PURESTORAGE®
Uncomplicate Data Storage, Forever

portworx
by Pure Storage

PURE VALIDATED DESIGN

Kubernetes Storage and Data Management for Amazon EKS with Portworx

Architecting a robust, reliable, and secure Kubernetes storage layer for Amazon Elastic Kubernetes Service with Portworx.

Pure Validated Design

Contents

Executive Summary

Organizations are increasingly adopting containers and Kubernetes to build their modern applications, along with 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. This document outlines how Portworx® by Pure Storage® can help augment Amazon EKS and provide capabilities like high availability and replication, best-in-class performance, and automated capacity management for containerized applications running on Amazon EKS. It discusses architectural patterns that help organizations customize their Amazon EKS clusters and optimize them for their applications. It also focuses on the need to build an automated capacity management solution to run containerized applications at scale on Amazon EKS.

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

A Pure Validated Design (PVD) designation 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, provides design consideration, and shares 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 consideration and deployment best practices for Amazon EKS and Portworx to provide a modern infrastructure platform to run Kubernetes.

Introduction

This validated design describes the benefits of using Portworx with Amazon EKS as the Kubernetes storage layer for running stateful applications, as well as design considerations, deployment specifics, and configuration best practices for building an enterprise grade Kubernetes storage solution for Amazon EKS.

This document lays out the different architectural patterns that organizations can leverage to build a Kubernetes storage and data management solution for Amazon EKS. It also describes a disaggregated or a hyperconverged deployment model that allows organizations to build their Amazon EKS clusters based on the resource requirements for their containerized applications. It also offers a solution that allows organizations to build an infrastructure stack that provides the best-in-class performance for their stateful applications, while also condensing their storage footprint and saving costs. This document also discusses why stateful containerized applications need automated capacity management and how a solution like Portworx can help set policies in place that eliminate the manual operational overhead involved in monitoring and managing storage capacity at scale in production.

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. We will walk through the steps involved in deploying Portworx on Amazon EKS as the Kubernetes storage and data management layer for containerized applications.

Solution Overview

Kubernetes has become the de facto standard for orchestrating and running containerized applications, from the dev-test stage all the way up to production. Organizations are also now getting comfortable with running stateful applications on their Kubernetes clusters in production. This includes modern distributed databases and data services like Cassandra, CockroachDB, Redis, and Kafka being deployed and used by developers for their modern applications. To accommodate these different stateful and distributed applications, organizations need a Kubernetes storage solution that helps them extract the best performance from the underlying infrastructure, while also ensuring that these applications are deployed with high availability and replication in mind.

Portworx provides a Kubernetes storage and data management layer that is built using containers and runs on Amazon EKS clusters to provide block and file persistent volumes to stateful applications. Installing Portworx on any Amazon EKS cluster requires users to generate a specification using Portworx Central and deploying the Portworx Operator and the StorageCluster Custom Resource (CR) using the kubectl commands generated at the end of the spec generator.

Portworx allows organizations to create custom Kubernetes StorageClass objects for their stateful applications, enabling them to set their own replication factors, filesystem types, snapshot schedules, IOPS or throughput limits, etc. Defining replication factors at the Portworx layer allows organizations to deploy applications that are tolerant to availability zone (AZ) failures, as volume replicas are spread across different AZs.

Since Portworx provides both block and file storage from the same underlying storage pool, users don't have to configure individual CSI plugins and backend filesystems for block (Amazon EBS) and file (Amazon EFS) storage. Portworx offers best-in-class performance for stateful applications that need shared file storage, and it allows organizations to deploy hundreds of persistent volumes on a single Amazon EKS worker node, rather than having to add more worker nodes because of EBS mount limits per EC2 instance.

Portworx also helps simplify and streamline Day 2 operations—like monitoring individual persistent volumes and underlying storage pools for consumed capacity—and have rules in place to perform volume or storage pool expansion operations automatically when certain thresholds set by the administrator are met. This eliminates the need for an operations team to continuously monitor 1000s of persistent volumes across multiple Amazon EKS clusters and manually expand each persistent volume to avoid any application downtime.

This document dives into each of the following use cases to help organizations build a reliable, scalable, and cost-effective infrastructure stack to run their containerized applications:

- Architecting a highly available Amazon EKS cluster using Portworx
- Optimizing infrastructure resources for Amazon EKS
- Automating capacity management for Amazon EKS

Solution benefits

This solution enables organizations to build robust and reliable Amazon EKS clusters with Portworx. Using Portworx, organizations can get the following benefits for their Amazon EKS clusters.

Availability: Portworx allows users to define replication factors as part of their Kubernetes StorageClass definition. Any persistent volumes provisioned using this StorageClass automatically create and store the number of replicas specified in the StorageClass across your Amazon EKS worker nodes. Portworx spreads out these replicas across different availability zones (AZs) as well. This makes containerized applications highly available and protects against any data loss in case a worker node or an AZ goes offline.

Scalability: Portworx allows users to customize their Amazon EKS cluster, such that only a subset of the worker nodes is contributing storage to containerized applications. This allows users to scale up their compute capacity as needed without having to pay for additional storage. Portworx also allows users to get more bang for their buck by provisioning hundreds of persistent volumes per Amazon EKS worker node vs adding more Amazon EKS worker nodes to provision more storage.

Operational management: Portworx automates the Day 0 deployment and simplifies the Day 2 operations for Amazon EKS clusters. Portworx automates the deployment and attachment of Amazon EBS volumes to Amazon EKS worker nodes and aggregates individual volumes into a unified storage pool to provision block and file storage. Users are no longer required to deploy and configure individual CSI drivers for EBS (block) and EFS (file) storage for their applications.

In addition to automating Day 0 deployment, Portworx also allows users to perform non-disruptive upgrades of Kubernetes versions, Amazon Machine Image (AMI) versions, and Portworx itself.

Portability: One of the key benefits of Kubernetes is the uniform orchestration platform it provides to deploy containerized applications anywhere. This also applies to Amazon EKS, as users can deploy Amazon EKS clusters on any AWS region—in fact, users can even deploy it on-prem using Amazon EKS Anywhere or Amazon EKS for AWS Outposts. However, Amazon EKS can't migrate already running applications across these different clusters. Portworx allows users to migrate their applications across any Amazon EKS, Amazon EKS Anywhere, or Amazon EKS on AWS Outposts clusters, unlocking true application portability across different Kubernetes clusters, different Kubernetes versions, and different infrastructure stacks.

Disaster recovery: Disaster recovery is one of the key requirements for any application running in production. Portworx allows users to create asynchronous and synchronous disaster recovery solutions for their Amazon EKS clusters that insure them against any data loss as well as any node, cluster, availability zone, or region failures. Portworx allows users to customize their recovery point and recovery time objectives for their applications to meet the most demanding service level agreements.

Cost optimization: Portworx allows users to start small and scale on demand for their storage needs. Using Portworx Autopilot, users can configure "IFTTT" rules, where Portworx Autopilot will monitor the storage utilization for individual Kubernetes persistent volumes and the underlying storage pool and automatically expand the volumes and the storage pool to accommodate the increasing storage needs without any application downtime.

