Portworx on Red Hat OpenShift Bare Metal

A validated architecture and design model to deploy Portworx® on Red Hat OpenShift running on bare metal hosts.

Authors: Sanjay Naikwadi, Vijay Bhaskar Kulari, Chris Crow, Ryan Wallner

Table of Contents

Table of Figures

Table of Tables

Executive Summary

Modern applications are built using containers and orchestrated by Kubernetes, but they still need a layer of persistence. Red Hat OpenShift, the industry's leading hybrid cloud application platform powered by Kubernetes, brings together tested and trusted services to reduce the friction of developing, modernizing, deploying, running, and managing applications. Red Hat OpenShift delivers a consistent experience across public cloud, on-premises, hybrid cloud, and edge architecture. To run stateful applications on Red Hat OpenShift, organizations need a robust data services platform like Portworx® by Everpure®. Portworx provides features like replication, high availability, security and encryption, capacity management, disaster recovery, and data protection to Red Hat OpenShift deployments. Instead of spending resources architecting and managing a custom Kubernetes storage layer, organizations can accelerate their modernization journeys by adopting a solution like Red Hat OpenShift with Portworx on bare metal servers.

About This Document

This Portworx reference architecture contains a validated architecture and design model to deploy Portworx on Red Hat OpenShift running on bare metal hosts. The document is intended for Kubernetes administrators and cloud architects who are familiar with Portworx. The audience must be familiar with basic Red Hat OpenShift concepts, like MachineSets, and hardware concepts, like disk and network configurations. The document has three main technical areas:

- **Planning and architecture overview**: This section presents the high-level architecture overview on how Portworx will be deployed on Red Hat OpenShift. It discusses the requirements related to OpenShift MachineSets for storage and storageless nodes and also physical disk and network configuration recommendations.
- **Design considerations**: This section provides more detailed requirements and recommendations that must be considered during the design phase. It covers Red Hat OpenShift requirements, networking, capacity planning, high availability, resource, and performance considerations, security, and monitoring.
- **Operations considerations**: This section covers operational tasks such as installation methods, upgrades, data protection, scaling, logging, and monitoring considerations. It also discusses considerations for running stateful containerized applications in a production environment.

Value Proposition

Benefits of Red Hat OpenShift

Red Hat OpenShift is a modern application development and deployment platform. It delivers a consistent, scalable, and secure platform for both virtualized and containerized applications from edge to core to cloud environments. It is designed to allow applications and the data centers that support them to scale from small form factor deployments at the edge to very large clusters consisting of thousands of nodes serving applications to millions of clients. Red Hat OpenShift is often deployed in cloud and virtualized environments but can also be deployed on bare metal infrastructure which allows clusters to support additional features like Red Hat OpenShift Virtualization for running VMs alongside containers on a single cloud-native platform. These bare metal deployments share many of the features of other Red Hat OpenShift environments including ease of deployment and rapid scaling to meet customer needs. In addition, by deploying on bare metal, you avoid the additional overhead necessary to manage a host hypervisor or cloud environment.

Key benefits of Red Hat OpenShift include:

- **Trusted DevSecOps platform**: Red Hat engineers and tests the underlying Kubernetes engine along with additional product features to deliver a complete hardened, tested and trusted platform. It is delivered upon Red Hat Enterprise Linux, a secure foundation with a comprehensive set of security capabilities to protect applications. Red Hat also offers Advanced Cluster Security for Kubernetes, providing full application lifecycle security.
- **A comprehensive application platform**: Red Hat OpenShift delivers a zero-configuration development platform that increases productivity and innovation. As a unified platform, Red Hat OpenShift accelerates your application modernization strategy with a single application platform to rehost, re-platform, or refactor existing monolithic or n-tier applications alongside the development of new cloud-native applications. With extensions to the edge, and the ability to build and manage intelligent applications with Red Hat OpenShift AI, organizations can realize benefits faster and across the hybrid cloud.
- **A consistent platform experience:** From on-premises to the cloud, or at the edge Red Hat OpenShift delivers a consistent experience across all environments. By standardizing on Red Hat OpenShift for both containerized workloads and virtual machines, organizations can reduce operational complexity, while also streamlining processes.

Benefits of Portworx

Portworx cloud-native storage offers a transformative solution for managing stateful applications in Kubernetes environments, far surpassing traditional hardware storage solutions and their basic CSI (container storage interface) drivers. While CSI connectors often fall short in handling creates, updates, and deletes (CRUD) at the scale provided by Kubernetes and containers, Portworx excels in delivering enterprise-grade features that enhance both operational efficiency and platform engineering productivity for containerized environments.

Key benefits of Portworx cloud-native storage include:

- **Accelerated time to revenue**: By streamlining storage management and reducing complexities, Portworx helps organizations achieve faster deployment and time to market.
- **Data resiliency**: Portworx ensures high availability and disaster recovery with advanced data replication and backup capabilities, protecting against data loss and downtime.
- **Enterprise scalability**: Designed to scale with your needs, Portworx supports seamless growth, whether you're operating in a single data center or across multiple clouds.
- **Self-Service storage access**: Developers gain the flexibility to provision and manage storage independently through storage classes, empowering them to innovate without waiting for IT intervention.
- **Integrated storage management:** Portworx offers comprehensive management features, including rule-based automation, thin provisioning, and support for multi-cloud, hybrid-cloud, and on-premises environments agnostic to the hardware providers.
- **Boosted platform engineering productivity**: By providing a unified and efficient storage solution, Portworx enhances the productivity of platform engineers, allowing them to focus on building and maintaining robust applications in multiple environments.

With Portworx, organizations can achieve unmatched data agility and reliability, ensuring their Kubernetes storage and databases are always performant and available, regardless of the underlying infrastructure.

Benefits of Portworx on Red Hat OpenShift Bare Metal

As part of their digital transformation efforts, organizations are modernizing their applications and infrastructure by adopting containers and Kubernetes, leveraging Red Hat OpenShift for a robust, enterprise-grade application platform. When deployed on bare metal, Red Hat OpenShift and Portworx together deliver unparalleled performance, security, and scalability for modern applications.

Key benefits of running Portworx and Red Hat OpenShift on bare metal include:

- **High performance**: Bare-metal deployments eliminate the overhead of virtualization, providing direct access to hardware resources. This results in superior performance for both containerized applications and stateful workloads managed by Portworx.
- **Full-stack automated operations**: Red Hat OpenShift offers full-stack automation, streamlining operations from the underlying hardware to the application layer. This automation simplifies management and reduces operational complexity.
- **Consistent experience across environments**: With Red Hat OpenShift, organizations benefit from a consistent application platform that works seamlessly across on-premises, hybrid-cloud, and multi-cloud environments. Portworx provides a similar data platform for running persistent apps across hybrid and multi-cloud environments. This consistency ensures smooth transitions and easier management.
- **Self-service provisioning**: Red Hat OpenShift enables developers to provision resources on-demand, accelerating the development lifecycle. Combined with Portworx, developers can also easily manage storage, enhancing productivity and innovation.
- **Enhanced data management**: Portworx adds a robust, secure, and highly available data management layer to Red Hat OpenShift. This integration ensures that applications have reliable and scalable storage, essential for mission-critical workloads.
- **Scalability and flexibility**: Both Red Hat OpenShift and Portworx are designed to scale effortlessly with your organization's growth. They provide the flexibility needed to handle dynamic workloads and evolving business needs.
- **OpenShift Virtualization**: OpenShift Virtualization allows you to run and manage virtual machines alongside containers. This capability is particularly valuable in bare metal environments, where performance is critical. It enables a seamless transition of legacy applications to modern infrastructure, supporting a hybrid approach and unifying application management.
- **Secure boot:** Portworx signed images now support the secure boot. Enabling secure boot adds an extra layer of security by verifying module authenticity before loading.

By combining the strengths of Red Hat OpenShift and Portworx on bare metal, organizations achieve a powerful and flexible infrastructure that supports their digital transformation goals. This setup ensures high performance, reliability, and scalability, making it an ideal solution for modern enterprise applications.

Target Use Cases

This document offers detailed guidelines and best practices for deploying Portworx on Red Hat OpenShift when utilizing bare metal servers as the underlying infrastructure. By following the instructions and recommendations outlined in this document, Red Hat OpenShift users will be equipped to deploy Portworx in a manner that ensures stability, reliability, and optimal performance.

Once Portworx is deployed according to these guidelines, users will be able to confidently deploy any type of stateful application within their Red Hat OpenShift environment. The robust data management capabilities of Portworx, combined with the powerful orchestration and management features of Red Hat OpenShift, will enable seamless and efficient storage operations for a wide variety of applications.

It is important to note that the scope of this document is focused on providing a general, stable deployment framework for Portworx on Red Hat OpenShift. While it does not delve into specific recommendations for individual applications, the deployment strategy presented here is designed to be universally applicable. This ensures that any application requiring reliable and scalable storage can be supported effectively within the Red Hat OpenShift and Portworx ecosystem.

By adhering to the best practices and guidelines provided, organizations can achieve a highly resilient and performant storage solution that meets the needs of diverse and demanding stateful applications.

Planning and Architecture Overview

In this section, we provide a planning and architecture overview for deploying Red Hat OpenShift with Portworx on bare metal. Our focus is on creating a high-level design that ensures scalability, reliability, security, and performance. We will outline the key architectural components and their interactions, supported by logical diagrams. These visual aids will illustrate the overall structure of the system, enabling a clear understanding of how the various elements work together to support your production workloads. This section of the reference architecture will give you a quick look at the overall architecture before the following sections explain items in greater detail.

Reference Architecture High Level Design

The diagram below shows a high level design of the Portworx reference architecture deployed on Red Hat OpenShift running on bare metal hardware. By implementing this design, a platform engineering team can automate the provisioning of a well defined architecture following best practices that includes high availability, operations management, observability, business continuity, performance, and security.

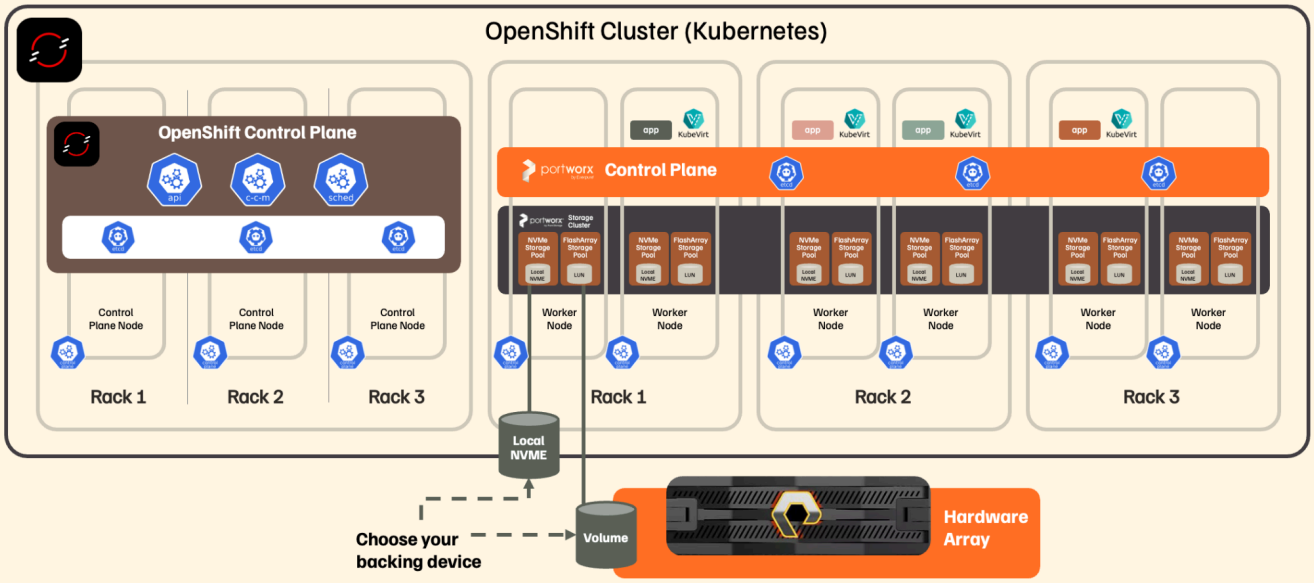

Figure 1. General reference architecture

Portworx Cluster Configuration

Portworx requires a minimum of three storage nodes in the cluster to create a quorum and to prevent a split-brain scenario that can occur with two nodes. Three nodes are also needed to provide three redundant copies of data for persistent volumes. However, for a production environment, this reference architecture recommends starting with six storage nodes. Although a Portworx cluster with three storage nodes may work well in some environments, there are significant advantages to deploying with six storage nodes initially:

There are multiple advantages to using at least six nodes, including:

- Cluster capacity:
  - In a cluster with three storage nodes, losing one node results in losing one-third of its capacity, increasing the load on the remaining two nodes until the lost node recovers, which could be days in the case of hardware failures.
  - In a six-node cluster, losing one node affects only one-sixth of its capacity, allowing the remaining five nodes to distribute the load more evenly, especially across fault domains like a rack.

- I/O load distribution:
  - A three-node cluster may experience more frequent I/O latencies during peak times.
  - A six-node cluster can distribute I/O requests more effectively, reducing the risk of I/O latencies.

Preparing Red Hat OpenShift

Before deploying Portworx, consider the following points to prepare Red Hat OpenShift and physical nodes:

- MachineSet configuration:
  - A MachineSet with six storage nodes is the minimum recommended for production environments (fewer nodes are not recommended unless the load is minimal such as in a development environment).
  - Ensure the Red Hat OpenShift cluster spans multiple availability zones or fault domains and equally distribute storage nodes among them (at least three zones are recommended).
  - Use the label `portworx.io/node-type:storage` on all nodes to enable Portworx to automatically provision storage for new nodes.
    ***Note**: This is only needed if you will also deploy storageless worker nodes which will not participate in a Portworx storage cluster. If all worker nodes will be storage nodes such as in hyper-converged architectures, this label is not needed.*
  - Portworx recommends that each node must have a minimum of eight CPU cores and 32 GB RAM (refer to the Resource Considerations section for more details).

Considerations for Red Hat OpenShift on Bare Metal

When it comes to an architectural footprint that a customer must choose to adopt, they will find that the experience that is provided by Red Hat OpenShift, whether deployed virtualized, in the cloud, or on bare metal resources within a customer's data center are all very similar. Deployments can be automated through the Assisted Installer, or an IPI based install, or fully customized with the agent-based or UPI based installation method. Due to this, the cluster deployment experience is similar no matter what form factor the cluster is deployed in.

One consideration that a customer must take into account is how Red Hat OpenShift is subscribed based on the form factor of the deployment. With a virtualized or hosted deployment of Red Hat OpenShift, subscriptions are based on vCPU core count on the compute nodes where application workloads run. A bare metal deployment differs in that it now becomes a socket-based entitlement with one subscription supporting up to two physical sockets, and up to 64 physical cores per node.

Another major benefit of adopting a bare-metal architecture is that it allows for additional features such as OpenShift Virtualization to be supported. Enabling this feature on a cluster allows it to run both virtual machines and containerized applications side-by-side, in a single cloud-native environment.

Considerations for Portworx on Red Hat OpenShift

Portworx is a comprehensive data platform for Kubernetes and it is integrated with Red Hat OpenShift. All features that Portworx provides to Kubernetes platforms are also available for Red Hat OpenShift clusters, ensuring seamless functionality and enhanced capabilities. Additionally, Portworx offers a plugin specifically designed for OpenShift, which integrates the Portworx Console UI with the OpenShift console.

The Portworx Console provides a detailed overview of the Portworx Storage Cluster, including critical metrics and status information. This integration enriches the OpenShift console by adding panels that display Portworx volume details in the OpenShift console Storage tabs.

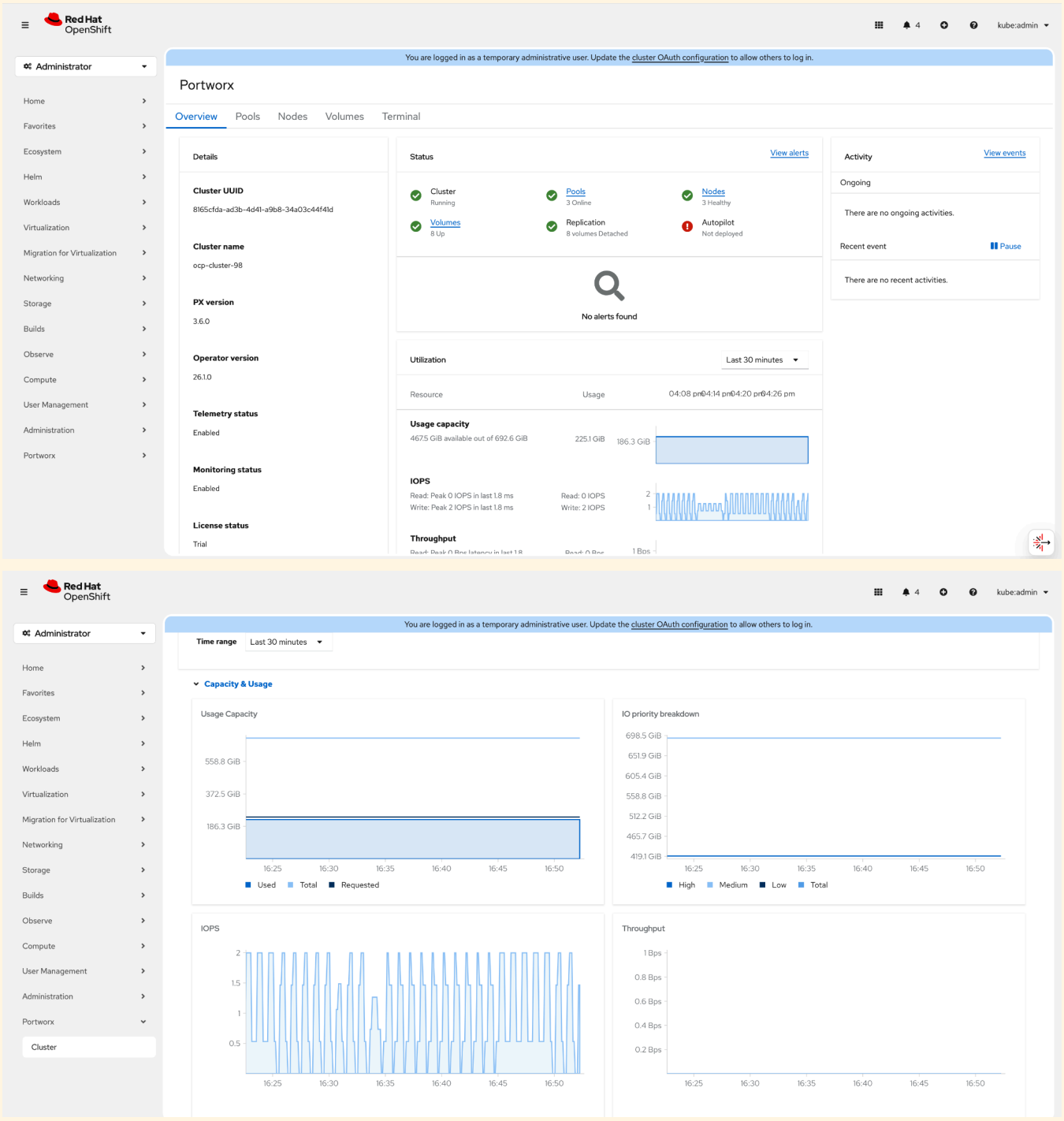

Figure 2. Portworx plugin for Red Hat OpenShift

The **Portworx Console plugin** for Red Hat OpenShift (OCP 4.12 and later) enables **native storage observability directly within the OpenShift web console**. It can be activated during deployment or upgrade of the Portworx Operator, eliminating the need for separate monitoring interfaces.

This integration leverages **OpenShift's built-in Prometheus and monitoring stack**, allowing administrators to visualize **cluster health, storage capacity, performance metrics, and workload-level usage** from a single interface. The plugin provides deep insights into **volumes, nodes, storage pools, and Kubernetes resources**, along with real-time alerts and events for proactive troubleshooting.

Disaggregated vs Hyper-converged Architectures

When designing the storage architecture for your Kubernetes cluster with Portworx, two primary approaches can be considered: disaggregated and hyper-converged architectures. Each approach has its own set of advantages and trade-offs, and the choice between them depends on your specific requirements and constraints.

For bare metal deployments, Portworx strongly recommends adopting a hyper-converged architecture. This approach ensures consistent storage performance and availability while simplifying the deployment process. Unlike cloud environments, bare metal installations cannot easily leverage the elasticity offered by cloud resources, making the benefits of a disaggregated infrastructure less significant. Conversely, a disaggregated architecture is typically preferred in cloud environments where elasticity is crucial, allowing worker nodes to scale in or out based on resource demands.

