

PURE VALIDATED DESIGN

# Application Migration Solution for Amazon EKS with Portworx

Architecting a hybrid cloud application migration solution for Amazon Elastic Kubernetes Service with Portworx.





# Contents

<b>Executive Summary .....</b>	<b>3</b>
<b>Introduction .....</b>	<b>3</b>
Solution Overview.....	4
Use Cases .....	5
Solution Benefits.....	5
<b>Amazon Elastic Kubernetes Service Overview .....</b>	<b>5</b>
Amazon EKS Control Plane Architecture .....	6
Amazon EKS Deployment Options .....	6
<b>Portworx Overview .....</b>	<b>6</b>
PX-Store .....	7
PX-Backup.....	8
PX-DR.....	9
PX-Autopilot.....	9
Deployment Options .....	9
<b>Amazon EKS Deployment with Portworx.....</b>	<b>10</b>
<b>Amazon EKS Anywhere Deployment with Portworx .....</b>	<b>14</b>
<b>Application Migration.....</b>	<b>21</b>
<b>Conclusion .....</b>	<b>26</b>
<b>Additional Resources .....</b>	<b>26</b>
<b>About the Author .....</b>	<b>27</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 those applications in production. Amazon EKS alleviates the management overhead involved in building and operating Kubernetes clusters, both in the public cloud using Amazon EKS and on-premises using Amazon EKS Anywhere. However, Amazon EKS doesn't provide the storage and data management features needed to run stateful applications in production, nor does it allow organizations to migrate their applications from one Kubernetes cluster to another.

This Pure Validated Design outlines how Portworx can complement Amazon EKS and provide capabilities like cloud migration for containerized applications running on Amazon EKS or Amazon EKS Anywhere. It discusses the different use cases that demand seamless application migration between Kubernetes clusters. It also explains how organizations can leverage the consistent Kubernetes storage layer that Portworx® by Pure Storage® provides to build an application migration or portability solution across different Kubernetes distributions, Kubernetes versions, and infrastructure environments.

This document is intended for DevOps engineers and administrators, cloud architects, system architects who are interested in building a flexible application migration solution for their Amazon EKS clusters, and architects who are looking to adopt Amazon EKS and want to move away from their existing Kubernetes environments.

When a solution gets the Pure Validated Design (PVD) designation, it means that Pure has integrated and validated our 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 considerations and deployment best practices to accelerate deployment. Our methodology ensures that your 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 Pure Validated Design includes design considerations, deployment specifics, and configuration best practices for building a migration solution for Amazon EKS. We will walk through the process of deploying Portworx on Amazon EKS and on Amazon EKS Anywhere in the public cloud and inside on-prem data center environments, respectively, to help build a consistent hybrid cloud solution. This validated design also offers guidance and steps that organizations should follow to build a migration solution that helps them seamlessly migrate their applications from Amazon EKS Anywhere on-prem to Amazon EKS clusters in the public cloud. Finally, it documents the steps organizations need to follow to build an application portability solution, which allows users to adopt Amazon EKS for their containerized applications by moving away from their Red Hat OpenShift or OSS Kubernetes clusters.

To follow along with the deployment steps listed in this document, an architect will need to deploy an Amazon EKS cluster in a region of their choice and an Amazon EKS Anywhere cluster inside their data center. The on-prem data center environment should be connected to the public cloud network by using either an AWS Direct Connect connection or an IPSEC VPN tunnel to ensure that any traffic between the two sites is transferred over a secure connection. Red Hat OpenShift clusters or OSS Kubernetes clusters are optional and are only needed in scenarios where the architect wants to migrate away from them and onto Amazon EKS or Amazon EKS Anywhere clusters.

## Solution Overview

Application migration is a key requirement of modern IT departments, as it gives organizations the flexibility to move their applications across different environments. These migrations can help organizations avoid vendor lock-in and allow them to utilize the flexibility and elasticity of the public cloud, while also ensuring that their mission-critical applications are running on a secure and compliant infrastructure. Kubernetes was built to provide a consistent orchestration layer to run containerized applications, but that consistency only works for new application deployments. Kubernetes lacks the ability to move running stateful applications across different clusters. This is a huge gap, as organizations don't want to build migration solutions for their workloads using scripts or custom tools; organizations prefer to utilize their resources, both monetary and human capital, on adding value to their end customers. At Portworx, we understand this requirement and challenge, which is why we offer an application migration solution that allows you to migrate your running stateful applications across different Kubernetes clusters without the need to build custom tools or scripts. Using Portworx, you can build application migration or application portability solutions running across different Kubernetes clusters, different Kubernetes versions and distributions, and across multiple sites.





## Use Cases

Portworx offers solutions around a multitude of different use cases, including:

- **Blue-green application deployment:** DevOps teams who want to reduce the risk of downtime or errors associated with production deployments leverage blue-green application deployment, which allows them to maintain two identical production environments (called blue and green) that differ only with respect to new changes being deployed. Only one environment is live at a time, and traffic can be directed between these two environments as part of the deployment. Using Portworx, organizations can leverage the blue-green application deployment method for stateful applications running on Kubernetes clusters.
- **Cloud bursting:** Public cloud environments are known for their elasticity and ability to scale on demand. Organizations can use Portworx to leverage this additional capacity to scale their applications on demand. Portworx allows organizations to create a copy of their production environments and use it to serve additional traffic in scenarios of peak customer demand.
- **Root cause analysis:** Using Portworx, DevOps teams can have an exact replica of the production environment to perform root cause analysis as well as identify bugs and troubleshoot issues. This can help organizations identify bugs sooner, try different fixes in the duplicate environment, and easily push those changes to production once the issue is fixed.
- **Avoiding vendor lock-in:** Portworx allows organizations to avoid vendor lock-in by giving them the ability to migrate their applications off Kubernetes clusters as needed. This allows organizations to choose the most cost-effective solution to run their containerized applications on any Kubernetes cluster.

