenkins
secret "jenkins-encryption-key" deleted
❯ NODE=`kubectl get pods -l app=jenkins-encrypt -n jenkins -o jsonpath='{.items[0].spec.nodeName}'`
kubectl cordon $NODE
kubectl delete pod jenkins-encrypted-0 -n jenkins
kubectl uncordon $NODE
node/ip-10-28-67-235.ec2.internal cordoned
pod "jenkins-encrypted-0" deleted
node/ip-10-28-67-235.ec2.internal uncordoned
❯ kubectl describe po jenkins-encrypted-0  -n Jenkins
Name:         jenkins-encrypted-0
Namespace:    jenkins
Priority:     0
```

```
                    name=jenkins-encrypted
                statefulset.kubernetes.io/pod-name=jenkins-encrypted-0
Annotations:    kubernetes.io/psp: eks.privileged
Status:         Error
IP:             10.28.75.180
IPs:
  IP:           10.28.75.180
Controlled By:  StatefulSet/jenkins-encrypted
Init Containers:
  init:
    Container ID:  docker://828822c2251e78f69a28c1f1563b8449616fd02d1cc2647c0c8b974945b1ffac
    Image:         jenkins/jenkins:2.293
    Image ID:      docker-
pullable://jenkins/jenkins@sha256:7eafcc2583688b6dd6d8f614bbfe5a67c0d2f8b2ee0563364d5b2f310a74c021
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      /var/jenkins_config/apply_config.sh
    State:          Terminated
      Reason:       Error
      Exit Code:    1
      Started:      Tue, 08 Jun 2021 09:54:36 -0400
      Finished:     Tue, 08 Jun 2021 09:54:40 -0400
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     3
      memory:  8Gi
    Requests:
      cpu:        500m
      memory:     512Mi
    Environment:  <none>
    Mounts:
      /usr/share/jenkins/ref/plugins from plugins (rw)
      /var/jenkins_config from jenkins-config (rw)
      /var/jenkins_home from jenkins-home (rw)
      /var/jenkins_plugins from plugin-dir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from jenkins-token-zxhfd (ro)
Containers:
  jenkins:
    Container ID:  docker://35724d04ccea6a8b91aa15acbc3a861d708779ab73aea95dac4224df6ee1c055
    Image:         jenkins/jenkins:2.293-jdk11
    Image ID:      docker-
pullable://jenkins/jenkins@sha256:439516c825946e9252a0b65049a3bfc500df8d599a57be9bfce08213c837170a
    Ports:         8082/TCP, 50010/TCP
    Host Ports:    0/TCP, 0/TCP
    Args:
```

```
      --httpPort=8080
    State:          CrashLoopBackoff
      Started:      Tue, 08 Jun 2021 09:54:51 -0400
    Ready:          True
    Restart Count:  4
    Limits:
      cpu:     2
      memory:  4Gi
    Requests:
      cpu:      500m
      memory:   512Mi
    Liveness:   http-get http://:8082/login delay=0s timeout=5s period=10s #success=1 #failure=5
    Readiness:  http-get http://:8082/login delay=0s timeout=5s period=10s #success=1 #failure=3
    Startup:    http-get http://:8082/login delay=0s timeout=5s period=10s #success=1 #failure=12
    Environment:
      POD_NAME:                  jenkins-encrypted-0 (v1:metadata.name)
      JAVA_OPTS:                 -Dcasc.reload.token=$(POD_NAME)
      JENKINS_OPTS:
      JENKINS_SLAVE_AGENT_PORT:  50010
      JENKINS_UC:                https://updates.jenkins.io
      JENKINS_UC_DOWNLOAD:       https://ftp-nyc.osuosl.org/pub/jenkins
      CASC_JENKINS_CONFIG:       /var/jenkins_home/casc_configs
    Mounts:
      /run/secrets/chart-admin-password from admin-secret (ro,path="jenkins-admin-password")
      /run/secrets/chart-admin-username from admin-secret (ro,path="jenkins-admin-user")
      /usr/share/jenkins/ref/plugins/ from plugin-dir (rw)
      /var/jenkins_config from jenkins-config (ro)
      /var/jenkins_home from jenkins-home (rw)
      /var/jenkins_home/casc_configs from sc-config-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from jenkins-token-zxhfd (ro)
  config-reload:
    Container ID:   docker://5e5c6c988e6c325123766d380ffa2ffa53d5db8fd2a6652b34179b0359575520
    Image:          kiwigrid/k8s-sidecar:1.12.0
    Image ID:       docker-pullable://kiwigrid/k8s-
sidecar@sha256:89739be9ff3894910b29fa505b8726372c8433a6b3786b8e769631d8146b4035
    Port:           <none>
    Host Port:      <none>
    State:          Failed
      Started:      Tue, 08 Jun 2021 09:54:54 -0400
    Ready:          False
    Restart Count:  0
    Limits:
      cpu:     500m
      memory:  512Mi
    Requests:
      cpu:     50m
      memory:  256Mi
    Environment:
```

```
      POD_NAME:           jenkins-encrypted-0 (v1:metadata.name)
      LABEL:              jenkins-config
      FOLDER:             /var/jenkins_home/casc_configs
      NAMESPACE:          jenkins
      REQ_URL:            http://localhost:8082/reload-configuration-as-code/?casc-reload-
token=$(POD_NAME)
      REQ_METHOD:         POST
      REQ_RETRY_CONNECT:  10
    Mounts:
      /var/jenkins_home from jenkins-home (rw)
      /var/jenkins_home/casc_configs from sc-config-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from jenkins-token-zxhfd (ro)
Conditions:
  Type              Status
  Initialized       False
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  plugins:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  jenkins-plugins
    ReadOnly:   false
  jenkins-config:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      jenkins
    Optional:  false
  plugin-dir:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  jenkins-plugin-dir
    ReadOnly:   false
  jenkins-home:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  jenkins
    ReadOnly:   false
  sc-config-volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  sc-config-volume
    ReadOnly:   false
  admin-secret:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  jenkins
    Optional:    false
  jenkins-token-zxhfd:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  jenkins-token-zxhfd
    Optional:    false
QoS Class:       Burstable
```