Disaggregated Architecture

In a disaggregated architecture, a subset of Kubernetes worker nodes is designated to provide storage for the entire cluster. Not all worker nodes participate in storage provisioning; instead, specific nodes are dedicated to handling storage tasks. This segregation allows for a clear distinction between compute and storage resources.

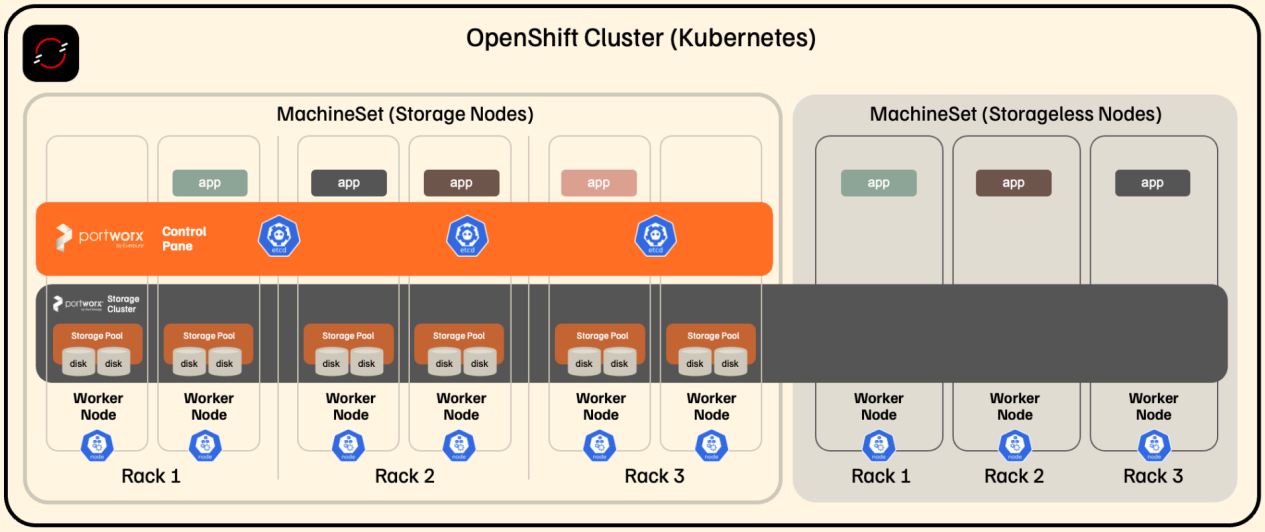

Figure 3. Disaggregated architecture

Key characteristics of a disaggregated architecture include:

- **Dedicated storage nodes**: Only a selected group of nodes are responsible for storage, while the rest focus on compute tasks.
- **Resource Specialization**: Storage nodes can be optimized with hardware specifically suited for storage tasks, such as high-performance NVMe disks and additional RAM, without impacting compute nodes.
- **Isolation:** Faults or performance issues in the compute nodes do not directly affect the storage nodes, and vice versa, providing a layer of operational isolation.

Advantages of a disaggregated architecture include:

- **Optimized resource allocation**: Allows for tuning and optimizing nodes specifically for storage without affecting compute performance.
- **Scaling**: Storageless nodes can be scaled down easily since they do not participate in cluster quorum decisions and Portworx will automatically decommission a scaled down storageless node after 10 minutes.

There are also a few potential disadvantages of using distributed architectures, like:

- **Potential bottlenecks**: As storage responsibilities are limited to fewer nodes, these nodes can become bottlenecks under high I/O workloads. Accessing replicas on a local node provides better performance than going over a network device.
- **Complexity in scaling**: Scaling storage independently of compute resources can add complexity to cluster management and scaling strategies.

Hyper-converged Architecture

In a hyper-converged architecture, every worker node in the Kubernetes cluster participates in providing storage for applications. This means that both compute and storage tasks are handled by the same set of nodes, distributing storage responsibilities across the entire cluster.

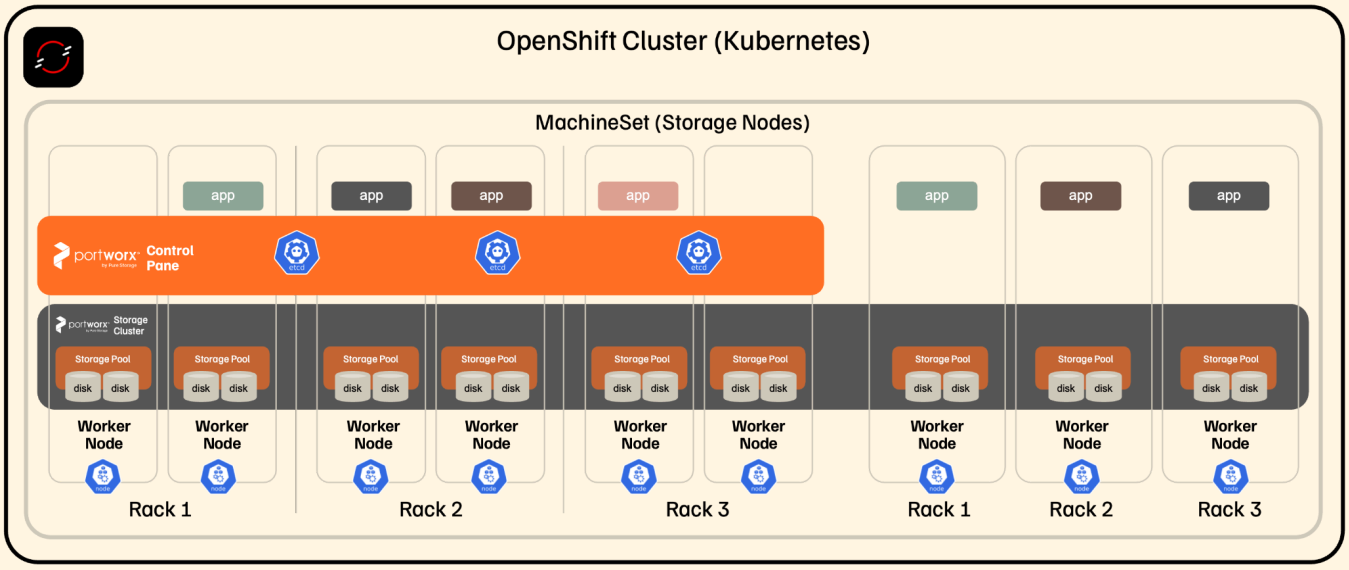

Figure 4. Hyper-converged architecture

Key characteristics of hyper-converged architecture include:

- **Uniform node role**: All worker nodes serve dual roles, providing both compute and storage resources.
- **Balanced resource utilization**: Storage load is evenly distributed across all nodes, preventing any single set of nodes from becoming a bottleneck.
- **Integrated scaling**: Adding more nodes to the cluster increases both compute and storage capacity simultaneously.

Advantages of hyper-converged architecture are:

- **Enhanced scalability**: As you add more nodes to the cluster, you increase both compute and storage capacity, making it easier to scale out your infrastructure.
- **Improved performance**: Distributing storage tasks across all nodes can lead to better I/O performance and lower latency due to the collective resources of the entire cluster. Applications will also be local to their replicas preventing the network from becoming a storage bottleneck.
- **Simplified architecture**: With every node performing the same role, the architecture can be simpler to design and implement.

Disadvantages of using hyper-converged architecture include:

- **Resource utilization**: Compute and storage tasks share the same resources, which means that a portion of each node's CPU and Memory must be dedicated to the storage cluster components.
- **Elasticity**: The Portworx storage cluster holds persistent data for applications which can not be removed easily. If the Red Hat OpenShift cluster is configured to scale in/down during times of inactivity, the storage cluster prevents this from happening due to its requirement to hold onto stateful data.

Design Considerations

Red Hat OpenShift Considerations

A Red Hat OpenShift Container Platform deployment on bare metal requires the following nodes:

Hosts	Description
Bootstrap machine	The cluster requires the bootstrap machine to deploy the OpenShift Container Platform cluster on the three control plane machines. You can remove the bootstrap machine after you install the cluster.
Control plane machines	The control plane machines run the Kubernetes and OpenShift Container Platform services that form the control plane.
Compute machines, also known as worker machines	The workloads requested by OpenShift Container Platform users run on the compute machines. These are also the machines where Portworx Enterprise will be deployed.

Table 1. Red Hat OpenShift node types

***Note**: Although Red Hat OpenShift supports configurations with two worker machines, Portworx requires a minimum of three worker nodes to create quorum for the storage cluster.*

***Note:** Portworx supports two node configurations with Arbiter nodes only. Please refer to Guide.*

Important: To maintain high availability of your cluster, use separate physical hosts for these cluster machines.

Requirements for node operating systems, versions, instruction set requirements, and minimum compute, memory, storage, and IOPS requirements should be reviewed for the version of Red Hat OpenShift you are planning to run. Please refer to Red Hat documentation.

This reference architecture was written based on testing of Portworx Enterprise 3.6.0 with Red Hat OpenShift version 4.20. Please refer to the Portworx support matrix for supported versions of Portworx for Red Hat OpenShift on bare metal implementations.

Networking Considerations

Designing an efficient network is essential for the optimal performance and reliability of your Portworx deployment. This section outlines key considerations and best practices for configuring the network in a Portworx environment running on Red Hat OpenShift. By carefully planning the network architecture, you can ensure seamless communication, high availability, and enhanced security for your storage cluster. The following subsections will cover critical aspects of network design, including the separation of management and data interfaces, as well as important firewall considerations to protect and optimize your infrastructure.

Management and Data Interfaces

When designing the networking configurations for a Portworx deployment, several key considerations must be taken into account to ensure optimal performance, reliability, and scalability. Portworx leverages the bare metal server's physical network interface card (NIC) for serving remote storage connections, as well as for keeping persistent volume replicas in sync across nodes. Special care should be taken when designing the network configurations for Portworx on Red Hat OpenShift.

- **Default NIC usage:** By default, Portworx utilizes the same network interface cards used by the Kubernetes cluster for both management traffic and data replication. This configuration simplifies the initial setup of the Portworx storage cluster but may lead to potential network congestion as both types of traffic share the same physical network resources. Using the default NIC for Portworx should be used for labs, or development environments to ease setup, but should be avoided for use cases where storage performance is critical, such as in production environments.

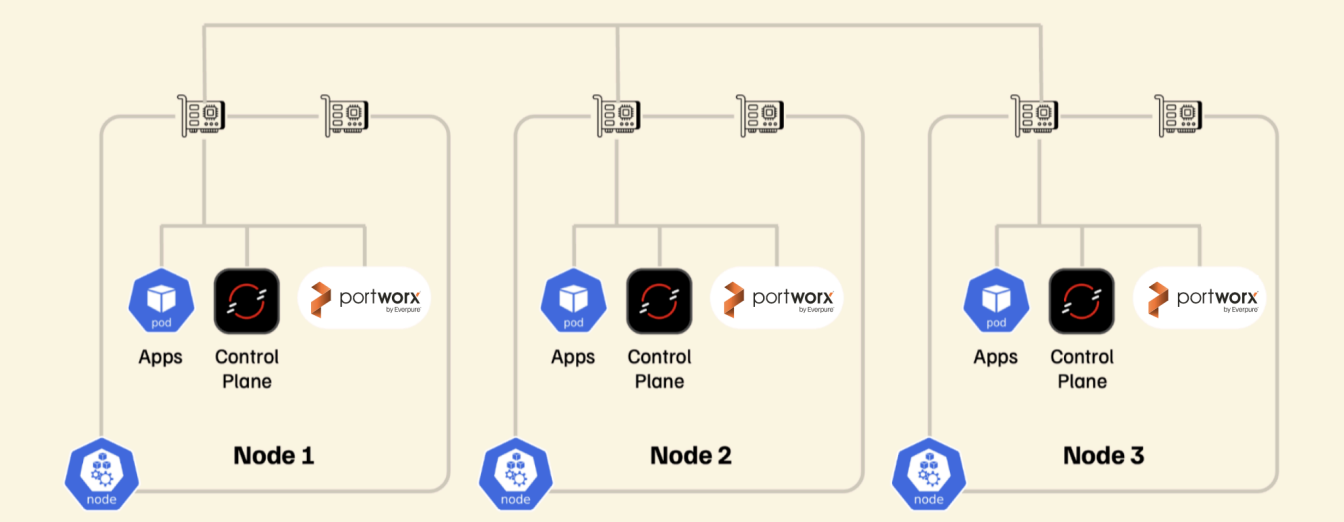

Figure 5. Portworx default network configuration

- **Dedicated data networks:** To optimize network performance and reduce congestion, Portworx can be configured to use a dedicated NIC specifically for handling data replication between nodes. This approach segregates data replication traffic from the management and application traffic, ensuring that the NICs used by Red Hat OpenShift for managing containerized applications, remain uncongested and perform efficiently. Implementing a dedicated data network can enhance the overall throughput and reliability of the storage cluster, especially with high data replication demands.

Note: *Bonded NICs would be preferred for the Portworx data network in order to prevent a disruption in the event that a NIC fails.*

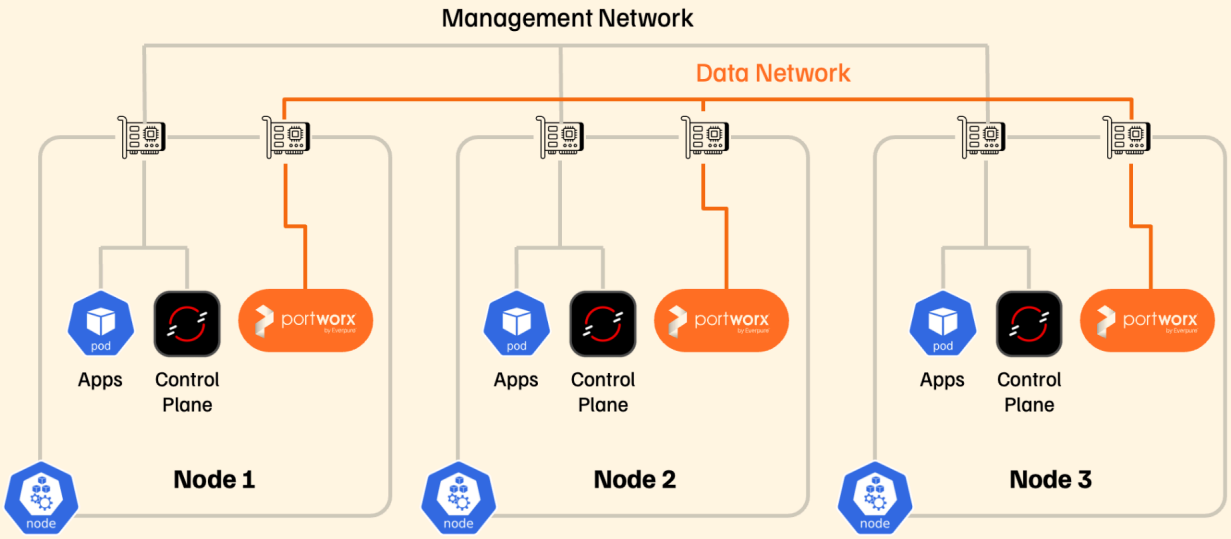

Figure 6. Portworx dedicated data network

- **Network performance**: The physical network has a great deal to do with how fast data can be written to replicas across nodes in the Red Hat OpenShift cluster. Writes to a persistent volume must be replicated to a second node before committing the writes back to the application, meaning the network is involved in how fast applications can write data to disk when using Portworx for data availability. Portworx recommends a network bandwidth of 10Gbps, with a minimum requirement of 1Gbps with latency less than 5 ms. Ensure that all storage nodes participating in data replication for Portworx use the same speed NICs to provide consistent storage performance.

***Note**: The network bandwidth has an important effect on performance of the storage cluster to maintain replicas. Networks with 25Gbps or more would be preferred for high intensity data workloads.*

Firewall Considerations

When deploying Portworx in a Red Hat OpenShift environment, ensuring proper network communication is essential for the seamless operation of your storage infrastructure. A critical aspect of this setup involves configuring your firewall to allow the necessary ports that facilitate communication between Portworx nodes and management interfaces. Properly opening these ports ensures that data replication, monitoring, and management tasks can be performed without interruption, maintaining the reliability and performance of the storage cluster. Below, the essential firewall ports required for running Portworx are listed.

Port	Protocol	Direction	Description
17001	TCP	Inbound and East-West Traffic	PX Management
17002	TCP and UDP	East-West Traffic	PX node to node gossip communication
17003	TCP	East-West Traffic	PX store data port

Port	Protocol	Direction	Description
17004	TCP	East-West Traffic	PX namespace
17005	TCP	East-West Traffic	PX control plane server
17006	TCP	East-West Traffic	PX data plane server
17008	TCP	East-West Traffic	KVDB monitor port
17009	TCP	East-West Traffic	PX node to node communication
17010	TCP	East-West Traffic	PX Namespace driver
17011	TCP	East-West Traffic	PX diagnostic service
17014	TCP	East-West Traffic	Watchdog server
17015	TCP	East-West Traffic	PX etcd peer-to-peer
17016	TCP	East-West Traffic	PX etcd client service
17017	TCP	East-West Traffic	PX gRPC SDK gateway
17018	TCP	Inbound and East-West Traffic	Portworx management port
17019	TCP	East-West Traffic	PX health monitor

Table 2. Portworx ports requirement

Additionally, the optional but recommended telemetry services have specific network requirements as outlined below:

Port	Protocol	Direction	Description
17021	TCP	East-West Traffic	Telemetry log uploader
20002	TCP	East-West Traffic	Telemetry phone home

Table 3. Portworx telemetry ports requirement

Telemetry also requires outbound access over port 443 to the following public domains and should be reachable by the worker nodes.

Port	Protocol	Direction	URL
443	TCP	outbound	https://register.cloud-support.purestorage.com
443	TCP	outbound	https://rest.cloud-support.purestorage.com
443	TCP	outbound	https://logs-01.loggly.com

Table 4. Telemetry domain allowed list

The following URLs must be accessible over HTTPS. These URLs are essential for tracking license consumption by Portworx, ensuring accurate monitoring and compliance. Proper access to these addresses guarantees that Portworx can effectively manage and report on license usage, contributing to an optimized and well-regulated deployment.

Port	Protocol	Direction	URL
443	TCP	outbound	https://rest.zuora.com
443	TCP	outbound	https://flex1327.compliance.flexnetoperations.com

Table 5. Licensing Domain Allowed List

Storage Considerations

Proper sizing considerations are crucial to the success of any storage solution. Ensuring that your storage resources are adequately sized helps maintain performance, reliability, and prevents potential bottlenecks and inefficiencies that can disrupt operations. When deploying a solution like Portworx on Red Hat OpenShift, it is essential to evaluate the storage needs of your applications carefully. This involves assessing factors such as current and projected data growth, I/O performance requirements, redundancy, high availability, and the capacity to handle peak loads. By taking these factors into account you can design a storage architecture that not only meets the demands of today but is also resilient to support future growth.

Initial Cluster Capacity

Portworx offers mechanisms to scale both vertically and horizontally, ensuring your storage cluster can dynamically adapt to changing demands. Vertical scaling involves increasing the capacity of individual storage nodes, while horizontal scaling adds additional nodes to the cluster to enhance overall capacity and redundancy. These scaling operations can be seamlessly managed through automation routines powered by AutoPilot and Cloud Drives depending on the underlying storage providers. These processes can be fully automated, allowing the cluster to respond in real-time to workload demands.

The availability of these advanced scaling features significantly alleviates the pressure of initial capacity planning. However, a thorough initial capacity sizing exercise remains crucial. Proper planning helps to ensure that the cluster is adequately provisioned to handle the expected workloads from the outset, providing a stable foundation upon which dynamic scaling can operate effectively. This initial step sets the stage for smooth operations and helps to identify any potential limitations or bottlenecks before they impact performance.

The following factors should be considered when creating the initial capacity planning:

- Number of volumes (PVCs) in the cluster
- Average size of volumes
- Number of nodes
- Replication factor (Portworx recommends a replication factor of 2 or 3)

Identify Total Cluster Capacity Needs

The first step to identify initial cluster capacity needs is to review the total amount of storage necessary for your applications and the associated replicas for those applications to provide high availability. Use the table below to total up the total cluster volume sizes necessary for operations. Be sure to also include a growth factor to account for increases in capacity expected. Sample entries have been added as an example.