## Solution Benefits

This solution enables organizations to build a flexible solution for migrating modern applications running on Red Hat OpenShift or OSS Kubernetes to Amazon EKS and Amazon EKS Anywhere. Using Portworx, organizations can migrate applications across Kubernetes distributions inside their own data center environments or across the hybrid cloud from an on-prem Kubernetes cluster to an Amazon EKS cluster running in the public cloud. Deploying this application migration solution allows organizations to avoid Kubernetes vendor lock-in, leverage the elasticity of the public cloud for additional capacity, and easily replicate production environments to perform root cause analysis or to leverage the blue-green deployment model.

## Amazon Elastic Kubernetes Service Overview

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that runs Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. 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 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 those running on any standard Kubernetes environment, no matter whether they are running, whether that is 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. Here is what Amazon EKS does:

- Actively monitors the load on control plane instances and automatically scales them to ensure high performance.
- Automatically detects and replaces unhealthy control plane instances, restarting them across the availability zones within the Region as needed.
- Leverages 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

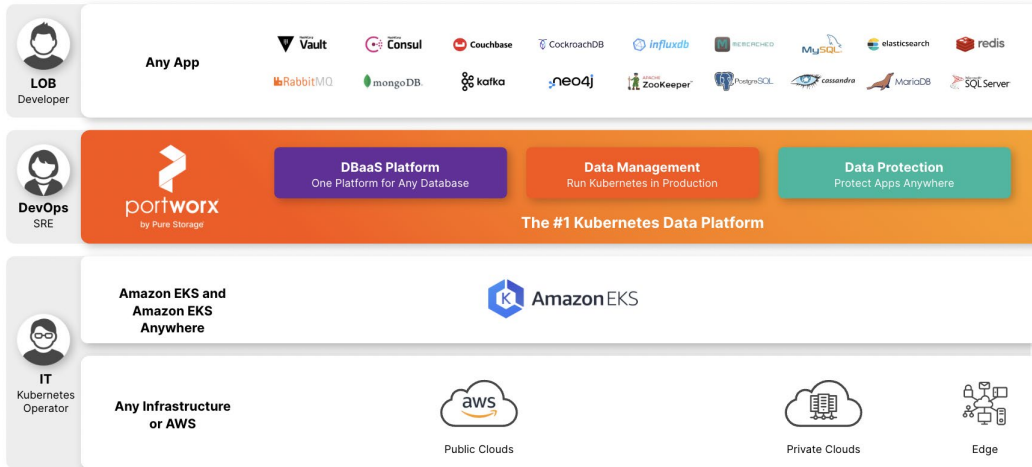
Organizations can use Amazon EKS with any, or all, of the following deployment options:

- **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 that serves applications and deployments in Kubernetes clusters. Portworx is deployed natively within Kubernetes and extends the automation capabilities into the infrastructure to eliminate 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 is 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, or Azure Blob, etc.—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. The key features of PX-Store include:

- **Storage virtualization:** The storage made available to each worker node is effectively virtualized such that each worker node can host pods that use up to hundreds of thousands of persistent volumes per Kubernetes cluster. This benefits Kubernetes clusters deployed to the cloud 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 pools of storage based on performance: low, medium, and high. Applications can select storage based on performance by specifying one of these pools at the 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 key features including:

- **App-consistent backup and restore:** Easily protect and recover applications regardless of how they are initially deployed on or rescheduled by Kubernetes.
- **Seamless migration:** Move a single Kubernetes application or an entire namespace between clusters (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, delivering key benefits, including:

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

## PX-Autopilot

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

- **Grow storage capacity on demand:** Automate your applications' growing storage demands while also minimizing disruptions. Set growth policies to automate cloud drive and Kubernetes integration to ensure your application's storage needs are met without performance or availability degradations.
- **Slash storage costs:** 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:

- **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). But we recommend using an external highly available KVDB database for these 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 intend to use 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.

## Amazon EKS Deployment with Portworx

For organizations running modern applications on Amazon EKS, Portworx provides a unified Kubernetes storage and data management layer with features like dynamic provisioning for block and file storage, high availability and replication, authentication and authorization, encryption at rest, io-profiles for different applications, snapshots, and topology awareness. To get started with Portworx on Amazon EKS, use the following steps to deploy an Amazon EKS cluster using eksctl:

1. Install eksctl on your jump host that you will be using to deploy Amazon EKS. In addition to eksctl, install and configure the AWS cli utility.
2. As part of the deployment, Portworx creates and attaches EBS volumes to your EKS worker nodes. Therefore, we need to grant Portworx the correct set of permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "<stmt-id>",
      "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": [
```

```

        "*"
    ]
}
]
}

```

3. Create an eksctl ClusterConfig, where can customize our EKS cluster and attach the necessary IAM policies. Below is a sample cluster config that deploys a 3-node Amazon EKS cluster.

**NOTE:** Please update the ARN for your IAM policy resource you create in your AWS account

