

PURE VALIDATED DESIGN

# Data Protection and Disaster Recovery for Amazon EKS with Portworx

Architecting a disaster recovery and data protection solution for Amazon Elastic Kubernetes Service with Portworx.



# Contents

|  |           |
|--|-----------|
| <b>Executive Summary .....</b>   | <b>3</b>  |
| <b>Introduction .....</b>  | <b>3</b>  |
| <b>Solution Overview .....</b>   | <b>4</b>  |
| Disaster Recovery .....  | 4         |
| Data Protection .....  | 5         |
| <b>Disaster Recovery and Data Protection for Modern Applications .....</b>             | <b>6</b>  |
| Solution Benefits.....   | 7         |
| <b>Amazon Elastic Kubernetes Service Overview .....</b>                                | <b>7</b>  |
| Amazon EKS Control Plane Architecture .....  | 7         |
| Amazon EKS Deployment Options.....   | 8         |
| <b>Portworx Overview .....</b>   | <b>8</b>  |
| PX-Store .....   | 9         |
| PX-Backup.....   | 9         |
| PX-DR.....   | 10        |
| PX-Autopilot.....  | 10        |
| Deployment Options .....   | 11        |
| <b>Disaster Recovery for Amazon EKS.....</b>   | <b>11</b> |
| <b>Synchronous DR.....</b>   | <b>12</b> |
| Sync DR: Deployment and Validation.....  | 13        |
| Asynchronous DR .....  | 18        |
| Async DR Failover.....   | 23        |
| Async DR Failback.....   | 23        |
| <b>Data Protection for Amazon EKS.....</b>   | <b>24</b> |
| Deployment and Validation .....  | 24        |
| <b>Kubernetes Backup and Restore .....</b>   | <b>28</b> |
| Amazon EKS with GP2-Backed Block Storage.....  | 28        |
| Amazon EKS with Amazon EFS-backed File Storage .....                                   | 30        |
| Application Migration from Google Kubernetes Engine (GKE) Clusters to Amazon EKS ..... | 32        |
| <b>Conclusion .....</b>  | <b>34</b> |
| <b>Additional Resources .....</b>  | <b>34</b> |
| <b>About the Author .....</b>  | <b>35</b> |

## Executive Summary

Organizations are increasingly adopting containers and Kubernetes to build their modern applications, and they are leveraging managed public cloud services like Amazon Elastic Kubernetes Service (EKS) to build and run their modern applications in production. Amazon EKS alleviates the management overhead involved in building and operating Kubernetes clusters but doesn't provide storage and data management features needed to run stateful applications in production. This document outlines how Portworx® by Pure Storage® can help complement Amazon EKS and provide capabilities like data protection and disaster recovery for containerized applications running on Amazon EKS. It discusses the need for modern data protection and disaster recovery solutions, the difference between the two, and how organizations can meet their service level agreements (SLAs) for modern applications running on Amazon EKS using Portworx.

This document is intended for DevOps engineers and administrators, cloud architects, and system architects who are interested in building a robust and resilient data protection and disaster recovery solution for their Amazon EKS clusters.

When a solution gets the Pure Validated Design (PVD) designation, it means that Pure has integrated and validated their leading-edge storage technology with an industry leading application solution platform to simplify deployment, reduce risk, and free up IT resources for business-critical tasks. The PVD process validates a solution and provides design consideration and deployment best practices to accelerate deployment. The PVD process assures the chosen technologies form an integrated solution to address critical business objectives. This document provides design guidelines and deployment steps for deploying Amazon EKS and Portworx to provide a modern infrastructure platform to run Kubernetes.

---

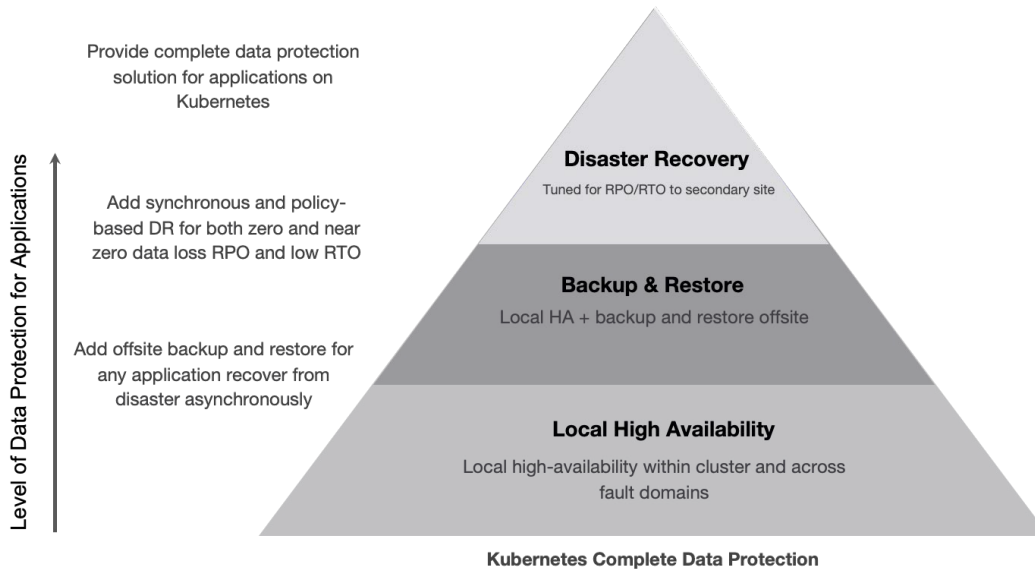
## Introduction

This document describes the benefits of using Portworx with Amazon EKS to run and protect stateful containerized applications. It is a validated design that includes design considerations, deployment specifics, and configuration best practices for building a data protection and disaster recovery solution for Amazon EKS.

It also explains the difference between disaster recovery and data protection and describes the steps involved in architecting a disaster recovery and a data protection solution for Amazon EKS. It offers different tactics to meet your SLA requirements when it comes to recovery time objective (RTO) and recovery point objective (RPO) by leveraging Portworx PX-DR and PX-Backup to build a disaster recovery and data protection solution respectively. To follow along with the deployment steps listed in this document, an architect will need to deploy one or multiple Amazon EKS clusters in a region of their choice and a secondary recovery Amazon EKS cluster in the same or different region. The Amazon EKS clusters should be running Portworx Enterprise as the Kubernetes storage layer for the disaster recovery solution, but Portworx Enterprise can be optional as the Kubernetes storage layer for the data protection solution.

## Solution Overview

Disaster recovery and data protection are key requirements to build and run applications in production. This requirement extends to the world of modern applications, as well. Kubernetes provides a reliable platform for orchestrating and maintaining the desired state for modern applications. But organizations still need a data protection and disaster recovery solution to manage scenarios like accidental deletion, data corruption, node failures, and availability zone and region outages.



Before we talk about how to design and architect such solutions using the Portworx portfolio, let's talk about what we mean by data protection and disaster recovery, and discuss some terminologies that we will use throughout this document.

## Disaster Recovery

Disaster recovery (DR) is the process of preparing for and recovering business critical infrastructure and applications from a disaster event. A disaster event can be anything that disrupts the normal operations of your applications. These disasters can be due to:

- **Human errors:** Accidental deletions, misconfigurations, pushing wrong code to production, etc.
- **Technical failures and attacks:** Hardware failures, software faults, power failures, denial of service attacks, cyber security attacks like ransomware, network connectivity issues, etc.
- **Natural disasters:** Floods, earthquakes, hurricanes, etc.

The main objective of building a disaster recovery plan is to ensure minimal disruption to the business and bring your applications back online as quickly as possible.

**Recovery Point Objective (RPO):** This metric is used to determine the amount of data loss that your organization can tolerate in the event of a disaster. Mission-critical applications demand an RPO of zero to ensure that there is no data loss if a disaster occurs.

**Recovery Time Objective (RTO):** This metric is used to determine the amount of time it takes your organization to fully recover from a disaster. Each application can have a different RTO requirement, which feeds into the designing the disaster recovery plan.

These RPO and RTO metrics can differ for all organizations, but as a rule of thumb, mission-critical applications demand a near zero or zero RPO, with an RTO of less than 15 mins. For Tier-2 applications, RPO can be two hours and an RTO of two to four hours. And for all the Tier-3 applications, RPOs can be four hours and an RTO of eight to 24 hours. To determine this for your organization, operation teams that are responsible for your production infrastructure need to meet with your application owners and developers and understand how the application is built. Your operations teams and your business owners will need to agree on the right balance between SLAs and cost of achieving that SLA requirement. Once this has been achieved, the operations team can then implement the right plan to manage different disaster events.

To meet these different RPO requirements, you can choose from any one of the following solutions:

- **Synchronous DR (sync DR)/Metro DR:** This scenario is useful for applications that need an RPO of zero. Sync DR gives you zero data loss in case of a disaster. Any data that is written to your primary site is automatically copied to your secondary or recovery site before the write operation is acknowledged. Designing a Sync DR architecture usually requires a latency requirement of a sub-10 ms round trip. Having a secondary copy of your data also allows you to restore your applications quickly, thus allowing you to meet the strict RTO requirement at the same time.
- **Asynchronous DR (async DR):** This scenario is useful for Tier 2 applications that can sustain data loss of up to a couple of hours. Async DR solution copies over your application data from the primary to the secondary site on a predetermined schedule. This schedule usually aligns with the RPO requirement guaranteed as part of the SLA.
- **Backups:** This scenario is useful for Tier 3 applications, where you only need to copy your data and store it in a backup repository every eight to 24 hours. This method falls under the data protection umbrella that we will discuss in the next section.

All disaster recovery or data protection plans should be evaluated frequently to identify and correct any gaps before an actual disaster event. This involves running DR drills occasionally to prepare your organization for a disaster event and verify that your DR plans work as expected.