Volume Size	# of Volumes	Replication Factor (HA)	Growth Factor	Total Size (Volume Size * Volumes * Replication Factor * Growth Factor)
50 GiB	30	3	1.3	5.85 TiB
100 GiB	50	2	1.3	13 TiB
Total Size				**Sum = 18.85 TiB**

Table 6. Cluster Capacity Worksheet

Once you have determined the total amount of capacity that is needed for your workloads, the storage node sizing can be completed based on the results. The number of storage nodes * the total disk capacity available (not including journal devices or the operating system disks) should be greater than the desired cluster size from the previous exercise. An example can be found below.

Desired Cluster Size	Number of Storage Nodes	Total Disk Capacity available per Node not counting Operating System Disks or Journal Devices	Total Disk Capacity (Nodes * Disk Size * Disk Count)
18.85 TiB	6	4 TiB	**24 TiB**

Table 7. Node Sizing Worksheet

The two calculations presented above should identify a baseline for the initial storage capacity necessary to run your workloads.

Backing Disks for the Portworx Storage Cluster

When configuring a Portworx storage cluster on Red Hat OpenShift, the choice and configuration of backing disks are critical for achieving optimal performance, reliability, and scalability. Below are key considerations to guide you in selecting and configuring backing disks for your Portworx storage cluster:

- **Block devices:** Portworx requires block storage devices as the backing storage for the Portworx storage cluster. Each storage node must be provisioned with raw block devices rather than pre-formatted file systems or logical volumes. These block devices can be sourced from local disks within the bare metal worker nodes or from a hardware storage array, such as a FlashArray or other SAN systems, that can present block devices to the servers. Ensure that the block devices to be used by Portworx can be recognized by the host operating system of the worker nodes.
- **Disk type and performance**: The block devices used by Portworx determine the overall capacity, I/O performance, and throughput of the storage cluster. When selecting disks, consider the type, size, and performance capabilities. Faster devices, such as NVMe SSDs, will provide better performance for Kubernetes applications utilizing Portworx for persistent volumes (PVs) compared to slower devices like traditional hard disk drives.
- **Disk redundancy and fault tolerance**: Portworx allows application owners to specify the level of redundancy for their stateful data through replication factors (e.g., repl2, repl3). This ensures that replicas are stored on multiple nodes in the cluster. Additionally, using a RAID configuration (such as RAID1) at the hardware controller level can provide per-node disk redundancy. This setup allows for single disk failures within a node without triggering a re-sync of replicas to another node, providing an extra layer of protection and potentially reducing re-sync overhead, though it comes with higher hardware costs.
- **Capacity planning**: The backing disks directly impact the total capacity of the Portworx storage cluster. While Portworx requires a small amount of overhead for metadata and journal writes, most of the capacity is used for persistent volumes and replicas. When planning capacity, consider the expected number of replicas per application to ensure sufficient storage. Disks can be resized, or additional devices can be added to expand the total storage cluster capacity for future needs.
- **Storage pool configuration**: Portworx requires at least one storage pool per storage node to operate, but multiple pools can be configured to segregate different types of workloads or create storage tiers for performance optimization. Each storage pool should consist of drives with identical specifications to ensure consistent storage performance across the nodes in the cluster. This configuration helps maintain uniformity in storage performance for all replicas.

By carefully considering these factors, you can ensure that your Portworx storage cluster on Red Hat OpenShift is well-equipped to handle the demands of your applications, providing high performance, reliability, and scalability.

Storage Pools

By following these guidelines, organizations can deploy a robust and scalable Portworx architecture on Red Hat OpenShift, ensuring high availability and optimal performance for their applications.

A storage pool in Portworx is a logical grouping of a node's physical drives. Portworx uses the space in these storage pools to dynamically create virtual volumes for containers. Storage pools consist of a collection of drives with the same capacity and type. When you create a pool, Portworx categorizes it based on its latency and performance in random and sequential IOPS.

To set up a storage pool, you need a minimum of one drive per node. Portworx evaluates each drive through a benchmark process, categorizing it based on its throughput into one of three I/O priorities: low, medium, or high. Drives that share the same I/O priority and size within a node are grouped into a pool. This categorization allows you to align various applications with the appropriate tier of storage based on their performance requirements. For instance, database applications can be placed on flash devices for high performance, while applications managing logging data can utilize less performant options.

A single backing disk is required per node to create a storage pool. However, for future scalability, consider the available method to expand the storage capacity in the cluster. This can be achieved either by expanding an existing disk which is often feasible when using a hardware array for backing disks, or by adding additional storage nodes to the cluster. Expanding an existing disk will provide additional storage to the cluster and the node's pool, while adding additional storage nodes will add both storage and compute for the OpenShift cluster.

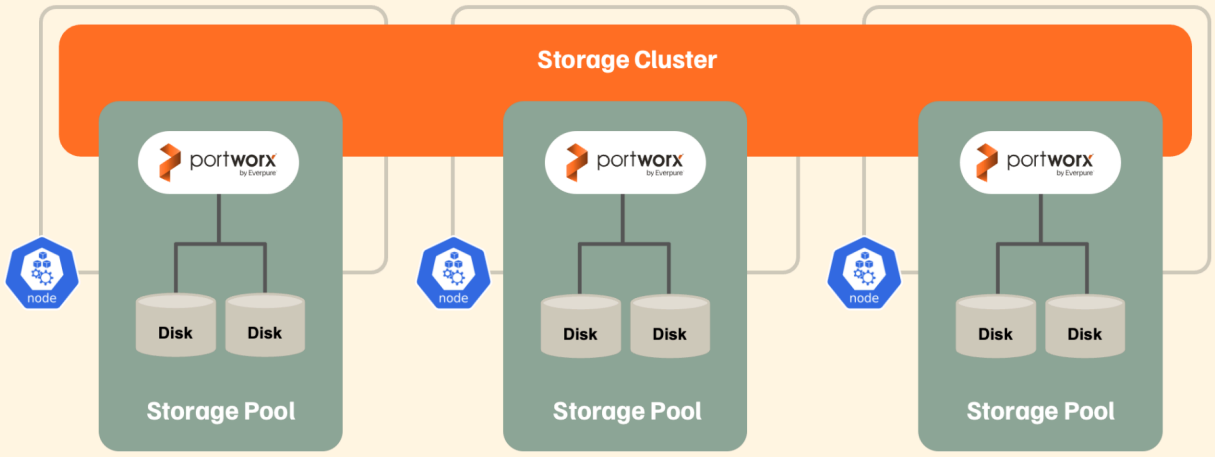

Figure 7. Portworx Storage Pools

Metadata Device

The PX-StoreV2 datastore for Portworx requires a 64GB or larger metadata drive to store general metadata related to storage activities. This drive can either be a physical device drive, or a logical partition, but it must be created on each storage node within your cluster. Using a dedicated disk separates the I/O to a different device from the storage cluster, reducing read and write contention and lowering latencies for applications running on the nodes. This metadata device may also store the Portworx KVDB (discussed below) as well as a journal to speed up writes. Due to the importance of this device, Portworx recommends using a dedicated 64GB device with low latency to handle the frequent reads/writes to this disk.

Recommended IOPS for Metadevice

- If IOPS are independent of disk size, Portworx recommends a minimum size of 64 GB and a minimum of 450 IOPs.
- If IOPS are dependent on disk size, Portworx recommends a size of 150 GB to ensure you get a minimum of 450 IOPS.

Portworx KVDB

Portworx relies on a key-value database (KVDB) to store critical information, including the cluster's state, configuration data, and metadata for storage volumes and snapshots. This data is essential for the operation of the storage cluster and must be highly available and protected from failures.

When configuring Portworx for your Kubernetes environment, you have two primary options for the key-value database (KVDB) that Portworx uses to store its internal metadata, an internal KVDB or an external etcd cluster:

> **Internal KVDB:** For most deployments, we recommend using the internal KVDB provided by Portworx. This option simplifies the deployment process by eliminating the need for an additional external cluster, thereby reducing complexity and potential points of failure. The internal KVDB is fully integrated with Portworx, providing a seamless and efficient solution for managing your storage metadata.

External etcd cluster: Alternatively, you can use an external etcd cluster as your KVDB. This option is particularly beneficial if you already have an existing etcd infrastructure in place or if you need advanced features such as SyncDR (Synchronous Disaster Recovery). SyncDR requires an external etcd cluster to ensure the high availability and consistency of your data across geographically dispersed clusters. While using an external etcd cluster can offer additional flexibility and scalability, it also introduces more complexity in terms of setup and management. If using an external etcd cluster be sure that the cluster is spread across multiple fault domains to ensure quorum can be maintained during an outage.

Portworx recommends using an internal KVDB database deployed as part of the storage cluster deployment unless there is a requirement to use the Portworx SyncDR solution.

Journal Device

To improve performance Portworx offers the ability to use a journal (or write-ahead log WAL) during write operations. This journal is a special storage location where changes are first recorded sequentially before being applied to the storage system. These sequential writes are much faster than writing random writes across the storage layer which speeds up transaction time. The sequential journal writes are then written to the main storage asynchronously. The journal device should be at least as fast as the fastest storage device on the node allocated for the Portworx storage cluster. If the journal device is slower, overall performance will degrade to match the slower device, so Portworx recommends a flash or NVMe drive for the journal.

The journal device should be 3GB, as Portworx will only utilize this amount of storage for journaling. Using a larger device does not provide any additional benefit unless the size of the disk also adds IOPS, like in many cloud environments. If using local drives for backing disks, it is recommended to have the journal device automatically created on the metadata drive instead of dedicating a large disk for a small 3GB requirement. This is simply to reduce the cost of hardware. If using a backing drive from a storage array where multiple volumes or LUNs can be created easily of any size, and presented to worker nodes, Portworx recommends creating a 3GB volume specifically for the journal device to get better performance.

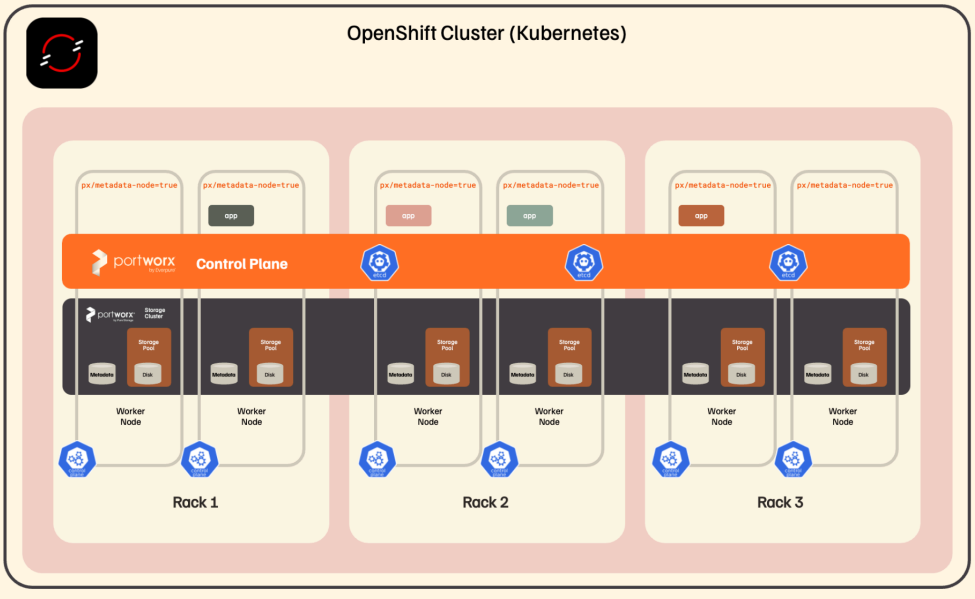

Figure 8. Metadata reference

In the event that the KVDB cluster goes offline, the Portworx storage cluster will enter a "run-flat" mode. During this state, Portworx workloads will continue to operate, but no changes can be made to the cluster, including the creation and deletion of PVCs and scaling operations.

Generic Third Party Array Considerations

A hardware storage array can be used for providing the block-devices as backing disks for the Portworx Storage cluster. Using a hardware storage array may provide benefits such as deduplication or compression for replicas stored in the Portworx Storage cluster but will depend on the capabilities of the storage array vendor.

When providing backing disks to Portworx storage nodes from a hardware array, special considerations should be taken. This section explains design considerations that should be taken into account when using an external Storage Array Network for Portworx Backing disks.

Block devices are typically presented to physical hosts (Red Hat OpenShift worker nodes) via Fibre-Channel or iSCSI connections. Portworx recommends using a minimum of two iSCSI or Fibre-Channel interfaces per node to present storage to the cluster. Using a pair of interfaces provides high availability in the case of a hardware failure, and with multi-pathing can provide more bandwidth to the backing storage array. Consider the configurations below for Fibre-Channel, NVMe over TCP and iSCSI connections:

Fibre-Channel: If using Fibre-Channel, ensure proper zoning in the SAN Fabric for the nodes.

iSCSI or NVMe/TCP: When using iSCSI, it is crucial to dedicate Ethernet interfaces solely for storage purposes, rather than sharing them with OpenShift traffic. Ensure that jumbo frames are configured on these interfaces, as well as any network devices in-line including the physical switches and storage array to enhance performance.

For the storage cluster disks, Portworx recommends presenting a single volume for each Portworx storage node based on your sizing decisions. Using a single volume makes it less impactful on performance if re-sizing operations are introduced later. Provided that your storage array is capable of resizing these volumes on day two, these volumes can be expanded to increase the total storage of the Portworx storage cluster without causing a rebalancing task to occur. Rebalancing is a costly storage operation which results in copying data.

It is simple to create additional volumes of a desired size with an external hardware array. In a scenario when you are using an external hardware array, Portworx recommends creating 3 GB volumes from the array and presenting them to the worker nodes to be used as the journal device.

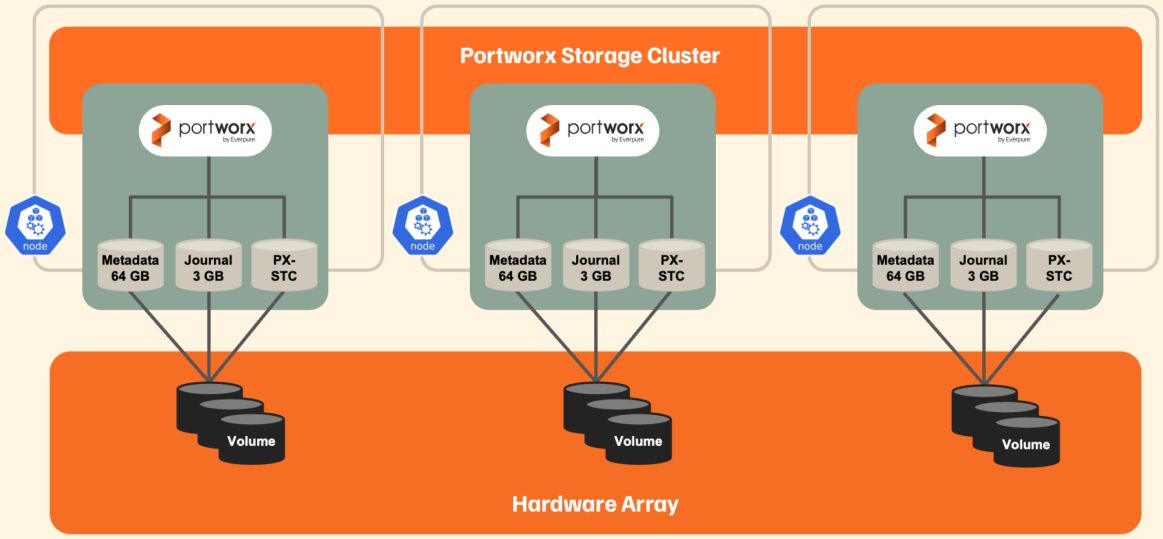

Figure 9. Third-party array diagram

FlashArray Considerations

For FlashArray customers, the process of managing the backing disks for Portworx is similar to any hardware array, but adds Portworx cloud drive functionality to simplify provisioning and scaling of backing disks. Similar to any hardware array, the Fibre-Channel, NVMe over TCP, iSCSI interfaces should be configured with proper zoning and masking, as well as configuring devices for Jumbo Frames if applicable, and proper multipathing should be configured to support high availability and balanced I/O.

Before configuring and provisioning backing volumes, consider whether to implement Secure Application Workspaces (SAW). SAW leverages Everpure's Secure Multi-Tenancy (SMT) feature to logically isolate tenants (realms) on a shared FlashArray. Each realm can be assigned its own users, volumes, QoS policies, and network interfaces, providing strict boundaries between Kubernetes clusters, namespaces, or teams. This isolation ensures that storage operations initiated by one tenant (via Portworx CloudDrives or FlashArray Direct Access Volumes) do not impact others, making it ideal for multi-tenant Kubernetes environments. SAW should be configured early in the design process to enforce governance, simplify reporting, and prevent provisioning conflicts. See the official Everpure FlashArray documentation to enable Secure Multi-Tenancy, and Secure Application Workspaces.

Portworx Enterprise has Cloud Drives which is a way for the Portworx control plane to manage Everpure disk arrays and persistent disks on Amazon Web Services (AWS), Azure, Google Compute Platform (GCP), Oracle Cloud, IBM Cloud, and VMware vSphere. Cloud Drives deployment on Red Hat OpenShift requires worker node configuration via MachineConfigs, including appropriate `multipathd` settings and Secure Boot considerations across storage-providing MachineSets. Support for Secure Boot is introduced in Portworx version 3.6; for earlier versions, Secure Boot must be disabled to ensure compatibility.