Data protection: Portworx PX-Backup, a Kubernetes-native tool that understands how modern applications are built and deployed on Amazon EKS, allows users to create backup and restore jobs for containerized applications running on Amazon EKS clusters. Portworx PX-Backup also allows users to protect their end-to-end applications, including not just the data, but any application configuration and Kubernetes objects as well. These backup snapshots are stored in Amazon S3 buckets and can be used to restore applications to the same or different Amazon EKS clusters.

Amazon Elastic Kubernetes Service

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. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. Amazon EKS is able to:

- Run and scale the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability.
- Automatically scale control plane instances based on load, detects and replaces 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 that are 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 can:

- Actively monitor the load on control plane instances and automatically scales them to ensure high performance
- Automatically detect and replace unhealthy control plane instances, restarting them across the availability zones within the region as needed
- Leverage 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

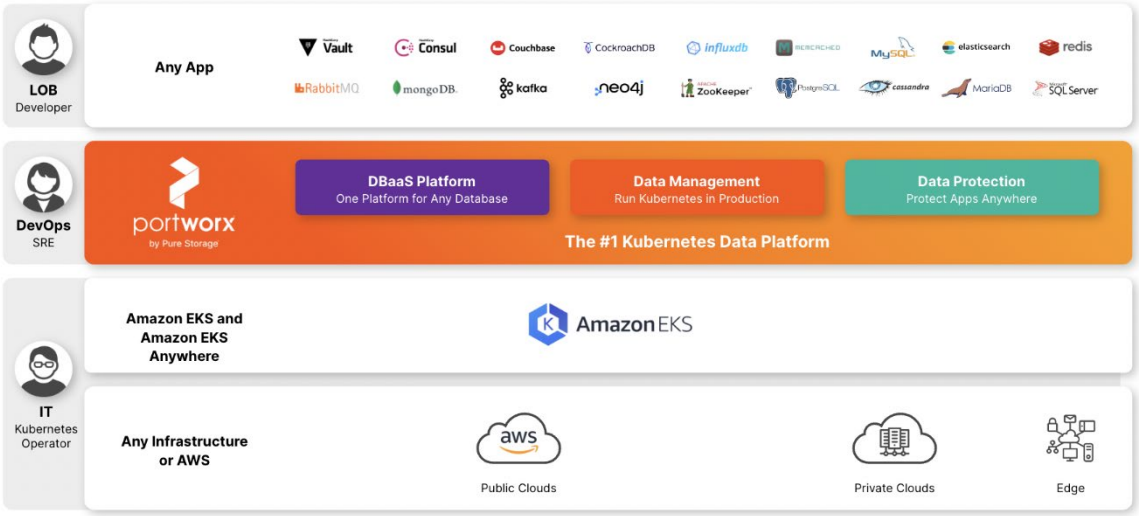
You can use Amazon EKS with any, or all, of the following deployment options:

- **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

Portworx is a data management solution that serves applications and deployments in Kubernetes clusters. Portworx is deployed natively within Kubernetes and extends the automation capabilities down into the infrastructure to eliminate all the complexities of managing data. Portworx provides simple and easy-to-consume 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**: A service that allows applications to have a zero RPO failover across data centers in a metro area as well as continuous backups across the WAN for even greater protection
- **PX-Backup**: A solution that allows enterprises to back up and restore the entire Kubernetes application—including data, app configuration, and Kubernetes objects—to any backup location, including AWS S3, or Azure Blob, with the click of a button
- **PX-Autopilot**: A service that provides rules-based auto-scaling for persistent volumes and storage pools

PX-Store

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

- **Storage virtualization**: The storage made available to each worker node is effectively virtualized such that each worker node can host pods that use up to hundreds of thousands of persistent volumes per Kubernetes cluster. This benefits Kubernetes clusters deployed to the cloud, in that larger volumes or disks are often conducive to better performance.
- **Storage-aware scheduling**: Stork, a storage-aware scheduler, 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

- **Compliance management**: Manage and enforce compliance and governance responsibilities with a single pane of glass for all your containerized applications
- **Streamlined storage integration**: Back up and recover cloud volumes with storage providers, including Amazon EBS, Google Persistent Disk, Azure Managed Disks, and CSI-enabled storage

PX-DR

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

- **Expand 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 applications' 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 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:

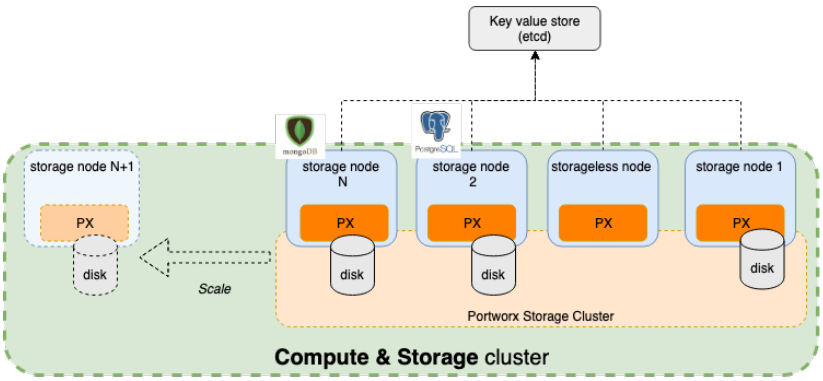
- **Use an existing key value database(KVDB):** For most deployments, you can create a deployment specification with the option of storing Portworx metadata in a separate etcd cluster. There are two exceptions to this:
  - The first scenario is when the PX-DR is used for Kubernetes clusters that are not within the same metro area, meaning the network round-trip latency between the primary and disaster recovery sites is greater than 10ms.
  - The second scenario in which a dedicated etcd cluster should be used is for large-scale deployment with 10 or more worker nodes, in which a heavy dynamic provisioning activity takes place.
- **Dedicated journal device:** A dedicated journal device can be specified to buffer metadata writes.

- **Dedicated cache device:** A dedicated cache device can be specified to improve performance by acting as a read/write-through cache.
- **Container storage interface (CSI) API compatibility:** You can choose the option to deploy Portworx with CSI enabled if PX-Security is to be used.
- **Stork:** Stork is a storage-aware scheduler that attempts to co-locate application pods onto the same nodes as the persistent volumes and persistent volume replicas use. Use Stork if your underlying infrastructure uses either servers with dedicated internal storage or servers with dedicated network-attached storage appliances.
- **Dedicated network:** Consider using a dedicated network for storage cluster traffic if the existing network infrastructure does not support quality of service.