```
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason       Age   From     Message
  ----     ------       ----  ----     -------
  Normal   Scheduled    34m   stork    Successfully assigned jenkins/jenkins-encrypted-0 to ip-10-
28-67-235.ec2.internal
  Warning  FailedMount  34m   kubelet  MountVolume.SetUp failed for volume "default-token-czw4j" :
failed to sync secret cache: timed out waiting for the condition. - Could not mount volume after
Secret was deleted
❯ kubectl delete -f jenkins-encrypted.yaml
kubectl delete pvc jenkins-encrypted -n jenkins
statefulset.apps "jenkins-encrypted" deleted
persistentvolumeclaim "jenkins-encrypted" deleted
```

PostgreSQL RWO

Tests: 2.01 through 2.03 Output

```
❯ kubectl create ns postgres
namespace/postgres created
❯ kubectl apply -f yamls/postgres-sc.yaml
storageclass.storage.k8s.io/px-postgres-sc created
❯ kubectl apply -f yamls/postgres-pvc.yaml
persistentvolumeclaim/postgres-data created
❯ kubectl get pvc -n postgres
NAME            STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS     AGE
postgres-data   Bound    pvc-e27f59af-a275-476c-90f0-3e0d0c681d15   2Gi        RWO            px-
postgres-sc   13s
❯ kubectl label nodes --all node-role.kubernetes.io/worker=true
node/ip-10-28-0-91.ec2.internal labeled
node/ip-10-28-43-50.ec2.internal labeled
node/ip-10-28-60-118.ec2.internal labeled
node/ip-10-28-67-235.ec2.internal labeled
node/ip-10-28-72-236.ec2.internal labeled
node/ip-10-28-8-123.ec2.internal labeled
❯ # this command assumes your nodes are labled correctly
NODE=`kubectl get nodes -l node-role.kubernetes.io/worker=true -o
jsonpath='{.items[0].metadata.name}'`
❯ cat << EOF | kubectl apply -f -
apiVersion: portworx.io/v1beta2
kind: VolumePlacementStrategy
```

```
spec:
  replicaAffinity:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - "$NODE"
EOF
volumeplacementstrategy.portworx.io/node-specific created
❯ cat << EOF | kubectl apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-node-specific-sc
  labels:
    app: postgres
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "1"
  placement_strategy: "node-specific"
allowVolumeExpansion: true
EOF
storageclass.storage.k8s.io/px-node-specific-sc created
❯ cat << EOF | kubectl apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: vps-test
spec:
  storageClassName: px-node-specific-sc
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
EOF
persistentvolumeclaim/vps-test created
❯ # check that the PVC is bound before running the inspect command
kubectl get pvc vps-test
# once it's bound you can inspect it
PVC=`kubectl get pvc vps-test --no-headers | awk '{print $3}'`
pxctl volume inspect $PVC
# and confirm that the IP of the Node in the replica set matches the node's IP
kubectl get node $NODE -o wide
zsh: correct 'pxctl' to 'tpctl' [nyae]? n
NAME       STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS          AGE
vps-test   Pending                                      px-node-specific-sc   21s
zsh: command not found: pxctl
```

```
NAME                          STATUS   ROLES    AGE   VERSION              INTERNAL-IP   EXTERNAL-IP
OS-IMAGE         KERNEL-VERSION                CONTAINER-RUNTIME
ip-10-28-0-91.ec2.internal    Ready    worker   28d   v1.19.6-eks-49a6c0   10.28.0.91    3.90.241.207
Amazon Linux 2   5.4.110-54.182.amzn2.x86_64   docker://19.3.13
❯ kubectl apply -f yamls/postgres-app.yaml
deployment.apps/postgres created
service/postgres created
❯ kubectl get po -n postgres
NAME                        READY   STATUS    RESTARTS   AGE
postgres-ddf7d7dfc-dkgww    1/1     Running   0          15s
❯ POSTGRES_POD=`kubectl get po -n postgres -l app=postgres -o jsonpath='{.items[0].metadata.name}'`
kubectl exec -it -n postgres $POSTGRES_POD -- psql -c "create database pxdemo;"
CREATE DATABASE
❯ kubectl exec -it -n postgres $POSTGRES_POD -- pgbench -i -s 50 pxdemo
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
creating tables...
100000 of 5000000 tuples (2%) done (elapsed 0.08 s, remaining 3.84 s)
200000 of 5000000 tuples (4%) done (elapsed 0.17 s, remaining 4.03 s)
300000 of 5000000 tuples (6%) done (elapsed 0.28 s, remaining 4.37 s)
400000 of 5000000 tuples (8%) done (elapsed 0.35 s, remaining 4.06 s)
500000 of 5000000 tuples (10%) done (elapsed 0.43 s, remaining 3.89 s)
600000 of 5000000 tuples (12%) done (elapsed 0.50 s, remaining 3.65 s)
700000 of 5000000 tuples (14%) done (elapsed 0.57 s, remaining 3.50 s)
800000 of 5000000 tuples (16%) done (elapsed 0.64 s, remaining 3.37 s)
900000 of 5000000 tuples (18%) done (elapsed 0.73 s, remaining 3.32 s)
1000000 of 5000000 tuples (20%) done (elapsed 0.82 s, remaining 3.28 s)
1100000 of 5000000 tuples (22%) done (elapsed 0.90 s, remaining 3.18 s)
1200000 of 5000000 tuples (24%) done (elapsed 0.97 s, remaining 3.08 s)
1300000 of 5000000 tuples (26%) done (elapsed 1.06 s, remaining 3.02 s)
1400000 of 5000000 tuples (28%) done (elapsed 1.15 s, remaining 2.96 s)
1500000 of 5000000 tuples (30%) done (elapsed 1.25 s, remaining 2.92 s)
1600000 of 5000000 tuples (32%) done (elapsed 1.36 s, remaining 2.90 s)
1700000 of 5000000 tuples (34%) done (elapsed 1.45 s, remaining 2.82 s)
1800000 of 5000000 tuples (36%) done (elapsed 1.54 s, remaining 2.74 s)
1900000 of 5000000 tuples (38%) done (elapsed 1.63 s, remaining 2.66 s)
2000000 of 5000000 tuples (40%) done (elapsed 1.70 s, remaining 2.55 s)
2100000 of 5000000 tuples (42%) done (elapsed 1.77 s, remaining 2.45 s)
2200000 of 5000000 tuples (44%) done (elapsed 1.85 s, remaining 2.36 s)
2300000 of 5000000 tuples (46%) done (elapsed 1.92 s, remaining 2.26 s)
2400000 of 5000000 tuples (48%) done (elapsed 2.00 s, remaining 2.17 s)
2500000 of 5000000 tuples (50%) done (elapsed 2.08 s, remaining 2.08 s)
2600000 of 5000000 tuples (52%) done (elapsed 2.14 s, remaining 1.98 s)
2700000 of 5000000 tuples (54%) done (elapsed 2.22 s, remaining 1.89 s)
2800000 of 5000000 tuples (56%) done (elapsed 2.29 s, remaining 1.80 s)
2900000 of 5000000 tuples (58%) done (elapsed 2.36 s, remaining 1.71 s)
```