```

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: px-eksctl
  region: us-east-1
  version: "1.21"
managedNodeGroups:
  - name: storage-nodes
    instanceType: m5.xlarge
    minSize: 3
    maxSize: 3
    volumeSize: 20
    #ami: auto
    amiFamily: AmazonLinux2
    labels: {role: worker, "portworx.io/node-type": "storage"}
    tags:
      nodegroup-role: worker
  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-of-your-portworx-aws-iam-policy> - Please replace this with your ARN
    withAddonPolicies:
      imageBuilder: true
      autoScaler: true
      ebs: true
      fsx: true

```



```

efs: true

albIngress: true

cloudWatch: true

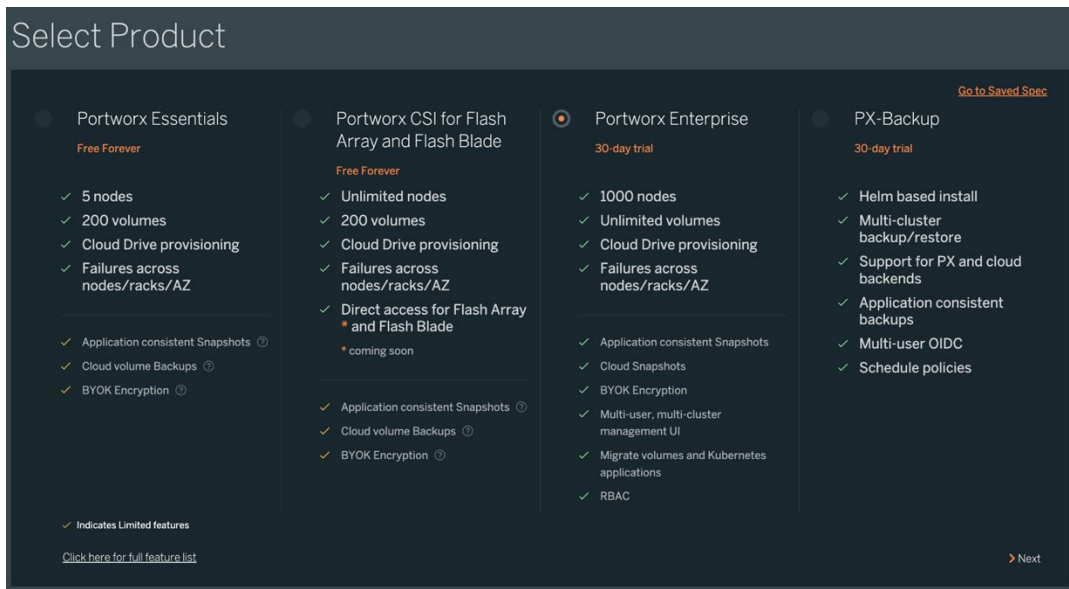
availabilityZones: [ 'us-east-1a', 'us-east-1b', 'us-east-1c' ]
    
```

4. Deploy the Amazon EKS cluster using eksctl create command:

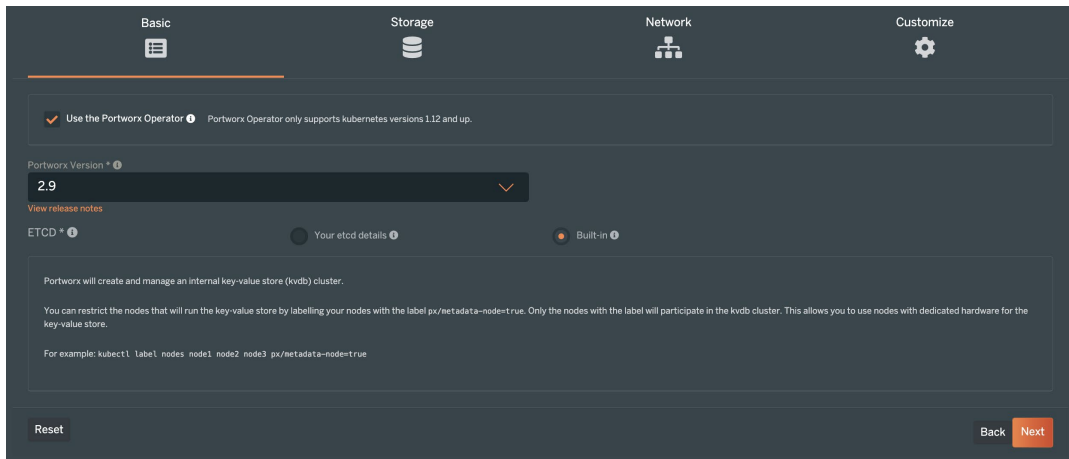
```