The configuration for multipath.conf can be found in the official Portworx documentation. Here is a sample YAML manifest for a MachineConfig:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: worker-enable-multipathing
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,YOUR_BASE64_ENCODED_MULTIPATH_CONF_HERE
        mode: 420
        path: /etc/multipath.conf
    systemd:
      units:
      - name: multipathd.service
        enabled: true
# The above manifest enables multipathing for FiberChannel connections. To add iSCSI based FlashArray
support, uncomment the lines below.
#      - name: iscsid.service
#        enabled: true
Ensure Secure Boot is disabled to allow Portworx to install required kernel modules (applicable to
versions prior to 3.6).
```

Before deploying Portworx, create a StorageAdmin account on the FlashArray and obtain the API key. This key will be used to create a JSON configuration file pure.json. After creating the JSON file, set up a namespace for Portworx and create a Kubernetes secret using this file, following the instructions provided in the Portworx documentation. This secret will be used by the Portworx control plane to send instructions to the FlashArray to create volumes, expand volumes, etc. providing automation between Portworx and the FlashArray.

When deploying Portworx, specify the desired configuration values as usual but choose FlashArray as the provider. When it comes to the capacity configuration section of the config, specify a 3 GB Journal volume, and the desired capacity for your Portworx storage pool. When you apply the storage cluster configuration to the Red Hat OpenShift cluster, the appropriate volumes will be created on the FlashArray and attached to the worker nodes.

Portworx AutoPilot can also be used in conjunction with Cloud Drives, to automatically scale not only persistent volumes, but the Portworx storage cluster as well.

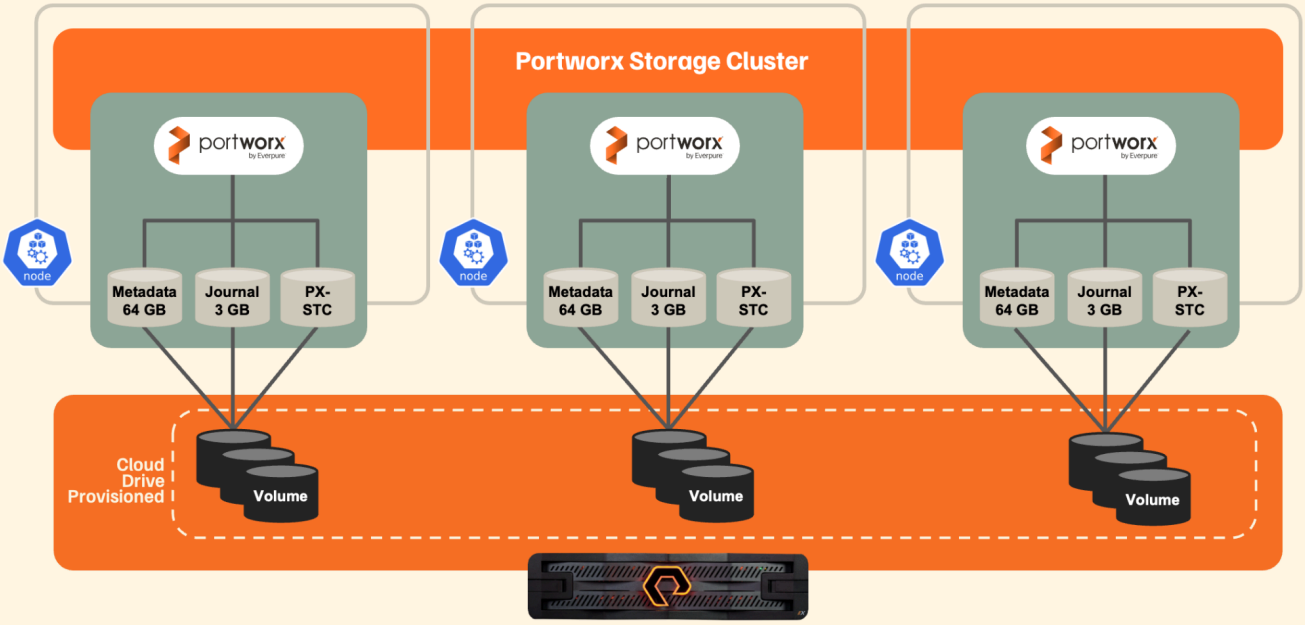

Figure 10. Everpure array diagram

FlashArray Direct Access (FADA)

When using a Everpure FlashArray, Kubernetes persistent volumes can be deployed in the Portworx storage cluster using backing disks from the FlashArray, or the persistent volumes can be created directly on the FlashArray. Users may decide to place some volumes on the Portworx storage cluster, and others on the FlashArray directly depending on the requirements for the application. Portworx Enterprise backed by a FlashArray provides a hybrid capability allowing Portworx to manage the massive scale usually found in Kubernetes environments, while also providing high performance with direct access to a Pure FlashArray.

Local Disks Considerations

Local disks (internal disks) can be utilized as block devices to back the Portworx storage cluster. Bare metal installations with local disks may require a different storage setup compared to clusters with a storage array. Since local disks cannot be partitioned like volumes on a storage array, it's important to use the entire physical disk to optimize I/O performance. This approach allows for the creation of a storage pool without the need to designate an individual journal device or metadata drive, thereby maximizing the use of physical hardware. For instance, servers equipped with four 1TB disks may not want to allocate a full 1TB disk solely for a 3GB journal device.

The trade-off between gaining performance by splitting I/O streams across devices and utilizing the full capacity of the disks is managed by combining metadata, journal, and storage space into a single pool in Portworx installations on local devices.

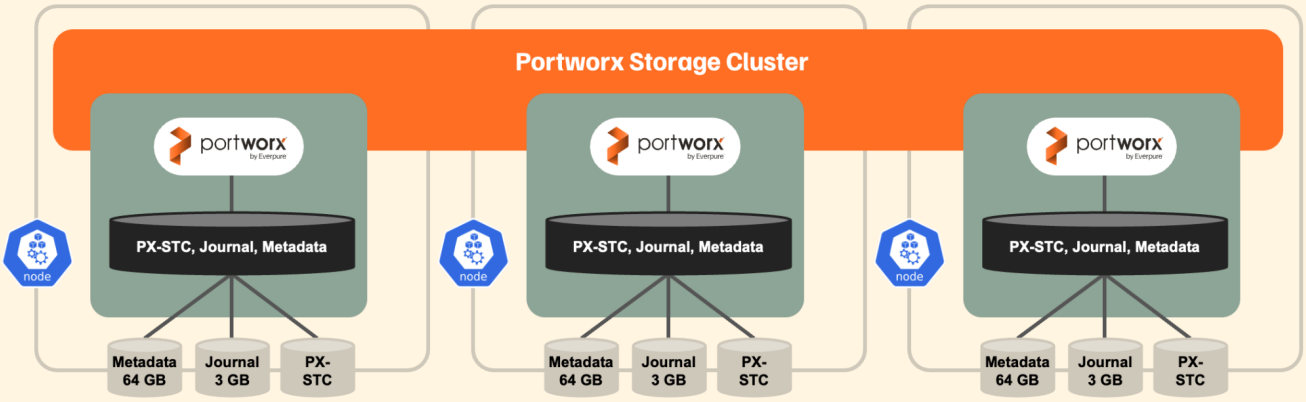

Figure 11. Local disk configuration diagram

Application availability is maintained through the replication factor within the Portworx storage cluster. Portworx replicates data between nodes, eliminating a requirement for mirroring or parity through RAID configurations at the physical disk level. However, using RAID—such as RAID 1, 5, 6, or 10—to add redundancy, while not necessary for maintaining high availability, can prevent a single disk failure from taking a Portworx node offline.

When considering RAID configurations, the decision hinges on whether you prefer the RAID configuration to degrade on a node or for the Portworx storage cluster to degrade if a single disk fails. In either case, data would be redundant as long as a replication factor with multiple replicas is used for the application.

Object Storage Options

While Portworx offers robust data management solutions for block and file storage within Kubernetes environments, it does not provide native object storage capabilities. For applications that require object storage, you can leverage solutions like FlashBlade® or Amazon S3. These options deliver scalable, high-performance object storage that complements your Kubernetes infrastructure.

To facilitate seamless integration with these object storage solutions, Portworx can proxy connection information to your applications using its scale-out object storage service. This service allows you to manage object storage for your Kubernetes workloads efficiently, providing a unified management experience. By utilizing this feature, you can easily connect your applications to the required object storage backend, ensuring smooth operation and optimal performance.

PX Store Configuration

Portworx uses **PX-StoreV2** as its datastore for supported deployments. PX-StoreV2 is designed for high-performance volume management, offering optimized metadata handling, improved performance tracking, and support for **PX-Fast**, which enables an accelerated I/O path for volumes.

High Availability Considerations

Ensuring high availability in your Red Hat OpenShift and Portworx environments is vital for minimizing downtime and maintaining continuous operation. This section delves into key considerations for achieving high availability, focusing on three critical areas: physical fault domains, OpenShift node topologies, and Portworx storage cluster topologies. By understanding and implementing best practices in each of these areas, you can design a resilient infrastructure that can withstand hardware failures and other potential disruptions, thereby ensuring that your applications and services remain consistently available.

Physical Fault Domains

High availability (HA) is a critical component in the design of any resilient storage and compute infrastructure. While Portworx and Red Hat OpenShift provide robust mechanisms to ensure data and application availability, the effectiveness of these mechanisms heavily depends on the proper configuration of physical fault domains. Understanding and correctly implementing physical fault domains is essential to mitigate risks associated with hardware failures and to ensure continuous operation of your applications.

A fault domain is essentially a grouping of hardware components that share a common risk of failure. These components can include servers, storage devices, network equipment, power supplies, and even entire racks or data centers. By appropriately configuring fault domains, you can isolate failures to specific segments of your infrastructure, thereby reducing the impact on overall system availability.

Portworx recommends using three isolated fault domains when possible, including using separate data centers or availability zones. In this architecture Portworx and Red Hat OpenShift control planes and replicas can be stretched across multiple independent fault domains and the loss of an entire data center will not create an outage to an application stack.

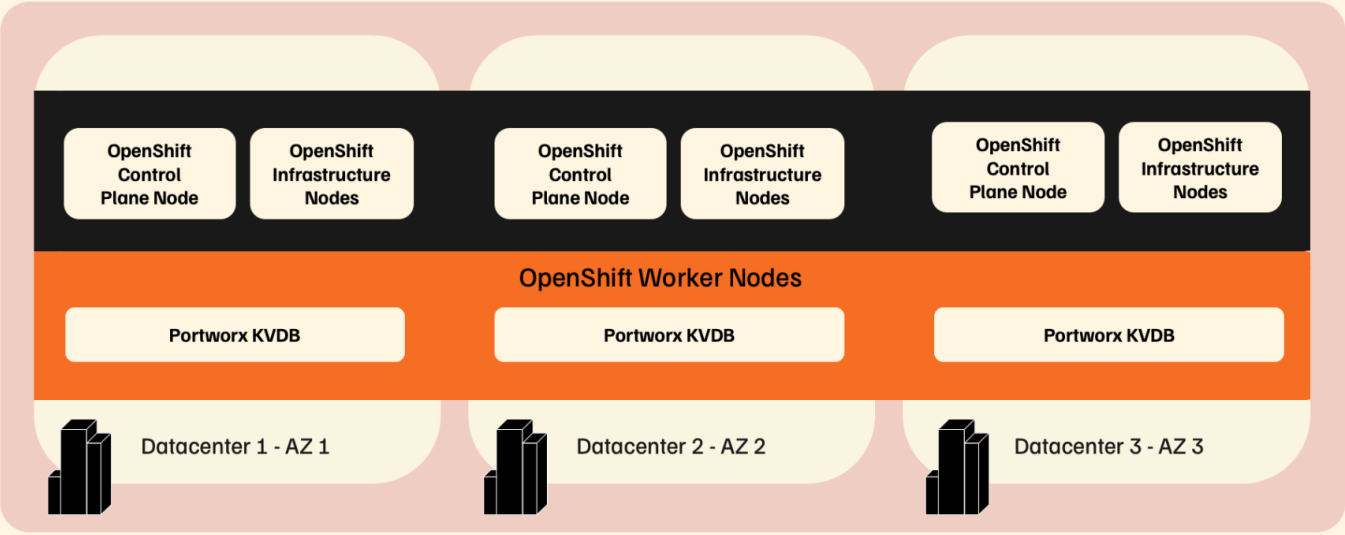

Figure 12. Data center fault domains

In on-premises environments, this is often cost prohibitive and exceptions are often made at the risk of possible outages. In those situations, Portworx recommends using isolated rack-level redundancies to provide a highly available physical design. This architecture assumes that Red Hat OpenShift and Portworx control plane nodes are distributed across racks and worker nodes are also distributed and labeled into their own fault domains. Each rack should have independently maintained power distribution units (PDUs), top of rack (TOR) switches, etc. and shared resources such as generators, cooling systems, etc. should be reduced as appropriate to reduce single points of failure.

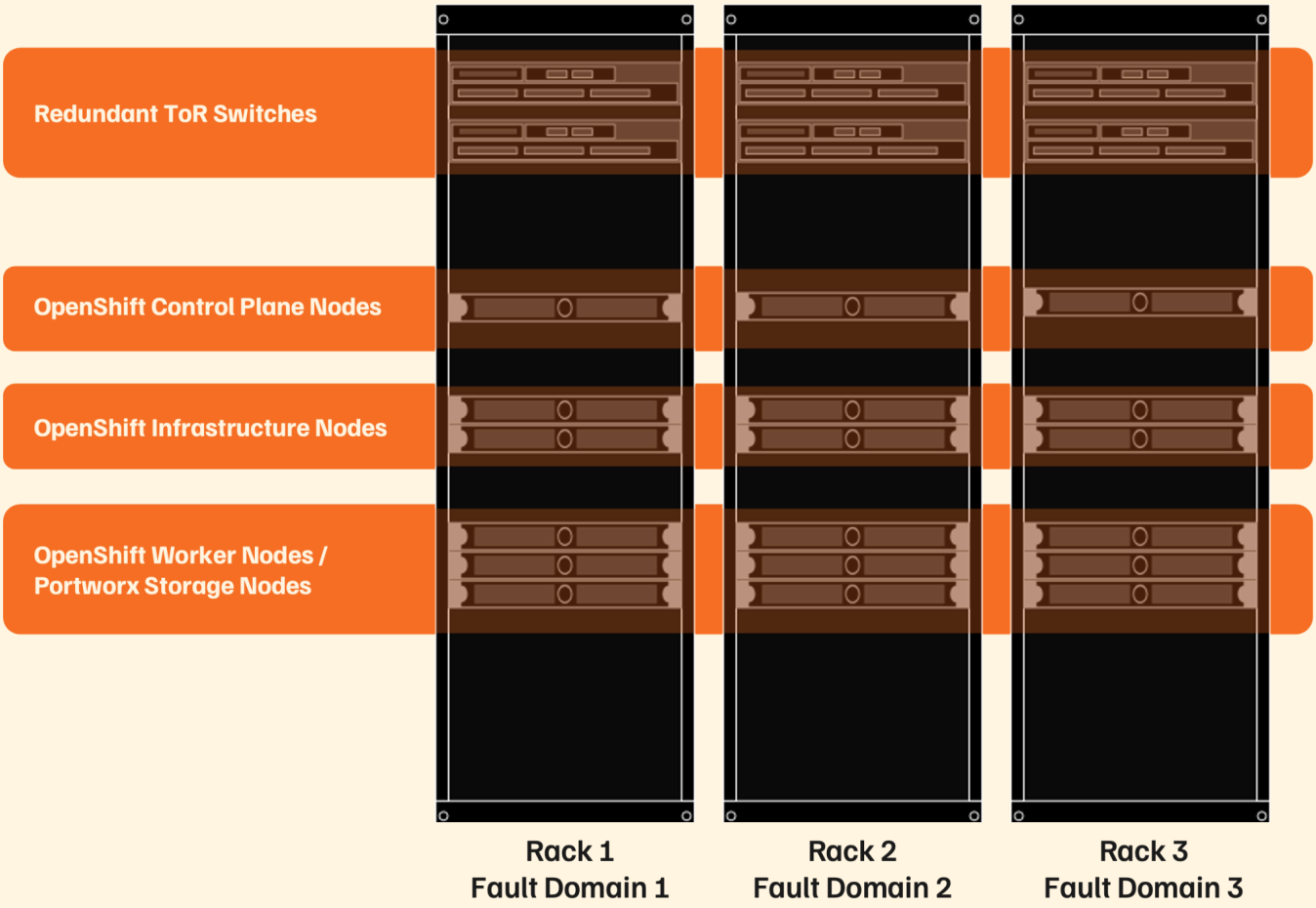

Figure 13. Rack fault domains

Red Hat OpenShift Node Topologies

A node is a virtual or bare-metal machine in the Red Hat OpenShift cluster. Worker nodes host your application containers, grouped as pods. The control plane nodes run services that are required to control the cluster. In OpenShift Container Platform, the control plane nodes contain more than just the Kubernetes services for managing the OpenShift Container Platform cluster.

In Red Hat OpenShift, there is also the concept of Infrastructure nodes. These are nodes that are labeled to run pieces of the OpenShift Container Platform environment in order to isolate infrastructure workloads for two primary purposes: preventing incurring billing costs against subscription counts and separating maintenance and management.

Infrastructure workloads are isolated workloads in addition to those services running on the control plane. At a minimum, a Red Hat OpenShift cluster contains 2 worker nodes in addition to 3 control plane nodes. While control plane components critical to the cluster operability are isolated on the masters, there are still some infrastructure workloads that by default run on the worker nodes—the same nodes on which cluster users deploy their applications.

To qualify as an infrastructure node and use the included entitlement, only components that are supporting the cluster, and not part of an end-user application, may be running on those instances. A list of workloads that can be executed in infrastructure nodes can be found in the "Red Hat OpenShift control plane and infrastructure nodes" section in OpenShift sizing and subscription guide for enterprise Kubernetes.

Portworx Storage Cluster Node Topologies

Portworx leverages node labels to identify fault domains and zones, which helps prevent availability issues in the event of an entire zone failure. In cloud environments such as AWS, GCP, Azure, IBM, or VMware, Kubernetes nodes come prepopulated with well-known failure domain labels. Portworx parses these labels to understand the cluster topology and manage data distribution accordingly.

For bare metal workloads, these labels are not automatically provided but can be manually added to achieve the same topology management. This allows for effective management of rack or datacenter configurations to enhance fault tolerance.

Key Node Labels for Fault Domain Management

Portworx looks for two primary node labels to automatically distribute replicas across fault domains:

- **px/region**: This label identifies the region in which a node is located.
- **px/zone**: This label identifies the zone within a region.

In addition to these legacy labels, Portworx also supports the Kubernetes-style topology labels:

- `topology.portworx.io/region`
- `topology.portworx.io/zone`

These topology labels provide the same logical grouping but follow a standardized naming convention aligned with Kubernetes topology awareness.

By default, Portworx ensures that replicas are distributed across different zones without requiring manual intervention beyond labeling the OpenShift worker nodes that host the Portworx Storage cluster. This distribution ensures that if an entire zone fails, the application data remains available as its replicas reside in other zones. Additionally, Portworx keeps replicas within the same region to avoid data synchronization across metered or higher latency network connections.

You can label your nodes with the px/region and px/zone labels based on your availability requirements. For example, you might label each rack in your datacenter as a different zone to ensure Portworx automatically deploys replicas across multiple racks.

Optional Rack-Level Label

Portworx also supports an optional third label called px/rack. This label provides more granular control over replica placement decisions:

- **px/rack**: This label specifies the rack information, allowing you to control which racks the replicas should be placed on during deployment. By default Portworx will use the rack label information to distribute your replicas but you may specify which racks the replicas should be deployed to through a storage class.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: px-postgres-sc
provisioner: pxd.portworx.com
parameters:
   repl: "2"
   shared: "true"
   racks: "rack1,rack2"
```

Using the px/rack label in conjunction with the px/region and px/zone labels allows you to implement specific placement logic tailored to your environment's needs.

P4

Optimizing the performance of your Portworx storage cluster involves configuring various parameters tailored to your deployment architecture and workload requirements. This section will explore several key areas for performance tuning, including disaggregated and hyper-converged architectures, the use of journal devices, I/O profiles, and the nodiscard option for managing data deletions efficiently. By carefully configuring these options, you can ensure that your Portworx storage cluster operates efficiently, delivering reliable and high-performing storage services for your OpenShift workloads.

Disaggregated Architecture

In a disaggregated deployment, where dedicated storage nodes are used, you can enable higher resource consumption by specifying the rt_ops_conf_high runtime option. This setting allows Portworx to utilize more resources on storage nodes, enhancing performance. This runtime option should be considered if there are at least 64 CPUs per node. The following example shows how to configure this option in your StorageCluster spec:

```
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  name: px-cluster
namespace: portworx
spec:
  image: docker.io/portworx/oci-monitor:3.6.0
    ...
  runtimeOptions:
        rt_opts_conf_high: "1"
```

Hyper-converged Architecture

In a hyper-converged architecture, where applications run on the same hosts as storage, resource allocation must be carefully managed to balance compute and storage performance. Configure the number of threads based on the number of cores available on the host. For example, if your host has 16 cores:

- num_threads=16 sets the total number of threads.
- num_io_threads=12 allocates 75% of the total threads for I/O operations.
- num_cpu_threads=16 allocates threads for CPU-bound tasks.

These values can be specified in the runtimeOptions field as shown below:

```
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
    name: px-cluster
namespace: portworx
spec:
    image: docker.io/portworx/oci-monitor:3.6.0
    ...
    runtimeOptions:
        rt_opts_conf_high: "1"
        num_threads: "16"
        num_io_threads: "12"
        num_cpu_threads: "16"
```

Journal Device

Portworx recommends using a journal device to handle metadata writes efficiently. Journal writes are frequent and small, benefiting significantly from the performance of flash or NVMe drives. The journal device should be 3GB, as Portworx will not utilize more than this amount.

```
kind: StorageCluster
...
storage:
   devices:
     - /dev/sdb
   journalDevice: auto
   systemMetadataDevice: /dev/sdc

...
```

***Note**: If you don't specify a journal device, Portworx will carve out 3GB out of the metadata disk to use automatically.*

I/O Profiles

Optimizing I/O profiles based on workload types can significantly enhance performance. I/O profiles determine how Portworx volumes interact with underlying storage to optimize traffic:

- **auto**: Switches between none (single replica) and db_remote (replication factor of 2 or higher).
- **db_remote**: Implements a write-back flush algorithm to coalesce multiple syncs within a 100ms window. Coalesced syncs are acknowledged after they have been copied to memory on all replicas. This mode assumes all replicas do not fail simultaneously in a 100ms window.
- **journal**: Performs stable writes to the journal and commits writes in batches to the backing storage, amortizing sync costs.
- **auto_journal**: Detects incoming write patterns to determine whether the journal profile can improve performance. It switches between none or journal I/O profiles based on data in the previous 24 seconds. Once a high enough confidence level is determined it configures the volume to use the journal profile or avoid it.
- **none**: No I/O optimizations are done for the volume.

If no I/O profile is specified, Portworx uses the default I/O profile set during cluster setup, typically auto.

By carefully configuring these options, you can tailor your Portworx storage cluster to meet specific performance requirements, ensuring efficient operation and optimal resource utilization across your Red Hat OpenShift environment.

Nodiscard Option

Certain applications, such as Kafka and Elasticsearch, frequently perform discard or delete operations, which can negatively impact the overall performance of the Portworx cluster. To mitigate this effect, it is recommended to use the nodiscard parameter.

When the nodiscard parameter is used, the Portworx volume is mounted with the nodiscard option. This configuration means that data deleted from the filesystem is not immediately removed from the underlying block device. By avoiding immediate discard operations, the system reduces the overhead associated with these operations, thereby improving performance.