## Data Protection

Data protection allows organizations to have a copy of their applications stored in a local or a remote backup repository, allowing them to restore their applications in case of accidental deletions, data corruption, ransomware attacks, etc. Architecting a robust backup and restore solution allows organizations to restore their applications to a previously known good state.

Data protection and disaster recovery are terms that are sometimes used interchangeably, which is not accurate. Disaster recovery means that you have a plan ready to execute in the event of a disaster, which allows for a low RPO and RTO metric. You need a secondary cluster (hot or cold) deployed and ready to failover your applications. Data protection is using your backup tool to store copies of your applications in a local or remote backup repository and leverage those snapshots to restore your applications.

## Disaster Recovery and Data Protection for Modern Applications

Modern applications that are built using containers and microservices and are deployed using Kubernetes have different sets of requirements when it comes to disaster recovery and data protection. Traditional tools won't work for containers because:

- **Traditional tools are machine-focused:** Traditional backup and DR solutions talk to the underlying machines (bare metal hosts or virtual machines) and protect them as the primary unit. These existing solutions, however, don't understand the microservices-based application architecture for the applications running on top. Containerized applications are distributed in nature, so each machine might have containers that might belong to different applications, and each application might have containers that are spread across multiple machines. If you are just protecting underlying machines, without understanding how modern applications are deployed and run in production, you might not be able to restore your applications as expected when needed.
- **Traditional tools don't speak Kubernetes:** Traditional backup and DR solutions are more focused on connecting directly with the physical servers or virtualization managers like vCenter server and inventorying all the different virtual machines running on top of it. A production Kubernetes cluster consists of multiple control plane nodes and multiple worker nodes that are responsible for running your application using constructs like Kubernetes pods, deployments, services, configmaps, etc. across different Kubernetes namespaces. If your tools are not able to understand and identify these constructs, you might not be able to restore your applications.
- **Traditional tools are centrally-managed:** Traditional backup and DR solutions don't have self-service or role-based access control built in. They are more focused on enabling the backup administrator or infrastructure administrator to create backup schedules and jobs and ensure that all the jobs are completed successfully. With modern applications, you need a more distributed approach where the backup administrator will add the backup locations and create backup schedules. But individual application owners or developers might best know how to protect a particular application and would prefer self-service access to have control over how their application is protected.

Modern disaster recovery and data protection tools like PX-DR and PX-Backup from Portworx provide a solution that understands and works with modern applications and Kubernetes because they are:

- **Container-granular:** PX-DR and PX-Backup run on top of your Kubernetes clusters. PX-DR runs on your primary and secondary Kubernetes clusters, whereas PX-Backup can run on the same Kubernetes cluster as your application or on a dedicated backup cluster. PX-DR provides single-click backup and restore capabilities for your containerized applications. These tools are built from the ground up for Kubernetes, instead of being retrofitted to work with containers, thus allowing you to fully protect your modern applications.
- **Kubernetes namespace aware:** PX-DR and PX-Backup talk to the Kubernetes API server, and can work at the namespace level inside your Kubernetes cluster. These tools identify all the different Kubernetes objects like pods, deployments, services, config maps, and secrets and help you protect everything that constitutes your containerized application.
- **Application consistent:** Containerized stateful applications are distributed in nature, so it is essential to have a backup and disaster recovery solution that can help take application-consistent snapshots across your whole cluster—not just crash-consistent snapshots. PX-Backup and PX-DR allow administrators to take application-consistent snapshots of your distributed applications, so there is no data loss in case of a disaster.
- **Capable of backing up data and app config:** PX-Backup and PX-DR allow you to protect your entire application from end to end. This includes all the Kubernetes objects, application configurations, and persistent volumes that store your application data.



- **Hybrid cloud ready:** PX-Backup and PX-DR work with all Kubernetes distributions, so you can run your applications on Amazon EKS in AWS or on EKS-Anywhere clusters on-prem. You can also use PX-Backup and PX-DR to restore your applications across a hybrid cloud (on-prem vs. public cloud) environment.
- **Unified solution for any application running anywhere:** PX-Backup helps protect any application running on any cloud or on-prem environment, rather than needing a point solution that is specific to any particular application or cloud environment.

## Solution Benefits

This solution enables organizations to build robust and resilient disaster recovery and data protection solutions for their modern applications running on Amazon EKS clusters. Organizations can use Portworx PX-DR to build synchronous and asynchronous disaster recovery solutions to recover quickly from a disaster event with no data loss, or with minimal data loss, respectively. Using PX-Backup, organizations can build a backup and restore solution that helps them tolerate accidental deletions, data loss, or even ransomware attacks. These architectures help organizations minimize the amount of downtime in the event of a disaster, so they can bring their applications back online and continue serving their customers.

## Amazon Elastic Kubernetes Service Overview

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Amazon EKS can:

- Run and scale the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability.
- Automatically scale control plane instances based on load, detect and replace unhealthy control plane instances, and provide automated version updates and patching for them.
- Integrate with many AWS services to provide scalability and security for your applications.
- Run up-to-date versions of the open-source Kubernetes software, so you can use all the existing plugins and tooling from the Kubernetes community. Applications running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, no matter whether they're running in on-premises data centers or public clouds.

## Amazon EKS Control Plane Architecture

Amazon EKS runs a single tenant Kubernetes control plane for each cluster. The control plane infrastructure is not shared across clusters or AWS accounts. The control plane consists of at least two API server instances and three etcd instances that run across three Availability Zones within a Region.

Amazon EKS is capable of:

- Actively monitoring the load on control plane instances and automatically scaling them to ensure high performance.
- Automatically detecting and replacing unhealthy control plane instances, restarting them across the Availability Zones within the region as needed.
- Leveraging the architecture of AWS Regions to maintain high availability. Because of this, Amazon EKS can offer an SLA for API server endpoint availability.

Amazon EKS uses Amazon VPC network policies to restrict traffic between control plane components to within a single cluster. Control plane components for a cluster can't view or receive communication from other clusters or other AWS accounts,

except as authorized with Kubernetes RBAC policies. This secure and highly available configuration makes Amazon EKS reliable and recommended for production workloads.

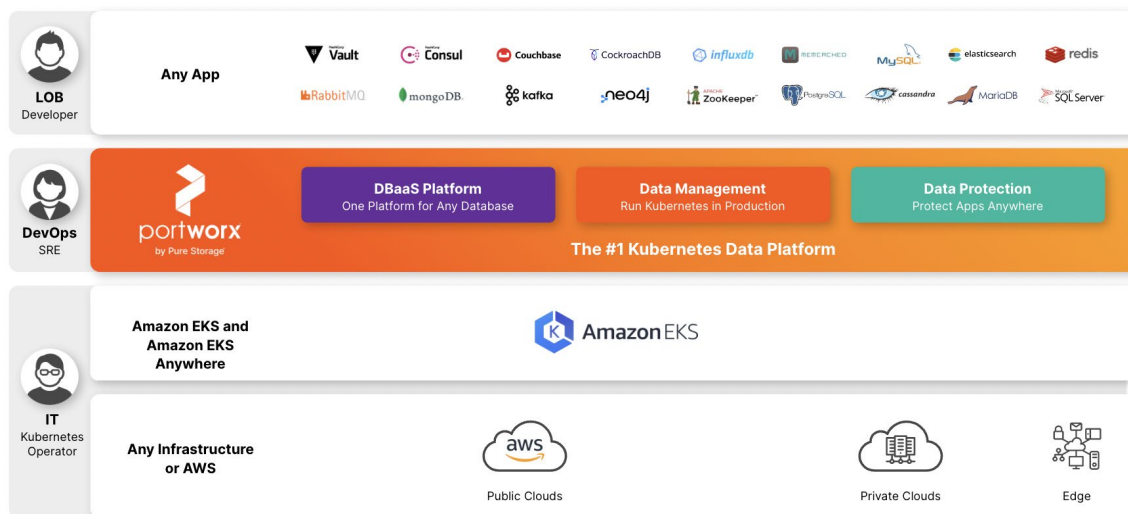
## Amazon EKS Deployment Options

You can use Amazon EKS with any, or all, of the following deployment options:

- **Amazon EKS:** Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes.
- **Amazon EKS on AWS Outposts:** AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities.
- **Amazon EKS Anywhere:** Amazon EKS Anywhere is a deployment option for Amazon EKS that enables you to easily create and operate Kubernetes clusters on-premises. Both Amazon EKS and Amazon EKS Anywhere are built on the Amazon EKS Distro.
- **Amazon EKS Distro:** Amazon EKS Distro follows the same Kubernetes version release cycle as Amazon EKS and is provided as an open-source project.

## Portworx Overview

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



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

- **PX-Migrate:** Allows applications to be easily migrated across clusters, racks, and clouds.
- **PX-Secure:** Provides access controls and enables data encryption at a cluster, namespace, or persistent volume level.





- **PX-DR:** Allows applications to have a zero RPO failover across data centers in a metro area as well as continuous backups across the WAN for even greater protection.
- **PX-Backup:** Allows enterprises to back up and restore the entire Kubernetes application, including data, app configuration, and Kubernetes objects, to any backup location—including AWS S3, Azure Blob, and others—with the click of a button.
- **PX-Autopilot:** Provides rules-based auto-scaling for persistent volumes and storage pools.

## PX-Store

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

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

## PX-Backup

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



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

- **App-consistent backup and restore:** Easily protect and recover applications regardless of how they are initially deployed on, or rescheduled by, Kubernetes.
- **Seamless migration:** Move a single Kubernetes application or an entire namespace between clusters (on-prem, cloud, or between on-prem and cloud).
- **Compliance management:** Manage and enforce compliance and governance responsibilities with a single pane of glass for all your containerized applications. This includes the ability to use role-based access control and create backup jobs to comply with the 3-2-1 backup policies.
- **Streamlined storage integration:** Back up and recover cloud volumes with storage providers, including Amazon EBS, Google Persistent Disk, Azure Managed Disks, and CSI-enabled storage.
  - **Support for CSI-compliant storage:** PX-Backup supports protecting applications running on any Kubernetes storage layer, allowing users to bring their own Kubernetes storage.
  - **Self-service and role-based access control:** PX-Backup allows developers and DevOps administrators to create their own backup jobs using role-based access control, thus allowing each application owner to customize data protection for their applications.