eksctl create cluster -f <my-clusterConfig>.yaml
    
```

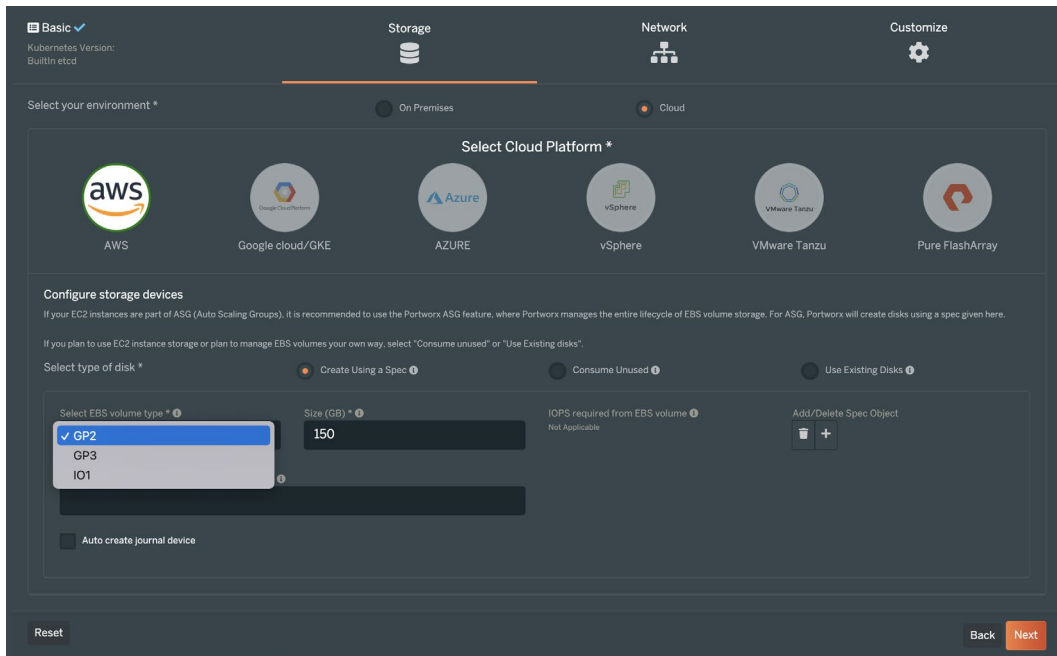
5. Once the cluster is deployed, you can navigate to [PX-Central](#) to generate a new specification for the Portworx cluster. Select **Portworx Enterprise** and click **Next**.



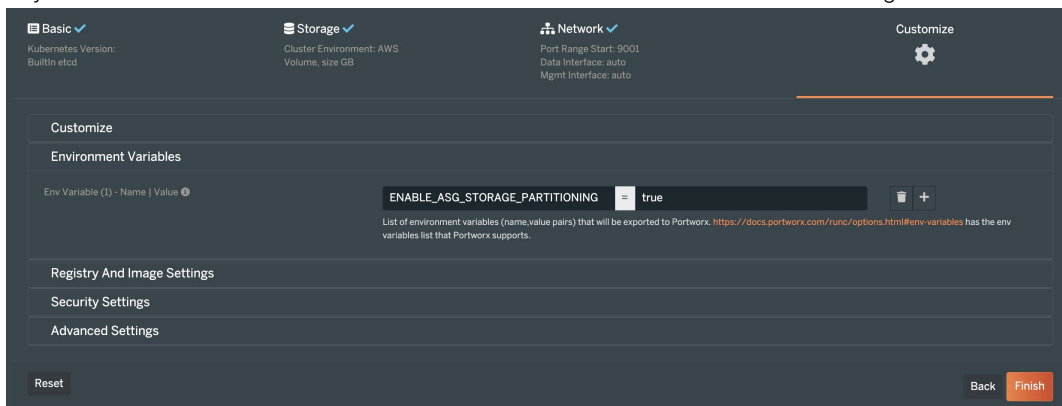
6. Check the box for Portworx Operator and then select the version of Portworx you want to deploy. You can also choose whether you want a built-in ETCD cluster or use an external non-Portworx managed ETCD cluster. Click **Next**.



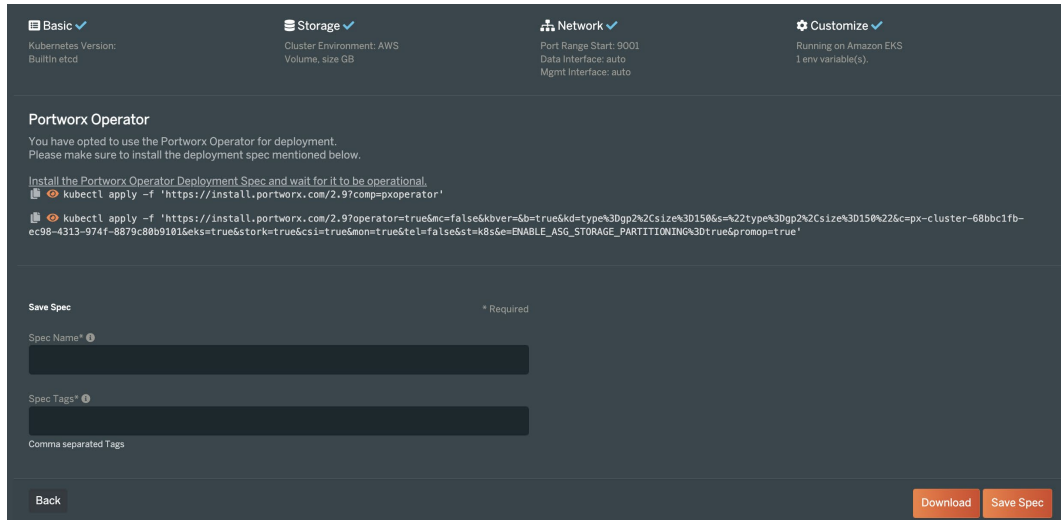
7. Select **Cloud** as your environment and select **AWS**. Here you can select the type of EBS volumes, size, and number of IOPS you need per volume. These configuration details will be used by Portworx to create EBS volumes, attach them to each EKS worker node, and aggregate them into a single storage pool. Click **Next**.



8. You can choose to leave the network settings to default and click **Next**.
9. In the Customize section, select **Amazon EKS** as the Kubernetes distribution, and then set an environment variable with key:value of `ENABLE_ASG_STORAGE_PARTITIONING=true`. Review the other settings and click **Finish**.



10. Read the End User License Agreement and click **Agree**. Next, you will be presented with a couple of kubectl commands. The first one deploys the Portworx Operator on your Amazon EKS cluster, and the second one creates the Portworx Storage Cluster Custom Resource and deploys all the Kubernetes objects needed in the kube-system namespace.



Basic ✓ Storage ✓ Network ✓ Customize ✓

Kubernetes Version: BuiltIn etcd

Cluster Environment: AWS  
Volume, size GB

Port Range Start: 9001  
Data Interface: auto  
Mgmt Interface: auto

Running on Amazon EKS  
1 env variable(s).

### Portworx Operator

You have opted to use the Portworx Operator for deployment. Please make sure to install the deployment spec mentioned below.

Install the Portworx Operator Deployment Spec and wait for it to be operational.