However, because the deleted data remains on the block device, it is necessary to periodically run a filesystem trim operation to clear out this data. This deferred deletion strategy helps maintain higher performance levels for applications that generate a high volume of discard operations.

Below is an example of a storage class definition that includes the nodiscard option:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-storage-class
provisioner: pxd.portworx.com
parameters:
  nodiscard: "true"
  # Other storage class parameters
```

By incorporating the nodiscard option, you can optimize the performance of your Portworx storage cluster, particularly for applications with heavy discard/delete workloads, while managing data deletion in a more controlled manner.

When using the nodiscard option, Portworx recommends configuring auto-fstrim in the cluster to periodically delete unused data blocks.

```
pxctl cluster options update --auto-fstrim on
```

Instead of letting auto fstrim trigger jobs automatically, you can schedule fstrim operations by defining a specific time and duration for fstrim to run. This is particularly beneficial if you're looking to perform operations when you know your node will have low traffic.

The following command will schedule a fstrim job schedule on the daily basis. To switch to a weekly schedule, modify the format accordingly.

```
pxctl cluster options update --fstrim-schedule-start <daily=hh:mm> --fstrim-schedule-duration <hrs>
```

Processor States

C-states, or CPU idle states, are power-saving modes that processors use to reduce energy consumption when the CPU is idle. Each C-state represents a different level of power savings, with deeper states saving more power but taking longer for the CPU to wake up from, potentially introducing latency in performance-sensitive applications.

For optimal performance, especially in high-throughput and low-latency environments like those managed by Portworx, it's recommended to set the CPU to C0 to ensure maximum responsiveness. If power savings are necessary, shallower C-states such as C1 or C1E can be considered. This approach minimizes latency and ensures that storage operations are not adversely affected by power state transitions. Properly configuring C-states helps maintain the desired performance levels and supports the overall efficiency of the Portworx deployment.

Storage I/O Contention

In a shared Kubernetes environment like Red Hat OpenShift, "noisy neighbors" can significantly impact the performance of other applications. "Noisy neighbors" refer to workloads that consume excessive I/O or network bandwidth, degrading the performance of other applications running on the same cluster. To mitigate this issue, Portworx offers a feature called Application I/O Control.

Application I/O Control allows you to set limits on I/O operations and throughput for individual applications, ensuring that no single application can monopolize the cluster's resources. By defining these limits, you can maintain a balanced and efficient environment, preventing performance degradation caused by resource-hungry applications.

Security Considerations

Securing your Portworx cluster involves two key areas: authorization, which protects Portworx volumes from unauthorized access and encryption, which secures the data within the volumes by encrypting it.

Authorization

Authorization in Portworx adds an extra layer of security by implementing role-based access control (RBAC) to protect volumes from unauthorized access. This ensures that only authenticated and authorized users can access the volumes. When security is enabled, Portworx creates a user token for the ‘kubernetes’ user by default.

To enable authorization in Portworx, add the spec.security.enabled: true stanza in the StorageCluster YAML configuration:

```
kind: StorageCluster
...
spec:
  ...
  security:
    enabled: true
...
```

Once authorization is enabled, only Kubernetes users will be able to access Portworx volumes if persistent volume claims (PVCs) are created using storage classes that include the authentication token. By default, “guest access” is allowed if no authentication token is included in the storage class. To disable guest access, refer to the Portworx documentation.

Encryption

Portworx recommends protecting your persistent volumes with encryption. All encrypted volumes are protected by a passphrase. Portworx uses this passphrase to encrypt the volume data at rest as well as in transit. It is recommended to store these passphrases in a secure secret store such as Hashicorp Vault but Portworx has support for additional secret store providers such as IBM Key Management, AWS KMS, Google Cloud KMS, Azure Key Vault, and the Kubernetes Secrets store.

The passphrases can be used in one of two ways for encrypting volumes, a per-volume secret, which uses a different secret passphrase for each encrypted volume and a cluster-wide secret, which uses a common secret for all encrypted volumes within the cluster. Each method has its own advantages and disadvantages, which should be considered when designing your Portworx deployment.

Per-volume Secret Encryption

Per-volume encryption provides the highest level of security, as exposing a single encryption passphrase would only affect one persistent volume, not the entire cluster. However, managing multiple encryption keys can be challenging, especially when using Portworx Disaster Recovery to move volumes to another cluster that might not have all the necessary encryption keys.

When using a per-volume encryption method, a secret needs to be created before each persistent volume is requested. The name of this secret must be maintained as it is used later on. For example:

```
oc create secret generic volume-secrets-1 -n portworx
--from-literal=mysql-pvc-secret-key-1=mysecret-passcode-for-encryption-1
```

Then a second secret must be created that refers to the initial secret from above. This secret identifies which secret to use for encryption, the namespace the secret exists in, the secret key, and the secret context.

```
oc create secret generic mysql-pvc-1 -n portworx --from-literal=SECRET_NAME=volume-secrets-1
--from-literal=SECRET_KEY=mysql-pvc-secret-key-1 --from-literal=SECRET_CONTEXT=portworx
```

Create a CSI Storage class for the encrypted PVCs:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-csi-db-encrypted-pvc-k8s
provisioner: pxd.portworx.com
parameters:
  repl: "3"
  secure: "true"
  io_profile: auto
  csi.storage.k8s.io/provisioner-secret-name: ${pvc.name}
  csi.storage.k8s.io/provisioner-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: ${pvc.name}
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
  #backend: "pure_block"  # Uncomment this line for FADA volumes.
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

Then request PVC's as normal, using this storage class. The templatized parameters in the storage class point to the name and namespace of the PVC itself. This ensures that each PVC requires a separate secret of the same name in the same namespace. This way, each PVC gets encrypted with its own passphrase.

Cluster-wide Encryption

Using a cluster-wide secret for encryption simplifies key management, as a single key is used for all volumes in the cluster. While this makes management easier, it means that if the key is compromised, all volumes are at risk. This approach is particularly useful for Portworx Disaster Recovery, as only one key needs to be migrated along with the applications.

When using the cluster-wide secret for encrypting volumes, simply create a secret in your secrets provider. This secret will house the passphrase for your clusterwide encryption.

```
oc -n portworx create secret generic <your-secret-name> \
  --from-literal=cluster-wide-secret-key=<value>
```

Run a command to set the new secret as the cluster-wide secret used by Portworx.

```
PX_POD=$(oc get pods -l name=portworx -n portworx -o jsonpath='{.items[0].metadata.name}')
oc exec $PX_POD -n portworx -- /opt/pwx/bin/pxctl secrets set-cluster-key \
  --secret cluster-wide-secret-key
```

Then specify the secure: “true” parameter in the storage classes to be encrypted and it will automatically use the passphrase in the cluster-wide secret to encrypt the volumes created from the storage class.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-secure-sc
provisioner: pxd.portworx.com
parameters:
  secure: "true"
  repl: "3"
  #backend: "pure_block"  # Uncomment this line for FADA volumes.
```

Monitoring Considerations

Monitoring is a vital aspect of managing and maintaining both Red Hat OpenShift clusters and Portworx Enterprise storage solutions. Effective monitoring ensures the health, performance, and reliability of your infrastructure, allowing for proactive issue resolution and optimized resource utilization.

Monitoring in Red Hat OpenShift

The OpenShift Container Platform monitoring stack is based on the Prometheus open source project and its wider ecosystem. The OpenShift Container Platform includes a preconfigured, preinstalled, and self-updating monitoring stack that provides monitoring for core platform components. You also have the option to enable monitoring for user-defined projects.

A set of alerts are included by default that immediately notify administrators about issues with a cluster. Default dashboards in the OpenShift Container Platform web console include visual representations of cluster metrics to help you to quickly understand the state of your cluster. With the OpenShift Container Platform web console, you can view and manage metrics, alerts, and review monitoring dashboards.

In the Observe section of the OpenShift Container Platform web console, you can access and manage monitoring features such as metrics, alerts, monitoring dashboards, and metrics targets.

After installing OpenShift Container Platform, cluster administrators can optionally enable monitoring for user-defined projects. By using this feature, cluster administrators, developers, and other users can specify how services and pods are monitored in their own projects. As a cluster administrator, you can find answers to common problems such as user metrics unavailability and high consumption of disk space by Prometheus, an open-source solution for monitoring and alerting in Red Hat OpenShift, in Troubleshooting monitoring issues.

Monitoring in Portworx Enterprise

Monitoring is a critical component of managing your Portworx storage cluster effectively. It is essential not only for leveraging advanced features such as AutoPilot and Application I/O control but also for ensuring the overall health, performance, and reliability of your storage infrastructure within your Kubernetes cluster.

A robust monitoring solution allows you to proactively identify and resolve issues, optimizing performance and ensuring continuous availability. By tracking key performance metrics, you can fine-tune configurations and ensure that applications run smoothly.

Effective monitoring also supports capacity planning by tracking usage trends, allowing you to anticipate future storage needs and allocate resources efficiently. Furthermore, it plays a vital role in compliance and auditing, helping you maintain logs and records to meet industry standards and regulatory requirements. Understanding resource utilization through monitoring can also optimize costs and improve overall efficiency.

Implementing a comprehensive monitoring strategy is crucial for maintaining a high-performing, reliable, and secure storage environment, which is essential for supporting your Kubernetes workloads.

Prometheus

Prometheus is an open-source monitoring and alerting toolkit designed specifically for reliability and scalability in dynamic environments like Kubernetes. In a Red Hat OpenShift cluster, Prometheus is used to collect and store metrics data, providing real-time insights into the performance and health of applications and infrastructure. It scrapes metrics from configured endpoints, stores them efficiently, and allows for powerful querying using its flexible query language, PromQL.

Portworx typically deploys its own version of Prometheus for monitoring activities of the storage cluster, but in a Red Hat OpenShift deployment, it is deployed with OpenShift itself. Portworx seamlessly integrates with the OpenShift Prometheus stack and provides several key metrics that can be used to monitor the health and performance of the Portworx cluster. Before deploying Portworx in the Red Hat OpenShift cluster, ensure that monitoring for user-defined projects is enabled. Follow the steps provided in the "Observe Cluster on Openshift" article of the Portworx documentation for guidance.

To query Portworx metrics in the Red Hat OpenShift web console:

1. Find the Observe dropdown, then select the Metrics dashboard.
2. On the Metrics page, enter any Portworx metric, all of which start with 'px_'. For example, to check CPU usage on the Portworx cluster, use the metric 'px_cluster_cpu_percent'.

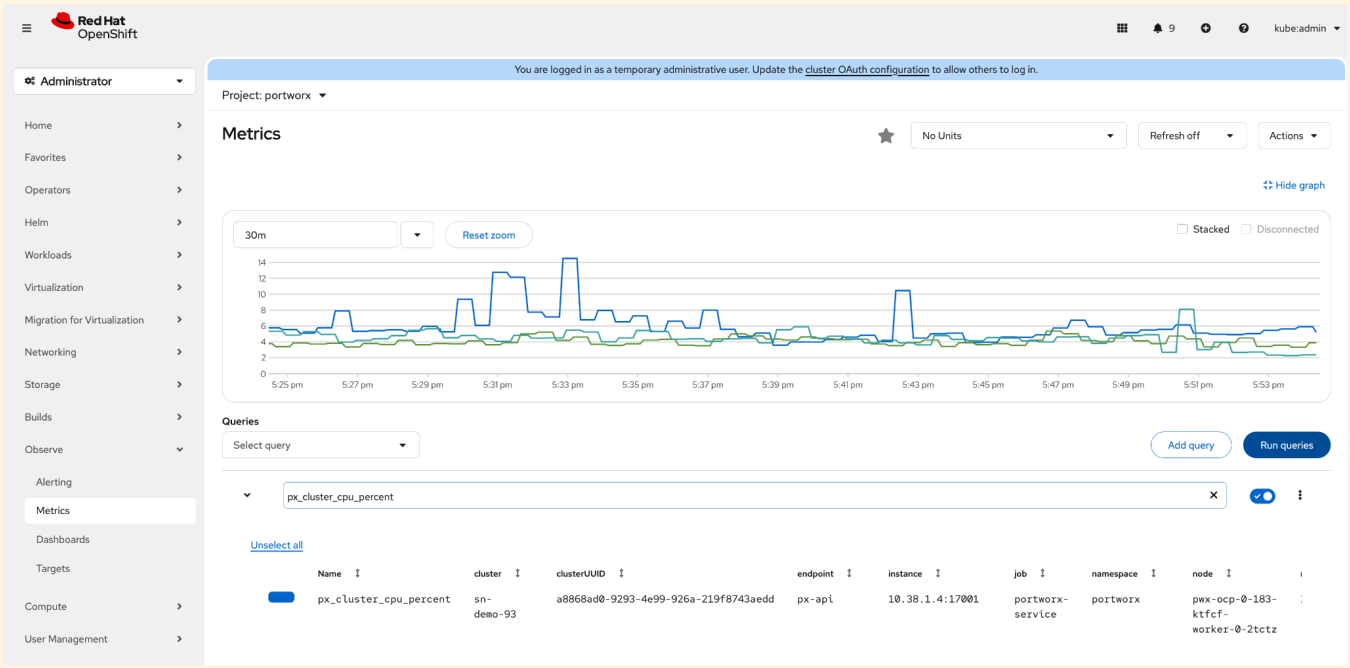

Figure 14. OpenShift metrics page

You can also check status of Portworx metrics targets:

1. Find the Observe dropdown, then select the Targets dashboard
2. Filter for “Portworx” in the text filter:

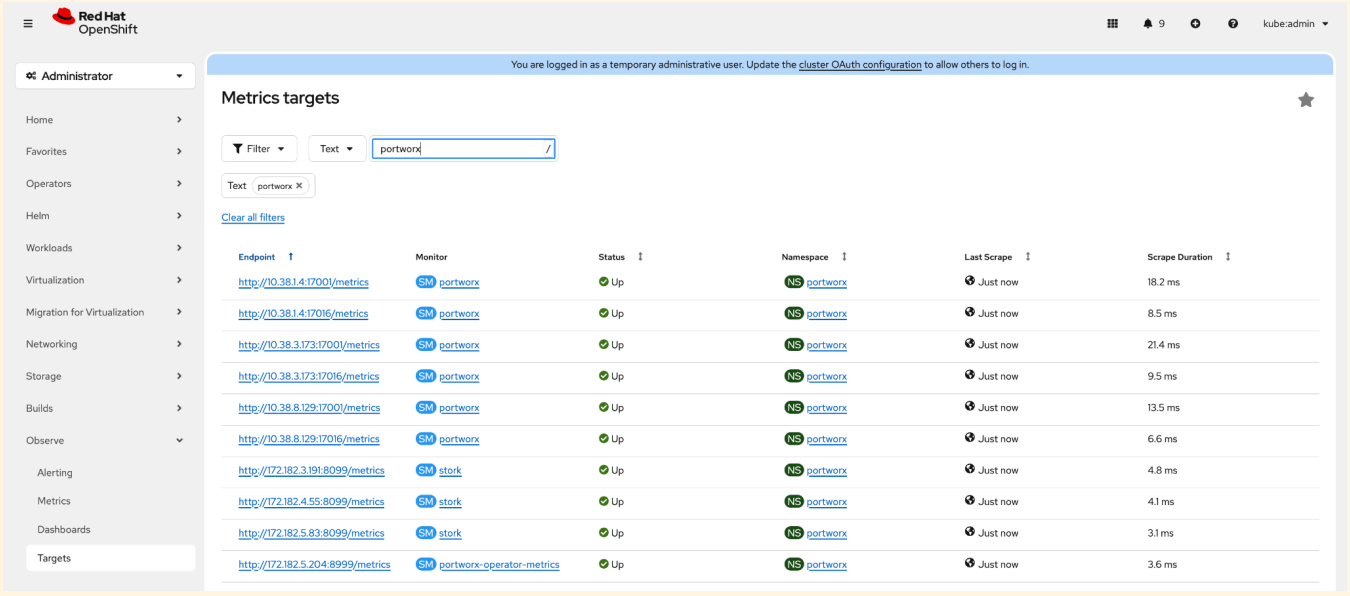

Figure 15. Portworx targets in the OpenShift console

Alertmanager

Alertmanager is a critical component of the Prometheus ecosystem, responsible for handling alerts generated by Prometheus. In a Red Hat OpenShift cluster, Alertmanager manages the lifecycle of alerts, including deduplication, grouping, and routing to the appropriate receiver endpoints such as email, Slack, or other notification systems. It ensures that alerts are delivered to the right people at the right time, facilitating rapid response to issues. With its flexible configuration, Alertmanager allows you to define sophisticated alerting rules and escalation policies, helping to maintain the stability and reliability of your environment.

Similar to Prometheus metrics, you can set up user-defined alerts in OpenShift to receive standard Prometheus alerts provided by Portworx. To enable these alerts in your Red Hat OpenShift cluster, follow the instructions in the Enabling Alert Routing for User-Defined Projects article of the OpenShift documentation.

After enabling user-defined alerts, access the Portworx alert rules via the OpenShift web console:

1. Under the Observe, select the Alerting dashboard
2. On the Alerting dashboard, select the 'Alerting rules' tab, and apply the 'namespace=portworx' filter to view the specific rules.

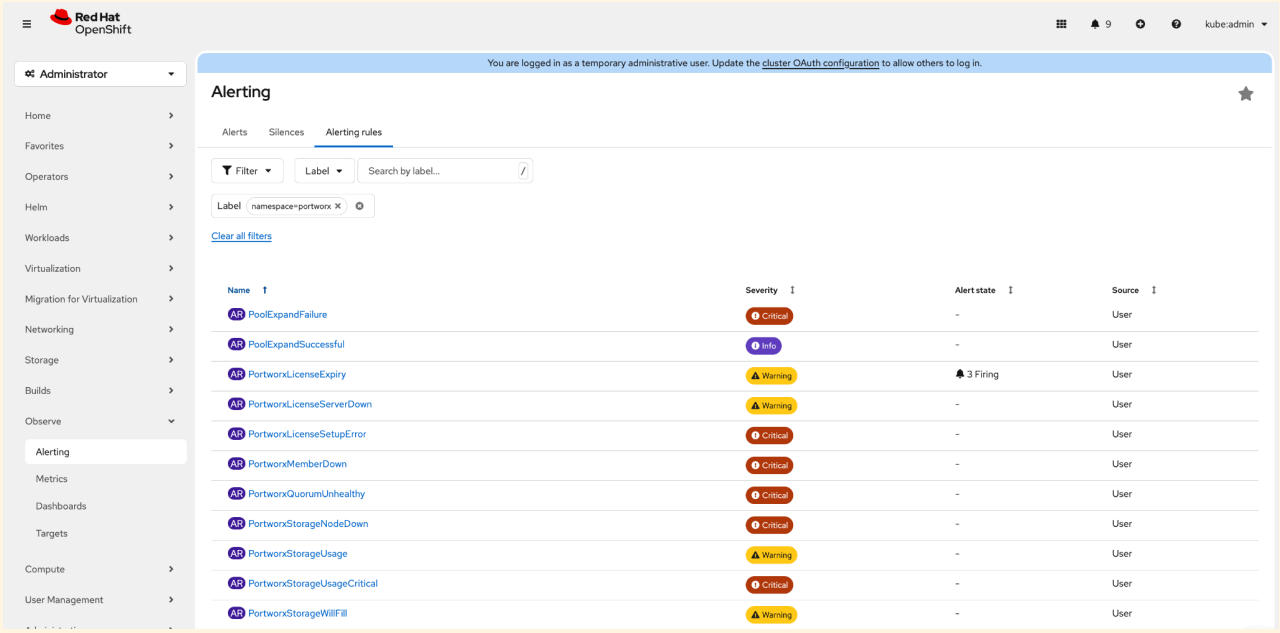

Figure 16. OpenShift alerting

Grafana Dashboards

Portworx provides five out-of-the-box Grafana dashboards to help monitor its status and performance. Since prometheus will be installed with your Red Hat OpenShift cluster, Grafana can connect to that instance to visualize the performance metrics by obtaining the Thanos route. To deploy these dashboards in your own Grafana instance on OpenShift, follow the steps outlined in the Portworx documentation. If Grafana is not already installed on your OpenShift cluster, refer to the Grafana documentation for installation and configuration instructions.

Portworx Grafana dashboards include:

- Internal KVDB (ETCD)
- Portworx Cluster
- Portworx Nodes
- Portworx Volumes
- Portworx Performance
- Portworx DR Dashboard

Figure 17. Grafana dashboard

Air-gapped Cluster Considerations

Operating Red Hat OpenShift and Portworx in air-gapped clusters presents unique challenges and considerations, as these environments are completely isolated from the Internet. This section explores the strategies and best practices for deploying and managing OpenShift and Portworx in such restricted settings. We will cover essential topics such as preparing the necessary installation files, configuring updates, and ensuring security and compliance without direct Internet access. By understanding and implementing these guidelines, you can maintain robust, secure, and efficient operations in your air-gapped clusters.