## PX-DR

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

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

## PX-Autopilot

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

- **Grow storage capacity on demand:** Automate your applications' growing storage demands while also minimizing disruptions. Set growth policies to automate cloud drive and Kubernetes integration to ensure your application's storage needs are met without performance or availability degradations.
- **Slash storage costs by half:** Intelligently provision cloud storage only when needed and eliminate the problem of paying

for storage when over-provisioned instead of when consumed. Scale at the individual volume or entire cluster level to save money and avoid application outages.

- **Integrate with All Major Clouds and VMware:** PX-Autopilot natively integrates with AWS, Azure, Google, and VMware Tanzu, enabling you to achieve savings and increase automated agility across all your clouds.

## Deployment Options

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

- **Key value database (KVDB):** Portworx requires a key-value database such as etcd for configuring storage. For most deployments, you can create a deployment architecture with an internal KVDB instance (highly available ETCD instance). However, we recommend using an external highly available KVDB database for the following two scenarios:
  - The first scenario is when the PX-DR is used to configure SyncDR for Kubernetes clusters, where the KVDB instance is shared across the two Kubernetes clusters.
  - The second scenario in which a dedicated etcd cluster should be used is for large-scale deployment, with 25 or more worker nodes, in which a heavy dynamic provisioning activity takes place.
- **Dedicated journal device:** A dedicated journal device can be specified to buffer metadata writes.
- **Dedicated cache device:** A dedicated cache device can be specified to [improve performance](#) by acting as a read/write-through cache.
- **Container storage interface (CSI) API compatibility:** You can choose the option to deploy Portworx with CSI enabled if you will be using PX-Security.
- **Stork:** Stork is a storage-aware scheduler that attempts to co-locate application pods onto the nodes that may have a replica of any persistent volumes that the application uses. Use Stork if your underlying infrastructure uses servers with dedicated internal storage or servers with dedicated network-attached storage appliances.

## Disaster Recovery for Amazon EKS

Portworx PX-DR allows organizations to build robust disaster recovery architectures for applications running on Amazon EKS. In this section, we will talk about how you can leverage PX-DR to build synchronous DR (Zero RPO) and asynchronous DR solutions. But before configuring PX-DR, we need to understand the following terms:

**ClusterPair:** To failover an application running on one Kubernetes cluster to another Kubernetes cluster, you need to migrate the resources between them. ClusterPair is an object that ensures mutual trust that is required for communication between the source and target Kubernetes clusters. This creates a pairing between the scheduler (Kubernetes), so that all resources can be migrated between them.

**SchedulePolicy:** Schedule policies are used to specify when Portworx should trigger a specific operation. Schedule policies themselves do not contain any actions specifically, but they can be referenced by MigrationSchedules (see below). A SchedulePolicy object has the following sections:

- **Interval:** For interval operations, it is how frequently Portworx will trigger the operation.
- **Daily:** For daily operations, Portworx will trigger the operation at the specified time every day.

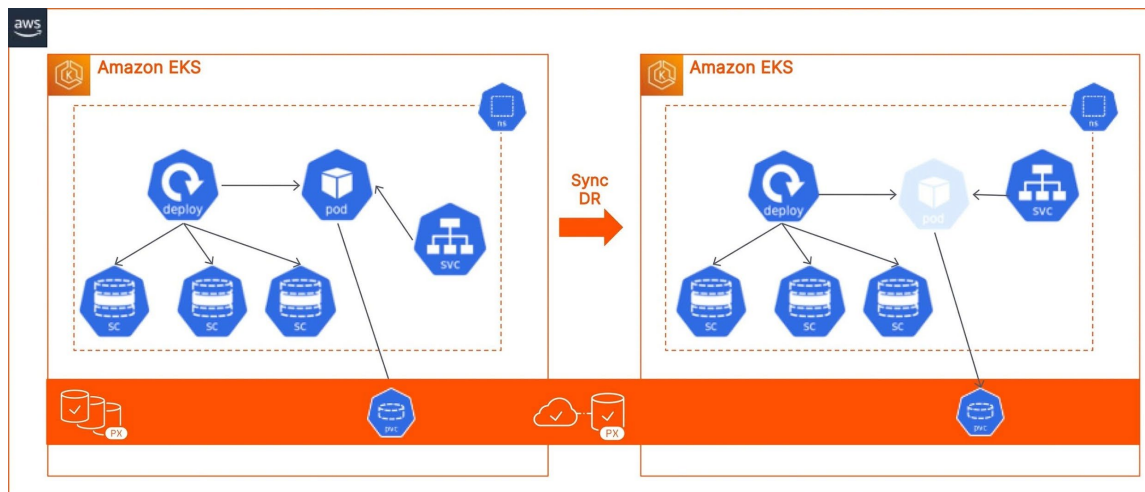
- **Weekly:** For weekly operations, Portworx will trigger the operation at the specified day and time every week.
- **Monthly:** For monthly operations, Portworx will trigger the operation at the specified day and time every month.

**MigrationSchedule:** A MigrationSchedule object defines the actions that need to be taken when migrating resources from the source to the target Amazon EKS cluster. A MigrationSchedule object has the following sections:

- **namespace:** A MigrationSchedule object is a namespaced object. You need to create a MigrationSchedule object for each application running in a different namespace.
- **clusterPair:** Each MigrationSchedule object uses a ClusterPair object to migrate resources between the source and the target Kubernetes clusters.
- **includeResources:** Setting this flag to “true” implies that we want each migration to include all the different Kubernetes objects inside a namespace.
- **startApplications:** Setting this flag to “false” tells Kubernetes not to start up any pods on the target cluster until we manually failover an application from the source to the target site.
- **IncludeVolumes:** Setting this flag to “true” implies that we need the migration object to copy all the data from the source to the target Kubernetes cluster.
- **schedulePolicyName:** Each MigrationSchedule needs to be associated with a SchedulePolicy object for Portworx to know when to trigger the migration operation given by the intervals defined in the SchedulePolicy object.

## Synchronous DR

For modern applications that need an RPO of 0, PX-DR can enable administrators to build synchronous DR solutions where a single Portworx cluster is stretched across two different Amazon EKS clusters. These Amazon EKS clusters can reside in the same or different AWS regions if they are able to meet the 10ms round trip latency requirement for Sync DR.



## Sync DR: Deployment and Validation

To deploy a Sync DR topology using two Amazon EKS clusters, use the following steps.

1. Deploy two Amazon EKS clusters using [eksctl](#), AWS CLI, or AWS Management Console.

For Sync DR, Portworx will deploy one *stretched* storage cluster. This requires the use of an external etcd instance as the key value database (KVDB). For testing purposes, you can deploy a single etcd instance on a Centos VM using the commands below.

```
yum install etcd

#Edit the /etc/etcd/etcd.conf file and set the ETCD_INITIAL_CLUSTER,
ETCD_INITIAL_ADVERTISE_PEER_URLS, ETCD_ADVERTISE_CLIENT_URLS, ETCD_LISTEN_CLIENT_URLS to the
management IP address of the controller node to enable access by other nodes via the management
network:
#[Member]
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://<<IP>>:2380"
ETCD_LISTEN_CLIENT_URLS="http://<<IP>>:2379"
ETCD_NAME="controller"
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://<<IP>>:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://<<IP>>:2379"
ETCD_INITIAL_CLUSTER="controller=http://<<IP>>:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster-01"
ETCD_INITIAL_CLUSTER_STATE="new"

systemctl enable etcd
systemctl start etcd
```

2. Verify that your Amazon EKS clusters can talk to the etcd instance over port 2379. For production workloads, we **always** recommend installing a 3-instance highly-available etcd cluster spread across sites. You may also wish to [enable TLS for etcd on production setups](#).
3. Allow communication between the two Amazon EKS clusters over ports 9001 to 9020 using Security groups. This allows the EKS worker nodes to communicate and enable creation of a single stretched Portworx cluster.
4. Next, generate a separate Portworx configuration with the already-deployed external etcd cluster using the spec generator on [PX-Central](#). While generating the installation spec, make sure you provide the same value on both clusters for the following arguments:
  - Cluster ID (StorageCluster: metadata.name)
  - KVDB endpoints (StorageCluster: spec.kvdb.endpoints), including any optional etcd security parameters

- Specify clusterdomains for the Amazon EKS clusters. A clusterdomain identifies a subset of nodes from the stretched Portworx cluster that are a part of the same failure domain. The site-a and site-b Amazon EKS clusters need to be respectively tagged with the cluster\_domain argument
- Edit the Portworx StorageCluster object manifests you obtained from the spec generator and add a `-cluster_domain` argument for each Amazon EKS cluster:

```
##### Site-A's StorageCluster object manifest #####
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  annotations:
    portworx.io/misc-args: "-cluster_domain site-a"

##### Site-B's StorageCluster object manifest #####
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  annotations:
    portworx.io/misc-args: "-cluster_domain site-b"
```

- Deploy the above yaml files against your Amazon EKS clusters and wait for the Portworx cluster to come online. You can then fetch the storkctl utility and verify that cluster domains are configured correctly, using the following:

```
STORK_POD=$(kubectl get pods -n kube-system -l name=stork -o jsonpath='{.items[0].metadata.name}')
&&
kubectl cp -n kube-system $STORK_POD:/storkctl/linux/storkctl ./storkctl
sudo mv storkctl /usr/local/bin &&
sudo chmod +x /usr/local/bin/storkctl

site-a# storkctl get clusterdomainsstatus
```

As we are building a DR solution for AWS EKS, we will need to configure a secret in the kube-system namespace and pass that secret to Stork running on the source cluster:

```
kubectl create secret generic --from-file=$HOME/.aws/credentials -n kube-system aws-creds
###
kubectl edit stc <<StorageClusterName>> -n kube-system
### Add the following under the stork section of the StorageCluster spec.
stork:
  enabled: true
  volumes:
```

```
- name: aws-creds
  mountPath: /root/.aws/
  readOnly: true
  secret:
    secretName: aws-creds
```

8. Apply the PX-DR license to the Portworx cluster from one of the Portworx pods:

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

kubectl exec -it $PXPOD -n kube-system -- /opt/pwx/bin/pxctl license activate <<license_key>>
```

Next, let's create an object store credential and a ClusterPair object by EXECing into the Portworx pod. Object store credentials can be created by using the following command:

```
/opt/pwx/bin/pxctl credentials create \
--provider s3 \
--s3-access-key <aws_access_key> \
--s3-secret-key <aws_secret_key> \
--s3-region <aws-region-name> \
--s3-endpoint s3.amazonaws.com \
--s3-storage-class STANDARD \
clusterPair_<UUID_of_destination_cluster>
```

You can get the UUID of the site-b cluster by using the following command:

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

Now, let's create a ClusterPair spec on site-b cluster, which is a namespaced object, meaning you need to ensure that you have the same namespace created on both sites.