Architecting a Highly Available Amazon EKS Cluster Using Portworx

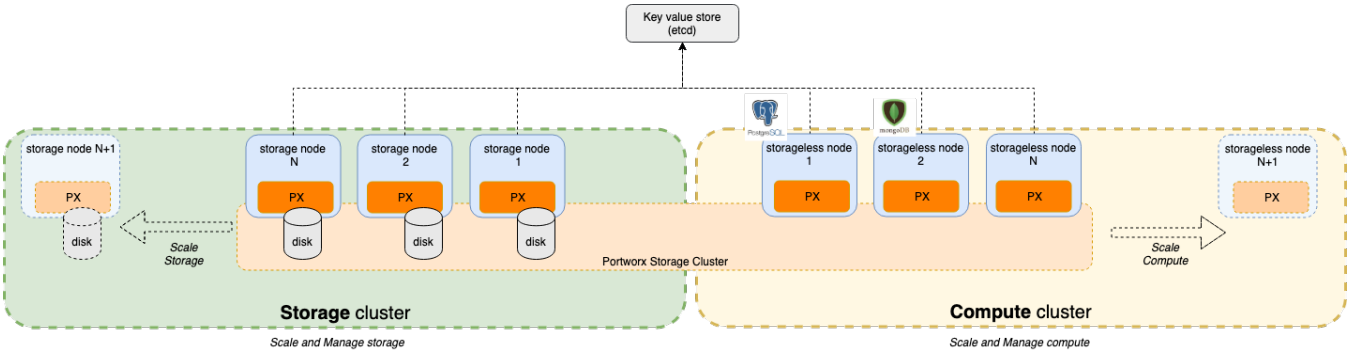
Portworx provides a Kubernetes storage and data management layer that is built using containers and runs on your Amazon EKS clusters, either as a Kubernetes Custom Resource (CR) or as a Kubernetes DaemonSet object. Portworx can be deployed on Amazon EKS in either one of the following modes:

Hyperconverged mode: Each Amazon EKS worker node also acts as a Portworx storage node and has locally attached Amazon EBS volumes that contribute storage to the overall Portworx storage pool capacity. This mode is best-suited for general purpose applications that need persistent storage.



Scale and Manage compute & storage

Disaggregated mode: Separate Amazon EKS worker node groups for storage nodes and storage-less (compute) nodes. The storage nodes provide storage capacity for all applications running on either of those Amazon EKS worker node groups. This topology allows organizations to scale their compute capacity independent of the storage capacity if they are working with compute-heavy applications.



Scale and Manage storage *Scale and Manage compute*

Once deployed, Portworx enables organizations to use the built-in Kubernetes StorageClass to dynamically provision block (ReadWriteOnce) and file (ReadWriteMany) persistent volumes for their stateful applications, or it allows organizations to create new StorageClass objects that are customized for their application.

You can customize a Portworx storage class based on the type of workload and the underlying storage layer. Here are a few settings that you can use to get the best out of Portworx:

- **fs:** xfs|ext4 – Filesystem to be laid out.
- **priority_io:** low|medium|high – IO priority for the volume. Use high for IOPS optimized volumes and use medium for throughput optimized.
- **shared_v4:** true – Flag to create a globally shared namespace volume that can be used by multiple pods over NFS with POSIX-compliant semantics.
- **repl:** 1|2|3 – Replication factor for the volume. This represents the number of copies stored on Portworx.
- **io_profile:** auto|db|db_remote|sequential|random – IO profiles change how a Portworx volume interacts with the underlying storage disks to improve traffic for different workloads. If you don't provide an IO profile, Portworx will set it to auto, and it will automatically apply an IO profile that is most appropriate to the data pattern it sees.
- **io_throttle_rd_bw** and **io_throttle_wr_bw:** Set read and write bandwidth limits in MB/s for persistent volumes, respectively.
- **io_throttle_rd_iops** and **io_throttle_wr_iops:** Set read and write IOPS limits for persistent volumes, respectively. Portworx allows you to set either the bandwidth limits or IOPS limits, but not both at the same time.

These are just a subset of all the parameters Portworx supports; for additional options, you can check out our documentation.

Stork: Storage Operator Runtime for Kubernetes

Stork is the storage scheduler from Portworx for Kubernetes that helps even tighter integration of Portworx and Amazon EKS. It allows users to co-locate pods with their data, provides seamless migration of data in case of errors, and makes it easier to create and restore snapshots of Portworx volumes. This is achieved by using a Kubernetes scheduler extender. So, every time a new pod is being scheduled on Amazon EKS, Stork will work with Portworx to ensure that the pod is being deployed on a worker node that has a local copy of the persistent volume. In case of a node failure, Stork also works with Amazon EKS to ensure that the new pod is deployed on a host with a replica of its persistent volume. All of this is automated, so as long as you specify 'schedulerName: stork' in your pod specification, you will get the best performance because of data locality using Stork.

Deploying Portworx on Amazon EKS

To get started with Portworx on Amazon EKS, you can 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. So, 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 we can customize our EKS cluster and attach the necessary IAM policies. Below is a sample ClusterConfig that deploys a 3-node Amazon EKS cluster:

```
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>
      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' ]
```

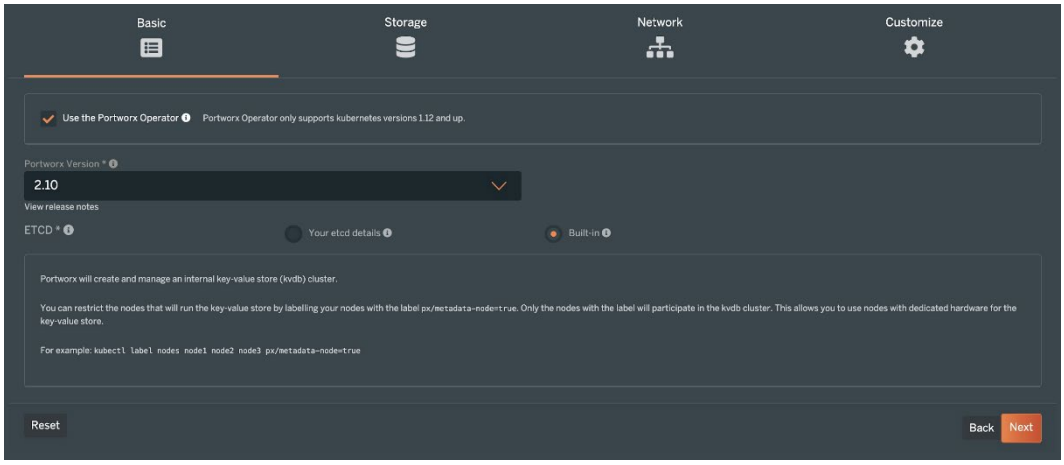
4. Deploy the Amazon EKS cluster using eksctl create command:

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

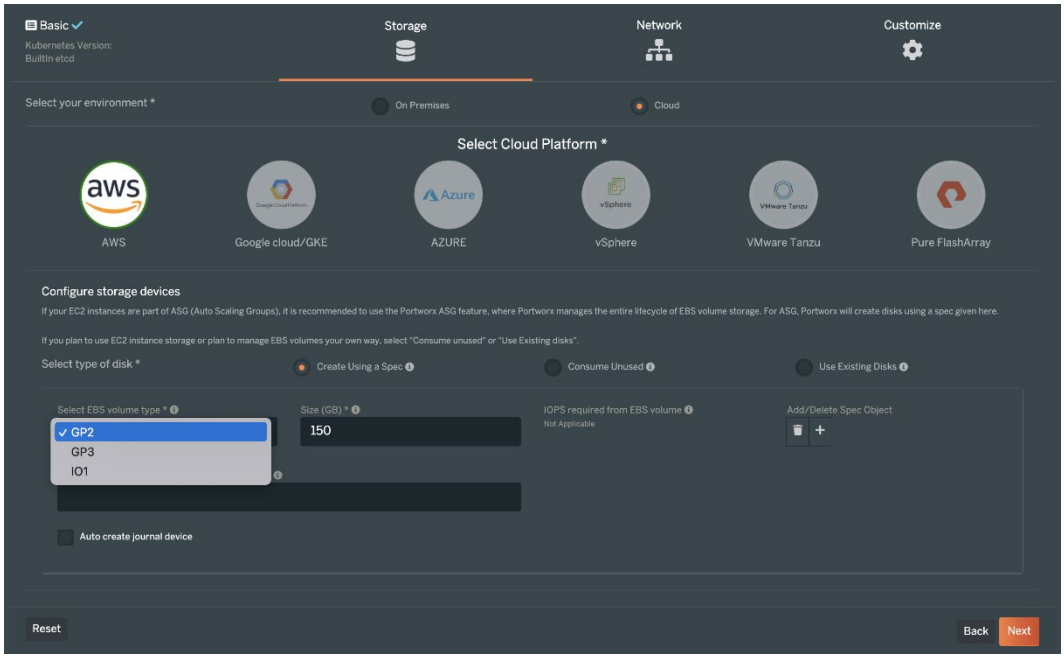
5. Once the cluster is deployed, you can navigate to PX-Central to generate a new specification for the Portworx cluster.

Select Portworx Enterprise and click **Continue.**

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.**

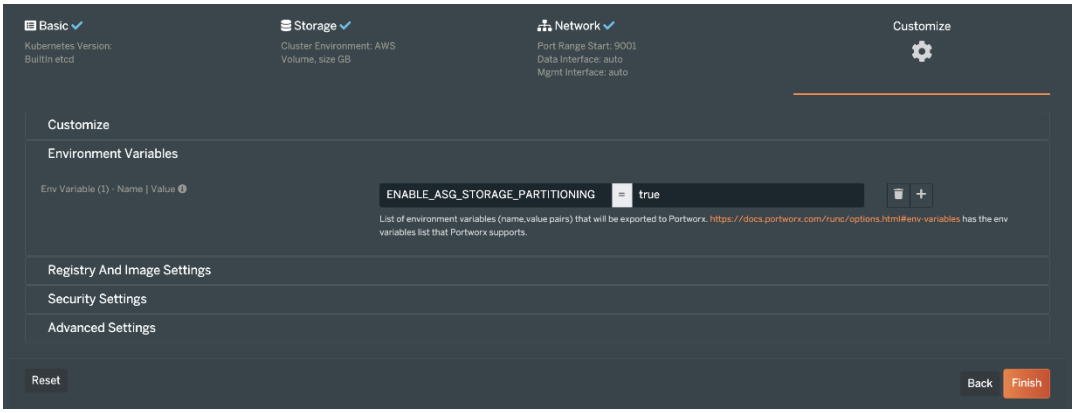


7. Select **Cloud** as your environment and select **AWS.** Here, you can select the type of EBS volumes and the size and number of IOPS you need per volume. These configuration details will be used by Portworx to create EBS volumes, attach them to each EKS worker node, and aggregate them into a single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

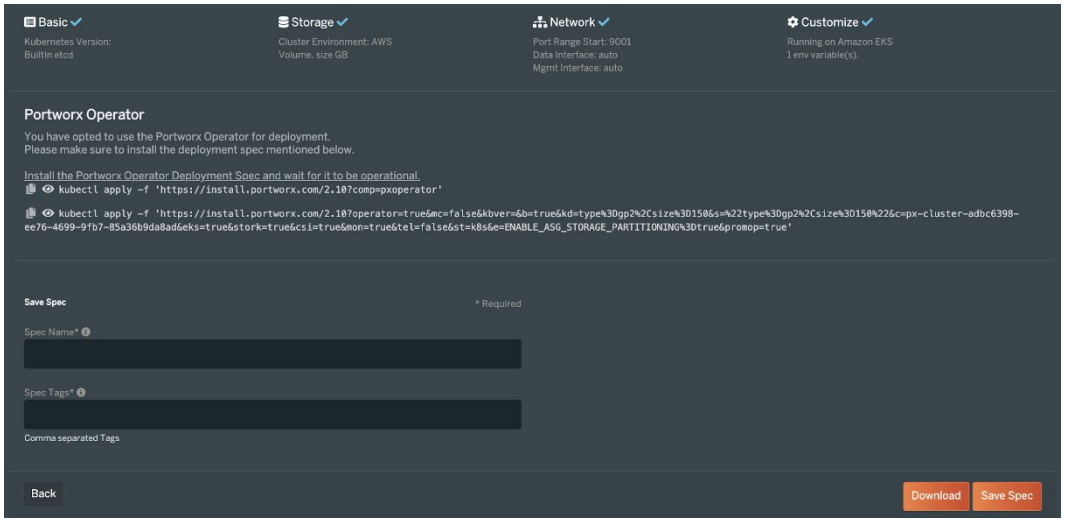


You can choose to leave the Network settings to default and click **Next.**

8. 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.**



9. 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.



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

High Availability and Replication

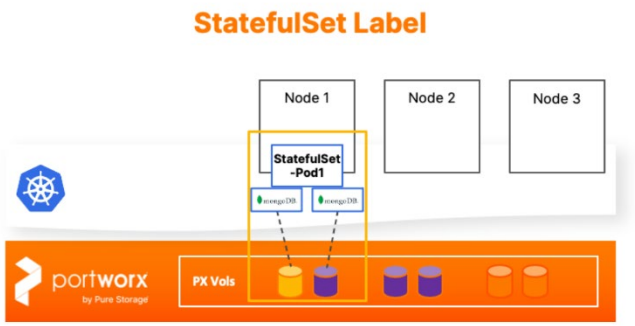
Portworx provides Kubernetes-native high availability and replication for stateful applications running on Amazon EKS. Portworx can store multiple copies of your persistent volumes and spread them across different Amazon EKS worker nodes running across multiple Availability Zones if your cluster is deployed in a cross-AZ method. If a node or an AZ goes down, Kubernetes, working with Stork, will automatically respawn the application pods to new nodes in the Amazon EKS cluster, which already have a replica of the persistent volume available locally. This decreases the amount of time it takes for applications to failover and come back online.