Red Hat OpenShift Considerations

When deploying a Red Hat OpenShift cluster in an air-gapped environment, a local image registry is essential for storing container images that are not accessible over the Internet. When deploying a Red Hat OpenShift cluster in an air-gapped environment, a local image registry is essential for storing container images that are not accessible over the Internet. If you do not have an existing local repository to use, Red Hat offers a container registry that can be integrated into the OpenShift installation, but Red Hat recommends an object storage solution for the registry's storage layer. In this scenario, refer to the Red Hat OpenShift documentation to determine a certified Object Storage solution to store container images.

Portworx Considerations

Portworx is often installed in an air-gapped environment to run critical applications that should not be made available over public networks such as the Internet. The primary consideration for installing Portworx in an air-gapped environment is to store the Portworx images in a local repository to be used for installations and upgrades.

In an air-gapped environment administrators should download the Portworx cluster images needed and push them into a local container registry such as the OpenShift image registry. Instructions for this can be found in the official Portworx documentation. Portworx has provided scripts to download these images for each particular Kubernetes and Portworx version. The images should be available to the OpenShift cluster before installing, scaling, or upgrading Portworx.

Telemetry and integration with Pure1® is not available for air-gapped clusters.

Operational Considerations

When deploying Portworx on Red Hat OpenShift bare metal nodes, it's important to understand the unique operational considerations that come into play. This section delves into critical aspects that ensure smooth and efficient operation, including installation information, data protection information, availability best practices, and monitoring. By addressing these factors, you can maximize the reliability and efficiency of your Portworx deployment, ensuring that your storage solutions are robust, secure, and performant in a bare metal environment.

Installation Methods and Procedures

This section of the reference architecture explains instructions for proper installation of a Portworx Enterprise storage cluster on Red Hat OpenShift bare metal nodes. This section includes installation procedures as well as monitoring and troubleshooting information for the installation process.

Red Hat OpenShift

Red Hat OpenShift User-Defined Monitoring (UDM) allows developers and administrators to extend the monitoring capabilities of OpenShift by integrating their own metrics and monitoring configurations. This feature enables the collection and analysis of custom application metrics alongside the default metrics gathered by OpenShift, providing a comprehensive view of application performance and health.

Portworx uses prometheus to monitor storage metrics and is critical for features like Autopilot and ApplicationI/O Control. The latest releases of Red Hat OpenShift don't allow applications to deploy their own Prometheus instances, so you must enable user-defined workload monitoring in OpenShift so that Portworx may use the OpenShift instance for monitoring metrics.

Consult the Red Hat OpenShift documentation for enabling User-Defined Monitoring, before installing Portworx Enterprise.

Portworx

To start the installation of Portworx on your deployed OpenShift cluster, you first need to create the Portworx storage cluster configuration using Portworx Central. Portworx Central offers a graphical user interface (GUI) that simplifies the process of building the necessary YAML configuration file for your OpenShift cluster.

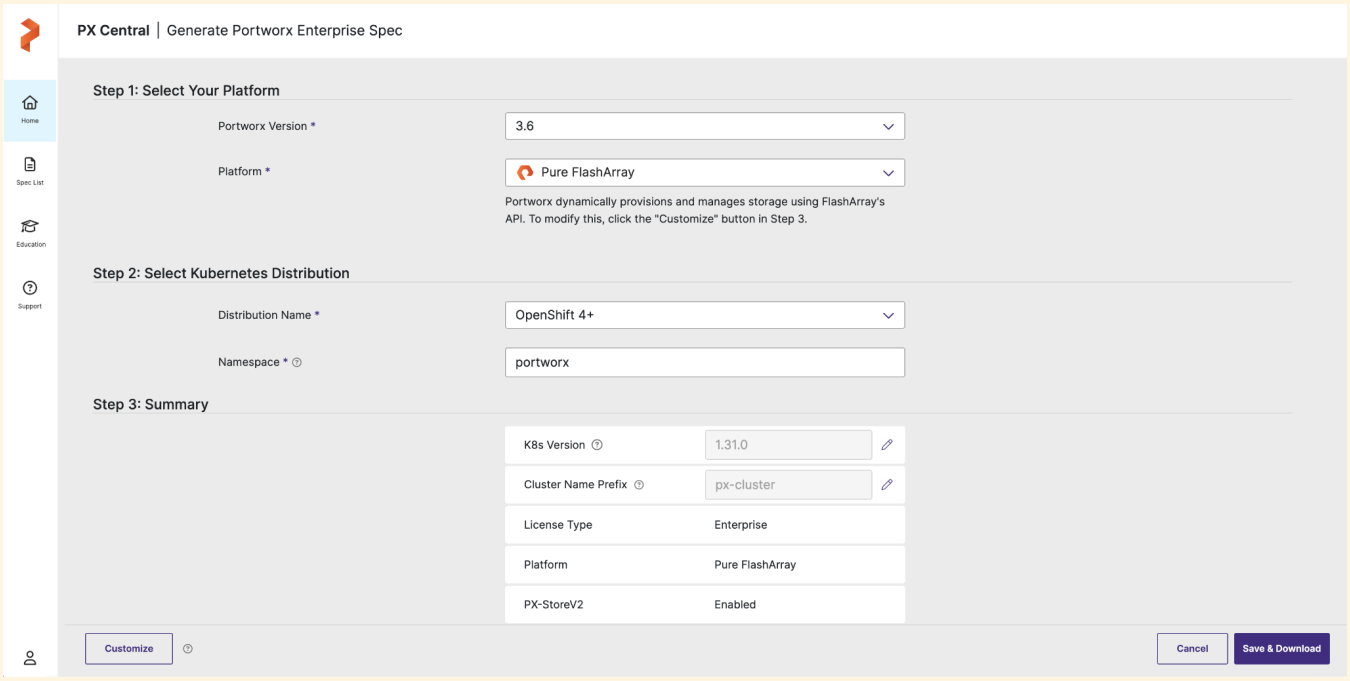

Figure 18. Portworx Central specification generator

Through the GUI, you can customize various aspects of your Portworx Storage Cluster configuration, ensuring it meets your specific requirements before applying it to the cluster. When you're done with the configuration wizard, you'll be presented with a screen displaying the Portworx Operator YAML link, and the Storage Cluster Configuration file.

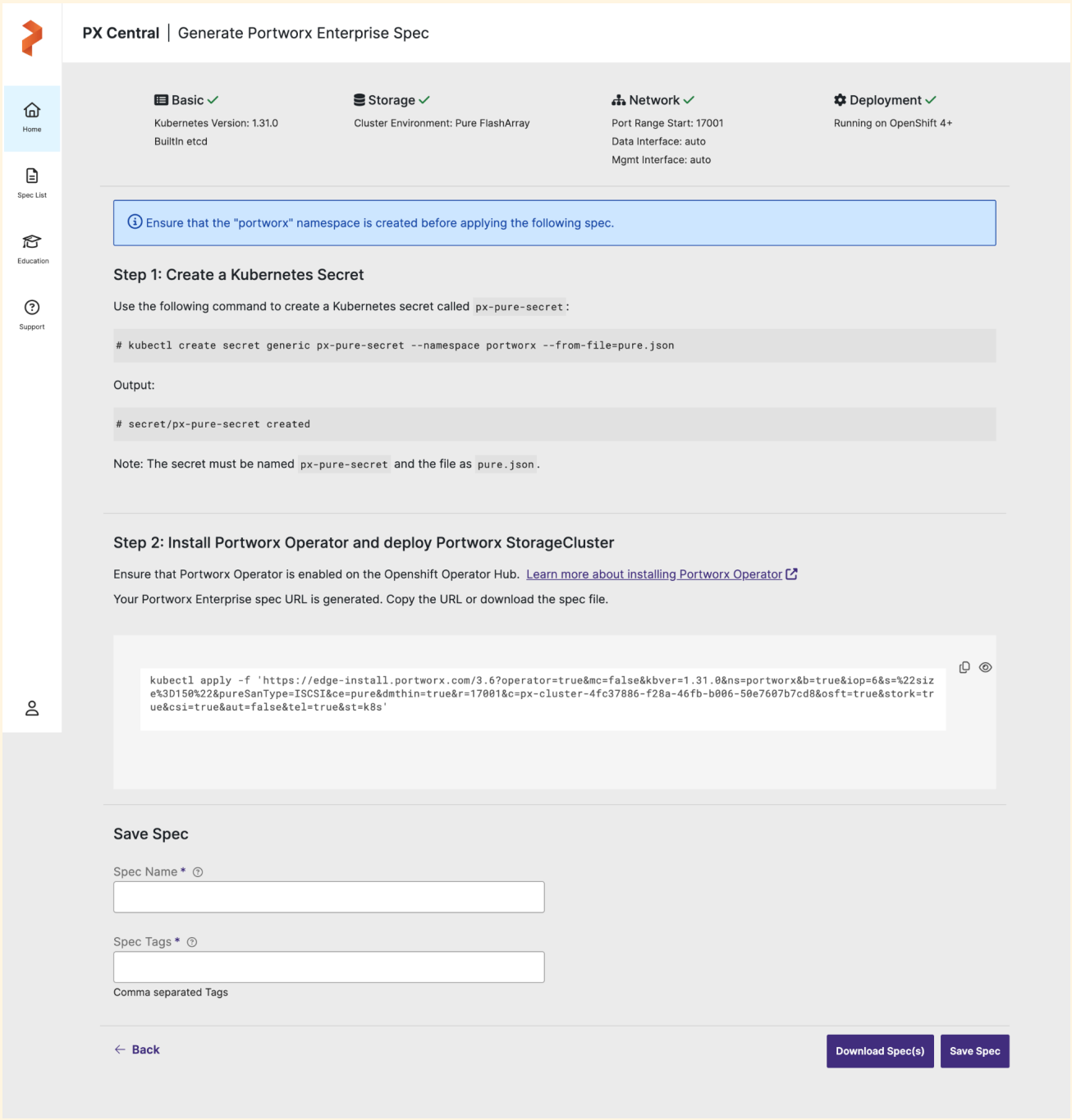

Figure 19. Portworx Central specification file

Portworx recommends installing the Portworx Operator from Ecosystem → Software Catalog in the OpenShift Console instead of through the link in Portworx Central. When deploying the Portworx operator, create a new project called “portworx” and then deploy the operator through the Ecosystem into this namespace. Also be sure to enable the Console plugin to get additional Portworx data in the OpenShift Console.

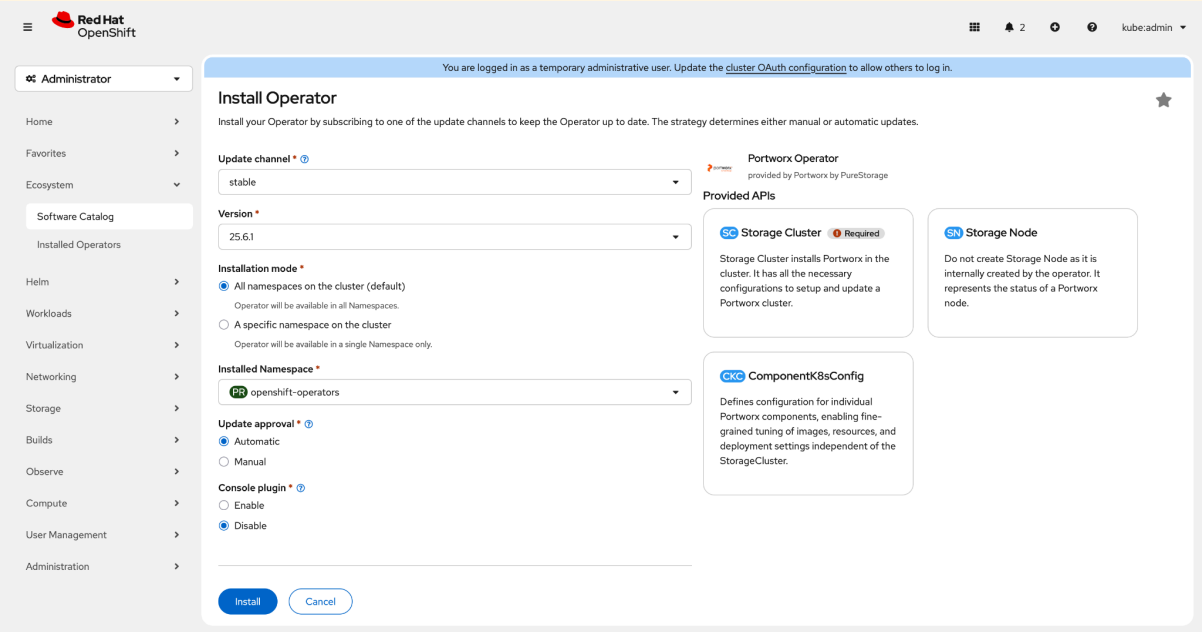

Figure 20. OpenShift Operator installation

***Note**: For air-gapped clusters, the Portworx Operator can be downloaded and stored in the local repository to be used instead of operator-hub.*

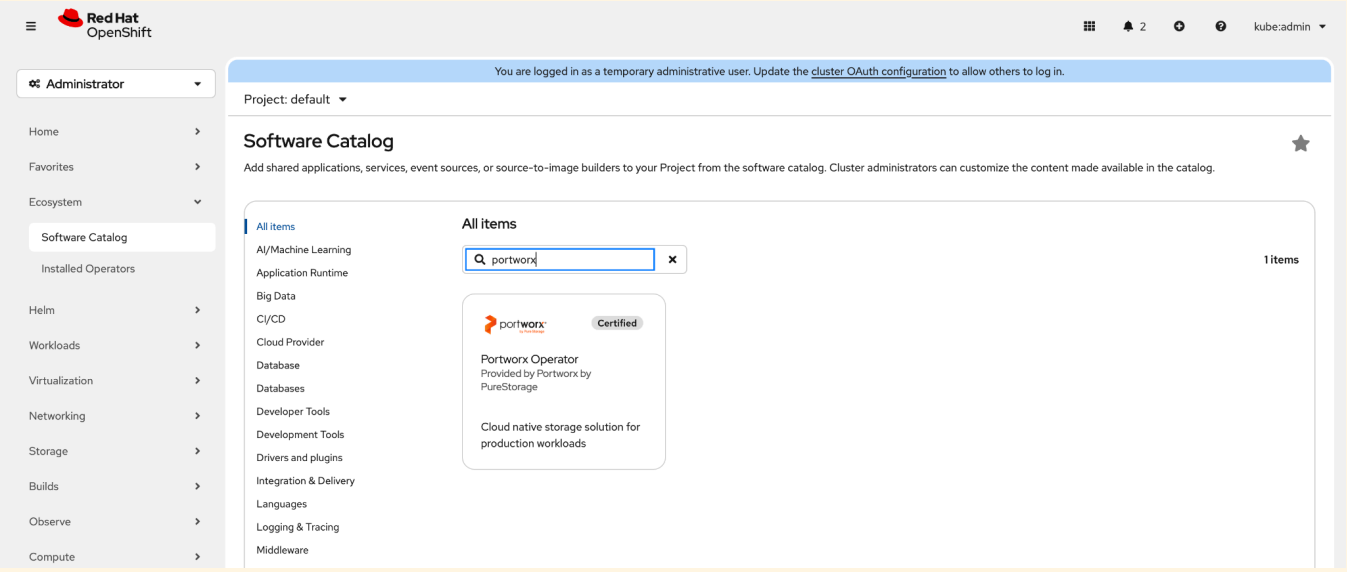

Figure 21. Portworx Operator in OperatorHub

When the Portworx Operator has been installed successfully, you can create a StorageCluster config from the OpenShift Console by pasting in the storage cluster config from the Portworx Central wizard.

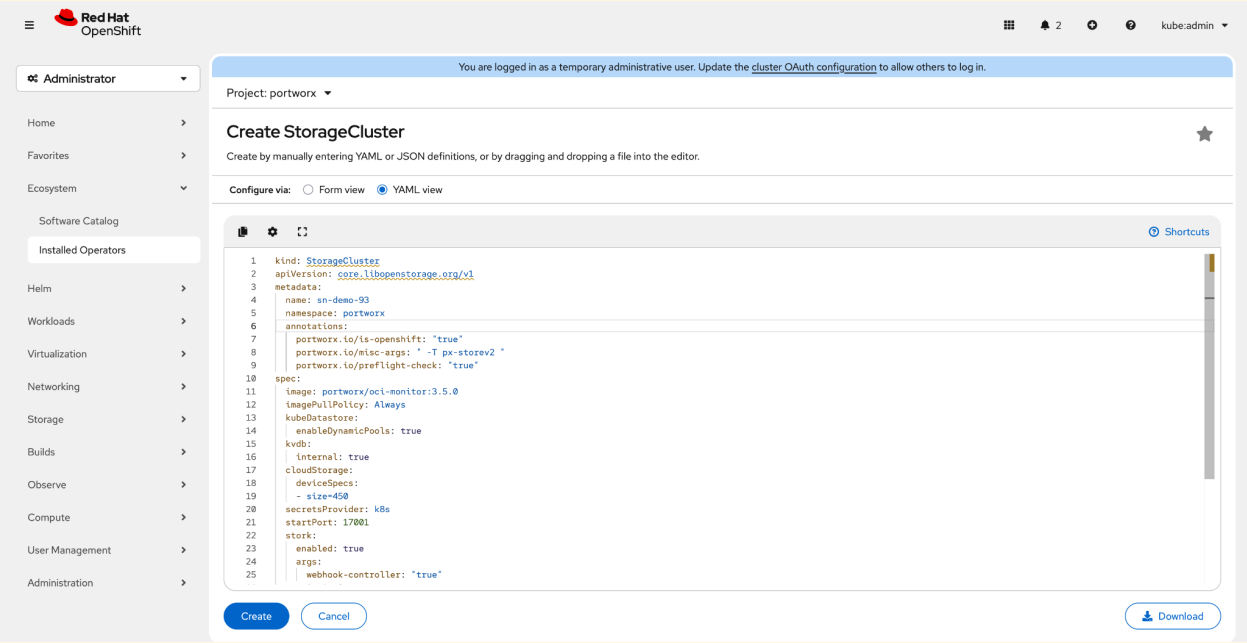

Figure 22. Creating a storage cluster in OpenShift Console

***Note**: Portworx images can be downloaded and stored in a local repository before installation in an air-gapped environment.*

An example storage cluster configuration file created from Portworx Central can be found below and can be edited to meet your environment's needs. Administrators should always create a new spec from Portworx Central for deploying a storage cluster. The information below is for informational purposes only.

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
 name: px-cluster-OMITED
 namespace: portworx
   portworx.io/is-openshift: "true"
   portworx.io/misc-args: " -T px-storev2 "
spec:
 deleteStrategy:
   type: UninstallAndWipe
 image: docker.io/portworx/oci-monitor:3.6.0
 imagePullPolicy: Always
 security:
   enabled: true
 kvdb:
   internal: true
 storage:
   devices:
     - /dev/sdb
   journalDevice: auto
   systemMetadataDevice: /dev/sdc
 network:
   dataInterface: br-ex
   mgmtInterface: ens224
 secretsProvider: k8s
 startPort: 17001
 stork:
   enabled: true
   args:
     webhook-controller: "true"
 autopilot:
   enabled: true
 csi:
   enabled: true
 monitoring:
   telemetry:
     enabled: false
   prometheus:
     enabled: false
     exportMetrics: true
```

Note the following details on the example specification above that matches the recommendations from this reference architecture.

This section of the cluster config enables security and authorization for the storage cluster.

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

The storage cluster is using an internal etcd instance for a KVDB. This can be configured with the “internal: true” key-value pair. If using an external etcd cluster, each of the endpoints for the etcd cluster can be listed under the kvdb specification.

```
spec:
 kvdb:
   internal: true
...
```

The following drives are used as storage devices on our nodes to form the storage cluster. In this instance we only have a single drive used for the node. Additional devices can be added as needed for your environment. It also uses this same drive for the journal device since it's configured for auto. The journal device can be specified manually here if a 3GB or larger device with the same or better storage characteristics can be used. Lastly, a metadata device to use is specified.

The storage section of the Portworx storage cluster specification may look different if using CloudDrives. In those instances, device sizes will be specified instead of device identifiers.

```
storage:
...
   devices:
     - /dev/sdb

   journalDevice: auto
   systemMetadataDevice: /dev/sdc

...
```

The networking was specified to use a different NIC for the storage replication network vs the management network. This is used to segment storage traffic on a different storage network to avoid network congestion/contention.