```
storkctl generate clusterpair -n demo clusterpairname > clusterpairname.yaml
```

Edit the generated yaml file and remove the <<insert\_storage\_options\_here>>: "" line. In SyncDR configuration, the replication is handled at the storage layer.

Next, let's log in to the site-a cluster and apply this configuration file. Verify that you have created the "demo" namespace on the primary cluster as well:

```
kubectl create ns demo
kubectl apply -f clusterpairname.yaml
```

Verify that the ClusterPair is online, using the following:

```
storkctl get clusterpair -n demo
```

Now, let's go ahead and create a schedule policy that will be used by our MigrationSchedule object to copy Kubernetes objects and metadata between the two clusters:

```
cat > testpolicy.html <<_EOF
apiVersion: stork.libopenstorage.org/v1alpha1
kind: SchedulePolicy
metadata:
  name: testpolicy
  namespace: demo
policy:
  interval:
    intervalMinutes: 15
_EOF

kubectl apply -f testpolicy.yaml
```

Let's create a MigrationSchedule object that will replicate our application from the site-a cluster to the site-b cluster:

```
cat > migrationschedule.yaml <<_EOF
apiVersion: stork.libopenstorage.org/v1alpha1
kind: MigrationSchedule
metadata:
  name: demomigrationschedule
  namespace: demo
spec:
  template:
    spec:
      clusterPair: clusterpairname #Reference to the clusterpair object
      includeResources: true # Copies all Kubernetes objects for every migration
      startApplications: false # Stops from deploying app pods at the DR site
```



```

    includeVolumes: false # Replication handled at the storage layer
  namespaces:
    - demo
  schedulePolicyName: testpolicy
_EOF
kubectl apply -f migrationschedule.yaml

```

Next, use the following commands to verify the status of the migration schedules and see if the different migration jobs are successful:

```

storkctl get migrationschedule -n demo
kubectl get migrationschedule -n demo

storkctl get migrations -n demo
kubectl get migrations -n demo

```

You have successfully deployed a SyncDR topology for your Amazon EKS clusters. Use the following steps to failover and failback your application from the site-a cluster to the site-b cluster and vice versa.

### Sync DR Failover

To failover an application, you need to instruct Stork and Portworx that one of your Amazon EKS clusters is down and inactive.

To initiate a failover, we need to first deactivate or mark our source cluster as inactive. After deactivating, the primary cluster goes out of sync.

```

storkctl deactivate clusterdomain site-a
storkctl get clusterdomainsstatus

```

If you are just simulating a failover to test Sync DR, you need to scale down your applications on the source side. In case of a disaster event, your source cluster will be offline, and the pods won't be in a running state.

```

kubectl scale --replicas 0 deployment/<<deployment_name>> -n demo

```

Now we can log in the site-b cluster and activate the migration to bring the application online on the secondary site.

```

storkctl activate migrations -n demo

```

Verify that your application is online by using the following command:

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

You have now successfully performed a failover operation in your SyncDR topology.

### Sync DR Failback

Once the site-a cluster is back up and running, the Portworx nodes in that cluster will not immediately rejoin the stretched cluster. They will stay in the “Out of Quorum” state until you explicitly activate the cluster domain.

1. Activate the site-a cluster using storkctl:

```
storkctl activate clusterdomain site-a  
storkctl get clusterdomainsstatus
```

2. Stop the application on the secondary site-b cluster:

```
kubectl scale --replicas 0 deployment/⟨⟨deployment_name⟩⟩ -n demo
```

3. Start the application on the primary site-a cluster:

```
kubectl scale --replicas 1 deployment/⟨⟨deployment_name⟩⟩ -n demo
```

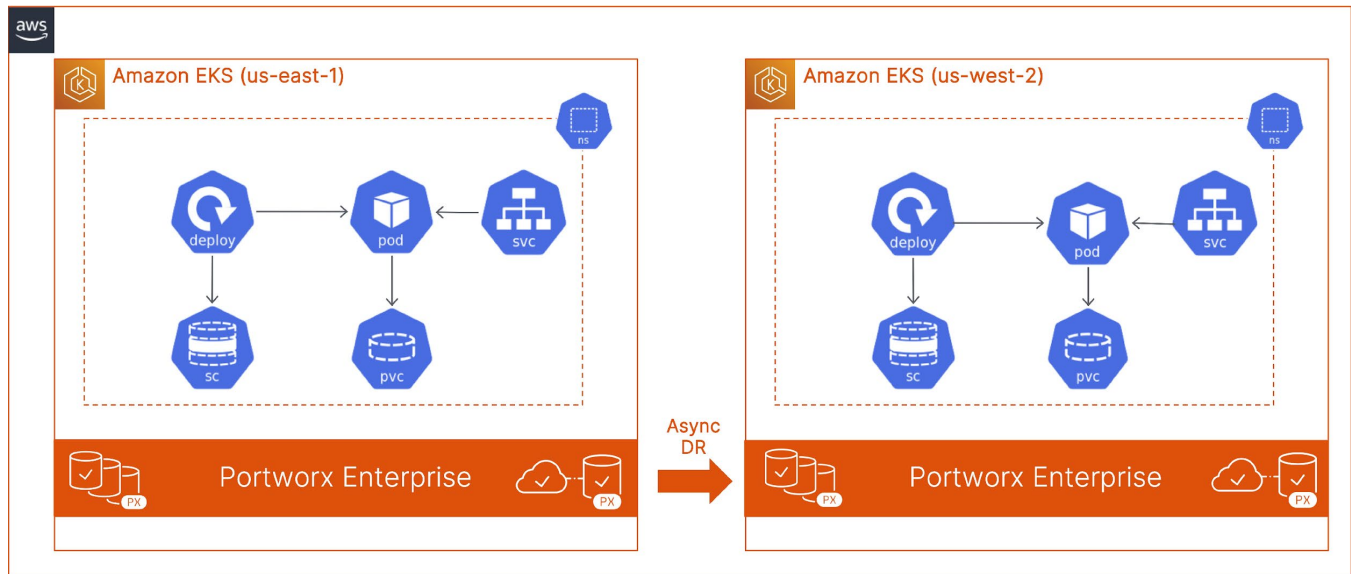
4. Next, let's resume the MigrationSchedule, using the following:

```
storkctl resume migrationschedule demomigrationschedule -n demo
```

You have now successfully performed a failback operation in your SyncDR topology.

### Asynchronous DR

In addition to SyncDR, PX-DR can be deployed in an AsyncDR configuration. This is useful in scenarios where your Amazon EKS clusters might be running in different AWS regions (and/or you otherwise cannot meet the sub-10 ms requirements for SyncDR).



### Async DR: Deployment and Validation

Deploy a separate source and a destination Amazon EKS cluster with a minimum configuration of 3 worker nodes in each cluster.

Install Portworx using [PX-Central](#). Unlike SyncDR, we don't need a shared etcd instance between the two Amazon EKS clusters. You can generate spec files for each cluster that by default use built-in etcd instances [deploying one Portworx storage cluster](#) per Amazon EKS cluster.

Once the Portworx pods are up and running in the kube-system namespace, exec into the Portworx pod on both the clusters and apply the PX-DR license by using the following command:

```
kubectl exec -it <<portworx_pod>> -n kube-system -- /opt/pwx/bin/pxctl license activate
<<license_key>>
```

Once you have your EKS clusters running, you will need to create appropriate inbound rules in your AWS Security Groups to allow communication between the two clusters over ports 9001 and 9010.

Next, since we are building a DR solution for AWS EKS, we will need to configure a secret in the kube-system namespace and pass that secret to Stork running on the source cluster:

```
kubectl create secret generic --from-file=$HOME/.aws/credentials -n kube-system aws-creds
###
kubectl edit stc <<StorageClusterName>> -n kube-system
### Add the following under the stork section of the StorageCluster spec.
stork:
  enabled: true
  volumes:
```

```
- name: aws-creds
  mountPath: /root/.aws/
  readOnly: true
  secret:
    secretName: aws-creds
```

Add a new annotation in the Portworx Storage Cluster specification, to set the service type for Portworx-service to LoadBalancer. Edit the Portworx-service in the kube-system namespace and change the Service Type to LoadBalancer.

**NOTE:** Do not enable load balancing without authorization enabled on the Portworx cluster.

```
kubectl edit stc <<storageclustername>> -n kube-system

metadata:
  annotations:
    portworx.io/service-type: "portworx-service:LoadBalancer"
```

Next, let's create an object store credential and a ClusterPair object. Object store credentials can be created by using the following command. You need to create the same credential on both Portworx storage clusters.