```
kubectl apply -f 'https://install.portworx.com/2.9?comp=pxoperator'
```

```
kubectl apply -f 'https://install.portworx.com/2.9?operator=true&mc=false&kbver=6b=true&kd=type%3Dgp2%2Csize%3D150&s=k22type%3Dgp2%2Csize%3D150%22&c=px--cluster-68bbc1fb-ec98-4313-974f-8879c88b9101&eks=true&stork=true&csi=true&mon=true&tel=false&st=k8s&e=ENABLE_ASG_STORAGE_PARTITIONING%3Dtrue&promop=true'
```

Save Spec \* Required

Spec Name\*

Spec Tags\*

Comma separated Tags

Back Download Save Spec

11. You can choose to save this specification or download the storage cluster specification in yaml format.
12. In the background, Portworx will create and attach the EBS volumes that match your specification to each of the EKS worker nodes and then aggregate them into a single storage pool and configure a few storage classes on your cluster that you can use for your stateful applications.
13. Once Portworx is up and running, you can use the following commands to validate the deployment:

```
kubectl get stc -n kube-system
kubectl get pods -n kube-system
kubectl get sc
```

## Amazon EKS Anywhere Deployment with Portworx

In this section, we will walk through the steps needed to deploy an Amazon EKS Anywhere cluster and then configure Portworx as the Kubernetes storage layer. Portworx provides a consistent set of features across Amazon EKS and Amazon EKS Anywhere, which is one of the key requirements needed to build application migration solutions across the hybrid cloud.

To deploy an Amazon EKS Anywhere cluster, you will need an [admin machine](#), a [VMware vSphere cluster](#), eksctl, and have already installed the [eksctl-anywhere plugin](#). Once the prerequisites are met, we will use the following steps to create a management cluster and a workload Amazon EKS Anywhere cluster:

1. Generate a configuration for the Amazon EKS Anywhere management cluster.

```
CLUSTER_NAME=mgmt
eksctl anywhere generate clusterconfig $CLUSTER_NAME \
  --provider vsphere > eksa-mgmt-cluster.yaml
```

2. Edit the `eksa-mgmt-cluster.yaml` file and customize it to match your VMware vSphere environment details.

```
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: Cluster
metadata:
  name: mgmt-eks-demo-1
spec:
  clusterNetwork:
    cni: cilium
    pods:
      cidrBlocks:
        - 192.168.0.0/16
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneConfiguration:
    count: 3
    endpoint:
      host: "10.21.143.55"
    machineGroupRef:
      kind: VSphereMachineConfig
      name: mgmt
  datacenterRef:
    kind: VSphereDatacenterConfig
    name: mgmt-dc
  externalEtcdConfiguration:
    count: 3
    machineGroupRef:
      kind: VSphereMachineConfig
      name: mgmt
  kubernetesVersion: "1.21"
  managementCluster:
    name: mgmt
  workerNodeGroupConfigurations:
    - count: 3
      machineGroupRef:
        kind: VSphereMachineConfig
        name: mgmt
      name: md-0
  ---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
```

```

kind: VSphereDatacenterConfig
metadata:
  name: mgmt-dc
spec:
  datacenter: "eks-px"
  insecure: false
  network: "VM Network"
  server: "<<vcenter IP>>"
  thumbprint: "<<vcenter thumbprint>>"
---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: VSphereMachineConfig
metadata:
  name: mgmt
spec:
  datastore: "EKS-Infra-1"
  diskGiB: 25
  folder: "eksvm"
  memoryMiB: 8192
  numCPUs: 2
  osFamily: ubuntu
  resourcePool: "/eks-px/host/eks-cluster/Resources/eks-rp"
  users:
  - name: capv

```

- Before we create a management cluster using the above yaml file, let's set the environment variables for your vSphere username and password. Ensure that these credentials have the [needed privileges](#).

