

PURE VALIDATED DESIGN

Security for Amazon EKS with Portworx

Architecting a secure data management solution for Amazon Elastic Kubernetes service with Portworx.



Contents

Executive Summary	3
Introduction	3
Solution Overview	4
Role-based Access Control	4
Encryption	5
Ransomware Protection	5
Auditing	6
Data Security for Modern Applications	6
Solution Benefits	6
Amazon Elastic Kubernetes Service Overview	7
Amazon EKS Control Plane Architecture	7
Amazon EKS Deployment Options	7
Amazon KMS Use Cases	8
Amazon CloudWatch Container Insights	8
Amazon S3 Object Lock	8
Portworx Overview	9
PX-Store	10
Portworx Backup	10
PX-DR	11
PX-Autopilot	11
Portworx Client	12
Role-based Access Control for Amazon EKS with Portworx Enterprise	12
Role-based Access Control for Amazon EKS with Portworx Backup	22
Encryption with Portworx Enterprise	29
Encryption with Portworx Backup	33
Ransomware Protection with Portworx Backup	35
Create a Backup Location Bucket with Object Lock	35
Configure the Backup Location in Portworx Backup	38
Conclusion	40
About the Author	41



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 but doesn't provide the storage and data management features needed to run stateful applications in production. This Pure Validated Design is an overview of how Portworx® can complement Amazon EKS and provide capabilities like authorization and authentications, ownership, encryption, and auditing for containerized applications running on Amazon EKS. It discusses why organizations need modern security solutions and how they can meet their service level agreements (SLAs) for modern applications running on Amazon EKS using Portworx.

This Pure Validated Design is intended for DevOps engineers and administrators, cloud architects, and system architects who are interested in building a robust and secure solution for their Amazon EKS clusters.

When a solution gets the Pure Validated Design (PVD) designation, it means that Pure Storage® 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 consideration and deployment best practices to accelerate deployment. This process assures the chosen technologies form an integrated solution to address critical business objectives. This document provides design guidelines and deployment steps for deploying Amazon EKS and Portworx to provide a modern infrastructure platform for running Kubernetes.

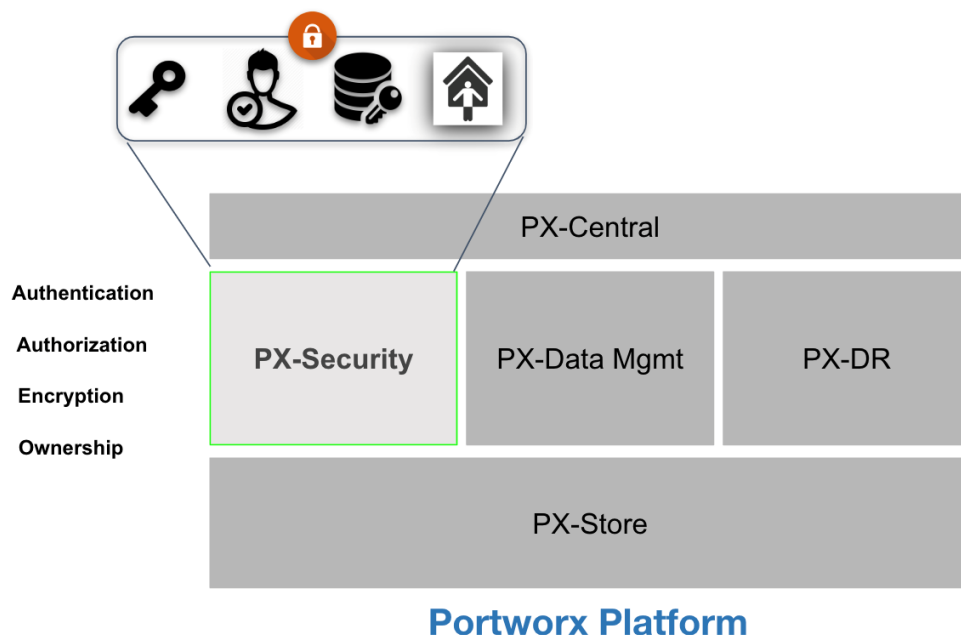
Introduction

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

This document lays out the different layers of security provided by Portworx, including role-based access control, authentication and authorization, ownership, auditing, encryption, and ransomware protection for Amazon EKS. To follow along with the deployment steps listed in this document, an architect will need to deploy one or multiple Amazon EKS clusters in a region of their choice. The Amazon EKS clusters should be running Portworx Enterprise as the Kubernetes storage layer for the secure data management solution, but Portworx Enterprise can be optional as the Kubernetes storage layer for the secure data protection solution.

Solution Overview

Data management and data protection platforms that enable enterprise grade security are key requirements to building and running applications in production. These requirements extend to the world of modern applications as well. Kubernetes provides a reliable platform for orchestrating and maintaining the desired state for modern applications, but organizations still need data management and data protection platforms that utilize the security mechanisms built into Kubernetes and seamlessly integrate with them. Before we talk about how you can design and architect such solutions using the Portworx portfolio, we will explain what we mean by security and discuss some terminologies that we will use throughout this document.



Role-based Access Control

Portworx role-based access control (RBAC) security is composed of three models:

- **Authentication:** A model for verifying the token is correct and generated from a trusted issuer
- **Authorization:** A model for verifying access to a particular request type according to the role or roles applied to the user
- **Ownership:** A model for determining access to resources

Portworx RBAC revolves around the ubiquitous JSON Web Token (JWT)-based authentication and authorization model. This technology is currently used by most major internet systems, providing a proven secure model for user and account identification.

A token is generated by a token authority (TA) and signed using either a private key or a shared secret. Then, the user should provide the token to Portworx for identification. **No passwords are ever sent to Portworx.**

As a result of this secure model, Portworx only has to verify the validity of the token to authenticate the user. Portworx then destroys the token, ensuring **tokens are never saved on a Portworx system.**

The token contains a section called "claims," which identifies the user and provides authorization information in the form of RBAC. Portworx uses the RBAC information to determine if the user is authorized to make the request.



Authentication: Authentication is based on role-based access control for all clients in the stack and an ownership model that is much like the familiar Unix-style permissions. Portworx will determine the validity of a user through a token-based model. The token will be created by the token authority and will contain information about the user in the claims section. When Portworx receives a request from the user, it will check the token validity by verifying its signature, using either a shared secret or public key provided during configuration.

NOTE: Authentication does not mean a requester has access to do something; the user must be authorized first.

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

Ownership: An owner, in this case, is a user. A user can be a person, system, or admin interacting with the Portworx platform. That owner can manage various aspects of the ownership model, such as who can access an object and how those users can interact with the resource. In this case, the object is likely a volume, snapshot, or clone.

Owners belong to groups themselves and contain a type of role. In other words, a user can have a specific role—such as an admin role, a user role, a view only role, or even a custom role—that defines how the user can interact with the resources in the platform.

An owner of a volume can add access via collaborators and groups. Each type of access can be restricted, so by default a collaborator or group does not have full access. The types of access are defined below:

- **Read access:** Allows access to the volume without modifying it. With a read access type, one can clone or inspect the volume.
- **Write access:** Allows modification of the volume and its metadata, for example allowing to mount, unmount, and restore the volume from a snapshot, in addition to all read access.
- **Admin access:** Allows full access to the volume, such as deleting the volume, in addition to all write access.

Encryption

Encryption can happen in a few different places; the first is encrypting data at rest with encryption keys for cluster-wide encryption or per-PVC encryption with Portworx. The second is to encrypt data in transit as it is transferred across the network. To enable encryption for volumes or in transit during backups with Portworx, a key must hold a passphrase used for encryption. This key can live in Kubernetes as a Kubernetes Secret or in external KMS systems such as AWS KMS.

Ransomware Protection

Immutability achieved by the object lock write once, read many (WORM) model is key to enabling ransomware protection for Kubernetes applications. Portworx Backup discovers your backup targets that have object lock enabled and allows you to set the retention period to prevent ransomware attacks from destroying any backups. Object lock is also good for compliance and governance as well as for preventing human operator error.



Auditing

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

Data Security for Modern Applications

Security is essential for organizations, and those using modern applications that are built using containers and microservices and deployed using Kubernetes infrastructure are no exception.

Kubernetes provides role-based access control authorization to help regulate access to specific resources within Kubernetes based on a set of roles. These mechanisms are great for regulating access to Kubernetes-specific objects—such as services, namespaces, quotas, and more—however, namespaces and pod security policies alone are not enough to restrict who can request changes to the underlying data management system.

Many enterprises plug into other platforms to provide the substrate for network and storage through APIs such as CNI or CSI. What Kubernetes lacks, however, is the ability to extend the RBAC into these subsystems—it is now the responsibility of the external system to ensure authorization and authentication as well. This is why we here at Portworx work together with Kubernetes to provide RBAC, encryption, and ownership through roles for the persistent volumes backing your PVCs in Kubernetes. This creates a seamless layer of protection that provides the following for your PVCs:

- Users within a namespace can be limited by their role, such that they could have read, write, admin or custom access.
- Users can be authenticated via a token automatically, so requests are authorized from specific namespaces for audit.
- Users can be placed into tenant-based namespaces to provide secure multi-tenancy for access to PVCs.
- Even if a user sees a storage class, it doesn't mean they are authorized to create a PVC.
- Using Portworx RBAC with encryption means data is secure on the host and cannot be accessed by non-authorized users within a namespace.
- If a request comes from outside of Kubernetes without a token, it will be blocked.

The importance of encryption is paramount to keeping information confidential—whether it is at rest or in transit. Data within the PVCs of a Kubernetes application should be encrypted for those applications using sensitive information—such as those in the healthcare space using PII and PHI. Data in transit should be encrypted so that no data is seen in plain text crossing the network, which could result in a man-in-the-middle attack.

Solution Benefits

This solution enables organizations to build robust and secure solutions for their modern applications running on Amazon EKS clusters. Using Portworx PX-Secure, organizations can build solutions that ensure the data is protected and secure. These architectures help organizations secure their application data and ensure only authorized users can access the data. Using Portworx Backup, organizations can build a backup and restore solution that helps them tolerate accidental deletions, data loss, or even ransomware attacks.

Amazon Elastic Kubernetes Service Overview

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications.