```
/opt/pwx/bin/pxctl credentials create \
--provider s3 \
--s3-access-key <aws_access_key> \
--s3-secret-key <aws_secret_key> \
--s3-region us-east-1 \
--s3-endpoint s3.amazonaws.com \
--s3-storage-class STANDARD \
clusterPair_<UUID_of_destination_cluster>
```

You can get the UUID of the site-b cluster by using the following command:

```
# you will need to use a different kubectl context to connect to the other cluster just for this
command and re-set the PXPOD variable

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

kubectl exec -it $PXPOD -n kube-system -- /opt/pwx/bin/pxctl status | grep 'Cluster UUID:'
```

Next, let's create a ClusterPair spec on the site-b cluster. ClusterPair specification is a namespaced object. So, ensure that you have the same namespace created on both sites.

```
storkctl generate clusterpair -n demo clusterpairname > clusterpairname.yaml
```

Once you have generated the above yaml file, edit it to update the following options:

**NOTE:** Ensure that all fields in the options are quoted.

```
options:
  ip: "<ip_of_remote_px_node>"
  port: "<port_of_remote_px_node_default_9001>"
  token: "<destination_cluster_token>"
  mode: "DisasterRecovery"
```

To fetch the token value, check the destination Portworx cluster, using the following:

```
# you will need to use a different kubectl context to connect to the other cluster just for this
command and re-set the PXPOD variable

kubectl exec -it $PXPOD -n kube-system -- /opt/pwx/bin/pxctl cluster token show
```

After filling in the missing information in the clusterpair yaml, apply it against your source cluster:

```
kubectl apply -f clusterpair.yaml
```

Verify that the clusterpair is online, using the following:

```
storkctl get clusterpair -n demo
```

Create a schedule policy object that will be used by our MigrationSchedule object to copy Kubernetes objects, metadata and persistent volumes between the two clusters.

```
cat > testpolicy.html <<_EOF
apiVersion: stork.libopenstorage.org/v1alpha1
kind: SchedulePolicy
metadata:
  name: testpolicy
  namespace: demo
```

```
policy:
  interval:
    intervalMinutes: 15
_EOF

kubectl apply -f testpolicy.yaml
```

Next, create a MigrationSchedule object on the source cluster.

```
cat > demomigrationschedule.yaml <<_EOF
apiVersion: stork.libopenstorage.org/v1alpha1
kind: MigrationSchedule
metadata:
  name: demomigrationschedule
  namespace: demo
spec:
  template:
    spec:
      clusterPair: clusterpairname
      includeResources: true
      startApplications: false
      includeVolumes: true
      namespaces:
        - demo
      schedulePolicyName: testpolicy
_EOF

kubectl apply -f demomigrationschedule.yaml
```

**NOTE:** In Async DR scenarios, we will not use the “includeVolumes:false” parameters because we want PX-DR to migrate incremental copies of our persistent volumes from the source to the destination cluster as well.

You can monitor the MigrationSchedule and migration objects by using the following commands:

```
kubectl get migrationschedule -n demo

kubectl get migrations -n demo

kubectl describe migrationschedule -n demo
```

```
storkctl get migrations -n demo

storkctl get migrationschedule -n demo
```

You have now successfully deployed an Async DR configuration for your Amazon EKS clusters.

To expedite failback operations for Async DR, you should also create object store credentials and a ClusterPair in the reverse direction (destination to source cluster) as well as all the other steps above.

## Async DR Failover

To perform a failover operation from the source to the destination cluster, use the following steps. If your source cluster is still online, use the following commands to suspend the MigrationSchedule and scale down your application.

```
storkctl suspend migrationschedule demomigrationschedule -n demo

kubectl scale --replicas 0 deployment/<<deployment-name>> -n demo
```

You can now log into your destination cluster and activate the migration.

```
storkctl activate migrations -n demo
```

You will see pods getting deployed for your application in the demo namespace. Use the following command to verify that your application is up and running on the destination cluster:

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

You have now successfully performed an Async DR failover operation from the source to the destination site.

## Async DR Failback

When your source cluster is back up and running, we can perform a failback operation to move the application back to the source cluster. Use the following steps to failback your application:

Create a new SchedulePolicy and a MigrationSchedule (demo-migrationschedule-failback) object on the destination cluster. This can be configured by using the commands in the Async DR deployment and validation section.

Once the first migration is successful, we can suspend the MigrationSchedule from the destination to the source site and scale down our application.

```
storkctl suspend migrationschedule demo-migrationschedule-failback -n demo

kubectl scale --replicas 0 deployment/⟨⟨deployment-name⟩⟩ -n demo
```

Activate the migration on the source cluster to bring your application back online.

```
storkctl activate migrations -n demo
```

You will see pods getting deployed for your application in the demo namespace. Use the following command to verify that your application is up and running on the destination cluster.

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

You have successfully performed an Async DR failback operation from the source to the destination site.

## Data Protection for Amazon EKS

This use case focuses on data protection for containerized applications running on Amazon EKS clusters. Portworx PX-Backup provides a modern Kubernetes-native, end-to-end backup and restore solution for Amazon EKS clusters.

### Deployment and Validation

To install PX-Backup on your Amazon EKS cluster, use the following steps:

**NOTE:** If you are running PX-Backup on a dedicated Amazon EKS cluster, you don't need to install Portworx on your cluster.

1. Navigate to PX-Central, create a "New Spec," select **PX-Backup** and click **Next**.
2. Enter a namespace where you want to install all the PX-Backup components. Select **Helm 3** and **Cloud**. Enter the name of the storage class that you want to use to install PX-Backup. This storage class will be used to deploy the persistent volumes needed by PX-Backup. Click **Next**.
3. Read through the License Agreement and, if acceptable, click **Agree**.

Follow the two-step process to install PX-Backup on your Amazon EKS cluster. We used the gp2 storage class to generate the commands below:

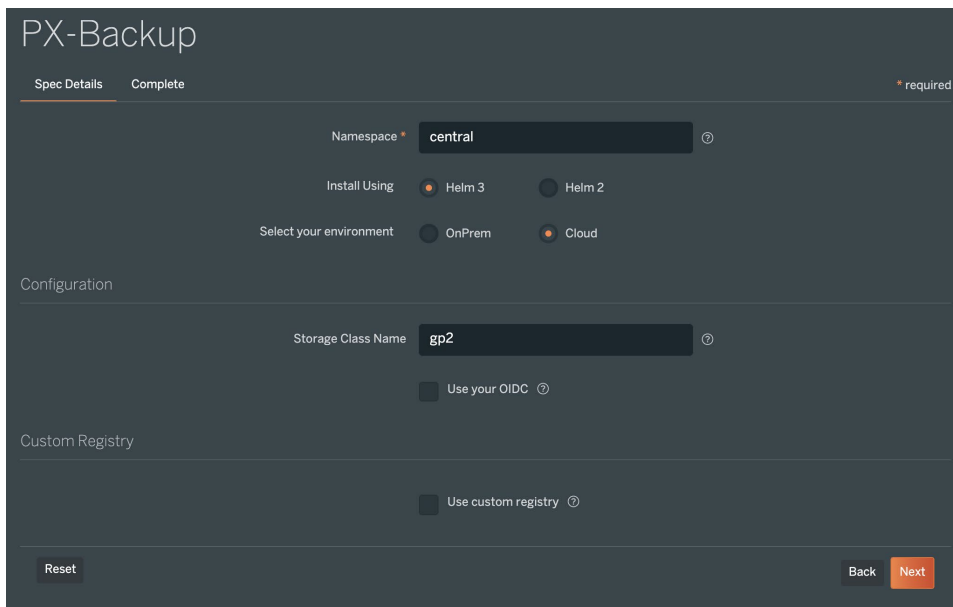
```
helm repo add portworx http://charts.portworx.io/ && helm repo update