```

export EKSA_VSPHERE_USERNAME='administrator@vsphere.local'
export EKSA_VSPHERE_PASSWORD='t0p$ecret'

```

- Deploy the Amazon EKS Anywhere management cluster using the following command:

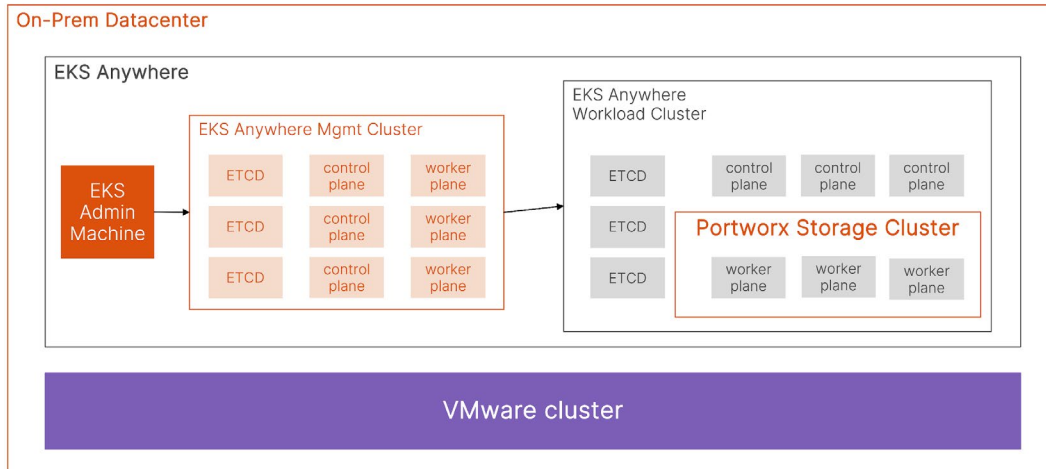
```

eksctl anywhere create cluster -f eksa-mgmt-cluster.yaml -v 6

```

- This creates a bootstrap cluster on the admin machine, followed by deployment of ETCD VMs, control plane VMs, and worker VMs on your VMware environment. Once all the VMs are up and running, the cluster management is moved from the bootstrap cluster to the new VM-based management cluster. Once the migration is successful, the bootstrap cluster is deleted, and you will see a "cluster creation successful" message.





- Now that you have your EKS Anywhere management cluster installed, you can deploy a workload cluster that will be used to install Portworx and run your stateful applications.
- We will follow a similar set of steps to deploy the workload cluster. We will generate a configuration file and customize the config file.

```

CLUSTER_NAME=w01
eksctl anywhere generate clusterconfig $CLUSTER_NAME \
  --provider vsphere > eksa-w01-cluster.yaml

## Example workload cluster configuration:
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: Cluster
metadata:
  name: w01
spec:
  clusterNetwork:
    cni: cilium
    pods:
      cidrBlocks:
        - 192.168.0.0/16
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneConfiguration:
    count: 1
    endpoint:
      host: "10.21.143.65"
  machineGroupRef:
    kind: VSphereMachineConfig

```

```
    name: w01-vm
  datacenterRef:
    kind: VSphereDatacenterConfig
    name: w01
  externalEtcdConfiguration:
    count: 1
    machineGroupRef:
      kind: VSphereMachineConfig
      name: w01-vm
  kubernetesVersion: "1.21"
  managementCluster:
    name: w01
  workerNodeGroupConfigurations:
  - count: 3
    machineGroupRef:
      kind: VSphereMachineConfig
      name: w01-vm
    name: md-0
---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: VSphereDatacenterConfig
metadata:
  name: w01
spec:
  datacenter: "eks-px"
  insecure: false
  network: "VM Network"
  server: "<<vCenter IP>>"
  thumbprint: "<<vCenter thumbprint>>"
---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: VSphereMachineConfig
metadata:
  name: w01-vm
spec:
  datastore: "EKS-Infra-1"
  diskGiB: 25
  folder: "eksworkload"
  memoryMiB: 8192
  numCPUs: 4
```

```

osFamily: ubuntu
resourcePool: "/eks-px/host/eks-cluster/Resources/eks-workload-rp"
users:
- name: capv

```

8. Next, we will deploy the workload cluster using the config file.

```

eksctl anywhere create cluster \
  -f eksa-w01-cluster.yaml \
  --kubeconfig mgmt/mgmt-eks-a-cluster.kubeconfig

```

9. Once the workload cluster is deployed, you can find the kubeconfig file to access the cluster in the generated subdirectory. The path should be `$(PWD)/$(CLUSTER_NAME)/$(CLUSTER_NAME)-eks-a-cluster.kubeconfig`. You can access your EKS workload cluster either from the admin machine or any other VM that has kubectl installed on it. You can use the following commands to access it from the admin machine.

```

export CLUSTER_NAME=w01
export KUBECONFIG=$(PWD)/$(CLUSTER_NAME)/$(CLUSTER_NAME)-eks-a-cluster.kubeconfig

```

10. Now that you have your workload cluster up and running, the next step is to install Portworx. Since EKS Anywhere is running on VMware vSphere, Portworx can automate the provisioning of disks and attach them to your EKS Anywhere worker nodes, and then it can create a Portworx storage cluster that can be leveraged by your stateful applications.

11. Before we generate a Portworx specification, let's go ahead and create a secret in the kube-system namespace for your vSphere username and password.

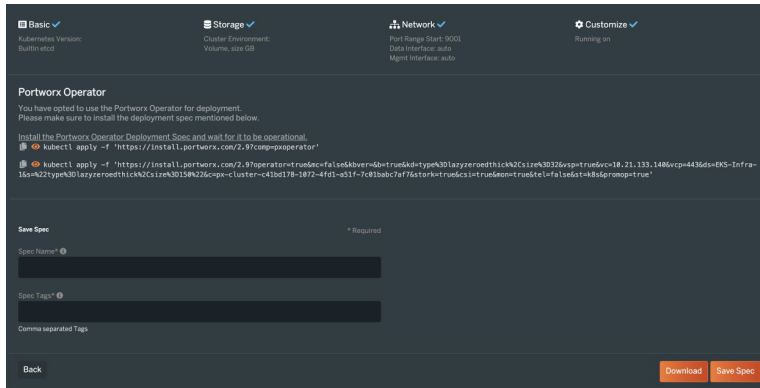
```