```
spec:
...
   network:
      dataInterface: ens224
      mgmtInterface: br-ex
...
```

Telemetry and Monitoring are enabled.

```
spec:
...
 monitoring:
   telemetry:
     enabled: true
   prometheus:
     enabled: false
     exportMetrics: true
...
```

If the cluster architecture is disaggregated, the storage cluster config should have two sections for nodes. Each section correlates with the storage or storage less nodes since storage less nodes don't need backing disks. An example would be the snippet below where we've identified the storage and storageless nodes based on a label, which would be set in the OpenShift cluster's machine set, discussed earlier in this reference architecture.

```
spec:
 image: docker.io/portworx/oci-monitor:3.6.0
 storage:
   devices:
   - /dev/sdb
 nodes:
 - selector:
     labelSelector:
       matchLabels:
         portworx.io/node-type: "storage"
   storage:
     devices:
     - /dev/sdb

 - selector:
     labelSelector:
       matchLabels:
         portworx.io/node-type: "storageless"
   storage:
     devices: []
```

Monitoring During Installation

Ensuring a smooth and successful installation of Portworx on Red Hat OpenShift bare metal nodes includes monitoring throughout the process. This section outlines the steps and tools needed to oversee the installation from initial setup to final deployment.

Portworx

To monitor the status of the Portworx storage cluster, the pods in the portworx namespace can be monitored. The command below will show all the pods being deployed for the storage cluster. These pods may show up in a failed state and restart for a few minutes until other pods in the cluster are available.

```
oc get pods -n portworx
```

The storage cluster pods will take some time to initialize the disks and build the cluster.

```
Every 2.0s: kubectl get pods -n portworx

NAME                                                READY   STATUS    RESTARTS      AGE
autopilot-6c868cbf74-vqvmf                          1/1     Running   0             2m29s
portworx-api-dg98v                                  1/2     Running   2 (31s ago)   2m3s
portworx-api-kds2c                                  0/2     Error     1 (60s ago)   2m3s
portworx-api-shvqf                                  1/2     Running   2 (25s ago)   2m3s
portworx-operator-85c89c8b75-sf8t6                  1/1     Running   0             5m36s
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-h564k   0/1     Running   0             2m13s
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-szs7g   0/1     Running   0             2m13s
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-tft7f   0/1     Running   0             2m13s
px-csi-ext-5b995d6f7d-6rcl2                         0/4     Pending   0             2m25s
px-csi-ext-5b995d6f7d-q79kb                         0/4     Pending   0             2m25s
px-csi-ext-5b995d6f7d-s8wvw                         0/4     Pending   0             2m25s
px-plugin-99597c6-62s55                             1/1     Running   0             2m13s
px-plugin-99597c6-t49qt                             1/1     Running   0             2m13s
px-plugin-proxy-79d5ffc6df-mn8sw                    1/1     Running   0             2m13s
stork-54c67c747c-8p685                              1/1     Running   0             2m48s
stork-54c67c747c-jt7h8                              1/1     Running   0             2m48s
stork-54c67c747c-xcwv5                              1/1     Running   0             2m48s
stork-scheduler-57bff8bc76-4sp6w                    1/1     Running   0             2m47s
stork-scheduler-57bff8bc76-drpzb                    1/1     Running   0             2m48s
stork-scheduler-57bff8bc76-vzffx                    1/1     Running   0             2m47s
```

Figure 23. Portworx Storage cluster deploying

When all of the pods are running, you can proceed to the Post-installation Validation section below to ensure the cluster is running in a healthy state.

```
NAME                                                READY   STATUS
autopilot-6c868cbf74-vqvmf                          1/1     Running
portworx-api-dg98v                                  2/2     Running
portworx-api-kds2c                                  2/2     Running
portworx-api-shvqf                                  2/2     Running
portworx-operator-85c89c8b75-sf8t6                  1/1     Running
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-h564k   1/1     Running
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-szs7g   1/1     Running
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-tft7f   1/1     Running
px-csi-ext-5b995d6f7d-6rcl2                         4/4     Running
px-csi-ext-5b995d6f7d-q79kb                         4/4     Running
px-csi-ext-5b995d6f7d-s8wvw                         4/4     Running
px-plugin-99597c6-62s55                             1/1     Running
px-plugin-99597c6-t49qt                             1/1     Running
px-plugin-proxy-79d5ffc6df-mn8sw                    1/1     Running
px-telemetry-phonehome-dznc2                        2/2     Running
px-telemetry-phonehome-lzldc                        2/2     Running
px-telemetry-phonehome-vlwcm                        2/2     Running
px-telemetry-registration-85ff874df7-97wm2          2/2     Running
stork-54c67c747c-8p685                              1/1     Running
stork-54c67c747c-jt7h8                              1/1     Running
stork-54c67c747c-xcwv5                              1/1     Running
stork-scheduler-57bff8bc76-4sp6w                    1/1     Running
stork-scheduler-57bff8bc76-drpzb                    1/1     Running
stork-scheduler-57bff8bc76-vzffx                    1/1     Running
```

Figure 24. Portworx Post-installation Objects

If you need to troubleshoot the deployment, you may tail the logs of the px-cluster pods in the portworx namespace. These pods will show details about what is happening when trying to initialize the storage cluster. This includes benchmarking drives, configuring the network connections, and creating the storage pools.

```
oc logs -n portworx [px-cluster-pod-name-here]
```

Post-installation Validation

After successfully installing Portworx on your Red Hat OpenShift bare metal nodes, it is important to perform a series of validation checks to ensure the deployment is fully operational and configured correctly. This section provides a comprehensive guide to the post-installation validation process, detailing essential steps such as verifying pod statuses, checking storage pools, and confirming data replication. By following these validation procedures, you can identify and address any potential issues early, ensuring that your Portworx deployment is reliable, efficient, and ready to handle your storage needs.

Portworx Validation

Once your Portworx Storage Cluster pods have been deployed and are in a running and ready state, you can check the status of the storage cluster by using the pxctl command line tool that is installed within the storage cluster pods.

First, you must obtain a Token with permissions to run pxctl commands against the storage cluster. You may skip this step if you did not enable "security" when deploying your storage cluster.

```
ADMIN_TOKEN=$(oc -n portworx get secret px-admin-token --template='{{index .data "auth-token" | base64decode}}')
```

Once the Admin token has been obtained, run the following command to find one of the Portworx Storage Cluster nodes and put it into an environment variable.

```
PX_POD=$(oc get pods -l name=portworx -n portworx -o jsonpath="{.items[0].metadata.name}")
```

We'll use the Pod and the Admin token to create a cluster context. To do this run the command below:

```
oc -n portworx exec -ti $PX_POD -- /opt/pwx/bin/pxctl context create admin --token=$ADMIN_TOKEN
```

Finally, you can exec into the Pod with the pxctl command line utility to query the Portworx storage cluster with a pxctl status command.

```
oc -n portworx exec -ti $PX_POD -- /opt/pwx/bin/pxctl status
```

```
Status: PX is operational
Telemetry: Healthy
Metering: Disabled or Unhealthy
License: Trial (expires in 30 days)
Node ID: 6e1f8558-31e0-4b59-b9a1-8a3bddbfaa9f
        IP: 10.38.15.217
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE    REGION
        0       HIGH            raid0           231 GiB 75 GiB  Online  default default
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:0     /dev/sdb        STORAGE_MEDIUM_SSD      250 GiB         09 Apr 26 19:16 UTC
        total                   -                       250 GiB
        Cache Devices:
         * No cache devices
        Journal Device:
        1       /dev/sdc1       STORAGE_MEDIUM_SSD      3.0 GiB
        Metadata Device:
        1       /dev/sdc2       STORAGE_MEDIUM_SSD      67 GiB
        * Internal kvdb on this node is using this dedicated metadata device to store its data.
Cluster Summary
        Cluster ID: ocp-cluster-98
        Cluster UUID: 96743afe-7df1-40b4-bf7f-e76d801afc0f
        Scheduler: kubernetes
        Total Nodes: 3 node(s) with storage (3 online)
        IP              ID                                      SchedulerNodeName                       Auth    StorageNode     Used    Capacity        Status  StorageStatus   Version         Kernel          O
S
        10.38.4.66      c3f4683c-6210-4bb1-a703-7f6253c59beb    pwx-ocp-0-171-6jr44-worker-0-kbb69      Enabled Yes(PX-StoreV2) 75 GiB  231 GiB         Online  Up              3.6.0.0-a81cf43 5.14.0-570.99.1.el9
_6.x86_64       Red Hat Enterprise Linux CoreOS 9.6.20260314-0 (Plow)
        10.38.15.217    6e1f8558-31e0-4b59-b9a1-8a3bddbfaa9f    pwx-ocp-0-171-6jr44-worker-0-x7wm9      Enabled Yes(PX-StoreV2) 75 GiB  231 GiB         Online  Up (This node)  3.6.0.0-a81cf43 5.14.0-570.99.1.el9
_6.x86_64       Red Hat Enterprise Linux CoreOS 9.6.20260314-0 (Plow)
        10.38.13.145    5d0c435e-42c7-4a8f-b94c-83ad67b4e7ef    pwx-ocp-0-171-6jr44-worker-0-dj98c      Enabled Yes(PX-StoreV2) 75 GiB  231 GiB         Online  Up              3.6.0.0-a81cf43 5.14.0-570.99.1.el9
_6.x86_64       Red Hat Enterprise Linux CoreOS 9.6.20260314-0 (Plow)
Global Storage Pool
        Total Used      :  225 GiB
        Total Capacity  :  693 GiB
```

Figure 25. PXCTL status

To ensure your Portworx installation is successful, consult "Verify Your Portworx Installation" in the Portworx documentation for detailed command instructions. Once you confirm that Portworx is installed correctly, you can proceed to create your first Persistent Volume Claim (PVC).

Workload and Volume Considerations

When designing and deploying applications on Portworx Enterprise, understanding the nuances of workload and volume management is important for performance, storage costs, and reliability. This section delves into key considerations for managing workloads and storage volumes, with a particular focus on the differences between in-app replication and storage array replication. By exploring these approaches, we will provide insights into their respective benefits, use cases, and potential challenges, enabling you to make informed decisions to best support your application's storage needs.

In-app Replication vs Portworx Replication Factor

Data availability is a critical aspect of any storage and application infrastructure, ensuring that data is accessible whenever needed, despite hardware failures or other disruptions. High data availability is essential for maintaining business continuity, minimizing downtime, and ensuring that applications run smoothly without interruption. There are several ways to accomplish this but in all cases it involves having extra copies of your data in different fault domains. Using Portworx as your storage platform opens up more options for high data availability.

Portworx storage classes enable end-users to deploy workloads with multiple replicas, which are complete copies of the data stored on different storage nodes within the cluster. Each replica ensures data redundancy and availability. Storage classes can be configured with repl1, repl2, or repl3, where repl1 indicates a single copy of the data with no redundancies, repl2 provides two copies, and repl3 ensures three copies of the data.

The decision around when to use which replication factor comes down to the service level agreements around the availability of your applications, the cost to house multiple copies of the data, and what capabilities the backing disks for the storage pool contain. Under most circumstances Portworx recommends using repl2 or repl3 to provide data availability for your applications. This decision does mean that there are multiple copies of your data spread across nodes, and thus uses two or three times the amount of disk space. However, if the backing disks are provided by a storage array such as a FlashArray, deduplication can be enabled to remove this concern.

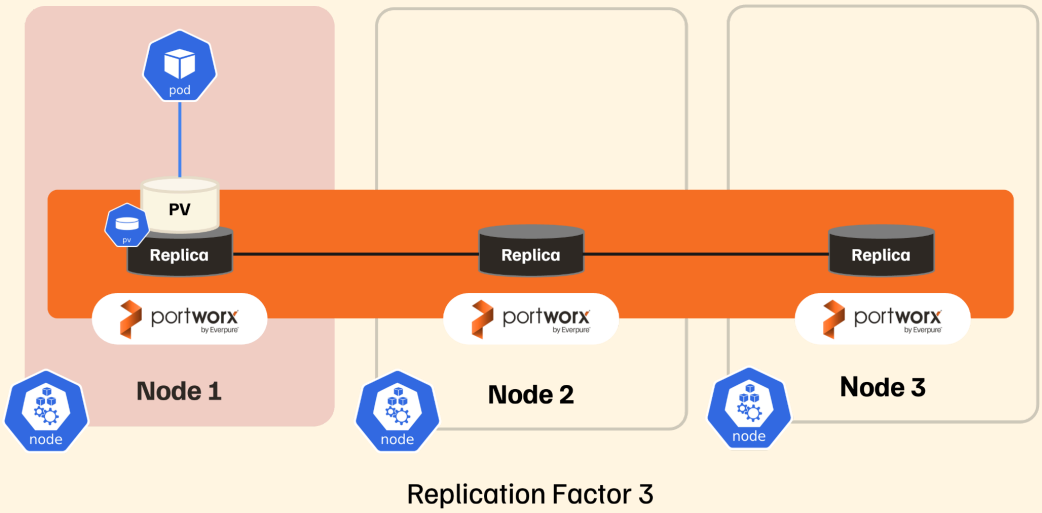

Figure 26. Portworx replication factor

In other cases, the applications themselves might perform replication on their own. Some databases work as part of a cluster and replicate the data above the storage layer. In this case there is already a copy of your data at the application level so using repl1 at the storage layer is sufficient. In this scenario the application (such as a database) is responsible for the high availability of the data and not the Portworx storage layer. It is possible to use repl 2 or repl 3 with app based replication, but you're increasing the number of copies of your data that exist.

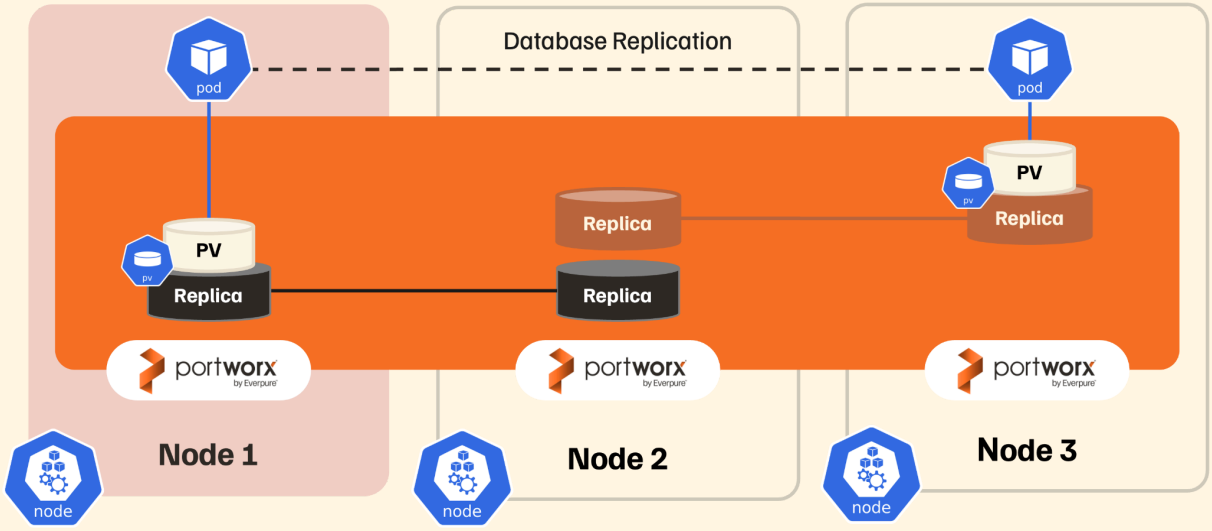

Figure 27. Portworx Replication with Application Replication

PX-Fast

PX-Fast allows more direct access to the underlying storage device using an accelerated I/O path. It is optimized for workloads requiring low latencies that provide their own application-level resilience. PX-Fast is designed to be a replacement for workloads that would benefit from direct access to local NVMe devices, but unlike a local NVMe device, it supports Portworx features such as Application I/O Control, Volume Snapshots, and Disaster Recovery making it a better choice than utilizing raw NVMe storage.

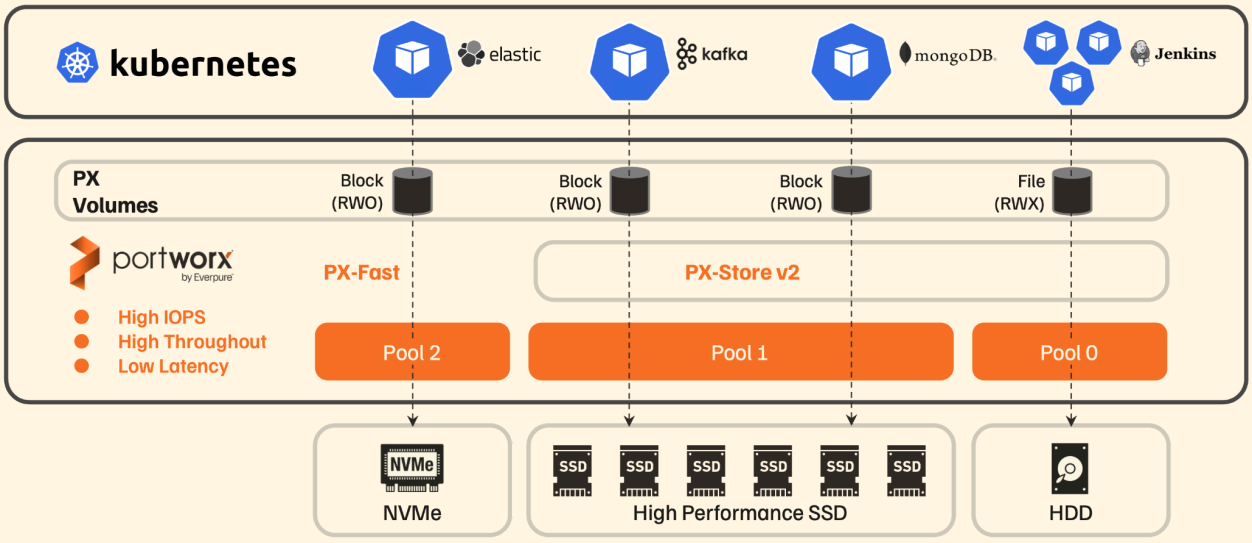

Figure 28. PX-Fast Direct Access

PX-Fast Requirement

PX-Fast has a few requirements and design considerations:

- **PX-StoreV2**: PX-Fast requires that PX-StoreV2 is deployed.
- **Replica 1**: PX-Fast requires that the storage class parameter repl be set to 1.
- **Locality**: Applications that utilize PX-Fast volumes must run on the nodes where the volumes reside. This can be enforced by STORK using the stork.libopenstorage.org/preferLocalNodeOnly annotation.

Example of following annotation set in deployment:

```
spec.template

annotations:
    stork.libopenstorage.org/preferLocalNodeOnly: "true"
```

Data Locality

Data locality refers to the practice of storing data close to where it is processed to minimize latency and improve performance. In a distributed storage system like Portworx, data locality is a key consideration for optimizing application performance and resource utilization.

Portworx will attempt to place replicas on the same nodes where the Kubernetes application is running to provide the best performance. This is completed with a tool maintained by Portworx, called Storage Orchestrator Runtime for Kubernetes, or STORK. There are other factors to consider with data locality and placement including the fault domains explained earlier in this reference architecture, but manual considerations can also be taken into account.

Portworx has affinity rules to keep replicas co-located on the same nodes. This might be important for an application mounting two volumes where data is being copied from one persistent volume to another. Using an affinity rule ensures that this data doesn't need to be replicated across nodes which might increase latency.

Similarly, the ability to keep volumes on different nodes is also possible with anti-affinity rules. Perhaps if you're using a replication factor of 1 with database replication, it might be important to keep those two volumes on separate nodes so a hardware failure doesn't affect both volumes for the same application simultaneously.

Affinity and anti-affinity rules can be configured on a per volume basis with the pxctl command line utility, or at the Kubernetes storage class.

Scaling

Scaling a Red Hat OpenShift and Portworx cluster is crucial for maintaining performance, availability, and efficient resource utilization in dynamic environments. This involves adjusting the number of nodes in the cluster and the storage capacity managed by Portworx to meet changing workloads and demands. While application scaling is important, ensuring that the underlying infrastructure can scale effectively is essential for supporting those applications. By implementing robust scaling strategies, you can optimize resource usage, reduce costs, and ensure that your OpenShift and Portworx clusters remain responsive and resilient, even during peak usage times.

Red Hat OpenShift can scale horizontally by adding more nodes to a cluster, which is ideal for handling increased workloads helping to ensure high availability. Vertical scaling involves increasing the resources of existing nodes and can be used for workloads that require more compute or memory. It is important to monitor cluster performance and resource utilization regularly so that future needs can be met. For more details, reference the Red Hat OpenShift documentation for their scaling guide.

Note: Red Hat OpenShift documentation may require a login.

Portworx

There are several objects that might need to be scaled during the normal course of operating Portworx on Red Hat OpenShift. This section will break these down into two categories: persistent volumes and the storage cluster.

Persistent Volume Scaling

Applications often require more storage over time. Whether this is more data in a database, more log files, or other artifacts being generated over time, administrators need to account for what happens when a persistent volume in Kubernetes runs out of capacity. Portworx allows users to modify the size of persistent volumes manually or automatically.

Persistent volumes created by Portworx through a storage class can be manually resized assuming the storage class has the parameter "allowVolumeExpansion: true" configured and the PVC must be in use by a pod. If these conditions are true, you can use the following Kubernetes command to resize the persistent volume claim.