helm install px-central portworx/px-central --namespace central --create-namespace --version 2.1.1 -
-set persistentStorage.enabled=true,persistentStorage.storageClassName="gp2",pxbackup.enabled=true
```



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

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



The PX-Backup configuration interface is shown with the following fields and options:

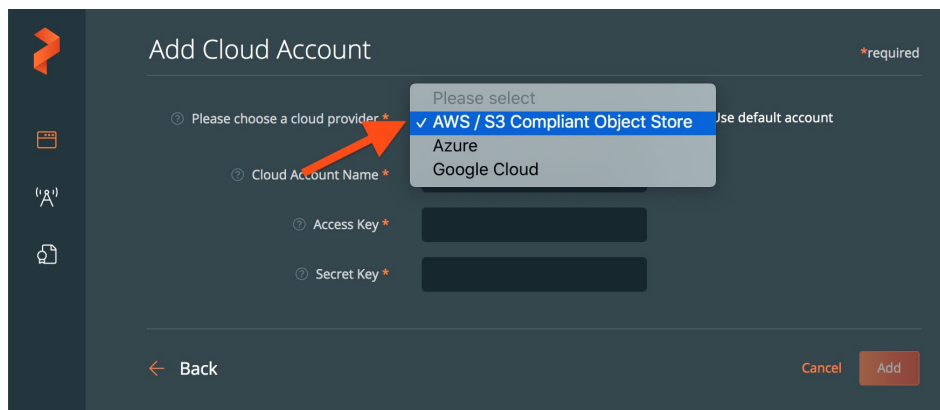
- Namespace \***: central
- Install Using**: Helm 3 (selected), Helm 2
- Select your environment**: OnPrem, Cloud (selected)
- Configuration**:
  - Storage Class Name**: gp2
  - ☐ Use your OIDC
- Custom Registry**:
  - ☐ Use custom registry
- Buttons**: Reset, Back, Next

Use the LoadBalancer IP for the px-backup-ui to access the PX-Backup interface.

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

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

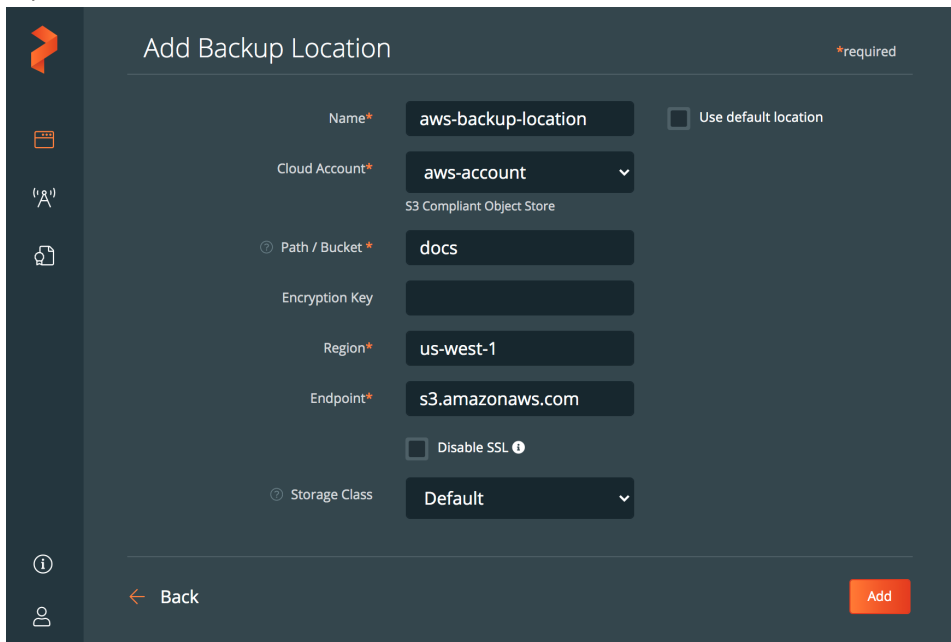
**Cloud accounts:** These credentials allow PX-Backup to authenticate with clusters for the purpose of taking backups and restoring to them as well as add and manage backup locations where backup objects are stored.



The Add Cloud Account interface is shown with the following fields and options:

- Please choose a cloud provider \***: AWS / S3 Compliant Object Store (selected), Azure, Google Cloud
- Cloud Account Name \***: [empty]
- Access Key \***: [empty]
- Secret Key \***: [empty]
- Buttons**: Back, Cancel, Add

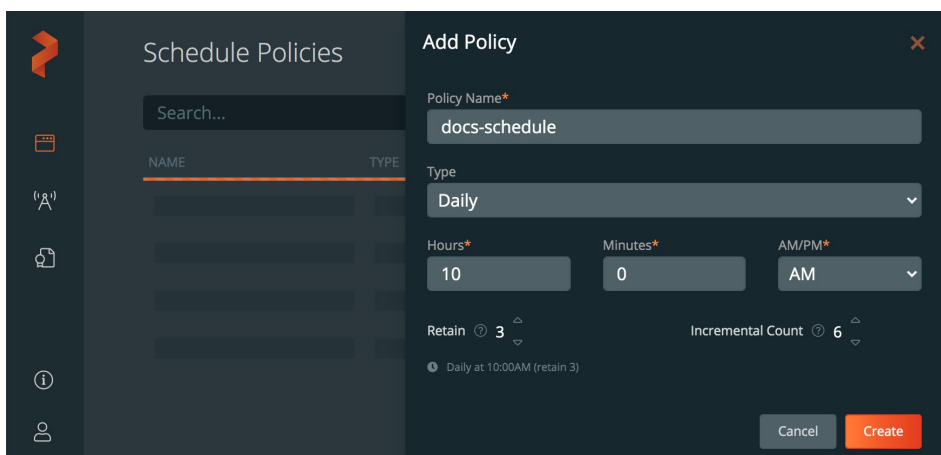
**Backup locations:** PX-Backup supports AWS S3 and S3-compliant object store as the backup repository to store your backup objects.



The screenshot shows the 'Add Backup Location' form. It includes the following fields and options:

- Name\***: aws-backup-location
- Cloud Account\***: aws-account (with a dropdown arrow)
- S3 Compliant Object Store**: (label below Cloud Account)
- Path / Bucket \***: docs
- Encryption Key**: (empty text field)
- Region\***: us-west-1
- Endpoint\***: s3.amazonaws.com
- Disable SSL**: (checkbox, currently unchecked)
- Storage Class**: Default (with a dropdown arrow)
- Use default location**: (checkbox, currently unchecked)
- Buttons**: Back (left arrow) and Add (orange button)

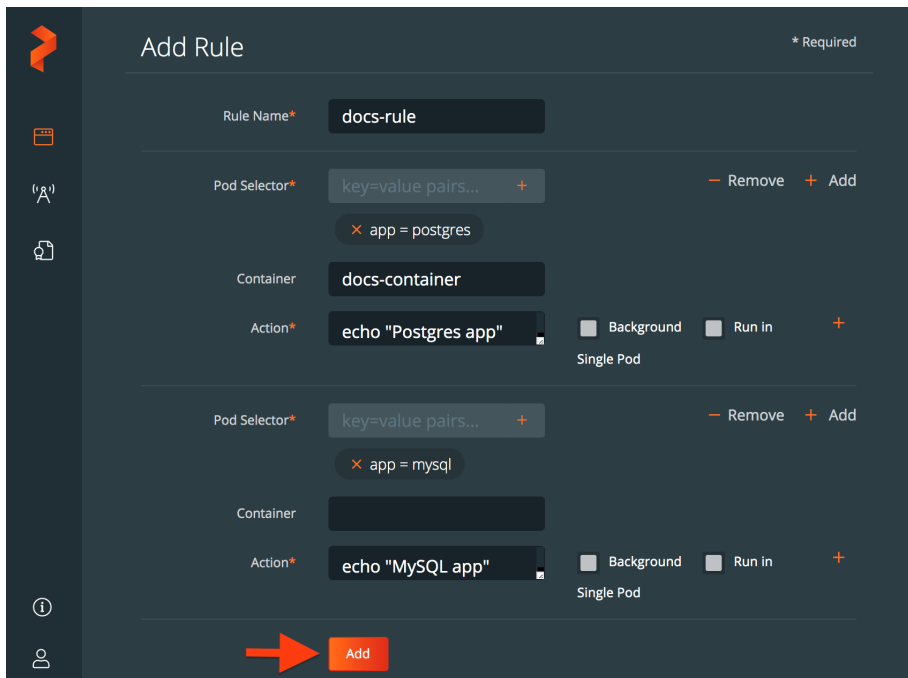
**Schedule policies:** PX-Backup allows administrators to create periodic, hourly, daily, weekly, and monthly schedule policies that will be leveraged by the application owners to create their backup jobs.



The screenshot shows the 'Schedule Policies' interface with the 'Add Policy' modal open. The modal contains the following fields and options:

- Policy Name\***: docs-schedule
- Type**: Daily (with a dropdown arrow)
- Hours\***: 10
- Minutes\***: 0
- AM/PM\***: AM (with a dropdown arrow)
- Retain**: 3 (with up/down arrows)
- Incremental Count**: 6 (with up/down arrows)
- Preview**: Daily at 10:00AM (retain 3)
- Buttons**: Cancel and Create (orange button)

**Backup Rules:** To ensure application consistency, PX-Backup allows administrators to create pre- and post-backup rules for their applications. Stateful and distributed applications like Cassandra, Elasticsearch, MongoDB, MySQL, PostgreSQL, etc. may need these backup rules to take application consistent snapshots. For example rules, please see this page or consult your applications' documentation.



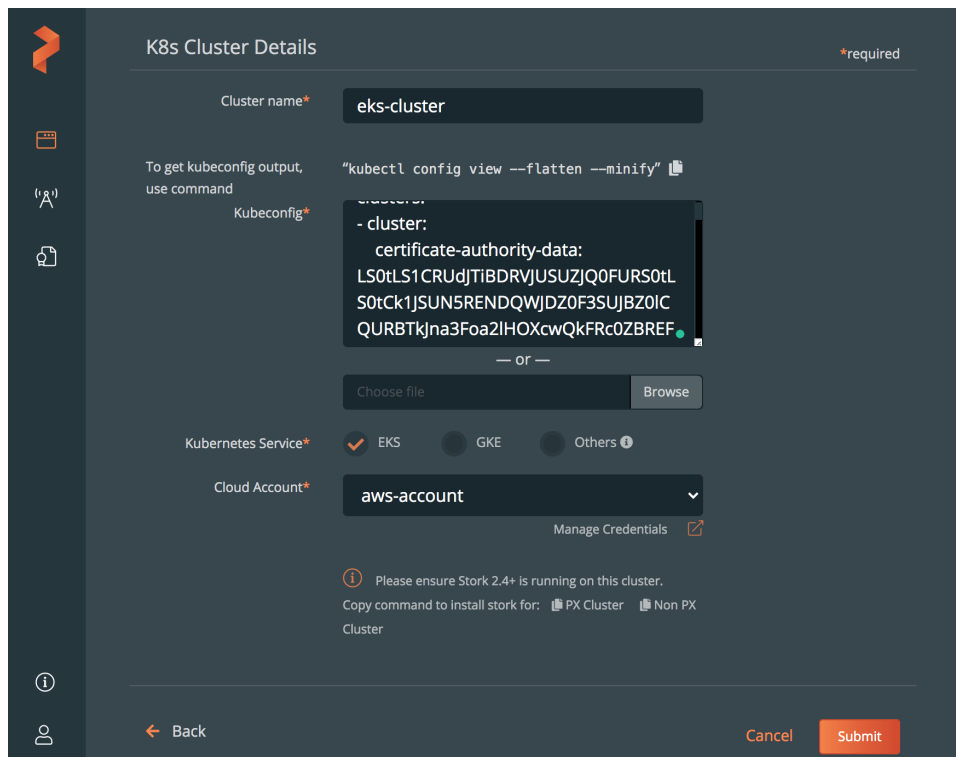
You can add your Amazon EKS clusters by using the PX-Backup UI:

1. Click on **Add Cluster** on the top right.
2. Enter the name of the Amazon EKS cluster.
3. Retrieve the Kubeconfig from your cluster and paste it in the Kubeconfig text frame or select the **Browse** button to upload it from a file.
4. Select **EKS** and click **Submit**.

If you are deploying PX-Backup on a cluster without Portworx, you will have to install our required Stork component, using the command that's available to be copied. Stork is deployed in pods on your Amazon EKS cluster using a deployment object, and it is a mandatory component on your Amazon EKS cluster to use PX-Backup.

Stork enables PX-Backup to perform the following tasks:

- Install the required CRDs on your target cluster
- Execute backup and restores on the target cluster
- Use data movers to push Kubernetes resources and data to the configured object storage location



**K8s Cluster Details** \*required

Cluster name\*

To get kubeconfig output, use command

Kubeconfig\* 