Amazon EKS delivers these features:

- Runs and scales the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability
- Automatically scales control plane instances based on load, detects and replaces unhealthy control plane instances, and provides automated version updates and patching for them
- Is integrated with many AWS services to provide scalability and security for your applications
- Runs 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.

Here is what Amazon EKS does:

- It actively monitors the load on control plane instances and automatically scales them to ensure high performance.
- It 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

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.

Amazon KMS Use Cases

You can use Amazon KMS with any, or all, of the following use cases:

- **Protect your data at rest:** Activate server-side encryption with AWS KMS using KMS keys that you control and manage.
- **Encrypt and decrypt data:** Use AWS Encryption SDK to securely handle cryptographic operations in your applications.
- **Sign and verify digital signatures:** Protect signing operations with AWS KMS using asymmetric KMS keys.
- **Validate JSON web tokens using HMAC:** Generate HMAC using AWS KMS to verify message integrity and authentication.

Amazon CloudWatch Container Insights

Use CloudWatch Container Insights to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. Container Insights is available for Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Kubernetes platforms on Amazon EC2. Container Insights supports collecting metrics from clusters deployed on Fargate for both Amazon ECS and Amazon EKS.

CloudWatch automatically collects metrics for many resources, such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects.

In Amazon EKS and Kubernetes, Container Insights uses a containerized version of the CloudWatch agent to discover all the running containers in a cluster. It then collects performance data at every layer of the performance stack.

Container Insights supports encryption with the customer master key (CMK) for the logs and metrics that it collects. To enable this encryption, you must manually enable KMS encryption for the log group that receives Container Insights data. This results in Container Insights encrypting this data using the provided CMK. Only symmetric CMKs are supported. Do not use asymmetric CMKs to encrypt your log groups.

Amazon S3 Object Lock

With S3 Object Lock, you can store objects using a write once, read many (WORM) model. Object Lock can help prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. You can use Object Lock to help meet regulatory requirements that require WORM storage or to simply add another layer of protection against object changes and deletion.

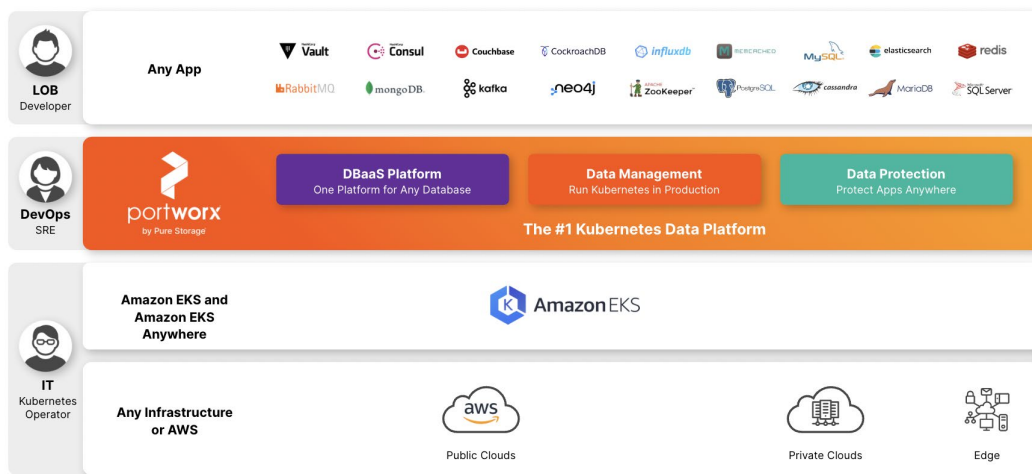
Object Lock provides two ways to manage object retention: retention periods and legal holds.

- **Retention period:** This specifies a fixed period of time during which an object remains locked. During this period, your object is WORM-protected and can't be overwritten or deleted. For more information, see [Object Lock retention](#).
- **Legal hold:** Provides the same protection as a retention period, but it has no expiration date. Instead, a legal hold remains in place until you explicitly remove it. Legal holds are independent from retention periods. For more information, see [Object Lock legal hold](#).

An object version can have both a retention period and a legal hold, one but not the other, or neither. For more information, see [How S3 Object Lock works](#).

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 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:** 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
- **Portworx Backup:** Allows enterprises to back up and restore the entire Kubernetes application, including data, app configuration, and Kubernetes objects, to any backup location—including AWS S3, Azure Blob, 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.

Portworx 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 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, Portworx 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:** Portworx 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:** Portworx 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 in half:** Intelligently provision cloud storage only when needed and eliminate the problem of paying for storage when over-provisioned instead of when consumed. Scale at the individual volume or entire cluster level to save money and avoid application outages.
- **Integrate with all major clouds and VMware:** PX-Autopilot natively integrates with AWS, Azure, Google, and VMware Tanzu, enabling you to achieve savings and increase automated agility across all your clouds.

Deployment Options

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

- **KVDB database:** 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 the following two scenarios:
 - The first scenario is when the PX-DR is used to configure SyncDR for Kubernetes clusters, where the KVDB instance is shared across the two Kubernetes clusters.
 - The second scenario, in which a dedicated etcd cluster should be used, is for large-scale deployment with 25 or more worker nodes, in which a heavy dynamic provisioning activity takes place.
- **Dedicated journal device:** A dedicated journal device can be specified to buffer metadata writes.
- **Dedicated cache device:** A dedicated cache device can be specified to [improve performance](#) by acting as a read/write-through cache.
- **Container storage interface (CSI) API compatibility:** You can choose the option to deploy Portworx with CSI enabled if PX-Secure is to be used.
- **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.

Portworx Client

DISCLAIMER: Portworx client is not supported by Pure Storage and is available “as-is.”

Portworx Client (pxc) is a client-side application that communicates with Portworx, Kubernetes, and other services to provide users with an integrated tool. It can be used as a stand-alone program or as a kubectl plugin. The Portworx Client (pxc) will allow us to highlight the security features of Portworx Enterprise.

To install Portworx Client, navigate to the [Releases](#) section on GitHub and select the correct operating system where you are running kubectl. Extract the binaries and copy kubectl-pxc anywhere in your PATH—for example, /usr/local/bin. You will now be able to run commands such as these:

```
$ kubectl pxc cluster describe
$ kubectl pxc node list
$ kubectl pxc pvc list
```

Role-based Access Control for Amazon EKS with Portworx Enterprise

Portworx PX-Secure allows organizations to authenticate, authorize, and determine ownership for applications running on Amazon EKS using role-based access controls (RBAC). In this section, we will talk about how you can leverage PX-Secure to securely access your application data.

Portworx RBAC revolves around the ubiquitous JSON Web Token (JWT)-based authentication and authorization model. This technology is currently used by most major internet systems, providing a proven secure model for user and account identification. A token is generated by a token authority (TA) and signed using either a private key or a shared secret. Then, the user should provide the token to Portworx for identification. No passwords are ever sent to Portworx.

As a result of this secure model, Portworx only must verify the validity of the token to authenticate the user. Portworx then destroys the token ensuring tokens are never saved on a Portworx system. The token contains a section called “claims,” which identifies the user and provides authorization information in the form of RBAC. Portworx uses the RBAC information to determine if the user is authorized to make the request.

Required JSON Web Token (JWT) Claims

Portworx requires a set of claims to be provided to authorize the user. The following table lists the set of required claims:

Claim	Type	Description
iss	string	Name of the issuer. Portworx utilizes this information to determine if the token was issued by a trusted token authority
sub	string	Unique identifier of the user. It is used to determine access control of resources.
Exp	date	Expiration date
iat	date	Issued at time
name	string	Name of the user
email	string	Email of the user. Portworx may be configured to use the email as the unique identifier of the user.
Roles	string array	(Custom claim) Portworx RBAC roles
groups	string array	(Custom claim) List of groups the user is part of. It is used to determine access control of resources.

Table 1. Required claims

With so many choices available for an OpenID Connect (OIDC) provider, we will highlight the RBAC controls using a self-signed configuration. One example of an OpenID Connect provider is AWS Cognito.

1. To get started, generate a separate Portworx configuration with the already-deployed EKS cluster using the spec generator on [PX-Central](#). While generating the installation spec, on the **Customize** tab, select **Security Settings**, and check the box for **Enabling Authentication**.



This will add the following section to the storageCluster YAML:

```
spec:
  security:
    enabled: true
```

- After applying the storageCluster, additional defaults will be added to the storageCluster customResource. These options can be customized for your OIDC provider.

```
Security:
  auth:
    guestAccess: Enabled
    selfSigned:
      issuer: operator.portworx.io
      sharedSecret: px-shared-secret
      tokenLifetime: 24h
  enabled: true
```

In addition, the following secrets will be generated in the kube-system namespace:

- Shared Secret stored under the secret px-shared-secret
- Admin token stored under the secret px-admin-token
- User token stored under the secret px-user-token

- First, we will configure an environment variable, PX_POD, to access a Portworx Enterprise pod:

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

- Test that you are unable to use `/opt/pwx/bin/pxctl` to review the status of the Portworx Enterprise cluster without authenticating with the admin token:

```
kubectl -n kube-system exec -ti $PX_POD -- /opt/pwx/bin/pxctl status
couldn't get: /config with error: Access denied token is empty
```

- We will create a Storage Admin that has the roles and responsibility of a full-access user that can interact with `pxctl`. There should only be one or two of these to limit full access.

Retrieve the admin token from the namespace in which Portworx was installed and store it in a variable `ADMIN_TOKEN`:

```
ADMIN_TOKEN=$(kubectl -n kube-system get \
  secret px-admin-token -o json \
  | jq -r '.data["auth-token"]' \
  | base64 -d)
```

Create the admin user using the `PX_POD` and `ADMIN_TOKEN` environment variables set earlier:

```
kubectl -n kube-system exec -ti $PX_POD -- /opt/pwx/bin/pxctl context create admin --
token=$ADMIN_TOKEN
```

NOTE: The `px-admin-token` secret contains a JWT token with the `system.admin` role in the “auth-token” data.