VSPHERE_USER: Use output of printf <vcenter-server-user> | base64
VSPHERE_PASSWORD: Use output of printf <vcenter-server-password> | base64

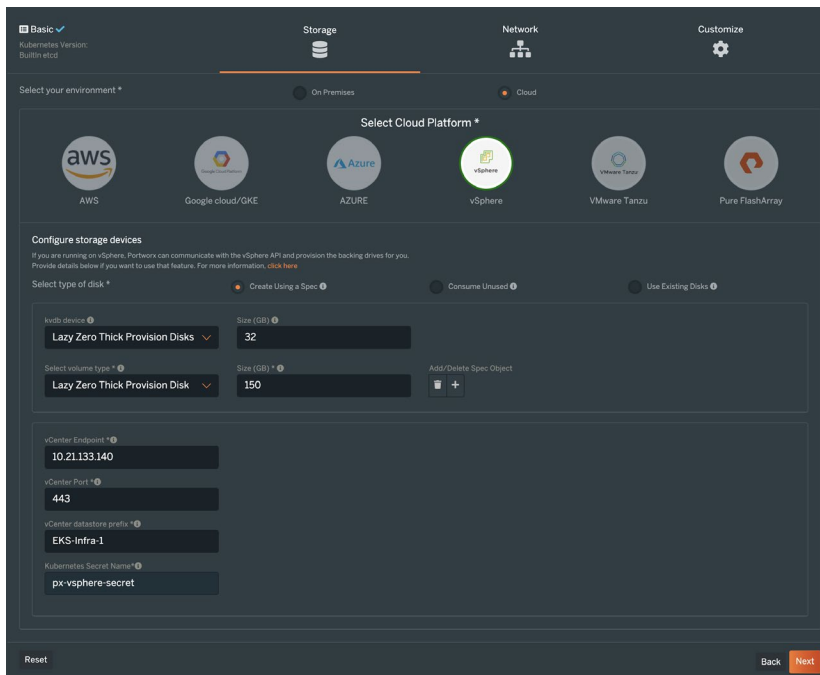
### px-vsphere-secret yaml file using the base64 versions of your vSphere username & password
apiVersion: v1
kind: Secret
metadata:
  name: px-vsphere-secret
  namespace: kube-system
type: Opaque
data:
  VSPHERE_USER: YWRtaW5pc3RyYXRgdkB2c3BoOPJ1LmxvY2Fs
  VSPHERE_PASSWORD: cLgxLjPuMVZUPw==

```

```
### Create the secret on your EKS Anywhere workload cluster
kubectl apply -f px-vsphere-secret.yaml
```



12. Next, let's navigate to [Portworx Central](#) and generate a specification. Select **Portworx Enterprise** and click **Next**. We will select the Portworx operator and select the latest version of Portworx. You can choose to use either a built-in or external etcd instance.
13. Next, let's select **Cloud** and **vSphere** and provide the vCenter server IP address and the vSphere datastore you want to use for Portworx virtual disks. Click **Next**.
14. You can leave the network settings as default or customize as needed. Click **Next**.
15. Here, select **None** and leave the other options as default. Click **Finish**, read the EULA, and click **Agree**.
16. You can use the two commands generated to deploy the Portworx operator followed by the Portworx storage cluster custom resource. You can generate your own using Portworx Central.



17. At this point, Portworx will automate the disk provisioning and mount operations and install a storage cluster on your Amazon EKS Anywhere cluster. You can monitor the deployment using the following commands:

```
kubectl get pods -n kube-system
kubectl get stc -n kube-system
kubectl get sc
```

## Application Migration

In this section, we will configure our source and destination clusters to implement an application migration solution. For this solution, we validated the following scenarios following these steps:

- Amazon EKS Anywhere cluster on-prem to Amazon EKS cluster in the public cloud
- On-prem Red Hat OpenShift cluster to on-prem Amazon EKS Anywhere cluster
- On-prem Red Hat OpenShift cluster to Amazon EKS cluster running in the public cloud
- On-prem vanilla Kubernetes cluster (open source) to on-prem Amazon EKS Anywhere cluster
- On-prem vanilla Kubernetes cluster (open source) to Amazon EKS cluster running in the public cloud

**NOTE:** Although we didn't cover the installation of Portworx on Red Hat OpenShift and OSS Kubernetes clusters, you can follow our documentation to install and configure Portworx as a prerequisite to the steps below. In addition to installing Portworx, ensure that your source and destination clusters can communicate with each other over ports 9001 and 9010.

1. Install storkctl on the jumphost that has access to your source and destination clusters.

```
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
```

2. Create Object Store credentials using Amazon S3 on both the source and destination clusters. Note that the UUID in both the scenarios will be for the destination cluster.

```
/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>

## To fetch the UUID for your destination cluster, use the 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}'
```

- Next, create a ClusterPair object. To create a ClusterPair object, use the following command to generate a specification on the destination cluster.