```
3000000 of 5000000 tuples (60%) done (elapsed 2.43 s, remaining 1.62 s)
3100000 of 5000000 tuples (62%) done (elapsed 2.51 s, remaining 1.54 s)
3200000 of 5000000 tuples (64%) done (elapsed 2.60 s, remaining 1.46 s)
3300000 of 5000000 tuples (66%) done (elapsed 2.69 s, remaining 1.38 s)
3400000 of 5000000 tuples (68%) done (elapsed 2.77 s, remaining 1.31 s)
3500000 of 5000000 tuples (70%) done (elapsed 2.87 s, remaining 1.23 s)
3600000 of 5000000 tuples (72%) done (elapsed 2.96 s, remaining 1.15 s)
3700000 of 5000000 tuples (74%) done (elapsed 3.05 s, remaining 1.07 s)
3800000 of 5000000 tuples (76%) done (elapsed 3.14 s, remaining 0.99 s)
3900000 of 5000000 tuples (78%) done (elapsed 3.22 s, remaining 0.91 s)
4000000 of 5000000 tuples (80%) done (elapsed 3.35 s, remaining 0.84 s)
4100000 of 5000000 tuples (82%) done (elapsed 3.45 s, remaining 0.76 s)
4200000 of 5000000 tuples (84%) done (elapsed 3.54 s, remaining 0.67 s)
4300000 of 5000000 tuples (86%) done (elapsed 3.65 s, remaining 0.59 s)
4400000 of 5000000 tuples (88%) done (elapsed 3.72 s, remaining 0.51 s)
4500000 of 5000000 tuples (90%) done (elapsed 3.80 s, remaining 0.42 s)
4600000 of 5000000 tuples (92%) done (elapsed 3.87 s, remaining 0.34 s)
4700000 of 5000000 tuples (94%) done (elapsed 3.96 s, remaining 0.25 s)
4800000 of 5000000 tuples (96%) done (elapsed 4.05 s, remaining 0.17 s)
4900000 of 5000000 tuples (98%) done (elapsed 4.13 s, remaining 0.08 s)
5000000 of 5000000 tuples (100%) done (elapsed 4.21 s, remaining 0.00 s)
vacuum...
set primary keys...
done.
❯ kubectl exec -it -n postgres $POSTGRES_POD -- df -m | grep postgres
/dev/pxd/pxd559963210498322444      1952   805      1030  44% /var/lib/postgresql/data
❯ kubectl exec -it -n postgres $POSTGRES_POD -- psql pxdemo -c "select count(*) from
pgbench_accounts"
  count
---------
 5000000
(1 row)
```

Failover Testing: 2.04, Postgres

```
❯ NODE=`kubectl get pods -l app=postgres -n postgres -o jsonpath='{.items[0].spec.nodeName}'`
kubectl cordon $NODE
kubectl delete pod $POSTGRES_POD -n postgres
node/ip-10-28-8-123.ec2.internal cordoned
pod "postgres-ddf7d7dfc-dkgww" deleted
❯ kubectl get pod -o wide -n postgres
NAME                        READY   STATUS    RESTARTS   AGE   IP             NODE
NOMINATED NODE   READINESS GATES
postgres-ddf7d7dfc-xgzd8    1/1     Running   0          23s   10.28.65.138   ip-10-28-67-
235.ec2.internal   <none>           <none>
❯ NODE=`kubectl get pods -l app=postgres -n postgres -o jsonpath='{.items[0].spec.nodeName}'`
```

```
kubectl cordon $NODE
kubectl delete pod $POSTGRES_POD -n postgres
node/ip-10-28-8-123.ec2.internal cordoned
pod "postgres-ddf7d7dfc-dkgww" deleted
❯ kubectl get pod -o wide -n postgres
NAME                        READY   STATUS    RESTARTS   AGE   IP             NODE
NOMINATED NODE   READINESS GATES
postgres-ddf7d7dfc-xgzd8    1/1     Running   0          23s   10.28.65.138   ip-10-28-67-
235.ec2.internal   <none>           <none>
❯ POSTGRES_POD=`kubectl get po -n postgres -l app=postgres -o jsonpath='{.items[0].metadata.name}'`
kubectl exec -it -n postgres $POSTGRES_POD -- df -m | grep postgres
/dev/pxd/pxd559963210498322444      1952   821      1014  45% /var/lib/postgresql/data
❯ kubectl exec -it -n postgres $POSTGRES_POD -- psql pxdemo -c "select count(*) from
pgbench_accounts"
  count
---------
 5000000
(1 row)
❯ kubectl uncordon $NODE
node/ip-10-28-8-123.ec2.internal uncordoned
```

Read Write Many: Nginix Deployment, 2.05 and 2.06

```
❯ kubectl apply -f yamls/nginx-sc.yaml
storageclass.storage.k8s.io/px-shared-sc created
❯ kubectl create ns nginx
kubectl apply -f yamls/nginx-pvc.yaml
namespace/nginx created
persistentvolumeclaim/nginx-pvc created
❯ kubectl apply -f yamls/nginx-app.yaml
deployment.apps/nginx created
service/nginx-svc created
❯ kubectl get po -n nginx
NAME                     READY   STATUS    RESTARTS   AGE
nginx-744b4745dc-w6pgz   1/1     Running   0          13s
❯ kubectl get svc -n nginx
NAME        TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)        AGE
nginx-svc   LoadBalancer   172.20.36.161   ac6065bf92048455ba37f6e9cd1b316b-1972197306.us-east-
1.elb.amazonaws.com   80:30096/TCP   26s
❯ cat << EOF | cat >> index.html
<html>
<h1>Hello World<\h1>
</html>
```