- Test that you are now able to use `/opt/pwx/bin/pxctl` to review the status of the Portworx Enterprise cluster as the admin user:

```
kubectl -n kube-system exec -ti $PX_POD -- /opt/pwx/bin/pxctl status

Status: PX is operational
Telemetry: Disabled or Unhealthy
...
Global Storage Pool
    Total Used      : 3.0 GiB
    Total Capacity  : 60 GiB

kubectl -n kube-system exec -ti $PX_POD -- /opt/pwx/bin/pxctl context list

contextconfig:
  current: admin
```

- Retrieve the shared secret from the namespace in which Portworx was installed and store it in a variable `PORTWORX_AUTH_SHARED_SECRET` in order to generate tokens for a user and view role:

```
PORTWORX_AUTH_SHARED_SECRET=$(kubectl -n kube-system get \
    secret px-shared-secret -o json \
    | jq -r '.data."shared-secret"' \
    | base64 -d)
```

- Next, we'll create a Kubernetes user that has the role and responsibility of acting as a verifying "user," confirming that Kubernetes is, in fact, allowed to interact with Portworx and that the request is coming from Kubernetes.

We will use `pxc` to generate a token and store it in a secret in the `kube-system` namespace:

```
KUBERNETES_TOKEN=$(kubectl pxc utilities token-generate \
    --token-email=kubernetes@local.net \
    --token-name="Kubernetes" \
    --token-roles=system.user \
    --token-groups=kubernetes \
    --token-duration=7d \
    --token-issuer=operator.portworx.io \
    --token-subject="kubernetes@local.net/kubernetes" \
    --shared-secret=$PORTWORX_AUTH_SHARED_SECRET)

kubectl -n kube-system create secret \
    generic px-kubernetes-token --from-literal=auth-token=$KUBERNETES_TOKEN
```

- Lastly, we'll create a simple view-only user to demonstrate how Portworx can limit access to the underlying data management APIs.

```
VIEWER_TOKEN=$(kubectl pxc utilities token-generate \
    --token-email=viewer@example.com \
    --token-name="Viewer" \
    --token-roles=system.view \
    --token-groups=viewer \
    --token-duration=7d \
    --token-issuer=operator.portworx.io \
    --token-subject="viewer@example.com/viewer" \
    --shared-secret=$PORTWORX_AUTH_SHARED_SECRET)

kubectl -n kube-system create secret \
    generic px-viewer-token --from-literal=auth-token=$VIEWER_TOKEN
```

- Use the viewer secret and attempt to create a volume via `pxc`; then switch to the Kubernetes secret and attempt again.

```
kubectl pxc --pxc.secret-name px-viewer-token --pxc.secret-namespace kube-system volume create
viewer-volume --size 5
Failed Create a volume: Access to /openstorage.api.OpenStorageVolume/Create denied: rpc error: code
= PermissionDenied desc = Access denied to roles: [system.view]
```

Multi-tenancy

This model is based on isolating tenant accounts by namespaces. You will need to create an account for the tenant in Kubernetes and restrict it to one or more namespaces. You will then store the tenant's token in each namespace they own.

The following steps provide instructions for creating and storing the token in a namespace for the tenant:

1. Retrieve the shared secret from the namespace in which Portworx was installed and store it in a variable PORTWORX_AUTH_SHARED_SECRET:

```
PORTWORX_AUTH_SHARED_SECRET=$(kubectl -n kube-system get \
    secret px-shared-secret -o json \
    | jq -r '.data."shared-secret"' \
    | base64 -d)
```

2. Create the tenant-a namespace:

```
kubectl create ns tenant-a
```

3. Generate the tenant-a token and save it in a secret called tenant-a/px-user-token:

```
TENANTA_TOKEN=$(kubectl pxc utilities token-generate \
    --token-email=tenant-a@example.com \
    --token-name="Tenant-a" \
    --token-roles=system.user \
    --token-groups=tenant-a \
    --token-duration=7d \
    --token-issuer=operator.portworx.io \
    --token-subject="tenant-a@example.com" \
    --shared-secret=$PORTWORX_AUTH_SHARED_SECRET)

kubectl -n tenant-a create secret \
    generic px-user-token --from-literal=auth-token=$TENANTA_TOKEN
```

4. Annotate the Kubernetes secret so that other components like Stork and Portworx Backup do not back up this resource.

```
kubectl -n tenant-a annotate secret px-user-token \
    stork.libopenstorage.org/skipresource=true
```

5. Create a storageClass for use by the tenants. With CSI, the secret-namespace will be set to a variable of `${pvc.namespace}`:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-tenant-sc
provisioner: pxd.portworx.com
parameters:
  repl: "2"
  csi.storage.k8s.io/provisioner-secret-name: px-user-token
  csi.storage.k8s.io/provisioner-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: px-user-token
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/controller-expand-secret-name: px-user-token
  csi.storage.k8s.io/controller-expand-secret-namespace: ${pvc.namespace}
```

6. Create a persistentVolumeClaim as tenant-a in the tenant-a namespace:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tenant-a-pvc
  namespace: tenant-a
spec:
  storageClassName: px-tenant-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

7. Review audit information to validate tenant-a can create a volume in their namespace:

```
Audit log information:
time="2022-08-31T18:24:21Z" level=info msg="Volume 493432375935037764 created" email=tenant-
```

```
a@example.com groups="[tenant-a]" method=volume.create name=Tenant-a roles="[system.user]"
subject=tenant-a@example.com username=tenant-a@example.com
time="2022-08-31T18:24:21Z" level=info msg=Authorized email=tenant-a@example.com groups="[tenant-a]"
method=volume.create name=Tenant-a roles="[system.user]" subject=tenant-a@example.com
username=tenant-a@example.com
time="2022-08-31T18:24:21Z" level=info msg=Authorized email=tenant-a@example.com groups="[tenant-a]"
method=volume.inspect name=Tenant-a roles="[system.user]" subject=tenant-a@example.com
username=tenant-a@example.com
```

8. Create the tenant-b namespace:

```
kubectl create ns tenant-b
```

9. Generate the tenant-b token and save it in a secret called tenant-a/px-user-token:

```
TENANTB_TOKEN=$(kubectl pxc utilities token-generate \
  --token-email=tenant-b@example.com \
  --token-name="Tenant-b" \
  --token-roles=system.user \
  --token-groups=tenant-b \
  --token-duration=7d \
  --token-issuer=operator.portworx.io \
  --token-subject="tenant-b@example.com" \
  --shared-secret=$PORTWORX_AUTH_SHARED_SECRET)

kubectl -n tenant-b create secret \
  generic px-user-token --from-literal=auth-token=$TENANTB_TOKEN
```

10. Annotate the Kubernetes secret so that other components like Stork and Portworx Backup do not back up this resource:

```
kubectl -n tenant-b annotate secret px-user-token \
  stork.libopenstorage.org/skipresource=true
```

We will make use of the tenant-a and tenant-b users in the next section on ownership.

Ownership

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

The below three access types are supported:

- **Read:** Has access to view or copy the resource. Cannot affect or mutate the resource.
 - **Write:** Has read access plus permission to change the resource.
 - **Admin:** Has write access plus the ability to delete the resource.
1. Generate persistentVolumeClaims in both namespaces tenant-a and tenant-b. We will verify that each tenant can only see their own volumes with pxc:

```
kubectl pxc --pxc.secret-name px-user-token --pxc.secret-namespace tenant-a volume list
```

Name	Size	HA	Shared	Status	State
Pods					
----	----	--	-----	-----	-----

pvc-9e3b51a1-a1a8-41ca-bc7b-4edae9eb9803	10 GiB	2	false	UP	on ip-192-168-46-73.ec2.internal tenant-a/postgres-7957478b7d-v5qqx

```
kubectl pxc --pxc.secret-name px-user-token --pxc.secret-namespace tenant-b volume list
```

Name	Size	HA	Shared	Status	State
Pods					
----	----	--	-----	-----	-----

pvc-48e1dd2b-3451-4570-916e-586e77f0071a	10 GiB	2	false	UP	on ip-192-168-72-88.ec2.internal tenant-b/postgres-7957478b7d-lr1nr

2. Next, we are going to make the tenant-a volume available to tenant-b user with read access:

```
kubectl pxc --pxc.secret-name px-user-token --pxc.secret-namespace tenant-a volume update pvc-9e3b51a1-a1a8-41ca-bc7b-4edae9eb9803 --add-collaborators tenant-b@example.com:r
Volume pvc-9e3b51a1-a1a8-41ca-bc7b-4edae9eb9803 parameter updated successfully
```

3. Validate that tenant-b can now see the tenant-a volumes:

```
kubectl pxc --pxc.secret-name px-user-token --pxc.secret-namespace tenant-b volume list
```

Name	Size	HA	Shared	Status	State
Pods					
----	----	--	-----	-----	-----

pvc-9e3b51a1-a1a8-41ca-bc7b-4edae9eb9803	10 GiB	2	false	UP	on ip-192-168-46-73.ec2.internal tenant-a/postgres-7957478b7d-v5qqx
pvc-48e1dd2b-3451-4570-916e-586e77f0071a	10 GiB	2	false	UP	on ip-192-168-72-88.ec2.internal tenant-b/postgres-7957478b7d-lr1nr

Guest Access

The guest role allows your users to access and manipulate public volumes. The `system.guest` role is now used whenever a user does not pass a token to any Portworx operation or SDK call. This role, by default, allows all standard user operations—such as Create, Delete, Mount, and Unmount—for public volumes.

Volumes are public if they are created before security was enabled or without a token. This simplifies the process for upgrading from “auth disabled” to “auth enabled” for Portworx 2.6+. Additionally, the admin may mark a volume as public to expose it to the guest role.

The role is mutable and is named `system.guest`. This allows the admin to change how the auth system behaves when an unauthenticated user interacts with it. Strict admins may disable the guest role entirely if they wish.

Disabling Guest Access

NOTE: Once the guest role is disabled, volumes created without a token will not be accessible without a token.

To disable guest access mode, set the value of the `spec.security.auth.guestAccess` field to `Disabled`:

```
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  name: px-cluster
  namespace: kube-system
spec:
  security:
    enabled: true
    auth:
      guestAccess: 'Disabled'
```

Managing the Guest Role Yourself

You can exercise finer control over the `system.guest` role by setting it to managed mode. This instructs the Operator to stop updating the `system.guest` role, allowing you to customize it yourself.

To enter managed mode, set the value of the `spec.security.auth.guestAccess` field to `Managed`:

```
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  name: px-cluster
  namespace: kube-system
spec:
  security:
    enabled: true
    auth:
      guestAccess: 'Managed'
```

Role-based Access Control for Amazon EKS with Portworx Backup

Portworx Backup Security allows you to control user access to certain resources by setting governance policies and managing permissions for the application owners on the platform. Portworx Backup Security is a role-based access control (RBAC) system that enables authorization for users or user groups through an existing OIDC authentication service such as Keycloak and Okta.

Portworx Backup allows mapping users or user groups to specific roles. These roles control actions and permissions that a user is allowed to perform. Administrators set the scope of access and allow users to share resources.

Portworx Backup Security allows administrators and application owners manage access to the following resources:

- Cloud accounts
- Backup locations
- Schedule policies
- Rules
- Roles

Kubernetes administrators and application owners use Portworx Backup Security to configure backups and restores by providing a granular level of authorization to Portworx Backup resources. Portworx Backup Security supports different levels of authorization across multiple Portworx Backup deployments while retaining the same user management and authentication.

Portworx Backup is managed using PX-Central, which provides OIDC integration. Portworx Backup Security for clusters is controlled by Kubernetes access control. Administrators can add their clusters in Portworx Backup with the credentials or Kubeconfig assigned to them. Portworx Backup inherits the permissions from Kubernetes and displays the resources that a user has permission to access.

Portworx Backup built-in roles: The Portworx Backup built-in roles match user personas managing the Kubernetes infrastructure and applications.

Infrastructure administrator (px-backup.infra.admin): This is the infrastructure owner with administrator privileges for Portworx Backup resources such as cloud accounts, backup locations, schedules, and rules. Infrastructure administrators create custom rules, in addition to the built-in rules in Portworx Backup. Infrastructure owners, or any user, can create a shared resource pool to share backup locations, schedules, and backup rules with other users.

Application administrator (px-backup.app.admin): Application administrators can manage applications they own. Application administrators contain privileges for schedules and rules and can use existing cloud accounts.

Application user (px-backup.app.user): The application user can back up and restore applications but cannot create a schedule policy or rules.

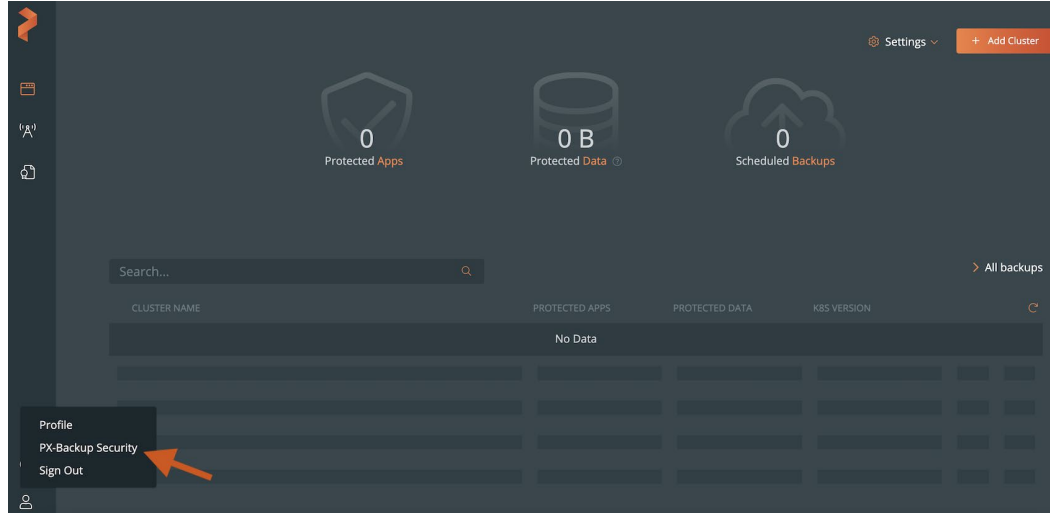
NOTE: Portworx Backup does not allow editing the built-in roles, but you can duplicate them.

If an Infrastructure administrator removes certain role permissions from a user, then the user is automatically assigned with the updated permissions. Thereafter, Portworx Backup restricts any actions (deleting, for example) on the objects created by the user using the old permissions.

Access Portworx Backup Security

Perform the following steps to access Portworx Backup Security:

1. Log into Portworx Backup using the infrastructure administrator credentials.
2. Select the Portworx Backup Security option from the profile menu in the lower left-hand corner of the Portworx Backup page.



Portworx Backup Security includes:

- **Role mapping:** Displays all existing OIDC users (when you integrate Portworx Backup with an external OIDC) and new users added by the infrastructure administrator using Portworx Backup Keycloak
- **Roles:** Displays the three built-in roles, by default, and new roles added by the infrastructure administrator

Add Users and User Groups

Using the Portworx Backup with Keycloak, you can add users and user groups in Portworx Backup. Additionally, when you integrate your OIDC provider into the Portworx Backup, all users and user groups that exist in your OIDC are added to the

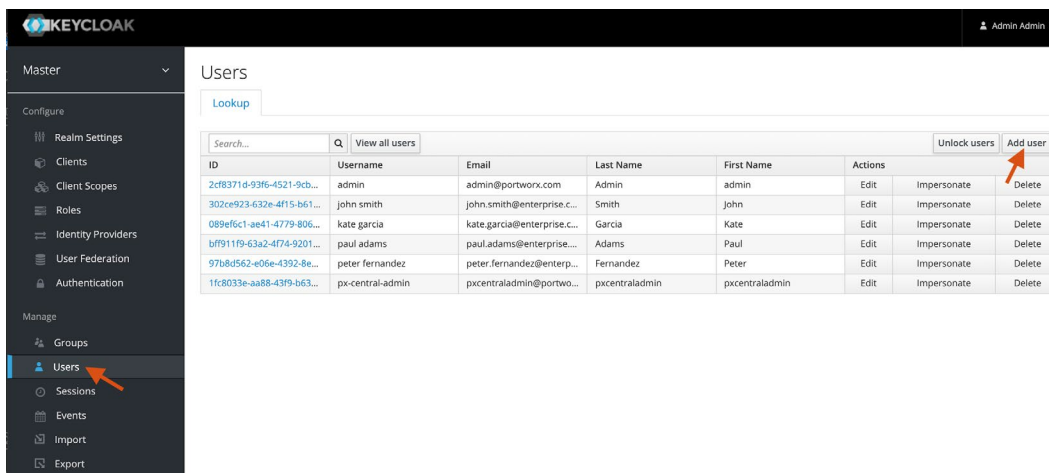
Portworx Backup. In this example, we will utilize Keycloak to create local users and groups. See note below regarding integration of Portworx Backup with an OIDC provider.

NOTE: If you are integrated with OIDC, then you do not need to create users and groups in Portworx Backup Keycloak.

Do not delete users and user groups that you added in the Keycloak.

To add users using Portworx Backup with Keycloak:

1. Login to Portworx Backup with Keycloak using the infrastructure administrator credentials.
2. In the Keycloak home page, select **Administration Console**.
3. From the **Master -> Manage** menu, select **Users**.
4. In the Users page that appears, click **Add User**:



In the Add Users page that appears, enter the Username and other user details.

5. Click **Save**.

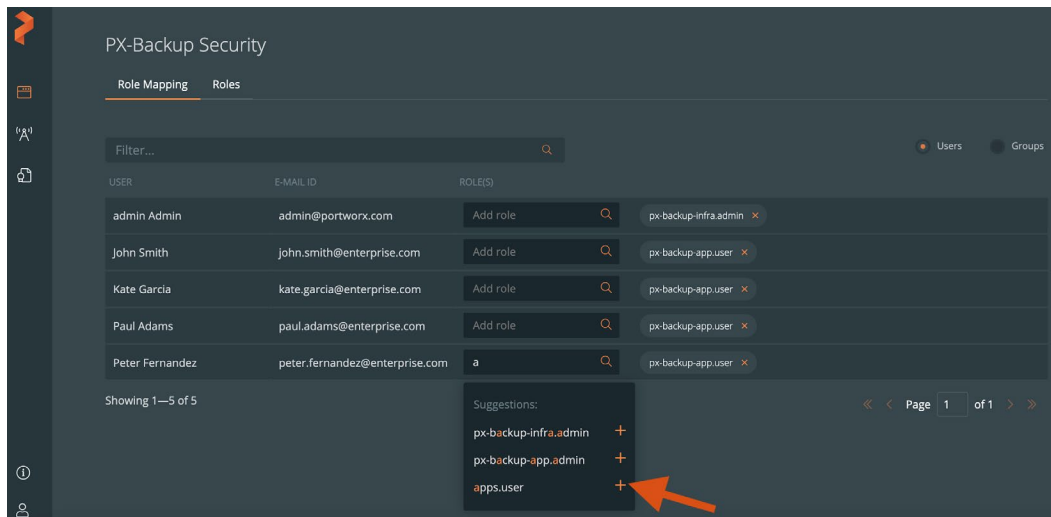
Map Roles to Users

Portworx Backup allows you to map any existing role to a user or user group. Users and groups would be mapped to existing or new roles for OIDC.

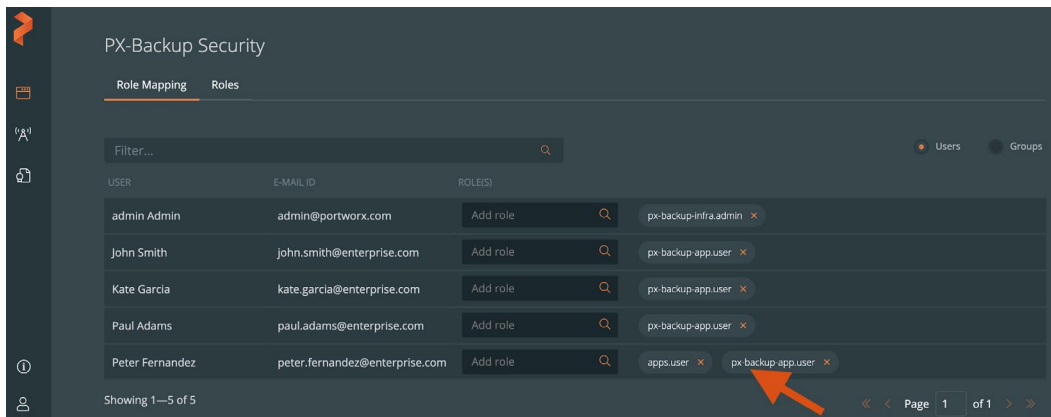
To map roles to users:

1. From the Role Mapping page -> Add role search box on a user row, type any character of an existing role that you want to map to a user. The Add role search box displays all existing roles with the character you typed.

- Click the + icon that appears along with a role.



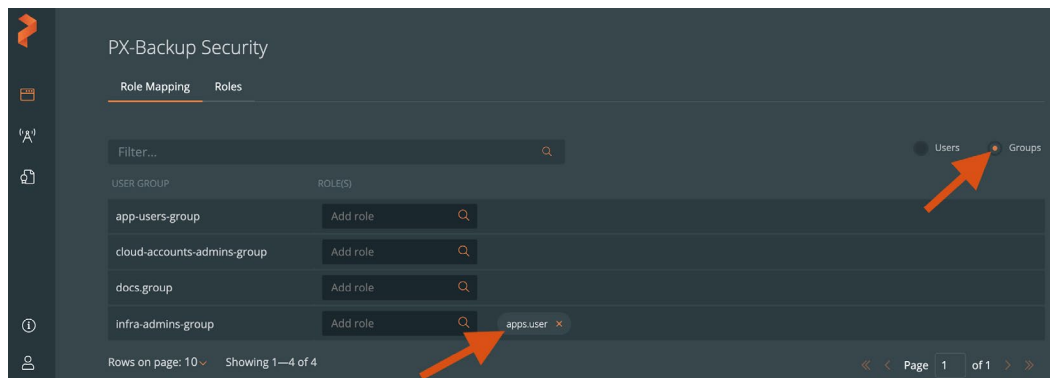
- The selected role is added along with the existing roles of a user:



Map Roles to User Groups

Before you map roles to user groups, choose the **Groups** radio button in the Role Mapping page. You can map any existing role to a user group by following the [Map roles to users](#) procedure.

The following example image displays a role added to a user group:



Auditing

Portworx provides security audit and access logs so organizations can help protect critical data, identify security loopholes, create new security policies, and track the effectiveness of security strategies. We will utilize AWS CloudWatch Container Insights to aggregate the log information in Amazon CloudWatch using Fluent Bit.

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

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

1. Enable AWS Cloudwatch CloudInsights (Fluent Bit). In this example the region is set to “us-east-1”:

```
ClusterName=$(eksctl get cluster -n <Cluster-Name> -o json | jq -r '.[].Name')
RegionName='us-east-1'
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off' || FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' || FluentBitHttpServer='On'
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/quickstart/cwagent-fluent-bit-quickstart.yaml | sed
's/{{cluster_name}}/'${ClusterName}';s/{{region_name}}/'${RegionName}';s/{{http_server_toggle}}/'${
FluentBitHttpServer}';s/{{http_server_port}}/'${FluentBitHttpPort}';s/{{read_from_head}}/'${
FluentBitReadFromHead}';s/{{read_from_tail}}/'${FluentBitReadFromTail}'/' | kubectl apply -f -
```

2. Enable Portworx logging by editing the fluent-bit-config configMap. There are multiple sections to be modified.

```
kubectl -n amazon-cloudwatch edit cm fluent-bit-config
```

3. Add a new @include to the fluent-bit.conf section in the fluent-bit-config configMap for a new portworx-log.conf file:

```
@INCLUDE portworx-log.conf\n
```

4. Create a new configuration file section, portworx-log.conf, in the fluent-bit-config configMap:

```
portworx-log.conf: |
  [INPUT]
    Name          systemd
    Tag           portworx.systemd.*
    systemd_filter _SYSTEMD_UNIT=portworx-output.service
    #Parser       syslog
    DB            /var/fluent-bit/state/flb_portworx.db
    Path          /var/log/journal
```

```

    Read_From_Tail      ${READ_FROM_TAIL}
    Strip_Underscores   true

[INPUT]
    Name                tail
    Tag                 portworx.tail.*
    Path                /var/lib/osd/log/security/openstorage-access*,
/var/lib/osd/log/security/openstorage-audit*
    Parser              logfmt
    DB                  /var/fluent-bit/state/flb_portworx_tail.db
    Mem_Buf_Limit       50MB
    Skip_Long_Lines     0n
    Refresh_Interval    10
    Rotate_Wait         30
    storage.type        filesystem
    Read_from_Head      ${READ_FROM_HEAD}

[FILTER]
    Name                modify
    Match               portworx.systemd.*
    Rename              _HOSTNAME             hostname
    Rename              _SYSTEMD_UNIT         systemd_unit
    Rename              MESSAGE               message
    Remove_regex        ^((?!hostname|systemd_unit|message).)*$

[FILTER]
    Name                aws
    Match               portworx.*
    imds_version        v1

[OUTPUT]
    Name                cloudwatch_logs
    Match               portworx.*
    region              ${AWS_REGION}
    log_group_name      /aws/containerinsights/${CLUSTER_NAME}/portworx
    log_stream_prefix   ${HOST_NAME}-
    auto_create_group   true
    extra_user_agent    container-insights

```

5. Add a new parser to the parsers.conf section in the fluent-bit-config configMap:

[PARSER]

```
Name      logfmt
Format    logfmt
```

6. Add /var/lib to fluentbit daemonSet:

```
volumeMounts:
- mountPath: /var/lib
  name: varlib
  readOnly: true
volumes:
- hostPath:
    path: /var/lib
    type: ""
  name: varlib
```

7. Verify that a new log group has been created in AWS CloudWatch, /aws/containerinsights/<clustername>/portworx.

Log groups (31)

By default, we only load up to 10000 log groups.

Search: 5 matches ☐ Exact match

<input type="checkbox"/>	Log group	Retention	Metric filters	Contributor Insights
<input type="checkbox"/>	/aws/containerinsights/[REDACTED]/application	Never expire	-	-
<input type="checkbox"/>	/aws/containerinsights/[REDACTED]/dataplane	Never expire	-	-
<input type="checkbox"/>	/aws/containerinsights/[REDACTED]/host	Never expire	-	-
<input type="checkbox"/>	/aws/containerinsights/[REDACTED]/performance	Never expire	-	-
<input type="checkbox"/>	/aws/containerinsights/[REDACTED]/portworx	Never expire	-	-

8. Below, you can see audit information for a create volume:

Search: create Clear 1m 30m 1h 12h Custom ☐

▼	Timestamp	Message
▼	2022-09-01T13:35:46.078-04:00	<pre>{ "time": "2022-09-01T17:35:46Z", "level": "info", "msg": "Volume 1149252364528486492 created", "email": null, "groups": [], "method": "volume.create", "name": null, "roles": ["system.guest"], "subject": null, "username": null, "az": "us-east-2a", "ec2_instance_id": "i-0746584b504f59599" }</pre> <div>Copy</div>
▼	2022-09-01T13:35:46.079-04:00	<pre>{ "time": "2022-09-01T17:35:46Z", "level": "info", "msg": "Authorized", "email": null, "groups": [], "method": "volume.create", "name": null, "roles": ["system.guest"], "subject": null, "username": null, "az": "us-east-2a", "ec2_instance_id": "i-0746584b504f59599" }</pre> <div>Copy</div>

Encryption with Portworx Enterprise

Portworx can integrate with AWS KMS to generate and use KMS data keys. This document will show you how to spin up a Portworx cluster that is connected to an AWS KMS endpoint. The data keys created in KMS can be used to encrypt Portworx volumes.

Configuring AWS KMS with Portworx

There are two ways you can set up Portworx so that it gets authenticated with AWS:

- Using AWS Environment Variable
- Using AWS EC2 Role Credentials

Using AWS Environment Variables

Following are the authentication details required by Portworx to use the AWS KMS service:

- AWS Access Key [AWS_ACCESS_KEY_ID] [required]
AWS Access Key ID of the account which has permissions to access KMS APIs
- AWS Secret Key [AWS_SECRET_ACCESS_KEY] [required]
AWS Secret Access Key of the account which has permissions to access KMS APIs
- AWS Secret Token Key [AWS_SECRET_TOKEN_KEY] [optional]
AWS Secret Token Key (if configured) of the account which has permissions to access KMS APIs
- AWS KMS key [AWS_CMK] [required]: The CMK can be found from AWS's resource ARN. Here is an example ARN for CMK: `arn:aws:kms:us-east-1:<aws-id>:key/<cmk-id>`. It specifies that the ARN is for the kms service for us-east-1 region. The trailing ID at the end of ARN is the actual CMK that needs to be provided to Portworx through the AWS_CMK field.
- AWS Region of the CMK [AWS_REGION] [required]: This is the AWS region to which the CMK is associated. CMKs are region specific and cannot be used across regions.

Generate the spec at PX-Central and specify the following two things:

1. Pass in all the above variables as-is in the Environment Variables section.
2. Specify the Secret Store Type in the Advanced Settings section as aws.

Example of env Section:

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
spec:
  env:
    - name: "AWS_ACCESS_KEY_ID"
      value: "<AWS-ACCESS-KEY>"
    - name: "AWS_SECRET_ACCESS_KEY"
      value: "<AWS-SECRET-ACCESS-KEY>"
    - name: "AWS_CMK"
      value: "arn:aws:kms:us-east-1:<AWS-ID>:key/<CMK-ID>"
    - name: "AWS_REGION"
```

```
value: "<AWS-REGION>"
```

Using AWS EC2 Role Credentials

Portworx can authenticate with AWS using AWS SDK's EC2RoleCredentials Provider. Follow these instructions to create an EC2 role. Make sure you provide the following access to KMS in your policy associated with the EC2 role.