Portworx also allows users to leverage advanced VolumePlacementStrategy rules to define affinity and anti-affinity rules for their applications. Users can define their VolumePlacementStrategy by creating a spec containing affinity or anti-affinity rules. Here is what each allows user to do:

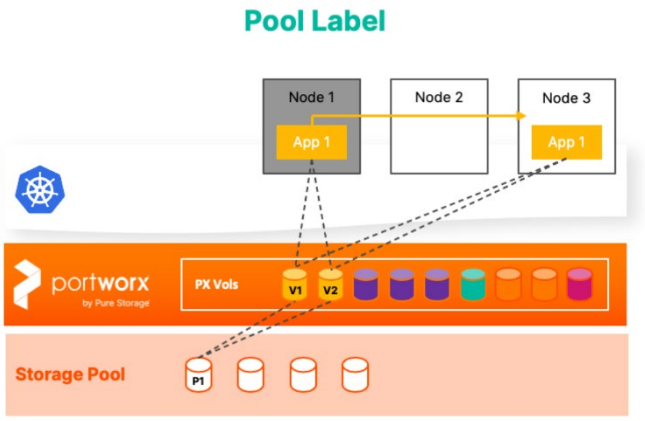
- **replicaAffinity:** Allows users to collocate volume replicas on nodes or storage pools that match the specified labels in the rule
- **replicaAntiAffinity:** Allows users to spread out the volume replicas across nodes or AZ domains
- **volumeAffinity:** Allows users to collocate volumes on the same nodes or storage pools that match the specified labels in the rule
- **volumeAntiAffinity:** Allows users to distribute the volumes across nodes, storage pools, or AZ domains

In addition to volume and replica affinity and anti-affinity rules, Portworx also allows users to add labels to their resources—like StatefulSets, storage pools, or namespaces—to collocate or distribute volumes across multiple nodes or AZs. Here is how each label works:

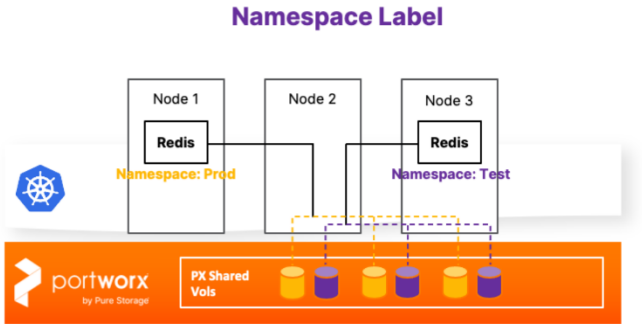
- **StatefulSet Label:** Defines volume affinity or anti-affinity rules to control volume placement relative to their parent StatefulSet pod



- **Pool Label:** Built-in pool label and topology key allows grouping or avoiding volumes on individual storage pools, not just Amazon EKS nodes



- **Namespace Label:** Allows users to filter placement rules by namespace labels, giving namespace awareness for volume placement rules and avoiding interference with other deployments



Using high availability and replication features along with volume placement strategies, Portworx can help you build architectures that are resilient to node failures, AZ failures, and any intra-cluster network failures or partitions.

Optimizing Infrastructure Resources for Amazon EKS

Portworx was built from the ground up for containers and Kubernetes, and it allows users to deploy hundreds of persistent volumes—block or file—on any single node in the Kubernetes cluster. By default, Amazon EKS uses the EBS CSI plugin, which allows users to automatically provision EBS volumes for each RWO persistent volume requested by the application pods. Because of this 1:1 mapping between persistent volumes and EBS volumes, users are restricted by the number of EBS volumes that can be mounted on an Amazon EKS worker node, which is just an Amazon EC2 instance.

According to AWS documentation, you can attach up to 40 EBS volumes to a Linux instance, which includes the root volume plus any attached instance store and EBS volumes. This limits the number of persistent volumes that can be provisioned and made available to application pods. Larger organizations running containerized applications in production will hit these limits quite easily and will need to scale their Amazon EKS worker node group to accommodate the requests for persistent volumes. This creates an unfavorable scenario that leads to an increase in the cloud spend on a per month basis to run the same application.

Architecturally, Portworx allows organizations to circumvent this issue. When users deploy Portworx on an Amazon EKS cluster, Portworx provisions, mounts, and aggregates larger Amazon EBS volumes across Amazon EKS worker nodes into a

single storage pool that will be used to provision block- and file-based persistent volumes, rather than deploying one EBS volume per persistent volume. Portworx allows users to provision hundreds of persistent volumes on a single Amazon EKS worker node, ensuring that organizations only toned to scale their Node Groups when they need more compute capacity for their application, instead of scaling up just as a workaround for mount point limits for EBS volumes.

As part of this solution validation testing, applications that requested 120 persistent volumes across a 3-node Amazon EKS cluster were never provisioned successfully, even if the number of requested persistent volumes was below the theoretical limits documented by Amazon.

In addition to the scale limits, Portworx also allows users to configure StorageClass objects with IOPS or throughput maximums for any persistent volumes deployed on Amazon EKS. These application IO control limits allow organizations to fine-tune their storage utilization and performance, which delivers more value to the organizations.

Achieving Better Performance for Shared File System Workloads

For applications that need access to a shared file system (ReadWriteMany) on Amazon EKS need to use the Amazon EFS CSI driver and manually provisioned backend Amazon EFS file systems. Amazon EFS file systems were not built for containers, and they take a performance hit when used at scale. As part of the solution validation testing, we performed scale testing comparing Amazon EFS-backed RWX persistent volumes to Portworx-backed RWX persistent volumes.

Test Setup

The following test setup was used, with the results below.

Kubernetes cluster: Three-node Amazon EKS cluster running M5.xlarge instances spread across us-east-1a, us-east-1b, and us-east-1d running Kubernetes version 1.21

Portworx storage cluster: Portworx version 2.10 running on Amazon EKS, with 1 Amazon EBS volume attached per Amazon EKS worker node. EBS volume used was type GP3 configured with 1000 MB/s throughput and 16000 IOPS, with a capacity of 150GB per node.

EFS storage backend: Amazon EFS file system running in us-east-1 with Max I/O mode configured. Max I/O mode for Amazon EFS supports 500000+ IOPS.

Storage class: Three different storage classes were configured:

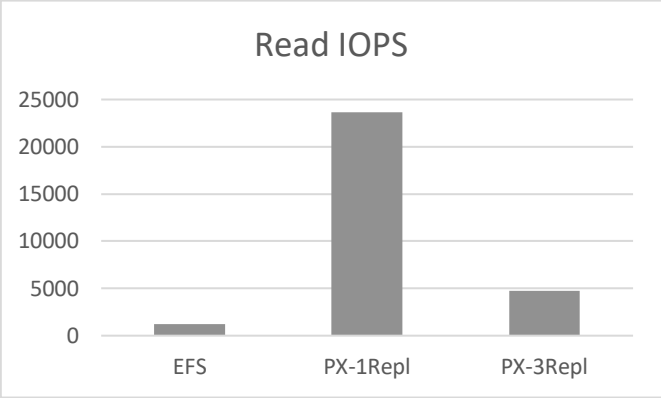
- EFS storage class
- Portworx storage class with replication factor set to 1
- Portworx storage class with replication factor set to 3

Test Harness: 10 FIO jobs running in parallel for the following IO profiles:

- Random R/W 60/40 Mix 4k block size
- Sequential R/W 60/40 Mix 256k block size
- Random Read 100% 4k block size
- Random Write 100% 4k block size
- Sequential Read 100% 256k block size
- Sequential Write 100% 256k block size