```
EOF
POD=`kubectl get pods -n nginx --no-headers | head -n 1 | awk '{print $1}'`
kubectl cp -n nginx index.html $POD:/usr/share/nginx/html/index.html
❯ kubectl scale deploy nginx -n nginx --replicas=3
deployment.apps/nginx scaled
❯ kubectl get po -n nginx -owide
NAME                     READY   STATUS    RESTARTS   AGE    IP             NODE
NOMINATED NODE   READINESS GATES
nginx-744b4745dc-8wg7w   1/1     Running   0          17s    10.28.37.164   ip-10-28-43-
50.ec2.internal    <none>           <none>
nginx-744b4745dc-g2c6w   1/1     Running   0          17s    10.28.79.33    ip-10-28-72-
236.ec2.internal   <none>           <none>
nginx-744b4745dc-w6pgz   1/1     Running   0          2m4s   10.28.20.149   ip-10-28-8-
123.ec2.internal    <none>           <none>
```

Volume Snapshot: Test 2.14

PostgreSQL used for testing:

```
❯ cat << EOF | kubectl apply -f -
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: px-postgres-snapshot
  namespace: postgres
spec:
  persistentVolumeClaimName: postgres-data
EOF
volumesnapshot.volumesnapshot.external-storage.k8s.io/px-postgres-snapshot created
❯ kubectl describe volumesnapshot px-postgres-snapshot -n postgres
Name:         px-postgres-snapshot
Namespace:    postgres
Labels:       SnapshotMetadata-PVName=pvc-e27f59af-a275-476c-90f0-3e0d0c681d15
              SnapshotMetadata-Timestamp=1622737975553645452
Annotations:  <none>
API Version:  volumesnapshot.external-storage.k8s.io/v1
Kind:         VolumeSnapshot
Metadata:
  Creation Timestamp:  2021-06-03T16:32:55Z
  Generation:          3
  Managed Fields:
    API Version:  volumesnapshot.external-storage.k8s.io/v1
    Time:         2021-06-03T16:32:55Z
    API Version:  volumesnapshot.external-storage.k8s.io/v1
    Manager:         kubectl-client-side-apply
```

```
    Operation:       Update
    Time:            2021-06-03T16:32:55Z
  Resource Version:  16829061
  Self Link:         /apis/volumesnapshot.external-
storage.k8s.io/v1/namespaces/postgres/volumesnapshots/px-postgres-snapshot
  UID:               c3dbd16b-3bfe-422e-9aa5-482f83287606
Spec:
  Persistent Volume Claim Name:  postgres-data
  Snapshot Data Name:            k8s-volume-snapshot-9790aa14-bf7e-410a-8306-c1915723ad99
Status:
  Conditions:
    Last Transition Time:  2021-06-03T16:32:56Z
    Message:               Snapshot created successfully and it is ready
    Reason:
    Status:                True
    Type:                  Ready
  Creation Timestamp:      <nil>
Events:                    <none>
```

Volume Snapshot Restore: Test 2.15

To test the snapshot restore, drop the px-demo database and show you can recover it:

```
❯ POSTGRES_POD=`kubectl get po -n postgres -l app=postgres -o jsonpath='{.items[0].metadata.name}'`
kubectl exec -it -n postgres $POSTGRES_POD -- psql -c "drop database pxdemo;"
DROP DATABASE
❯ kubectl exec -it -n postgres $POSTGRES_POD -- psql pxdemo -c "select count(*) from
pgbench_accounts"
psql: FATAL:  database "pxdemo" does not exist
command terminated with exit code 2
❯ cat << EOF | kubectl apply -f -
apiVersion: stork.libopenstorage.org/v1alpha1
kind: VolumeSnapshotRestore
metadata:
  name: postgres-snap-restore
  namespace: postgres
spec:
  sourceName: px-postgres-snapshot
  sourceNamespace: postgres
EOF
kubectl get po -n postgres -w
volumesnapshotrestore.stork.libopenstorage.org/postgres-snap-restore created
```

```
NAME                        READY   STATUS    RESTARTS   AGE
postgres-ddf7d7dfc-j28cd    0/1     Pending   0          1s
postgres-ddf7d7dfc-j28cd    0/1     Terminating   0          14s
postgres-ddf7d7dfc-j28cd    0/1     Terminating   0          14s
postgres-ddf7d7dfc-qlxd5    0/1     Pending       0          0s
postgres-ddf7d7dfc-qlxd5    0/1     Pending       0          0s
postgres-ddf7d7dfc-qlxd5    0/1     Pending       0          10s
postgres-ddf7d7dfc-qlxd5    0/1     ContainerCreating   0          10s
postgres-ddf7d7dfc-qlxd5    1/1     Running             0          11s
^C%
❯ POSTGRES_POD=`kubectl get po -n postgres -l app=postgres -o jsonpath='{.items[0].metadata.name}'`
kubectl exec -it -n postgres $POSTGRES_POD -- psql pxdemo -c "select count(*) from pgbench_accounts"
  count
---------
 5000000
(1 row)
```

Volume Resize: Automate with Autopilot, Test 2.15

```
❯ kubectl get po -n monitor -lapp=prometheus
kubectl get svc -n monitor prometheus
kubectl describe cm -n kube-system autopilot-config
NAME                                                READY   STATUS    RESTARTS   AGE
prometheus-prometheus-kube-prometheus-prometheus-0   2/2     Running   1          28d
Error from server (NotFound): services "prometheus" not found
Name:         autopilot-config
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>
Data
====
config.yaml:
----
providers:
- name: default
  type: prometheus
  params: url=http://prometheus-kube-prometheus-prometheus.monitor.svc:9090
Events:  <none>
❯ cat << EOF | kubectl apply -f -
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: volume-resize
  namespace: kube-system
spec:
  ##### selector filters the objects affected by this rule given labels
  selector:
    matchLabels:
```