Here is a sample AWS Policy that gives access to KMS:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1490047200000",
      "Effect": "Allow",
      "Action": [
        "kms:*"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:<aws-id>:key/<cmk-id>"
      ]
    }
  ]
}
```

Apply the EC2 role to all the AWS instances where Portworx will be running.

Along with the EC2 role, you will still need to provide AWS_CMK and AWS_REGION either through config.json or as environment variables. To provide them through config.json, add the following section to the config.json on all the nodes:

```
cat /etc/pwx/config.json
{
  "clusterid": "<cluster-id>",
  "secret": {
    "secret_type": "aws-kms",
    "aws": {
      "AWS_CMK": "your-aws-kms-key-id",
      "AWS_REGION": "you-aws-region-to-which-this-cmk-belongs"
    }
  }
}
```

```
    ...
}
```

Portworx Encrypted Volumes

Portworx has two different kinds of encrypted volumes:

- **Encrypted Volumes:** Encrypted volumes are regular volumes that can be accessed from only one node.
- **Encrypted Sharedv4 Volumes:** An encrypted Sharedv4 volume allows access to the same encrypted volume from multiple nodes.

The following steps will show how to encrypt a volume and validate it.

1. Create a storageClass with the **secure:** parameter set to true.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-secure-sc
provisioner: pxd.portworx.com
parameters:
  secure: "true"
  repl: "2"
```

2. Create a persistentVolumeClaim:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mysql-data
spec:
  storageClassName: px-secure-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

3. Validate that the persistent volume is encrypted and mounted via dm-crypt dev mapper:

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

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

```
kubectl exec -it $PXPOD -n kube-system -- /opt/pwx/bin/pxctl volume inspect <Volume ID>
```

Volume	: 710859371731701068
Name	: pvc-b5b86482-b383-40f3-94f4-8afe6dd6019e
Size	: 2 GiB
Format	: ext4
HA	: 2
IO Priority	: LOW
Creation time	: Aug 11 18:39:56 UTC 2022
Shared	: no
Status	: up
State	: Attached: 58a84ad6-3da5-41ce-89e6-bcf496585fff (192.168.9.103)
Last Attached	: Aug 11 18:40:07 UTC 2022
Attributes	: encrypted
Device Path	: /dev/mapper/pxd-enc710859371731701068
Labels	: namespace=default,pvc=postgres-data,rep1=2,secure=true
Mount Options	: discard
Reads	: 95
Reads MS	: 100
Bytes Read	: 1409024
Writes	: 1463
Writes MS	: 2967
Bytes Written	: 210259968
IOs in progress	: 0
Bytes used	: 267 MiB
Replica sets on nodes:	
Set 0	
Node	: 192.168.80.253 (Pool d0191f3d-4cea-407d-b18d-6087479dcb89)
Node	: 192.168.9.103 (Pool 077c13d6-1d59-4252-9f76-5755a8c99be6)
Replication Status	: Up
Volume consumers	:
- Name	: postgres-7957478b7d-rhdpq (aeadafa4-12a5-444c-afc0-5a8ab01d491a) (Pod)
Namespace	: default
Running on	: ip-192-168-9-103.ec2.internal
Controlled by	: postgres-7957478b7d (ReplicaSet)

Encryption with Portworx Backup

When backing up your applications, Portworx Backup admins can provide an “Encryption Key” to a Portworx Backup location so that data is sent encrypted in transit.



The example using Amazon S3 as a backup target shows the in-transit data when encrypted and unencrypted.

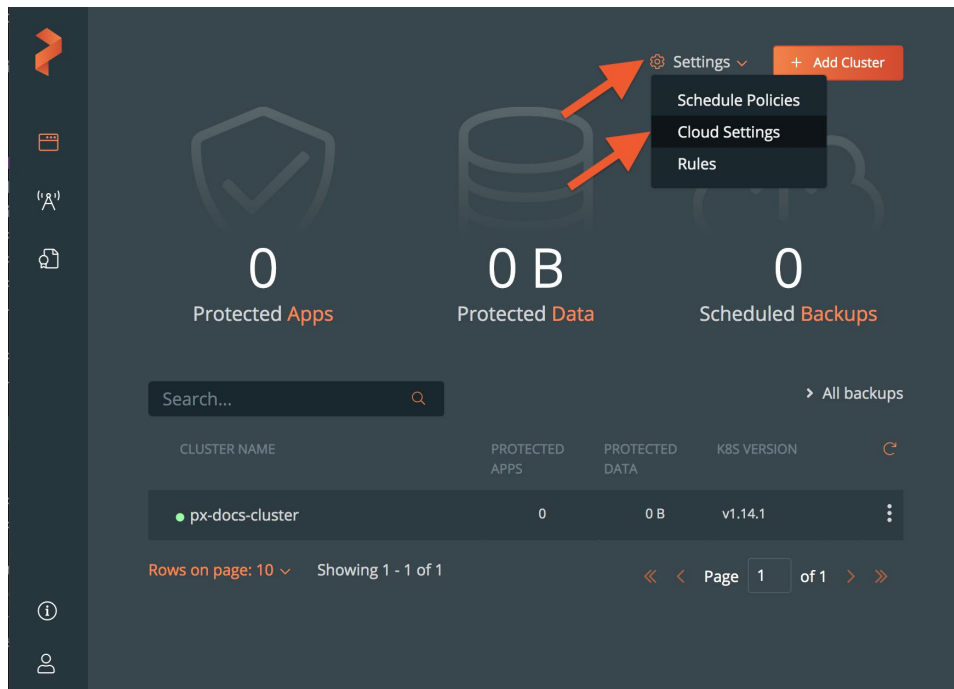
Encrypted

```
20:57:12.705491 IP ip-10-0-123-45.us-east-2.compute.internal.42328 > s3.us-east-2.amazonaws.com.https: Flags [P.], seq 1930:2511, ack 2829, win 1464, length 581
 0x0000: 4500 026d 423e 4000 3f06 8440 0a00 df57 E..mB>a.?.a...W
 0x0010: 34db 54da a558 01bb b978 b1b0 6872 bd02 4.T..X...x..hr..
 0x0020: 5018 05b8 756c 0000 1703 0302 4000 0000 P...ul.....a...
 0x0030: 0000 0000 2aad 9c1a e84a ee27 d1ec c624 ....*....J.'...$
 0x0040: 023a da24 b206 36d8 3954 adec b894 a729 ...$.6.9T.....)
 0x0050: b7fe 731c 3f7f 1981 bb58 4fbe 1f11 2226 ..s.?....X0..."&
```

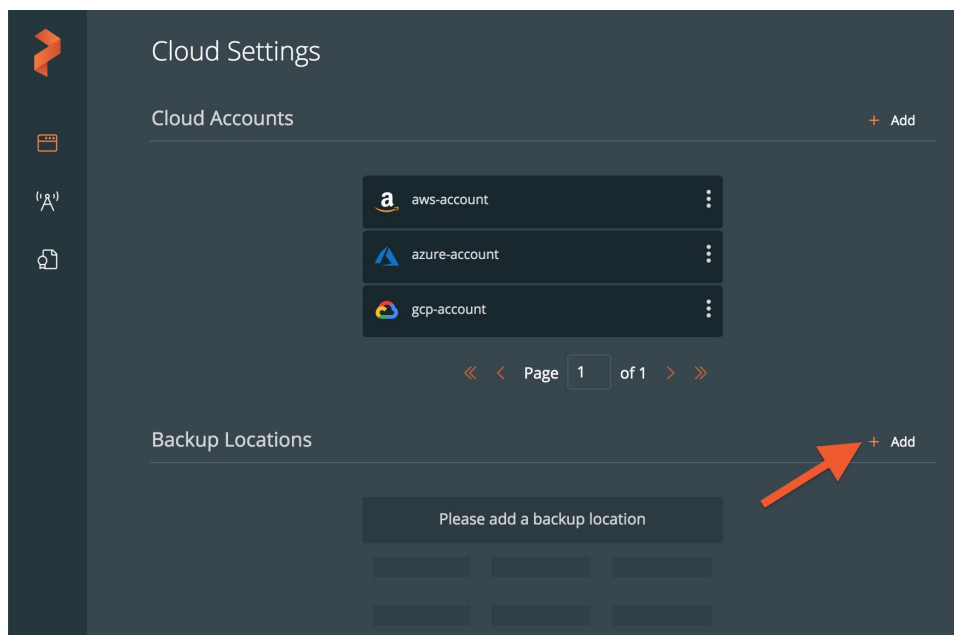
Unencrypted

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

1. From the home page, select **Settings**, then Cloud **Settings** to open the cloud settings page:



2. In the **Backup Locations** section, select **Add**:



3. Populate the following fields:

AWS users:

- **Name:** Specify the name the backup location will appear as in Portworx Backup.
- **Cloud Account:** Choose the AWS credentials this backup location will use to create backups.
- **Path / Bucket:** Specify the path of the bucket or the name of the bucket that this backup location will place backups into.

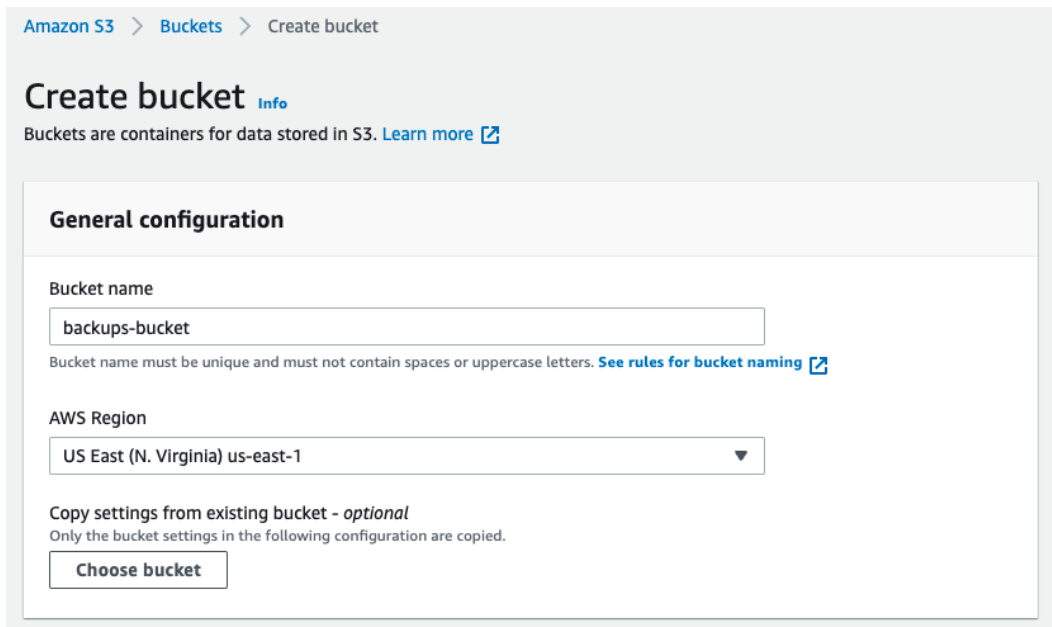
- **Encryption key** (*Optional*): Enter the optional encryption key to encrypt your backups in-transit.
- **Region**: Specify the name of your AWS region.
- **Endpoint**: Specify the URL of your cloud storage server or provider.
- **Disable SSL**: Select this option if your on-prem S3-compliant object store does not support SSL/TLS.
- **Storage class**: Choose the S3 storage class your cloud backups will use.