```
kubectl edit pvc mssql-data
```

When complete you should see a VolumeResizeSuccessful message in the PVC object.

```
Normal  VolumeResizeSuccessful  5s    volume_expand      ExpandVolume succeeded for volume
                                                          default/example-pvc
```

The ability to manually resize a persistent volume is important, but Portworx instead recommends using the Portworx AutoPilot feature to automatically resize volumes when they get low on disk space. This prevents an issue where OpenShift administrators aren't available immediately to resize a disk that is quickly running out of space, and reduces the operational overhead of bespoke updates to the cluster.

Installing AutoPilot involves deploying a configuration spec with a url to the thanos-querier route. This is needed because AutoPilot needs access to monitoring information stored in Prometheus to understand how much capacity a persistent volume has available.

An example Autopilot spec is below:

```
...
spec:
  autopilot:
    enabled: true
    image: <autopilot-image>
    providers:
    - name: default
      params:
        url: https://<THANOS-QUERIER-HOST>
      type: prometheus
...
```

If your Portworx StorageCluster was configured with Security, you'll also need to modify the storage cluster to include the PX_SHARED_SECRET information.

Example StorageCluster Config Snippet:

```
 autopilot:
...
    env:
    - name: PX_SHARED_SECRET
      valueFrom:
        secretKeyRef:
          key: apps-secret
          name: px-system-secrets
```

***Note**: To deploy AutoPilot in an air gapped environment, the autopilot images must be downloaded and placed in a local image registry like OpenShift Container Registry.*

For full installation instructions for AutoPilot, consult the Portworx documentation.

Once AutoPilot is installed and configured, Administrators can create rules that dictate how and when persistent volumes should be expanded. An AutoPilot rule has four main sections:

Section	Description
Selector	A key value pair (tag) on an object that AutoPilot should be monitoring such as a PVC.
Namespace Selector	A key value pair (tag) on the namespaces where AutoPilot should monitor.
Condition	The metric that should trigger an AutoPilot action to run.
Action	Defines what action to take when the conditions are met.

Table 8. AutoPilot configuration elements

If a PVC with the selector tag applied, is also in a namespace with the namespace selector applied, and it matches the condition, the action will be applied. So autopilot can automatically resize your volumes when free space becomes too low.

An example AutoPilot rule can be found below.

```
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: volume-resize
spec:
  selector:
    matchLabels:
      autoresize: true
  namespaceSelector:
    matchLabels:
      resize: true
  conditions:
    expressions:
    - key: "100 * (px_volume_usage_bytes / px_volume_capacity_bytes)"
      operator: Gt
      values:
        - "80"
  actions:
  - name: openstorage.io.action.volume/resize
    params:
      scalepercentage: "100"
      maxsize: "400Gi"
```

The autopilot rule above applies to any PVCs with an autoresize: true tag, in a namespace with a resize: true tag, and is using greater than 80% of the PVC's total capacity. The volume will be resized 100% (or doubled) with a maximum size of 400Gi.

Portworx recommends setting a maxsize for scaling so that volumes don't continually resize themselves in perpetuity, using up all the storage in the cluster. Applications that need resized continuously, likely have an issue that needs to be investigated by an Administrator to find out why it's using up so much capacity.

Please see the Portworx documentation for further information about installation and usage of AutoPilot rules in your OpenShift cluster.

Storage Cluster Scaling

Over time, the initial capacity provisioned to the Portworx Storage cluster may not be adequate to support the addition of new applications or application growth. If this occurs there are several ways to expand the Portworx storage cluster:

- **Resize backing disks**: The easiest way to increase the overall size of the Portworx storage cluster is increasing the size of the backing drives. If your backing devices come from a hardware storage array, vertically scaling the volumes or LUNs presented to the Portworx storage nodes is the preferred way to expand the volumes. This is because no data movement has to happen to rebalance the cluster. PX-StoreV2 does not support online pool resize; additionally, the maximum supported pool resize capacity is limited to 15 TB.
- **Expand the cluster**: In a hyper-converged cluster, every node in the cluster also provides storage. So another way to expand the storage cluster size is to horizontally scale the nodes in the cluster. In this model, be sure to expand the cluster across availability zones and in numbers that coincide with your preferred replication factor. For example, adding a single node to a cluster constrained by storage capacity, where applications use repl3 might not alleviate storage pressure because there isn't room for the replicas. In this case you'd want to add three nodes to the cluster.

Additional storage nodes can be added for a disaggregated storage cluster as well by adding additional OpenShift worker nodes to the cluster and tagging them with portworx.io/node-type: storage labels to identify that they will participate in Portworx the storage cluster.

Backup and Disaster Recovery

When storing data for production environments, it is imperative to implement backup and recovery strategies to protect against data loss and ensure business continuity. In a Kubernetes environment we can think of the cluster as ephemeral and not a critical item to protect. However, the applications, their metadata, and their persistent data must be protected from accidental deletions, site failures, and natural or manmade disasters.

While disaster recovery and data protection are critical concerns to address for any production environment they are outside the scope of this reference architecture. For information about design decisions for backup and disaster recovery please see the Portworx Backup and Disaster Recovery addendum document which compliments this reference architecture.

Upgrading

Upgrading Portworx on an OpenShift cluster involves two distinct components: the OpenShift cluster itself and the Portworx storage cluster. Each component has specific requirements for performing an in-place upgrade, including hardware specifications, kernel versions, and Kubernetes versions. Understanding and meeting these requirements is crucial to ensure a smooth and successful upgrade process without downtime.

Red Hat OpenShift

Before upgrading Red Hat OpenShift, it is essential to verify that the new version is compatible with Portworx. In some instances, you may need to upgrade Portworx before proceeding with the OpenShift upgrade. To ensure seamless operation of your OpenShift cluster with Portworx during and after the upgrade, please follow these best practices:

- **Kernel compatibility**: If the new OpenShift version includes a new kernel, ensure that the current version of Portworx is compatible with this kernel version.
- **Kubernetes compatibility**: If the new OpenShift version includes a new Kubernetes version, verify that the currently deployed version of Portworx is compatible with it.
- **OpenShift compatibility**: Confirm that the deployed version of Portworx is compatible with the new version of OpenShift you plan to use.
- **Cluster health**: Ensure that the Portworx cluster is healthy.
- **KVDB health**: Verify that the Portworx KVDB is healthy and that all three instances of KVDB are up and running.

For Red Hat OpenShift upgrade instructions please consult the official Red Hat OpenShift documentation.

Portworx

As with an OpenShift upgrade, it's important to check version compatibility with the underlying Red Hat OpenShift cluster. Ensure that the Kubernetes version, OpenShift version, and OS Kernel versions are supported prior to upgrading Portworx.

Before starting the upgrade process ensure that the Portworx deployment is healthy. As a pre-upgrade step run this command to check the status of all of the Portworx pods before attempting an upgrade.

```
oc get pods -n portworx
```

The results should show pods all in a running and ready status.

```
→  git oc get pods -n portworx
NAME                                                    READY   STATUS
autopilot-6c868cbf74-vqvmf                              1/1     Running
portworx-api-dg98v                                      2/2     Running
portworx-api-kds2c                                      2/2     Running
portworx-api-shvqf                                      2/2     Running
portworx-operator-85c89c8b75-sf8t6                      1/1     Running
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-h564k   1/1     Running
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-szs7g   1/1     Running
px-cluster-93aab04c-3b18-4793-adfa-62309b8281b8-tft7f   1/1     Running
px-csi-ext-5b995d6f7d-6rcl2                             4/4     Running
px-csi-ext-5b995d6f7d-q79kb                             4/4     Running
px-csi-ext-5b995d6f7d-s8wvw                             4/4     Running
px-plugin-99597c6-62s55                                 1/1     Running
px-plugin-99597c6-t49qt                                 1/1     Running
px-plugin-proxy-79d5ffc6df-mn8sw                        1/1     Running
px-telemetry-phonehome-dznc2                            2/2     Running
px-telemetry-phonehome-lzldc                            2/2     Running
px-telemetry-phonehome-vlwcm                            2/2     Running
px-telemetry-registration-85ff874df7-97wm2              2/2     Running
stork-54c67c747c-8p685                                  1/1     Running
stork-54c67c747c-jt7h8                                  1/1     Running
stork-54c67c747c-xcwv5                                  1/1     Running
stork-scheduler-57bff8bc76-4sp6w                        1/1     Running
stork-scheduler-57bff8bc76-drpzb                        1/1     Running
stork-scheduler-57bff8bc76-vzffx                        1/1     Running
```

Figure 29. Portworx objects before upgrades

If any pod is not in this state please fix the pod(s) before starting the upgrade. Check the Troubleshooting section of the Portworx official documentation or contact Portworx support for further assistance.

Just as with the post-installation steps, you must obtain a token with permissions to run pxctl commands against the storage cluster. You may skip this step if you did not enable “security” when deploying your storage cluster.

```
ADMIN_TOKEN=$(oc -n portworx get secret px-admin-token --template='{{index .data "auth-token" |
base64decode}}')
```

Once the Admin token has been obtained, run the following command to find one of the Portworx Storage Cluster nodes and put it into an environment variable.

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

We'll use the Pod and the Admin token to create a cluster context. To do this run the command below.

```
oc -n portworx exec -ti $PX_POD -- /opt/pwx/bin/pxctl context create admin --token=$ADMIN_TOKEN
```

Finally, you can exec into the Pod with the pxctl command line utility to query the Portworx Storage Cluster with a pxctl status command.

```
oc -n portworx exec -ti $PX_POD -- /opt/pwx/bin/pxctl status
```

```
Status: PX is operational
Telemetry: Healthy
Metering: Disabled or Unhealthy
License: Trial (expires in 30 days)
Node ID: 6e1f8558-31e0-4b59-b9a1-8a3bddbfaa9f
        IP: 10.38.15.217
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE    REGION
        0       HIGH            raid0           231 GiB 75 GiB  Online  default default
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:0     /dev/sdb        STORAGE_MEDIUM_SSD      250 GiB         09 Apr 26 19:16 UTC
        total                   -                       250 GiB
        Cache Devices:
         * No cache devices
        Journal Device:
        1       /dev/sdc1       STORAGE_MEDIUM_SSD      3.0 GiB
        Metadata Device:
        1       /dev/sdc2       STORAGE_MEDIUM_SSD      67 GiB
        * Internal kvdb on this node is using this dedicated metadata device to store its data.
Cluster Summary
        Cluster ID: ocp-cluster-98
        Cluster UUID: 96743afe-7df1-40b4-bf7f-e76d801afc0f
        Scheduler: kubernetes
        Total Nodes: 3 node(s) with storage (3 online)
        IP              ID                                      SchedulerNodeName                       Auth    StorageNode     Used    Capacity        Status  StorageStatus   Version         Kernel          O
S
        10.38.4.66      c3f4683c-6210-4bb1-a793-7f6253c59beb    pwx-ocp-0-171-6jr44-worker-0-kbb69      Enabled Yes(PX-StoreV2) 75 GiB  231 GiB         Online  Up              3.6.0.0-a81cf43 5.14.0-570.99.1.el9
_6.x86_64       Red Hat Enterprise Linux CoreOS 9.6.20260314-0 (Plow)
        10.38.15.217    6e1f8558-31e0-4b59-b9a1-8a3bddbfaa9f    pwx-ocp-0-171-6jr44-worker-0-x7wm9      Enabled Yes(PX-StoreV2) 75 GiB  231 GiB         Online  Up (This node)  3.6.0.0-a81cf43 5.14.0-570.99.1.el9
_6.x86_64       Red Hat Enterprise Linux CoreOS 9.6.20260314-0 (Plow)
        10.38.13.145    5d0c435e-42c7-4a8f-b94c-83ad67b4e7ef    pwx-ocp-0-171-6jr44-worker-0-dj98c      Enabled Yes(PX-StoreV2) 75 GiB  231 GiB         Online  Up              3.6.0.0-a81cf43 5.14.0-570.99.1.el9
_6.x86_64       Red Hat Enterprise Linux CoreOS 9.6.20260314-0 (Plow)
Global Storage Pool
        Total Used      :  225 GiB
        Total Capacity  :  693 GiB
```

Figure 30. Storage cluster status pre-upgrade

Similar to previous steps, ensure all Portworx nodes are online and errors or warnings are not displayed in the command above.

Next check all Portworx KVDB instances are running and healthy by running the command below.

```
oc -n portworx exec $PX_POD -- /opt/pwx/bin/pxctl sv kvdb members
```

```
Kvdb Cluster Members:
ID                                      PEER URLs                               CLIENT URLs                             LEADER  HEALTHY DBSIZE
6e1f8558-31e0-4b59-b9a1-8a3bddbfaa9f    [http://portworx-3.internal.kvdb:17015] [http://portworx-3.internal.kvdb:17016] false   true    1.4 MiB
5d0c435e-42c7-4a8f-b94c-83ad67b4e7ef    [http://portworx-1.internal.kvdb:17015] [http://portworx-1.internal.kvdb:17016] false   true    1.3 MiB
c3f4683c-6210-4bb1-a793-7f6253c59beb    [http://portworx-2.internal.kvdb:17015] [http://portworx-2.internal.kvdb:17016] true    true    1.3 MiB
```

Figure 31. List of KVDB members

All KVDB instances must be healthy and one of the nodes should be a leader.

If any pod is not in this state please fix the pod(s) before starting the upgrade. Check the Troubleshooting section of the Portworx official documentation or contact Portworx support for further assistance.

Portworx Operator Upgrade

After all prerequisites above have been completed the first component to upgrade is the Portworx operator. By default Red Hat OpenShift automatically upgrades the operator when a new version is released, but if you disable automatic upgrades then you have to manually upgrade the operator to the latest version. You can do that in the OpenShift web console:

1. Under the Operators dropdown, select the Installed Operators dashboard.
2. Select the Portworx project and approve the upgrade to the latest version.
3. Run the following command to confirm Portworx operator is running after the upgrade:

```
oc -n portworx get pod -l name=portworx-operator
```

The Portworx operator should be ready and in a running state before continuing with the upgrade process.

```
NAME                                READY   STATUS    RESTARTS      AGE
portworx-operator-85b959bb66-pm98w  1/1     Running   1 (32h ago)   32h
```

Figure 32. Portworx operator status

Once the operator has been upgraded, proceed to upgrade the Portworx Storage Cluster.

Portworx Storage Cluster Upgrade

Once you've upgraded the Operator, you're ready to begin upgrading Portworx and all its associated components. Portworx utilizes a rolling upgrade approach, upgrading one node at a time. It will only proceed to the next node after the previous one has been successfully upgraded. To upgrade Portworx, edit the StorageCluster resource and update the Portworx image.

```
oc edit -n portworx storagecluster
```

Modify the oci-monitor image version to the desired version of Portworx Enterprise.

For example:

"image: portworx/oci-monitor:3.1.2" would be changed to "image: docker.io/portworx/oci-monitor:3.6.0" if you were attempting to upgrade to Portworx 3.6.0.

```
spec:
...
  csi:
    enabled: true
    installSnapshotController: true
  image: docker.io/portworx/oci-monitor:3.6.0
  imagePullPolicy: Always
...
```

Once the storage spec has been modified with the desired version, the Portworx Level 5 Operator will do the work of upgrading the storage cluster, autopilot, and any other components necessary for the desired version.

***Note**: If this is an air-gapped installation, the new versions of the Portworx images must be pre-downloaded and placed into the local image registry.*

Logging and Monitoring

Effective logging and monitoring are fundamental to maintaining the health, performance, and security of an OpenShift cluster. These practices provide insights into the system's operation, enabling administrators to detect, diagnose, and resolve issues promptly. By continuously collecting and analyzing logs and metrics, organizations can ensure that their applications run smoothly, optimize resource utilization, and maintain compliance with regulatory requirements. Implementing standard logging and monitoring processes not only enhances operational efficiency but also fortifies the cluster's resilience against potential disruptions and security threats.

Portworx

All Portworx components run within the OpenShift cluster as containers and as such, Kubernetes Administrators can view logs for any of the components by interrogating pods through the oc logs command.

For example:

```
oc logs -n portworx [portworx-api-pod]
```

It is also possible to review these logs from the OpenShift Console by navigating to the specific pod and going to the Logs dashboard.

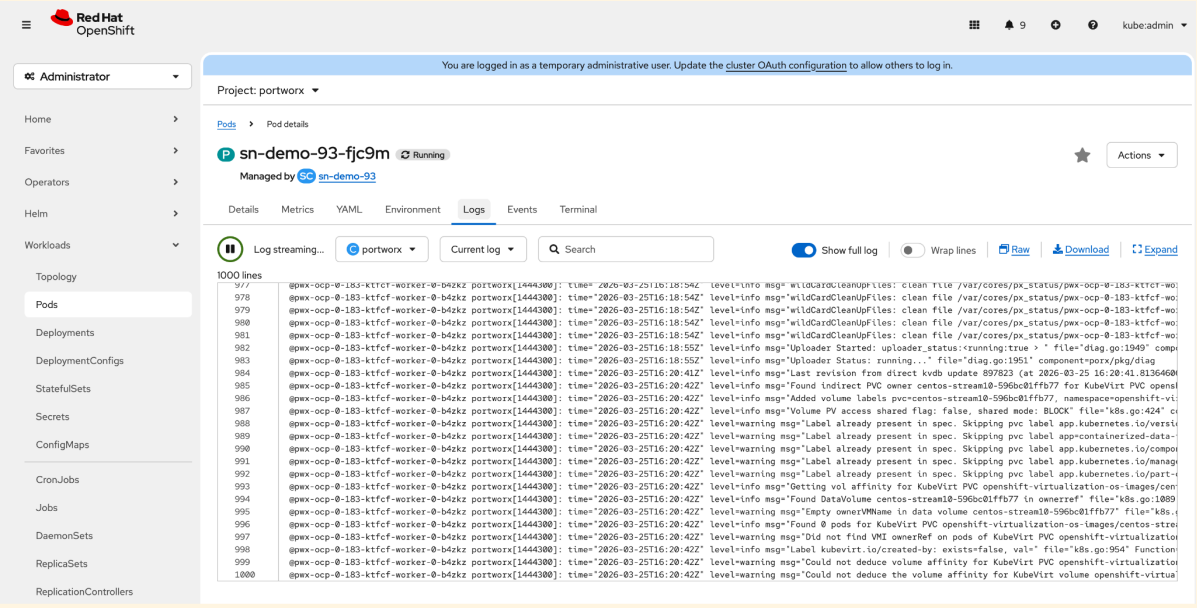

Figure 33. OpenShift console logs

If needed, or as advised by Portworx support personnel, you can increase log levels by updating the StorageCluster configuration for each component. For example, to enable debug level logging for the Stork component, you can add the spec.stork.args.verbose: true stanza to the StorageCluster resource.

```
spec
...
   stork:
      args:
         verbose: true
         webhook-controller: "true"
      enabled: true
...
```

To enable debug level in the Portwox container, add the PX_LOGLEVEL=debug environment variable to the StorageCluster specification:

```
spec
...
   env:
   - name: PX_LOGLEVEL
     value: debug
...
```

This will generate a tar.gz file that can be sent to Portworx support for investigation on issues. If you have Telemetry enabled the diags file is automatically sent to Pure1.

Summary

This reference architecture provides a comprehensive guide for deploying and managing bare metal OpenShift clusters with Portworx Enterprise, ensuring a robust, scalable, and high-performing infrastructure. The architecture emphasizes key considerations such as high availability, data locality, and efficient resource utilization. It outlines best practices for installation, configuration, and operational management, including crucial aspects like backup and recovery, monitoring, and logging. By following this reference architecture, organizations can confidently deploy Red Hat OpenShift and Portworx Enterprise on bare metal, achieving optimal performance, resilience, and reliability for their production workloads.

Legal Notice and Attributions