```
      app: postgres
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    # volume usage should be less than 50%
    expressions:
    - key: "100 * (px_volume_usage_bytes / px_volume_capacity_bytes)"
      operator: Gt
      values:
      - "50"
  ##### 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: "100Gi"
EOF
autopilotrule.autopilot.libopenstorage.org/volume-resize created
❯ kubectl get events --field-selector involvedObject.kind=AutopilotRule,involvedObject.name=volume-
resize --all-namespaces --sort-by .lastTimestamp
NAMESPACE   LAST SEEN   TYPE     REASON       OBJECT                         MESSAGE
default     1s          Normal   Transition   autopilotrule/volume-resize   rule: volume-resize:pvc-
e27f59af-a275-476c-90f0-3e0d0c681d15 transition from Initializing => Normal
```

Scale Up: Adding Storage Using Autopilot

Starting Capacity = 750GB

```
❯ cat << EOF | kubectl apply -f -
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 90%
    - key: "100 * ( px_pool_stats_used_bytes/ px_pool_stats_total_bytes)"
      operator: Gt
      values:
      - "10"
    # pool total capacity should not exceed 2TB
```

```
    - key: "px_pool_stats_total_bytes/(1024*1024*1024)"
      operator: Lt
      values:
       - "2000"
  ##### action to perform when condition is true
  actions:
  - name: "openstorage.io.action.storagepool/expand"
    params:
      # resize pool by scalepercentage of current size
      scalepercentage: "50"
      scaletype: "auto"
EOF
❯ pxctl cluster provision-status
>> Running pxctl on ip-10-28-8-123.ec2.internal
NODE                                    NODE STATUS     POOL
        POOL STATUS     IO_PRIORITY     SIZEAVAILABLE   USED    PROVISIONED     ZONE            REGION
                RACK
f7dfb0f8-322c-45fd-aebb-d3cc0322c398    Up              0 ( c4c6dfcb-5069-4c1e-a5f7-8b8ec7c58a70 )
        Online          HIGH            247 GiB 222 GiB         25 GiB  665 GiB         us-east-1b
        us-east-1       default
185717ed-734f-464b-af38-0fa9288350cd    Up              0 ( dc0a35ed-c6ec-469c-a474-64bc4e38d65f )
        Online          HIGH            247 GiB 226 GiB         21 GiB  648 GiB         us-east-1a
        us-east-1       default
c447d117-2ab1-41ec-9b6f-03782334c804    Up              0 ( c986f840-73a4-4b3e-9117-0413dfb8b09f )
        Online          HIGH            247 GiB 223 GiB         24 GiB  699 GiB         us-east-1c
        us-east-1       default
```

Add Storage Node: Test 3.0.2