```
"kubectl config view --flatten --minify"
- cluster:
  certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tL
S0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lC
QURBTklna3Foa2lHOXcwQkFRc0ZBRE
```

— or —

Choose file

Kubernetes Service\* ☒ EKS ☐ GKE ☐ Others ⓘ

Cloud Account\*

ⓘ Please ensure Stork 2.4+ is running on this cluster.  
Copy command to install stork for: ☐ PX Cluster ☐ Non PX Cluster

Once your Amazon EKS Cluster is added, you can start backing up your applications using the PX-Backup UI.

## Kubernetes Backup and Restore

As part of this document, we validated the following use cases with Amazon EKS:

- Amazon EKS with GP2-backed block storage
- Amazon EKS with Amazon EFS-backed file storage
- Application migration from Google Kubernetes Engine (GKE) clusters to Amazon EKS using PX-Backup

In addition to the above listed use cases, PX-Backup can also help protect applications that are running on Amazon EKS and leveraging Portworx as the Kubernetes storage layer for both file and block storage.

### Amazon EKS with GP2-Backed Block Storage

In this scenario, our containerized application was running on an Amazon EKS cluster and using the default GP2 storage class for block storage. You can use PX-Backup to back up your applications running on Amazon EKS and restore them to the same cluster in the case of accidental deletions and data loss. But with PX-Backup you can also use your backup snapshots stored in an Amazon S3 bucket and restore them to a completely different Amazon EKS cluster running in a different AWS region. This is important for Tier-3 applications, which might not have a DR requirement and need to be recovered to a new Amazon EKS cluster in a different AWS region.

To set this up, perform the following:

1. PX-Backup is deployed on the same or dedicated Amazon EKS cluster using native GP2 storage, and we have already configured our [AWS ObjectStore Account access](#) and an [S3 backup repository](#) and added our source Amazon EKS cluster that is running our containerized application (as per the previous section about data protection).

If you wish to restore your backup snapshot to a different Amazon EKS cluster running in a different AWS region, you need to edit the “kdmp-config” config map running in the kube-system on your source Amazon EKS cluster.

```
kubectl edit cm kdmp-config -n kube-system

###Add the following parameter in the data section:
BACKUP_TYPE: "Generic"
```

2. Next, you can navigate to your source Amazon EKS cluster from the PX-Backup UI to start the process of creating a backup job.
3. Select the namespace your application is running in. You can also customize which Kubernetes objects you want to protect by using labels and recourse types.

You can create different backup jobs. For example, if you have 1 application per namespace, you can create a backup job per namespace. If you have your Kubernetes objects labeled using a specific key value pair, you can use that to create a backup job. Or, if you only want to backup specific Kubernetes object types, you can do that too. PX-Backup offers complete flexibility to users on how they want to protect their applications.

1. Once you have selected the objects that you want to back up, click the **Backup** button.


Here you can enter and select the following details:

- Name for the backup job
- Backup location
- On-demand backup jobs or a scheduled backup job
- The pre- and post-backup rules for application-consistent backups
- Any labels you want to specify for the backup job

2. Next, click **Create**. You can monitor the backup jobs from the Backups tab.

The Backup Timeline gives administrators a single pane of glass view to monitor all backup jobs for a cluster over the past 24 hours or over the past 30 days.

Once your backup job is successful, you can use that backup job to restore your application.

To restore your application, select the backup job that you want to restore from, click the  icon on the right, and click **Restore**. You can restore an application to the same Amazon EKS cluster, or you can restore your application to a different Amazon EKS cluster. You can also restore it to the same namespace and replace existing resources, or you can restore your application to a different namespace. PX-Backup also allows you to just restore specific Kubernetes objects from a backup snapshot. Also, if

you are restoring to a new Amazon EKS cluster, you can customize additional things like the destination namespace and the destination storage class.

Once your application is restored, you can monitor the restore and all previous restore operations from the Restore tab.

## Amazon EKS with Amazon EFS-backed File Storage

In this scenario, our containerized application was running on an Amazon EKS cluster and using Amazon EFS CSI driver to dynamically provision ReadWriteMany persistent volumes. You can use PX-Backup to back up your applications running on Amazon EKS with Amazon EFS backed file storage. Here is how to set this up:

PX-Backup is deployed on the same or dedicated Amazon EKS cluster using native GP2 storage, and we have already configured our [AWS Cloud Account](#) and an [S3 backup repository](#) and added our source Amazon EKS cluster that is running our containerized application.

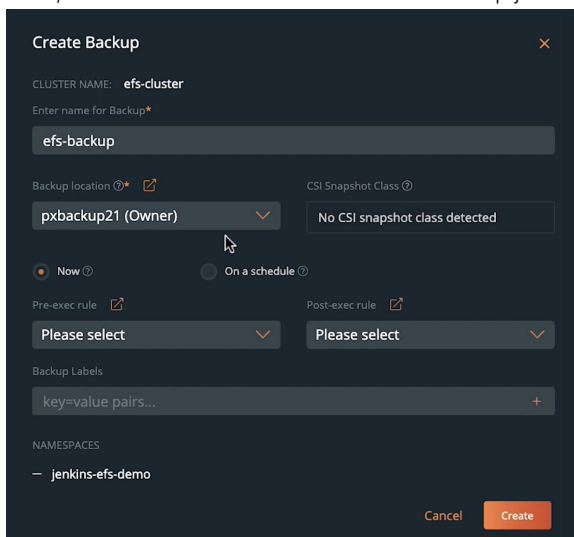
Next, you can navigate to your source Amazon EKS cluster from the PX-Backup UI to start the process of creating a backup job. You can also customize which Kubernetes objects you want to protect by using labels and resource types. Begin by selecting the namespace your application is running in.

You can create different backup jobs. For example, if you have one application per namespace, you can create a backup job per namespace. PX-Backup offers complete flexibility on how to protect your applications.

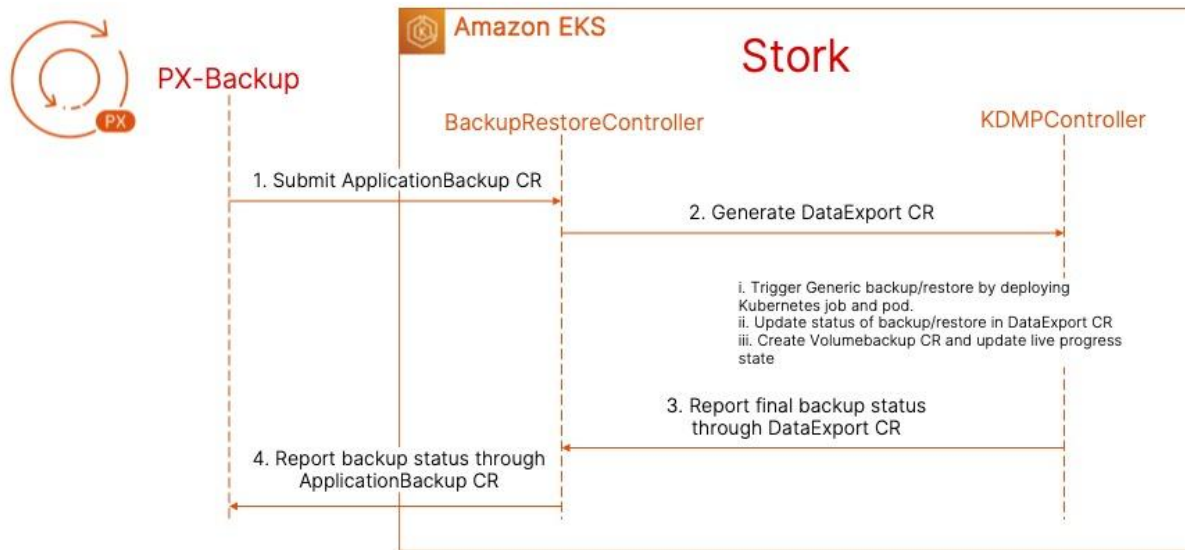
Once you have selected the objects that you want to back up, click the **Backup** button. Here you can enter and select the following details:

- Name for the backup job
- Backup location
- On-demand backup jobs or a scheduled backup job
- The pre- and post-backup rules for application-consistent backups
- Any labels you want to specify for the backup job

Next, click **Create**. You can monitor the backup jobs from the Backups tab.




PX-Backup uses the following workflow to backup Amazon EFS-backed persistent volumes running on Amazon EKS clusters.



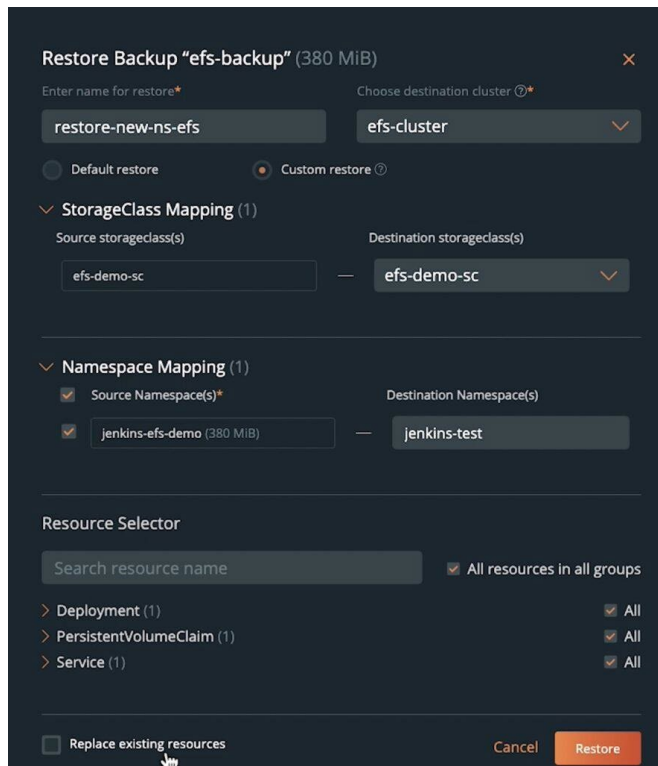
Upon initializing a backup job, PX-Backup will talk to Stork running on your target cluster and initiate the backup operation by creating an ApplicationBackup custom resource (CR).

Next, Stork will run a check to see if the persistent volume in the namespace is backed by Portworx PX-Store, Amazon EBS, Azure Managed Disk, or Google Persistent Disk. Since we are using an EFS-backed persistent volume, Stork will default to a generic data mover to copy your persistent volume and store it in the backup repository.

Next, the ApplicationBackup CR will create a DataExport CR, which creates a new Kubernetes job and a pod to perform the backup operation. The EFS-backed PVC will be mounted inside the backup pod, which copies the data to an S3-backed backup repository. Once the persistent volume is copied successfully, Stork will back up all the Kubernetes objects and application config and report a backup successful status to PX-Backup. The Backup Timeline gives administrators a single pane of glass view to monitor all backup jobs for a cluster over the past 24 hours or over the past 30 days.

Once your backup job is successful, you can use that backup job to restore your application. To restore your application, select the backup job that you want to restore from, click the  icon on the right, and click **Restore**. You can restore an application to the same Amazon EKS cluster, or to a different Amazon EKS cluster. You can also restore it to the same namespace and replace existing resources or to a different namespace. PX-Backup also allows you to just restore specific Kubernetes objects from a backup snapshot.

If you are restoring to a new Amazon EKS cluster, you can customize additional things like the destination namespace and the destination storage class.



Once your application is restored, you can monitor the restore and all previous restore operations from the restore tab.

## Application Migration from Google Kubernetes Engine (GKE) Clusters to Amazon EKS

In this scenario, we are still running our applications in a GKE cluster and will use PX-Backup to restore/migrate that application over to Amazon EKS. To set this up:

1. Deploy PX-Backup on a dedicated cluster either on GKE or Amazon EKS and configure the [AWS Cloud Account](#), [GCP Cloud Account](#), and an [S3 backup repository](#), and add our source GKE cluster and the destination EKS cluster using the respective cloud credentials.
2. Once your GKE and EKS clusters are added to your PX-Backup instance, the "kdmp-config" config map on your source GKE cluster needs to be updated. Use the following command to update the config map:

```
kubectl edit cm kdmp-config -n kube-system

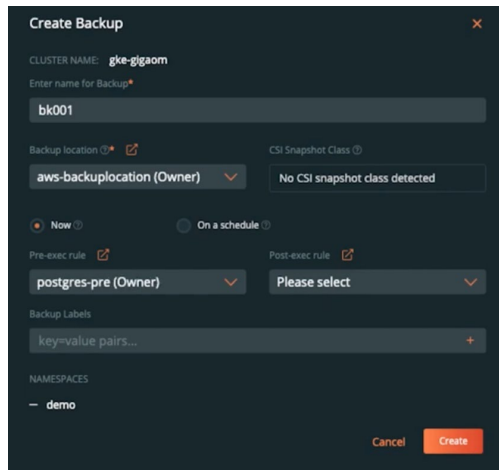
#Add the following parameter in the data section:
BACKUP_TYPE: "Generic"
```

3. Next, navigate to your source GKE cluster from the PX-Backup UI to start the process of creating a backup job. Select the namespace your application is running in. You can also customize which Kubernetes objects you want to protect by using labels and resource types.



Once you have selected the objects that you want to back up, you can click the **Backup** button. Here you can enter and select the following details:

- Name for the backup job
- Backup location
- On-demand backup jobs or a scheduled backup job
- The pre- and post-backup rules for application-consistent backups
- Any labels you want to specify for the backup job



**Create Backup**

CLUSTER NAME: gke-gigaom

Enter name for Backup\*

bk001

Backup location ⓘ **aws-backuplocation (Owner)** CSI Snapshot Class ⓘ No CSI snapshot class detected

☒ Now ⓘ ☐ On a schedule ⓘ

Pre-exec rule ⓘ **postgres-pre (Owner)** Post-exec rule ⓘ Please select

Backup Labels

key=value pairs... +

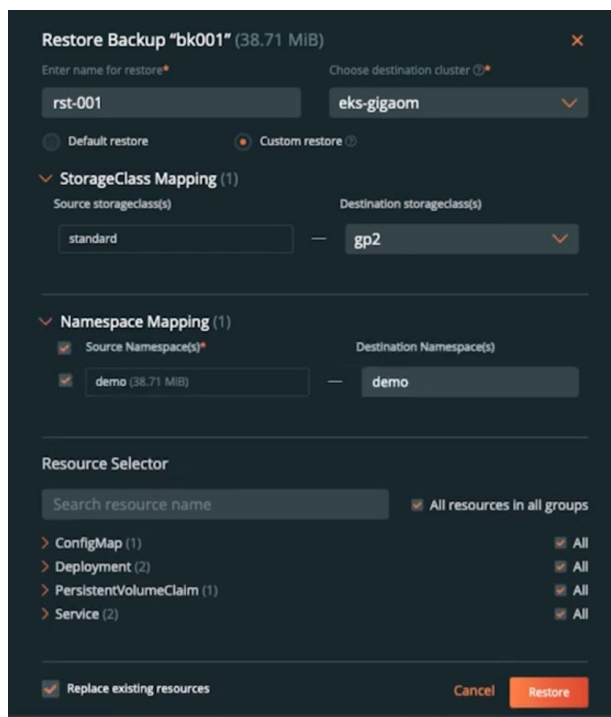
NAMESPACES

— demo

Cancel Create

4. Next, click **Create**. You can monitor the backup jobs from the Backups tab.

Next, let's try restoring your GKE backup to an EKS cluster. To do that, navigate to the GKE cluster and find the snapshot that you want to restore from. Select the backup snapshot and select **Restore**. Here, we can give the restore object a name and select a destination cluster. Since we are moving from GKE to EKS, we can also customize the way our application is restored.



**Restore Backup "bk001" (38.71 MiB)**

Enter name for restore\* **rst-001** Choose destination cluster ⓘ **eks-gigaom**

☐ Default restore ☒ Custom restore ⓘ

▼ **StorageClass Mapping (1)**

Source storageclass(s) **standard** — Destination storageclass(s) **gp2**

▼ **Namespace Mapping (1)**

☒ Source Namespace(s)\* **demo (38.71 MiB)** — Destination Namespace(s) **demo**

**Resource Selector**

Search resource name  ☒ All resources in all groups

> ConfigMap (1) ☒ All  
 > Deployment (2) ☒ All  
 > PersistentVolumeClaim (1) ☒ All  
 > Service (2) ☒ All

☒ Replace existing resources

Cancel Restore



We can select a destination storage class that we want to use for our persistent volumes. Since we are moving to EKS, we will select the gp2 storage class. We can also select a namespace that exists on the EKS cluster. Ptionally, you can choose to restore all resources or just a subset of resources from our backup job. Click **Restore**, and PX-Backup will instantiate a restore operation of your application on EKS.

Once you have your application restored, you can use kubectl and verify that all your application components and data have been successfully deployed and restored from GKE to EKS.

This scenario is useful when organizations want to protect their containerized applications running on one cloud platform and use another cloud platform as a restore site.

## Conclusion

Portworx provides the best-in-class enterprise-grade data services for any application running on Amazon EKS at any scale. Delivering speed, density, and scale, Portworx not only enables efficient, automatic provisioning on top of your Amazon EKS clusters; it also provides advanced features like high availability and replication, automated capacity management, and dynamic provisioning using application specific storage classes (IO\_profiles and IO\_priority, for example).

Portworx also provides a complete disaster recovery and business continuity solution with PX-DR. PX-DR allows organizations to build synchronous and asynchronous DR solutions for their Amazon EKS clusters. In addition to DR, PX-Backup also provides you with a Kubernetes-native backup and restore solution that can be leveraged to build architectures for local or remote backup and restore activities.

## Additional Resources

- [Portworx Blogs](#)
- [Portworx Demos](#)



## About the Author

Bhavin Shah is a Senior Technical Marketing Manager at Pure Storage. He is responsible for designing and architecting solutions around Portworx Enterprise, backup, and disaster recovery for Kubernetes. Bhavin has worked in the data management ecosystem for the past eight years, focusing on building solutions around converged infrastructure, hyper-converged infrastructure, cloud, and Kubernetes. Bhavin joined Pure Storage in March 2021 and works in the Cloud Native Business Unit.

The Pure Storage products and programs described in this documentation are distributed under a license agreement restricting the use, copying, distribution, and decompilation/reverse engineering of the products. No part of this documentation may be reproduced in any form by any means without prior written authorization from Pure Storage, Inc. and its licensors, if any. Pure Storage may make improvements and/or changes in the Pure Storage products and/or the programs described in this documentation at any time without notice.

THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. PURE STORAGE SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

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

[purestorage.com](https://purestorage.com)

800.379.PURE