```
storkctl generate clusterpair <<clusterpairname>> -n <<migrationnamespace>> > eks-clusterpair.yaml
```

**NOTE:** You can configure individual ClusterPair objects for each namespace that you want to migrate, or you can configure an admin namespace so any admin that has access to that namespace can migrate any other namespace on the source cluster.

- Edit the portworx-service to LoadBalancer by adding an annotation to the Portworx storage cluster specification on the destination cluster.

**NOTE:** Do not enable [load balancing](#) without authorization enabled on the Portworx cluster.

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

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

- Edit the above ClusterPair yaml file and update the storage options in the spec.options section:

```
options:
  ip: "<ip-address-of-node-in-the-destination-cluster>"
  port: "<port_of_remote_px_node_default_9001>"
  token: "<token_generated_from_destination_cluster>"
```

- To fetch the cluster token from the destination cluster, use 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 cluster token show
```

7. Next, use this updated ClusterPair yaml file and apply it against the source cluster:

```
kubectl apply -f eks-clusterpair.yaml -n <<migrationnamespace>>
```

**NOTE:** When the ClusterPair gets created, Portworx also creates a 100 GiB volume called ObjectstoreVolume. If you plan to migrate volumes that are significantly larger than 100GiB, make sure you first check out the [Migrating Large Volumes](#) section.

8. Monitor the ClusterPair creation using the following commands:

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

NAME	STORAGE-STATUS	SCHEDULER-STATUS	CREATED
eks-clusterpair	Ready	Ready	12 Mar 22 03:11 UTC

9. Once both the storage status and schedule status are ready, we can go ahead and create a migration spec yaml file:

```
cat > app-migration.yaml <<_EOF
apiVersion: stork.libopenstorage.org/v1alpha1
kind: Migration
metadata:
  name: <YOUR_MIGRATION_OBJECT>
  namespace: <YOUR_MIGRATION_NAMESPACE>
spec:
  clusterPair: <YOUR_CLUSTER_PAIR>
  includeResources: true # This migrates all Kubernetes objects.
  startApplications: true # Deploys app pods on the destination cluster
  namespaces:
  - <NAMESPACE_TO_MIGRATE>
  purgeDeletedResources: false # boolean value specifying if STORK should automatically purge a
  resource from the destination cluster when you delete it from the source cluster.
_EOF

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

10. Portworx also allows you to specify pre- and post-exec rules as part of your migration that enable you to run scripts before and after a migration operation is triggered. Use the following sample pre- and post-exec rules to build your own rules:

```
#Pre-Exec Rule for MySQL
apiVersion: stork.libopenstorage.org/v1alpha1
kind: Rule
```

```
metadata:
  name: mysql-pre-backup-rule
  namespace: mysql-1-pvc-mysql
  annotations:
    "stork/cmdexecutor-image": "openstorage/cmdexecutor:latest"
rules:
- podSelector:
  app: mysql
  actions:
- type: command
  # dummy action to test non-background command
  value: echo "hello stork pre-backup"
  runInSinglePod: true
- type: command
  background: true
  runInSinglePod: true
  # this command is just to test multiple background commands
  value: mysql --user=root --password=$MYSQL_ROOT_PASSWORD -Bse 'show databases;system
${WAIT_CMD};'
- type: command
  background: true
  # this command will flush tables with read lock
  value: mysql --user=root --password=$MYSQL_ROOT_PASSWORD -Bse 'flush tables with read
lock;system ${WAIT_CMD};'

#Post-Exec Rule for MySQL
apiVersion: stork.libopenstorage.org/v1alpha1
kind: Rule
metadata:
  name: mysql-post-backup-rule
  namespace: mysql-1-pvc-mysql
  annotations:
    "stork/cmdexecutor-image": "openstorage/cmdexecutor:latest"
rules:
- podSelector:
  app: mysql
  actions:
- type: command
  # dummy action to test non-background command
  value: echo "hello stork pre-backup"
```



```

    runInSinglePod: true
  - type: command
    runInSinglePod: true
    # this command is just to test multiple background commands
    value: mysql --user=root --password=$MYSQL_ROOT_PASSWORD -Bse 'show databases;'
```

11. To use pre- and post-exec rules, you can update your migration specification as follows:

```

cat > app-migration.yaml <<_EOF
apiVersion: stork.libopenstorage.org/v1alpha1
kind: Migration
metadata:
  name: <YOUR_MIGRATION_OBJECT>
  namespace: <YOUR_MIGRATION_NAMESPACE>
spec:
  clusterPair: <YOUR_CLUSTER_PAIR>
  includeResources: true # This migrates all Kubernetes objects.
  startApplications: true # Deploys app pods on the destination cluster
  preExecRule: mysql-pre-backup-rule
  postExecRule: mysql-post-backup-rule
  namespaces:
  - <NAMESPACE_TO_MIGRATE>
  purgeDeletedResources: false # boolean value specifying if STORK should automatically purge a
resource from the destination cluster when you delete it from the source cluster.
_EOF

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

12. You can monitor the migration progress by using the following commands:

```

storkctl get migrations -n demo
kubectl get migrations -n demo
kubectl describe migrations -n demo
```

13. Once the migration is successful, you should see the following status:

```

storkctl get migrations -n demo
```

NAME	CLUSTERPAIR	STAGE	STATUS	VOLUMES	RESOURCES	CREATED
appmigration	eks-cluster	Final	Successful	1/1	3/3	12 Mar 22 20:10 UTC



14. You can also verify the migration is successful by using the following commands on the destination cluster:

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

As part of each Migration, Portworx starts by replicating the persistent volumes between the two clusters and then copies over the Kubernetes objects.

## Conclusion

Portworx provides 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, IO\_priority, etc.).

Portworx provides a complete application migration solution that allows you to move your containerized applications across different Kubernetes clusters running different Kubernetes versions, different Kubernetes distributions, and different underlying infrastructure stacks. Portworx by Pure Storage is the gold standard when it comes to Kubernetes data services, and it brings all its capabilities to Amazon EKS clusters.

## Additional Resources

- [Portworx blog posts](#)
- [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, hyperconverged 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