```
Status: PX is operational
License: PX-Enterprise extended eval (expires in 59 days)
Node ID: 79bb6bbe-440d-4ac1-9ede-096181bba6ce
        IP: 10.28.28.73
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE            REGION
        0       HIGH            raid0           247 GiB 10 GiB  Online  us-east-1a      us-east-1
        Local Storage Devices: 1 device
```

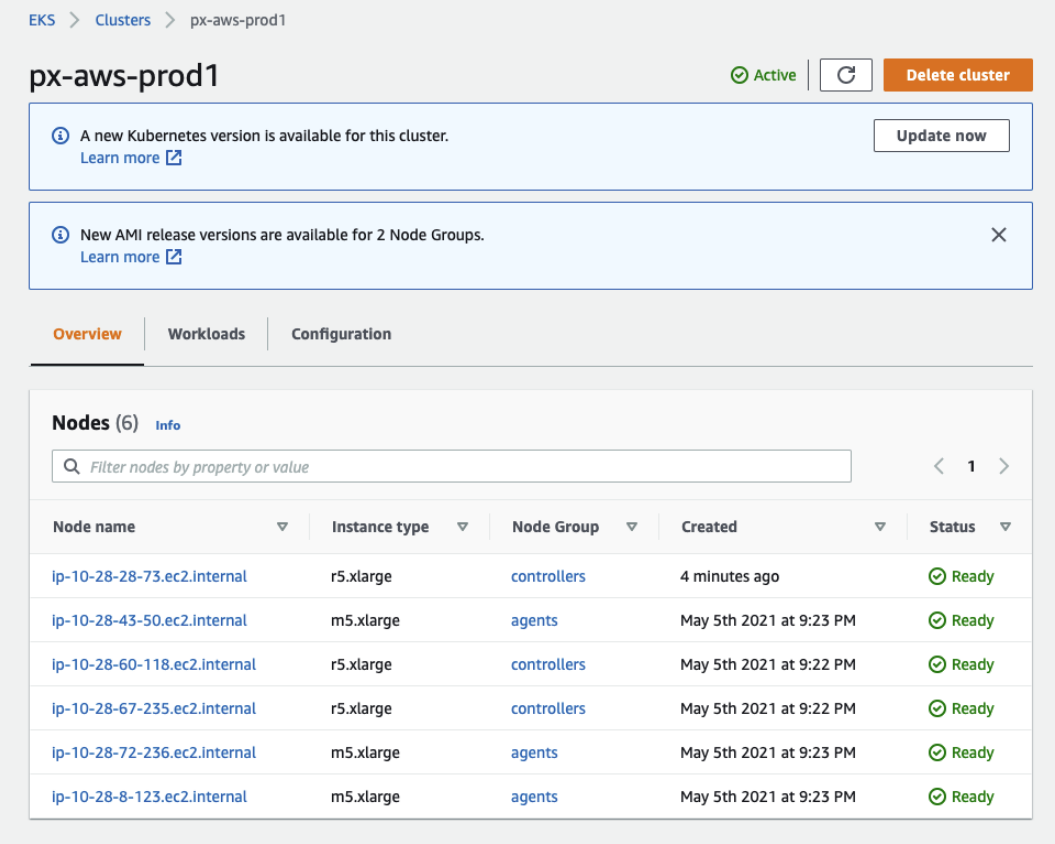
```
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/nvme1n1p2  STORAGE_MEDIUM_NVME     247 GiB         03 Jun 21 18:08 UTC
        total                   -                       247 GiB
        Cache Devices:
         * No cache devices
        Journal Device:
        1       /dev/nvme1n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 4 node(s) with storage (4 online), 2 node(s) without storage (2 online)
        IP              ID                                      SchedulerNodeName               Auth
                StorageNode     Used    Capacity        Status  StorageStatus   Version         Kernel
                                OS
        10.28.60.118    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-60-118.ec2.internal
        Disabled        Yes             25 GiB  247 GiB         Online  Up              2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.67.235    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-67-235.ec2.internal
        Disabled        Yes             24 GiB  247 GiB         Online  Up              2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.28.73     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-28-73.ec2.internal
        Disabled        Yes             10 GiB  247 GiB         Online  Up (This node)  2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.8.123     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-8-123.ec2.internal
        Disabled        Yes             21 GiB  247 GiB         Online  Up              2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.72.236    9937ed71-674a-421d-8226-0ae361ba5ebc    ip-10-28-72-236.ec2.internal
        Disabled        No              0 B     0 BOnline       No Storage      2.7.1.0-8a9e965
        5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.43.50     2931ea9b-9ade-4c82-8dfd-5dfcc84d4b0c    ip-10-28-43-50.ec2.internal
        Disabled        No              0 B     0 BOnline       No Storage      2.7.1.0-8a9e965
        5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
        Total Used      :  81 GiB
        Total Capacity  :  988 GiB
Upgrade - deploy new version of Portworx - Test 3.03
•       STC=$(kubectl get stc -n portworx | awk '{if(NR>1)print $1}')
•       kubectl patch stc $STC -n portworx --type json --patch '[{"op": "replace", "path":
"/spec/image", "value":"portworx/oci-monitor:2.7.1"}]'
Events:
  Type     Reason     Age                From     Message
  ----     ------     ----               ----     -------
  Normal   Pulling    15m                kubelet  Pulling image "portworx/oci-monitor:2.7.1"
  Normal   Pulled     15m                kubelet  Successfully pulled image "portworx/oci-
monitor:2.7.1" in 4.492196242s
  Normal   Created    15m                kubelet  Created container portworx
  Normal   Started    15m                kubelet  Started container Portworx
```

Storage Rebalance: Automated by Autopilot

This is not shown due to the small amount of data in the cluster.

OS Patch: Upgrade OS in Rolling Fashion, Test 3.06

Current state:

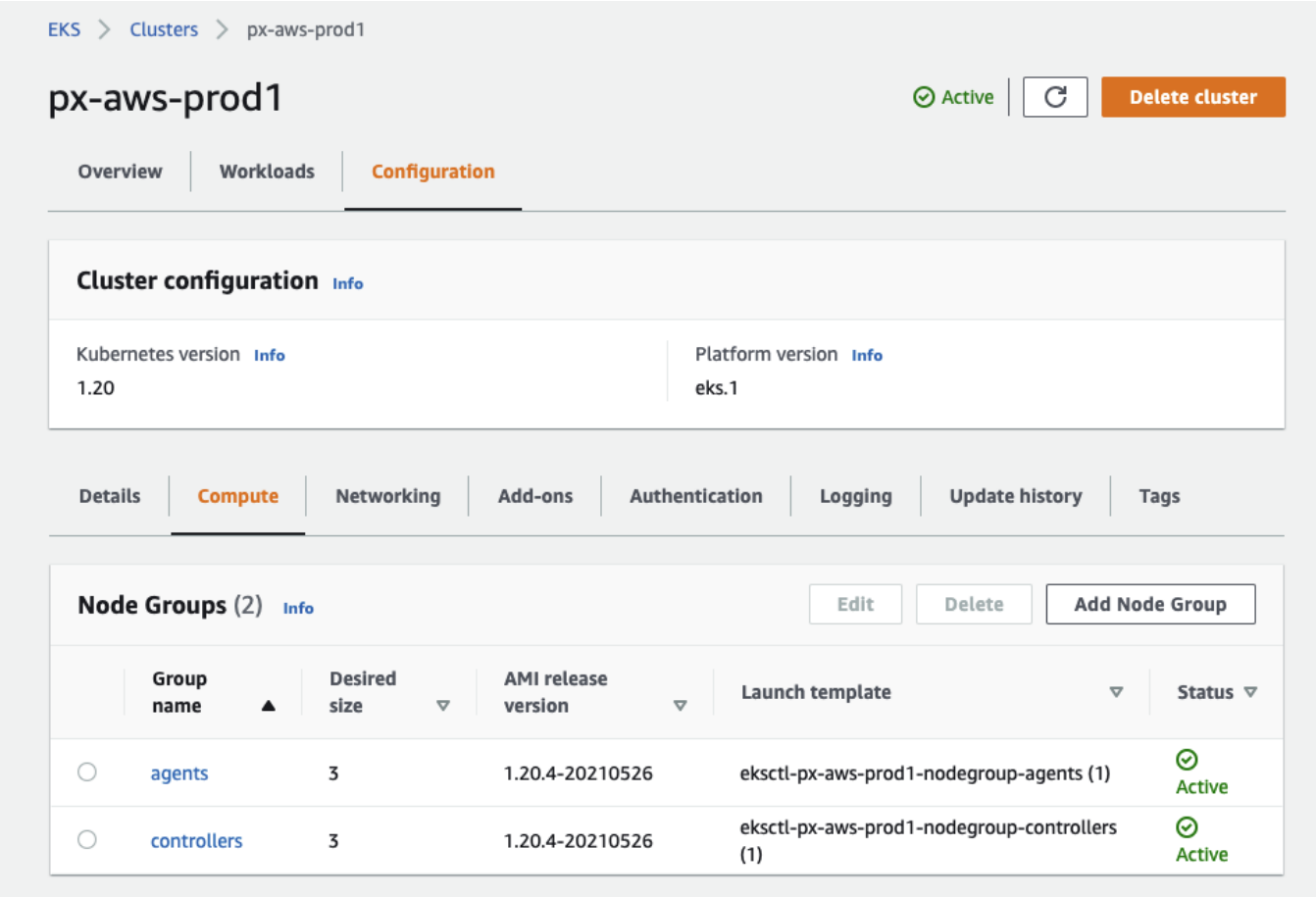


PXCTL Status:

```
Status: PX is operational
License: PX-Enterprise extended eval (expires in 59 days)
Node ID: f7dfb0f8-322c-45fd-aebb-d3cc0322c398
        IP: 10.28.60.118
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE            REGION
        0       HIGH            raid0           247 GiB 25 GiB  Online  us-east-1b      us-east-1
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/nvme1n1p2  STORAGE_MEDIUM_NVME     247 GiB         03 Jun 21 18:11 UTC
        total                   -                       247 GiB
        Cache Devices:
         * No cache devices
        Kvdb Device:
```