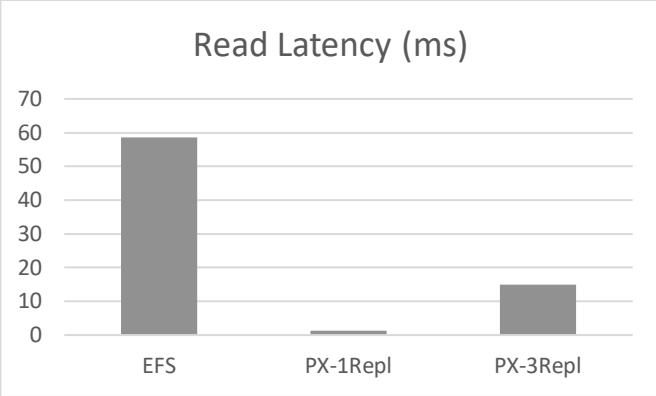
Results: Random R/W 60/40 Mix 4k Block Size

Read IOPS

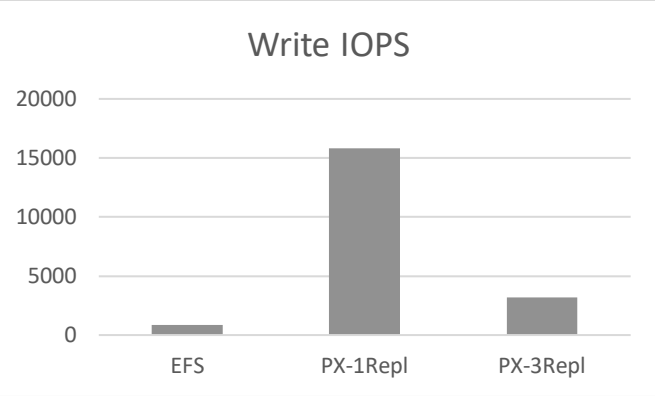


Read bandwidth (MB/s)

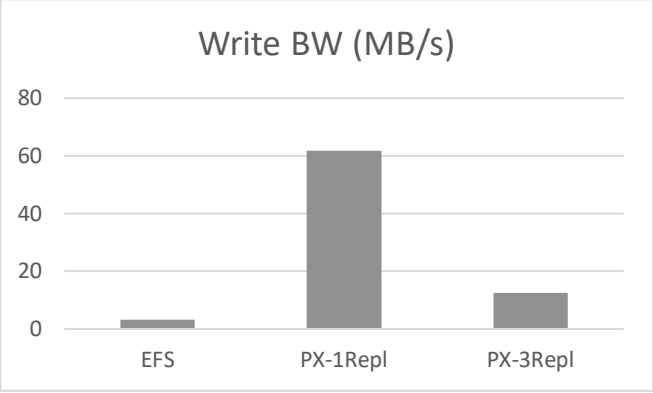
Read latency (ms)



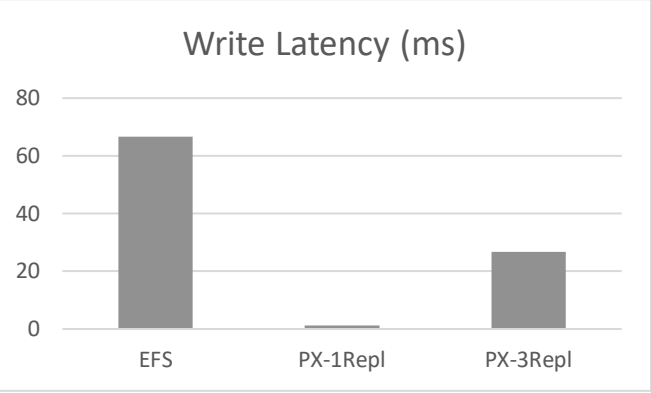
Write IOPS



Write bandwidth (MB/s)

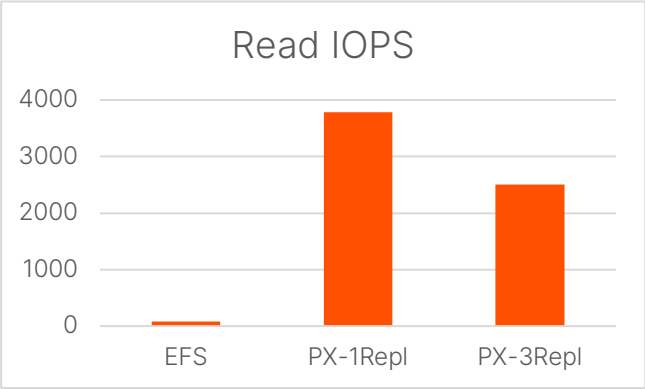


Write latency (ms)

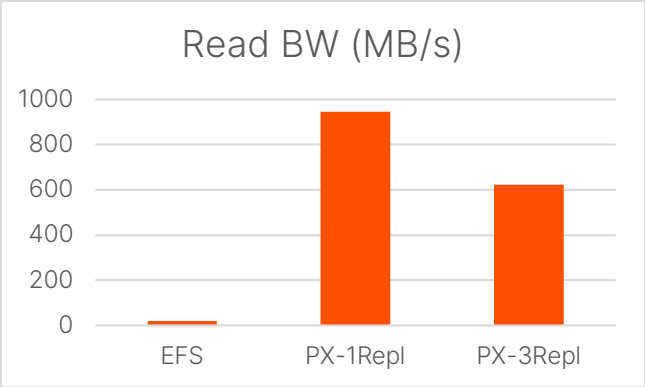


Results: Sequential R/W 60/40 Mix 256k Block Size

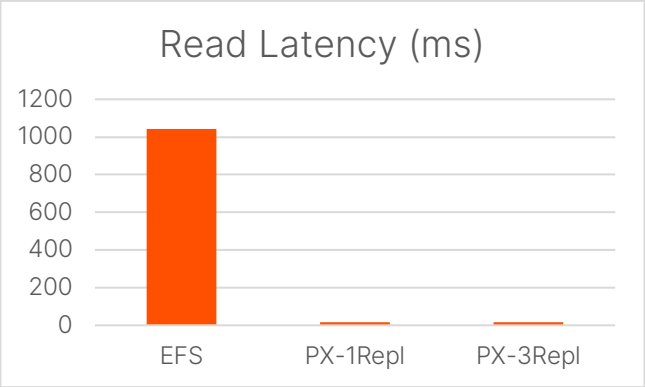
Read IOPS:



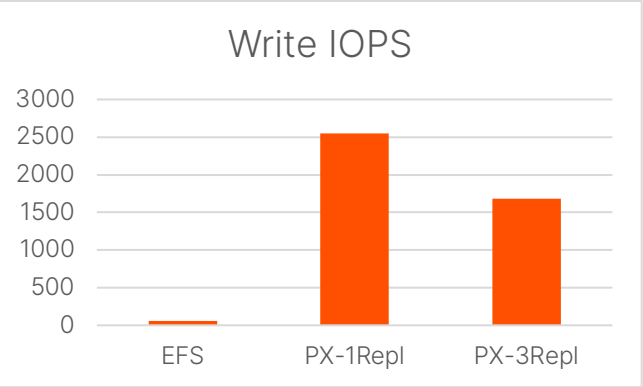
Read bandwidth (MB/s):