Ransomware Protection with Portworx Backup

Portworx Backup serves as a vital way for organizations to protect their Kubernetes-based applications and data against malicious malware, such as ransomware, by making use of the immutability achieved by the object lock write once, read many (WORM) model.

Create a Backup Location Bucket with Object Lock

1. First, create a new bucket with a name of your choosing:



Amazon S3 > Buckets > Create bucket

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

backups-bucket

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

US East (N. Virginia) us-east-1

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.

Choose bucket

- Next, scroll down to Advanced Settings and enable object lock. Then create your bucket. You will be notified that the bucket object lock needs additional configuration:

▼ Advanced settings

Object Lock

Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. [Learn more](#)

☐ Disable

☒ Enable

Permanently allows objects in this bucket to be locked. Additional Object Lock configuration is required in bucket details after bucket creation to protect objects in this bucket from being deleted or overwritten.

- The additional configuration for retention can be found by selecting your bucket, navigating to the Properties tab, and scrolling down to the object lock section. Click **Edit** on object lock. From here, you can select your default retention mode and what your retention period is.

Object Lock

Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. [Learn more](#)

Once Amazon S3 Object Lock is enabled, you can't disable Object Lock or suspend Bucket Versioning for the bucket.

Object Lock

Enabled

Default retention

Automatically protect new objects put into this bucket from being deleted or overwritten.

☐ Disable

☒ Enable

Default retention mode

☒ Governance

Users with specific IAM permissions can overwrite or delete protected object versions during the retention period.

☐ Compliance

No users can overwrite or delete protected object versions during the retention period.

Default retention period

Days ▼

Must be a positive whole number.

- The deletion of versioned objects will add only the delete marker to object-locked stored files, and the actual object version will become a non-current version. Users can invoke automation or manually delete these objects. With AWS, users can automatically clean up these delete markers and the non-current versions of the objects using a lifecycle rule within the lifecycle configuration. Navigate to Lifecycle configuration, then to Create lifecycle rule, and give the rule a name.

Amazon S3 > Buckets > backups-bucket-123 > Lifecycle configuration > Create lifecycle rule

Create lifecycle rule

Lifecycle rule configuration

Lifecycle rule name

Up to 255 characters

5. Select the two options below in Lifecycle rule actions and the one option in Delete expired object delete markers.

Lifecycle rule actions

Choose the actions you want this rule to perform. Per-request fees apply. [Learn more](#) or see [Amazon S3 pricing](#)

- ☐ Move current versions of objects between storage classes
- ☐ Move noncurrent versions of objects between storage classes
- ☐ Expire current versions of objects
- ☒ Permanently delete noncurrent versions of objects
- ☒ Delete expired object delete markers or incomplete multipart uploads

These actions are not supported when filtering by object tags or object size.

Delete expired object delete markers or incomplete multipart uploads

Expired object delete markers
This action will remove expired object delete markers and may improve performance. An expired object delete marker is removed if all noncurrent versions of an object expire after deleting a versioned object. This action is not available when "Expire current versions of objects" is selected. [Learn more](#)

- ☒ Delete expired object delete markers

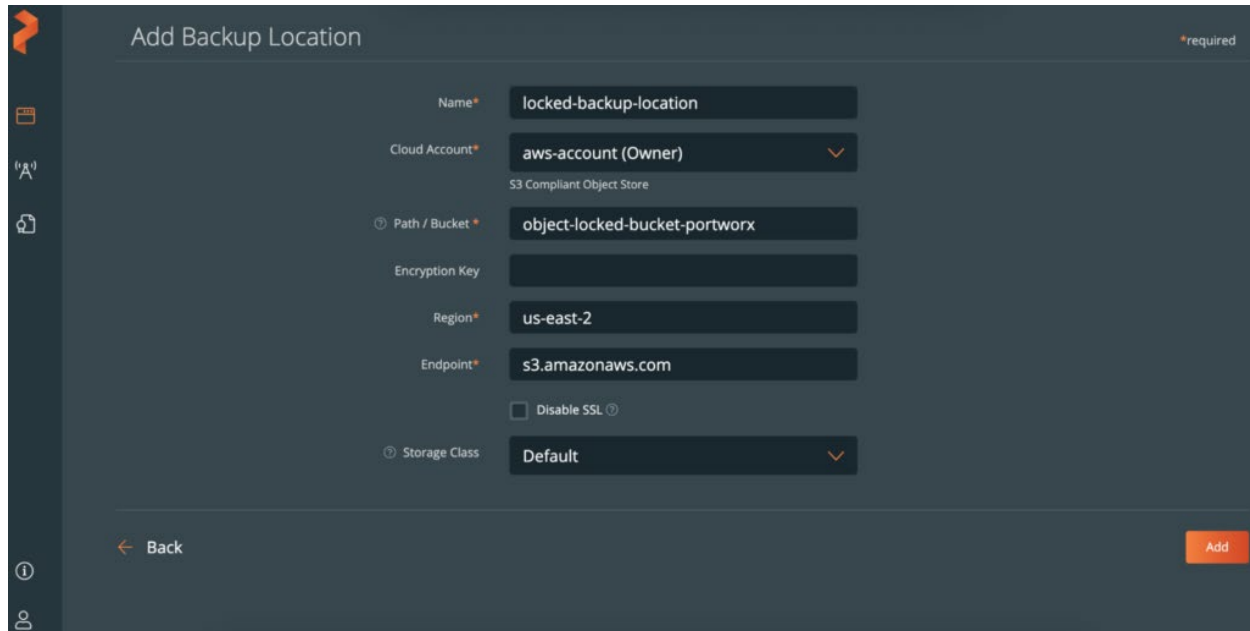
Incomplete multipart uploads
This action will stop all incomplete multipart uploads, and the parts associated with the multipart upload will be deleted. [Learn more](#)

- ☐ Delete incomplete multipart uploads

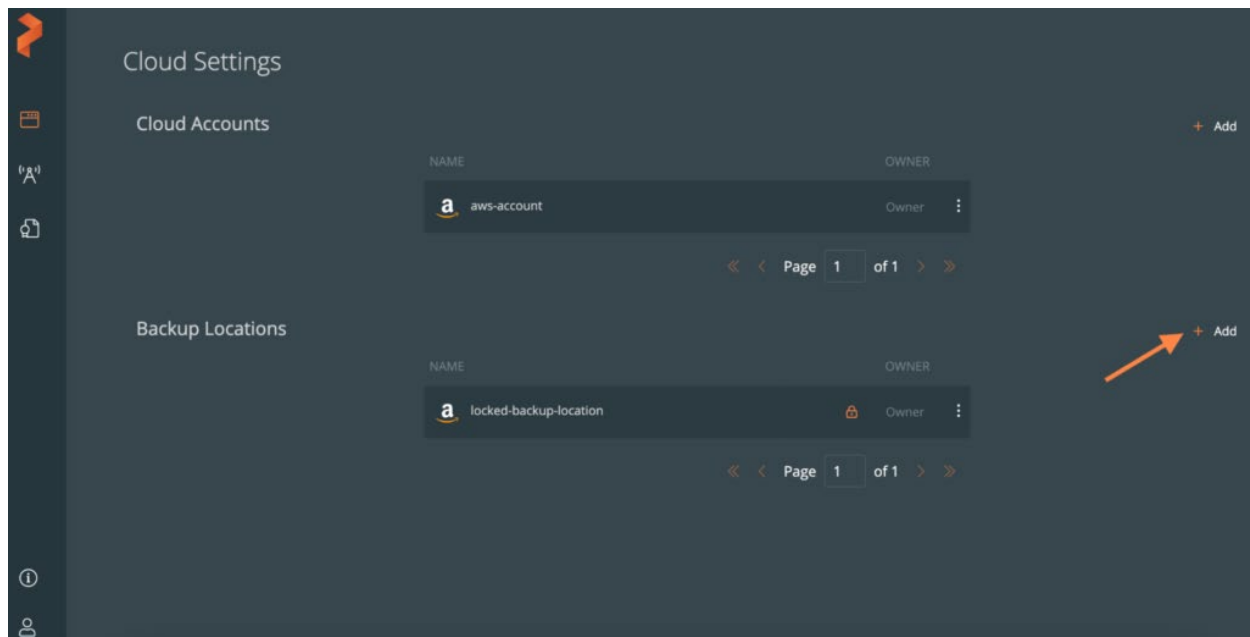
You are now ready to tell Portworx Backup to use this bucket for ransomware protection.

Configure the Backup Location in Portworx Backup

1. When you are creating a backup location for ransomware protection, the Path/Bucket field needs to be a previously created bucket that has object lock enabled.



2. When creating a ransomware protected backup location, you will see the lock symbol in the backup location row that signifies this is an object-locked location.