```
        Device Path     Size
        /dev/nvme2n1    150 GiB
         * Internal kvdb on this node is using this dedicated kvdb device to store its data.
        Journal Device:
        1       /dev/nvme1n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 4 node(s) with storage (4 online), 2 node(s) without storage (2 online)
        IP              ID                                      SchedulerNodeName               Auth
                StorageNode     Used    Capacity        Status  StorageStatus   Version         Kernel
                                OS
        10.28.60.118    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-60-118.ec2.internal
        Disabled        Yes             25 GiB  247 GiB         Online  Up (This node)  2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.67.235    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-67-235.ec2.internal
        Disabled        Yes             24 GiB  247 GiB         Online  Up              2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.28.73     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-28-73.ec2.internal
        Disabled        Yes             10 GiB  247 GiB         Online  Up              2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.8.123     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-8-123.ec2.internal
        Disabled        Yes             22 GiB  247 GiB         Online  Up              2.7.1.0-
8a9e965 5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.72.236    9937ed71-674a-421d-8226-0ae361ba5ebc    ip-10-28-72-236.ec2.internal
        Disabled        No              0 B     0 BOnline       No Storage      2.7.1.0-8a9e965
        5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
        10.28.43.50     2931ea9b-9ade-4c82-8dfd-5dfcc84d4b0c    ip-10-28-43-50.ec2.internal
        Disabled        No              0 B     0 BOnline       No Storage      2.7.1.0-8a9e965
        5.4.110-54.182.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
        Total Used      :  81 GiB
        Total Capacity  :  988 GiB
```

Upgrading Kubernetes via Web Console (Including AMI image to match versions):



Portworx Cluster status after update:

```
kubectl pxc pxctl status
>> Running pxctl on ip-10-28-81-112.ec2.internal
Status: PX is operational
License: PX-Enterprise extended eval (expires in 59 days)
Node ID: c92a6607-25f9-43a7-b38a-42e06dccfea8
        IP: 10.28.81.112
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE            REGION
        0       HIGH            raid0           247 GiB 10 GiB  Online  us-east-1c      us-east-1
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/nvme1n1p2  STORAGE_MEDIUM_NVME     247 GiB         03 Jun 21 21:44 UTC
        total                   -                       247 GiB
        Cache Devices:
         * No cache devices
        Kvdb Device:
        Device Path     Size
        /dev/nvme2n1    150 GiB
         * Internal kvdb on this node is using this dedicated kvdb device to store its data.
        Journal Device:
        1       /dev/nvme1n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 6 node(s) with storage (6 online)
        IP              ID                                      SchedulerNodeName               Auth            StorageNode     Used    Capa
city    Status  StorageStatus   Version         Kernel                          OS
        10.28.49.141    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-49-141.ec2.internal    Disabled        Yes             25 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.81.112    c92a6607-25f9-43a7-b38a-42e06dccfea8    ip-10-28-81-112.ec2.internal    Disabled        Yes             10 GiB  247
GiB             Online  Up (This node)  2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.82.219    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-82-219.ec2.internal    Disabled        Yes             22 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.11.87     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-11-87.ec2.internal     Disabled        Yes             10 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.36.85     5dfa752e-f366-4fac-b4fe-39554e51244e    ip-10-28-36-85.ec2.internal     Disabled        Yes             10 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.23.56     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-23-56.ec2.internal     Disabled        Yes             22 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
        Total Used      :  99 GiB
        Total Capacity  :  1.4 TiB
```

Resiliency Testing:

Node Reboot:

This was difficult to capture as the node went offline and came back online without interrupting operations outside of Userspace (non-daemonset); pods were rescheduled within seconds on a different node.

Node Shutdown:

```
Status: PX is operational
License: PX-Enterprise extended eval (expires in 58 days)
Node ID: 5dfa752e-f366-4fac-b4fe-39554e51244e
        IP: 10.28.36.85
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE            REGION
        0       HIGH            raid0           247 GiB 12 GiB  Online  us-east-1b      us-east-1
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/nvme1n1p2  STORAGE_MEDIUM_NVME     247 GiB         03 Jun 21 21:48 UTC
        total                   -                       247 GiB
        Cache Devices:
         * No cache devices
        Journal Device:
        1       /dev/nvme1n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 6 node(s) with storage (5 online)
        IP              ID                                      SchedulerNodeName               Auth            StorageNode     Used       C
apacity Status  StorageStatus   Version         Kernel                                  OS
        10.28.49.141    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-49-141.ec2.internal    Disabled        Yes             26 GiB     2
47 GiB          Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.81.112    c92a6607-25f9-43a7-b38a-42e06dccfea8    ip-10-28-81-112.ec2.internal    Disabled        Yes             11 GiB     2
47 GiB          Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.82.219    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-82-219.ec2.internal    Disabled        Yes             24 GiB     2
47 GiB          Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.11.87     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-11-87.ec2.internal     Disabled        Yes             25 GiB     2
47 GiB          Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.36.85     5dfa752e-f366-4fac-b4fe-39554e51244e    ip-10-28-36-85.ec2.internal     Disabled        Yes             12 GiB     2
47 GiB          Online  Up (This node)  2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.23.56     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-23-56.ec2.internal     Disabled        Yes             Unavailable
navailable      Offline Down            2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
```

AWS Auto-scaling replaced the node based on my definitions and the node was resynced and back online:

```
Status: PX is operational
License: PX-Enterprise extended eval (expires in 58 days)
Node ID: 185717ed-734f-464b-af38-0fa9288350cd
        IP: 10.28.9.164
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE            REGION
        0       HIGH            raid0           247 GiB 23 GiB  Online  us-east-1a      us-east-1
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/nvme2n1p2  STORAGE_MEDIUM_NVME     247 GiB         04 Jun 21 14:47 UTC
        total                   -                       247 GiB
        Cache Devices:
         * No cache devices
        Journal Device:
        1       /dev/nvme2n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 6 node(s) with storage (6 online)
        IP              ID                                      SchedulerNodeName               Auth            StorageNode     Used       Capa
city    Status  StorageStatus   Version         Kernel                                  OS
        10.28.49.141    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-49-141.ec2.internal    Disabled        Yes             26 GiB     247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.81.112    c92a6607-25f9-43a7-b38a-42e06dccfea8    ip-10-28-81-112.ec2.internal    Disabled        Yes             11 GiB     247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.82.219    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-82-219.ec2.internal    Disabled        Yes             24 GiB     247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.11.87     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-11-87.ec2.internal     Disabled        Yes             25 GiB     247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.36.85     5dfa752e-f366-4fac-b4fe-39554e51244e    ip-10-28-36-85.ec2.internal     Disabled        Yes             12 GiB     247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.9.164     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-9-164.ec2.internal     Disabled        Yes             23 GiB     247
GiB             Online  Up (This node)  2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
        Total Used      :  122 GiB
```

Disk Failure: (simulation)