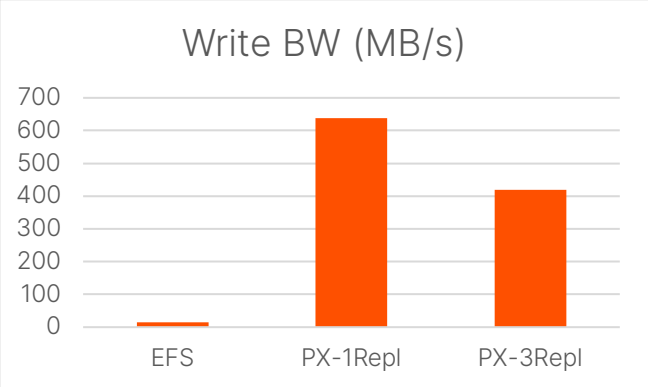
Read latency (ms):



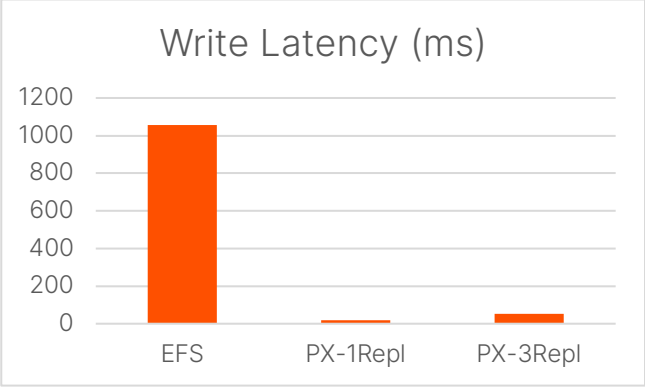
Write IOPS:



Write bandwidth (MB/s):

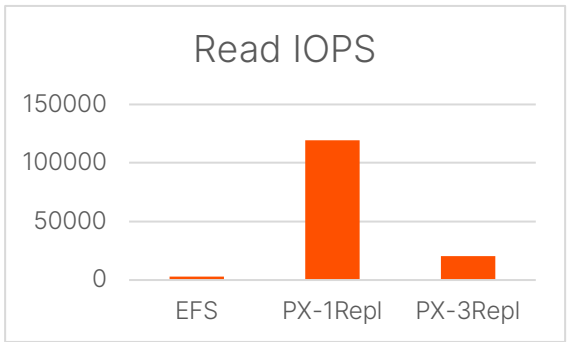


Write latency (ms):

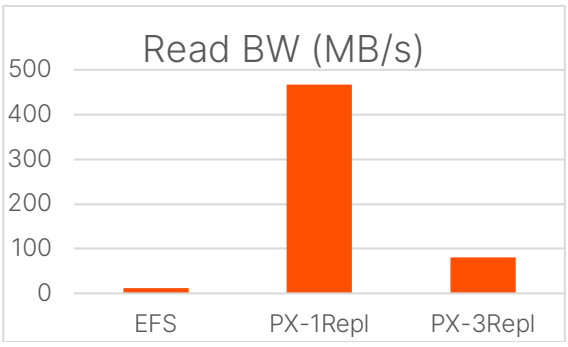


Results: Random Read 100% 4k Block Size

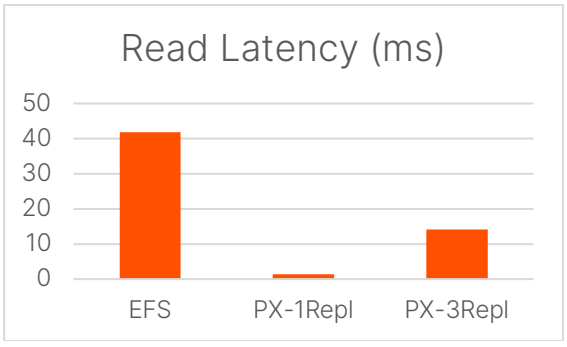
Read IOPS:



Read bandwidth (MB/s):

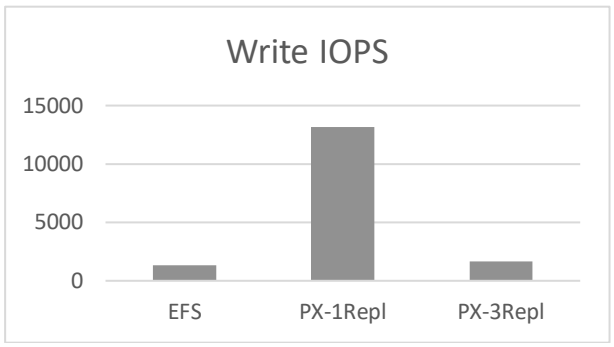


Ready latency (ms):

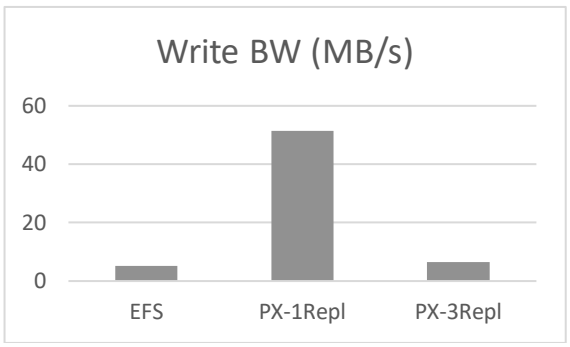


Results: Random Write 100% 4k Block Size

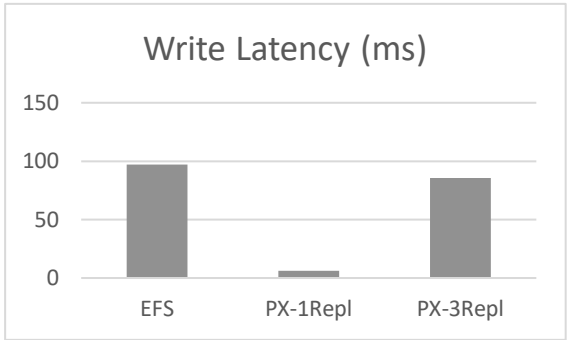
Write IOPS:



Write bandwidth (MB/s):

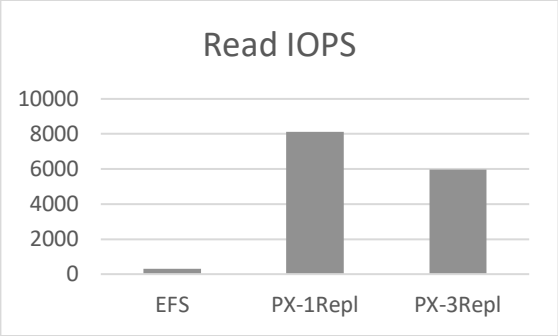


Write latency (ms):