NAME	OWNER
aws-account	Owner

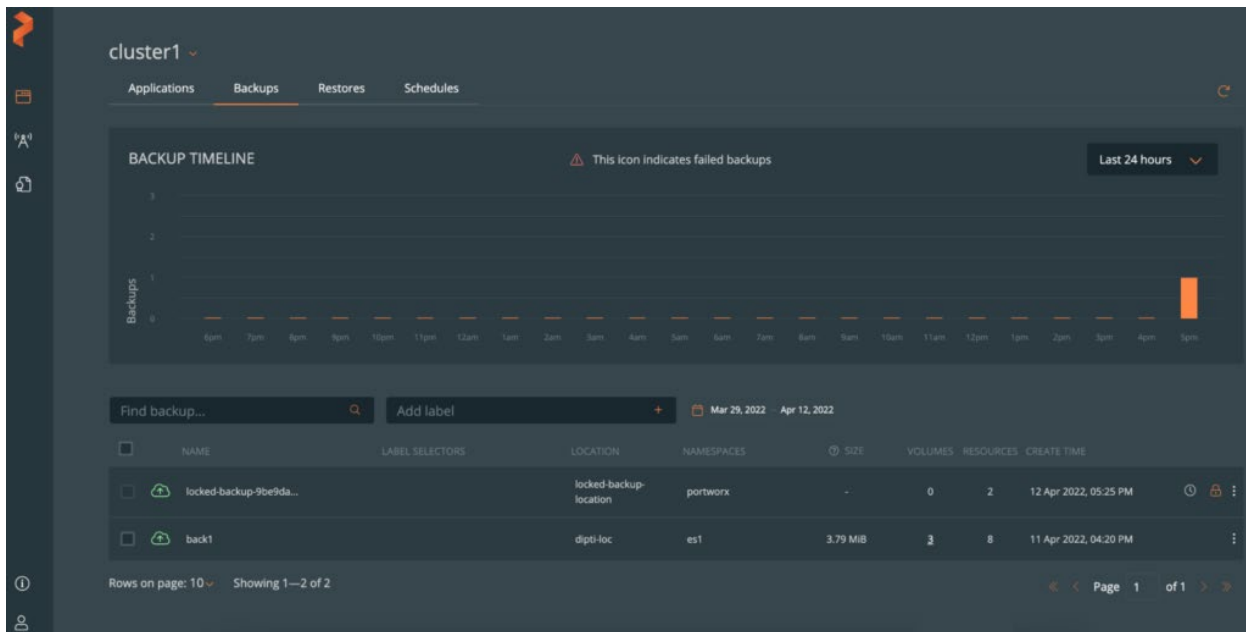
Page 1 of 1

NAME	OWNER
locked-backup-location	Owner

Page 1 of 1

Take an Ad-Hoc That Is Ransomware Protected

To create a manual backup, create a backup as you normally would using Portworx Backup. When you start the backup, you should see a lock icon, indicating that it is a ransomware-protected manual backup.

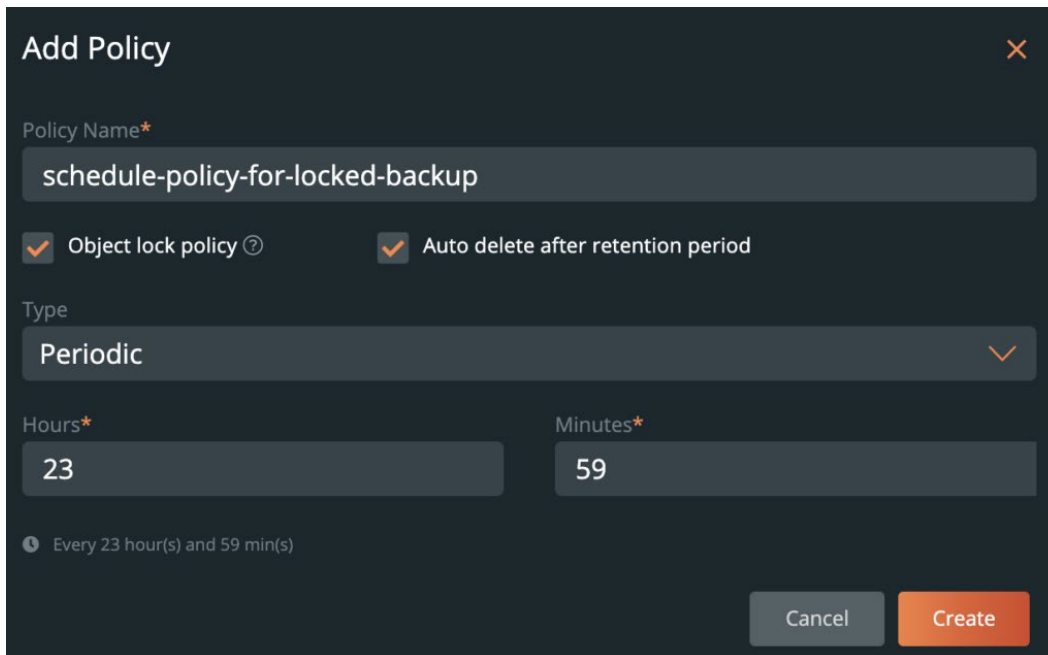


Create a Schedule Policy That Is Ransomware Protected

When adding a new policy, select the **Object lock policy box** underneath the policy name. This will remove the manual incremental and retain selection from the policy, as this will default to what the object lock is configured to on the bucket. After this, scheduled backups can be triggered using the ransomware protected schedule policy.

Users should be aware of the following points when using object-locked policies:

- Users can opt for auto-deletion. If they choose not to, deleting from the backup UI will be the user's responsibility.
- Users can set only Periodic or Daily backups. Yearly or Monthly backups are not supported yet in the case of object lock-enabled scheduled backup.



Add Policy

Policy Name*

schedule-policy-for-locked-backup

☒ Object lock policy ☒ Auto delete after retention period

Type

Periodic

Hours*

23

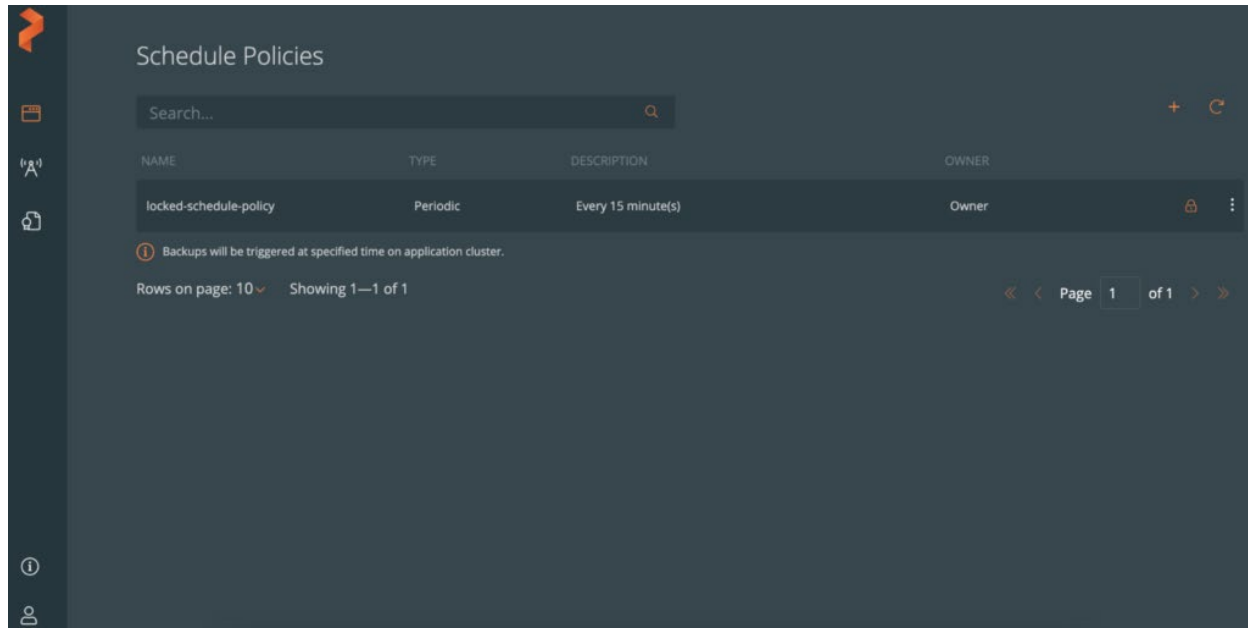
Minutes*

59

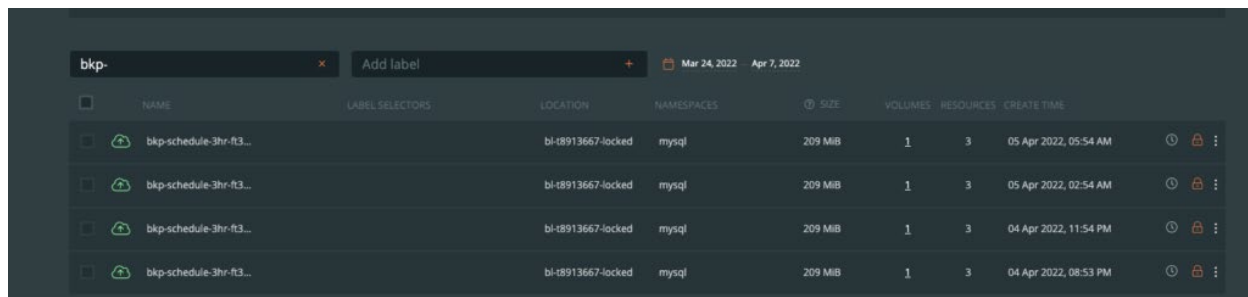
Every 23 hour(s) and 59 min(s)

Cancel Create

A lock icon will also appear adjacent to the new policy, as shown below.



In case of a successfully created scheduled backup using ransomware-based schedule policies, a lock icon will appear exactly as it does for manual backups. The below image shows backups taken based on a three-hour schedule policy.



NOTE: With all manual backups, they are full backups by nature of the ad-hoc request. For scheduled backups using ransomware protection, Portworx Backup will take full backups after five incremental backups.

Conclusion

Portworx provides the best-in-class enterprise-grade data services for any application running on Amazon EKS at any scale. Delivering speed, density, and scale, Portworx not only enables efficient, automatic provisioning on top of your Amazon EKS clusters, but 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 also provides customers with a secure data management solution with PX-Secure and Portworx Backup. With PX-Secure encryption and access controls, you can move securely at the speed of Kubernetes. PX-Secure includes cluster-wide encryption, container-granular or storage-class-based bring your own key (BYOK) encryption, role-based access control, and integration with active directory or LDAP. Portworx Backup brings role-based access control, encryption in flight, and ransomware protection.



About the Author

Matt LeVan is a Senior Solutions Engineer for the Cloud Native Business Unit at Pure Storage. He is responsible for designing and architecting solutions around Portworx Enterprise, backup, and disaster recovery for Kubernetes. Matt has worked in the data management ecosystem for the past 20 years, focusing on building solutions around Storage Infrastructure, Cloud, and Kubernetes. Matt joined Pure Storage in July 2022.

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

800.379.PURE