```
❯ NODE=`kubectl get pods -l app=postgres -n postgres -o jsonpath='{.items[0].spec.nodeName}'`
echo $NODE
ip-10-28-82-219.ec2.internal
```

```
❯ kubectl get pods -n postgres
NAME                          READY   STATUS    RESTARTS   AGE
nginx-6598b75bbf-zmsfs        1/1     Running   0          123m
postgres-ddf7d7dfc-slrsq      1/1     Running   0          123m
```

Node IP-10-28-82-219 placed into maintenance mode to simulate a disk failure:

```
        1       /dev/nvme1n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 6 node(s) with storage (5 online)
        IP              ID                                      SchedulerNodeName               Auth            StorageNode     Used
apacity Status                  StorageStatus   Version         Kernel                          OS
        10.28.49.141    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-49-141.ec2.internal    Disabled        Yes             25 GiB
47 GiB          Online                  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.81.112    c92a6607-25f9-43a7-b38a-42e06dccfea8    ip-10-28-81-112.ec2.internal    Disabled        Yes             11 GiB
47 GiB          Online                  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.82.219    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-82-219.ec2.internal    Disabled        Yes             Unavailabl
navailable      Online (StorageDown)    In Maintenance  2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.11.87     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-11-87.ec2.internal     Disabled        Yes             25 GiB
47 GiB          Online                  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.36.85     5dfa752e-f366-4fac-b4fe-39554e51244e    ip-10-28-36-85.ec2.internal     Disabled        Yes             12 GiB
47 GiB          Online                  Up (This node)  2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.9.164     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-9-164.ec2.internal     Disabled        Yes             23 GiB
47 GiB          Online                  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
        Total Used      :  98 GiB
        Total Capacity  :  1.2 TiB
❯ kubectl get pod -n postgres -o wide
NAME                     READY   STATUS    RESTARTS   AGE    IP             NODE                          NOMINATED NODE   READINESS GA
S
nginx-6598b75bbf-zmsfs   1/1     Running   0          131m   10.28.37.247   ip-10-28-49-141.ec2.internal   <none>           <none>
postgres-ddf7d7dfc-slrsq 1/1     Running   0          131m   10.28.67.18    ip-10-28-82-219.ec2.internal   <none>           <none>
```

Note the PostgreSQL pod was not restarted and is still running on the same node, but using the storage replica on another node. Note also that the database is still available.

```
❯ POSTGRES_POD=`kubectl get po -n postgres -l app=postgres -o jsonpath='{.items[0].metadata.name}'`
kubectl exec -it -n postgres $POSTGRES_POD -- psql pxdemo -c "select count(*) from pgbench_accounts"
  count
---------
 5000000
(1 row)
```

Bringing Portworx on the node back online resynchronizes any changes and marks node available:

```
>> Running pxctl on ip-10-28-11-87.ec2.internal
Status: PX is operational
License: PX-Enterprise extended eval (expires in 58 days)
Node ID: 79bb6bbe-440d-4ac1-9ede-096181bba6ce
        IP: 10.28.11.87
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE            REGION
        0       HIGH            raid0           247 GiB 25 GiB  Online  us-east-1a      us-east-1
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:1     /dev/nvme1n1p2  STORAGE_MEDIUM_NVME     247 GiB         03 Jun 21 22:19 UTC
        total                   -                       247 GiB
        Cache Devices:
         * No cache devices
        Kvdb Device:
        Device Path     Size
        /dev/nvme3n1    150 GiB
         * Internal kvdb on this node is using this dedicated kvdb device to store its data.
        Journal Device:
        1       /dev/nvme1n1p1  STORAGE_MEDIUM_NVME
Cluster Summary
        Cluster ID: px-jenkins-prod
        Cluster UUID: eb9e5c27-dde2-4c08-8577-4b92c97e0e26
        Scheduler: kubernetes
        Nodes: 6 node(s) with storage (6 online)
        IP              ID                                      SchedulerNodeName               Auth            StorageNode     Used    Capa
city    Status  StorageStatus   Version         Kernel                                  OS
        10.28.49.141    f7dfb0f8-322c-45fd-aebb-d3cc0322c398    ip-10-28-49-141.ec2.internal    Disabled        Yes             25 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.81.112    c92a6607-25f9-43a7-b38a-42e06dccfea8    ip-10-28-81-112.ec2.internal    Disabled        Yes             11 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.82.219    c447d117-2ab1-41ec-9b6f-03782334c804    ip-10-28-82-219.ec2.internal    Disabled        Yes             24 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.11.87     79bb6bbe-440d-4ac1-9ede-096181bba6ce    ip-10-28-11-87.ec2.internal     Disabled        Yes             25 GiB  247
GiB             Online  Up (This node)  2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.36.85     5dfa752e-f366-4fac-b4fe-39554e51244e    ip-10-28-36-85.ec2.internal     Disabled        Yes             12 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
        10.28.9.164     185717ed-734f-464b-af38-0fa9288350cd    ip-10-28-9-164.ec2.internal     Disabled        Yes             23 GiB  247
GiB             Online  Up              2.7.1.0-8a9e965 5.4.117-58.216.amzn2.x86_64     Amazon Linux 2
Global Storage Pool
```

Note that during this operation the database was able to respond to queries and was not rescheduled to another node:

```
❯ kubectl get pods -n postgres -l app=postgres -o wide
NAME                        READY   STATUS    RESTARTS   AGE    IP           NODE                           NOMINATED NODE   READINESS GATES
postgres-ddf7d7dfc-slrsq    1/1     Running   0          138m   10.28.67.18  ip-10-28-82-219.ec2.internal   <none>           <none>
```


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