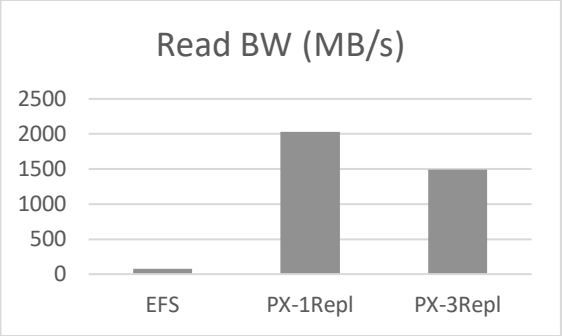


Results: Sequential Read 100% 256k Block Size

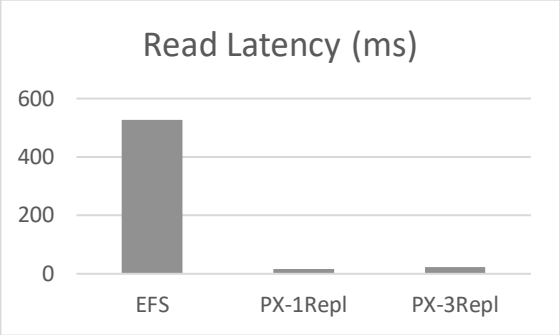
Read IOPS:



Read Bandwidth (MB/s):

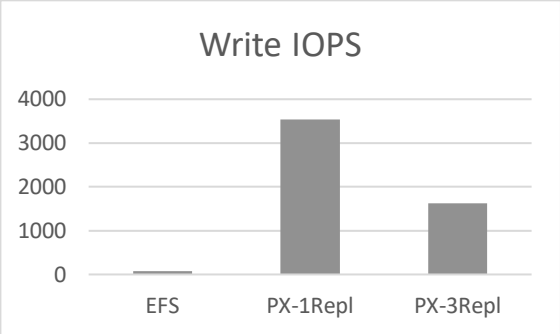


Read latency (ms):

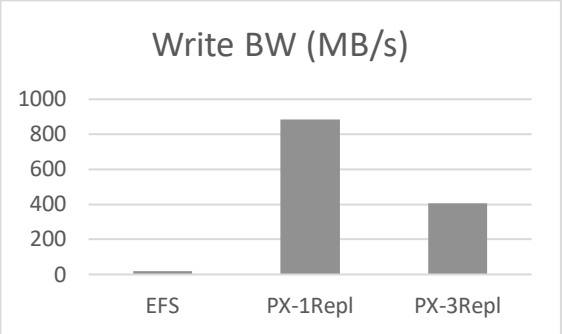


Results: Sequential Write 100% 256k Block Size

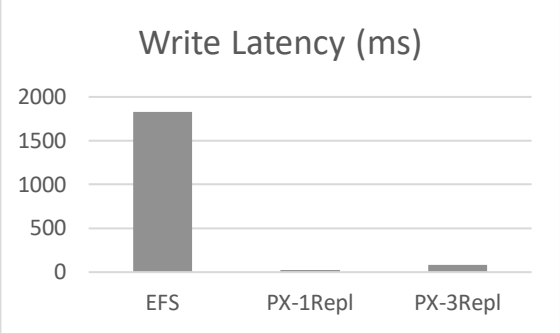
Write IOPS:



Write Bandwidth (MB/s):



Write Latency (ms):



Based on the testing we conducted, Portworx-based ReadWriteMany volumes perform well at scale compared to Amazon EFS-backed ReadWriteMany volumes. For applications like Jenkins, which need access to a RWX persistent volume on Amazon EKS, Portworx can significantly help improve application performance when running multiple CI/CD jobs in parallel.

Automated Capacity Management for Amazon EKS

Capacity management is a critical component to ensure application uptime. If your persistent volumes don't have any available capacity, your application will go offline. This usually involves users relying on monitoring tools to keep track of the capacity utilization of individual persistent volumes across multiple Amazon EKS clusters and then manually expanding persistent volumes when they are running low on capacity. This can be a tedious task, and it adds a lot of manual overhead when managing multiple Amazon EKS clusters at scale in production. Portworx allows users to leverage PX-Autopilot, which helps automate the storage capacity management for Amazon EKS clusters. Portworx PX-Autopilot provides a rule-based engine that responds to changes from a monitoring source. With PX-Autopilot, Amazon EKS clusters can react dynamically without admin intervention to events such as these:

- Resizing PVCs when they are running out of capacity
- Scaling Portworx storage pools to accommodate increased usage
- Rebalancing volumes across Portworx storage pools when they come unbalanced

Automatically Grow PVCs

PX-Autopilot allows administrators to create Autopilot rules that can execute certain actions if conditions are met for resources with specific labels. Below is a sample Autopilot rule that monitors resources with the label "app: postgres" in the namespace with the label "type:db" and automatically doubles the size of the persistent volume if the persistent volume is more than 70% full. PX-Autopilot will keep expanding the persistent volume until it hits a max size of 400GB.

```
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: volume-resize
spec:
  ##### selector filters the objects affected by this rule given labels
  selector:
    matchLabels:
      app: postgres
  ##### namespaceSelector selects the namespaces of the objects affected by this rule
  namespaceSelector:
    matchLabels:
      type: db
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    expressions:
```

```
    - key: "100 * (px_volume_usage_bytes / px_volume_capacity_bytes)"
      operator: Gt
      values:
        - "70"
  ##### action to perform when condition is true
  actions:
  - name: openstorage.io.action.volume/resize
    params:
      # resize volume by scalepercentage of current size
      scalepercentage: "100"
      # volume capacity should not exceed 400GiB
      maxsize: "400Gi"
```

Automatically Expand Portworx Storage Pools

With PX-Autopilot, we can also create Autopilot rules that allow administrators to automatically expand their Portworx storage pools by provisioning additional Amazon EBS volumes and attaching them to the Amazon EKS worker nodes. The YAML file lists an Autopilot rule that resizes your storage pool by adding new disks when the available capacity on your storage pool is less than 30%, for a maximum size of 900GB.

```
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: pool-expand
spec:
  enforcement: required
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    expressions:
    # pool available capacity less than 30%
    - key: "100 * ( px_pool_stats_available_bytes/ px_pool_stats_total_bytes)"
      operator: Lt
      values:
        - "30"
    # volume total capacity should not exceed 900GiB
    - key: "px_pool_stats_total_bytes/(1024*1024*1024)"
      operator: Lt
      values:
       - "900"
  ##### action to perform when condition is true
```

```
actions:
  - name: "openstorage.io.action.storagepool/expand"
    params:
      # resize pool by scalepercentage of current size
      scalepercentage: "50"
      # when scaling, add disks to the pool
      scaletype: "add-disk"
```

PX-Autopilot can completely automate storage capacity management using the YAML files listed above, or it can also help you put action approvals in place. In this case, PX-Autopilot will trigger approval requests for the administrators to approve using kubectl or by using the GitOps workflows. If the administrator approves these requests, PX-Autopilot will carry out the expansion operations and help administrators manage their storage capacity.

PX-Autopilot can work with AWS Karpenter to help build a solution that automatically expands storage capacity using PX-Autopilot, and it will automatically expand compute capacity by adding more Amazon EKS worker nodes using AWS Karpenter.

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 of 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, and others).

Portworx also offers a complete disaster recovery and business continuity solution with PX-DR. PX-DR allows customers 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.

Portworx from 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 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, hyperconverged infrastructure, cloud, and Kubernetes. Bhavin joined Pure Storage in March 2021 and works in the Cloud Native Business Unit.

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

PS2341-01 10/2022