Portworx on Red Hat OpenShift Bare Metal Reference Architecture: OpenShift Virtualization Addendum

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

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

Table of Contents

Introduction

The purpose of this document is to serve as a supplemental guide to the Everpure® "Red Hat OpenShift with Portworx on Bare Metal" Reference Architecture. This addendum builds upon the foundational design outlined in the original base reference architecture, providing specific details and considerations for deploying and running Red Hat OpenShift Virtualization on Portworx® Enterprise. It aims to offer additional design guidance that complements the official documentation, helping organizations effectively integrate OpenShift Virtualization with Portworx in a bare metal environment. This document is intended to ensure a seamless, optimized deployment that leverages the strengths of both OpenShift Virtualization and Portworx, delivering robust performance, scalability, and reliability for virtualized workloads.

Target Use Cases

This addendum is designed to address key use cases for organizations looking to enhance their infrastructure by integrating OpenShift Virtualization with Portworx. The following scenarios represent ideal applications of this architecture, enabling businesses to modernize, streamline operations, and future-proof their environments by:

Modernizing traditional virtualization platforms: Organizations with existing traditional virtualization platforms can leverage this architecture to re-platform their workloads onto a modern, cloud-native infrastructure. By running virtual machines (VMs) within Red Hat OpenShift, you gain the flexibility and scalability of Kubernetes while maintaining the ability to support traditional VM-based applications. This approach allows you to modernize your infrastructure without sacrificing existing investments, positioning your organization for future innovation.

Innovating for future growth: This architecture empowers businesses to innovate today while laying the groundwork for future advancements. By adopting a unified platform that supports both containerized and virtualized workloads, you ensure that your infrastructure is ready to adapt to new technologies and business needs as they arise. The integration of OpenShift Virtualization with Portworx provides a consistent and scalable foundation for driving digital transformation and fostering innovation.

Creating consistent management across workloads: One of the primary benefits of integrating OpenShift Virtualization with Portworx is the ability to achieve consistency in managing both containerized and virtualized workloads. With consistent management, you can simplify operations, reduce the risk of errors, and improve overall system reliability. By unifying your virtualization and containerization strategies under a single platform, this architecture significantly increases operational efficiency. The streamlined deployment and management processes reduce the time and resources required to maintain your environment, allowing your team to focus on higher-value activities. Additionally, the built-in automation and scalability features of Portworx and OpenShift help optimize resource utilization, leading to cost savings and improved performance.

These target use cases demonstrate the versatility and value of integrating OpenShift Virtualization with Portworx, making it an ideal solution for organizations seeking to modernize their infrastructure and enhance operational capabilities.

Technology Overview

OpenShift Virtualization is built on the open-source KubeVirt project, which integrates the Linux kernel hypervisor KVM into Kubernetes, enabling virtual machines (VMs) to be managed as native Kubernetes objects. KVM is a mature and trusted hypervisor, embedded in the Red Hat Enterprise Linux kernel for many years. By incorporating a type-1 hypervisor directly into OpenShift clusters, organizations can seamlessly run VMs and containers side by side within a unified Kubernetes environment, eliminating the need to manage separate infrastructure for different types of workloads.

The virtualization capabilities within Red Hat OpenShift are leverages core Linux virtualization technologies such as KVM and QEMU:

- **Kernel-based virtual machine (KVM)** – KVM is a core virtualization module within the Linux kernel that transforms the Linux operating system into a type-1 (bare-metal) hypervisor. It enables multiple VMs to run on a single physical machine, each with its own isolated operating system. KVM provides the foundational framework for creating and managing virtual CPUs and memory, while relying on other tools like QEMU for hardware emulation and VM lifecycle management.
- **Quick EMUlator (QEMU)** – QEMU is a user-space emulator and virtualizer that, when combined with KVM, provides the necessary hardware emulation to run guest operating systems as VMs on a host machine. It emulates essential hardware components such as CPUs, disks, and network interfaces. With KVM, QEMU leverages hardware-assisted virtualization (e.g., Intel VT-x or AMD-V) to deliver near-native performance for guest OSes. It can also operate in software emulation mode without KVM, though this results in slower performance.
- **libvirt** – libvirt is an open-source virtualization API and management layer commonly used with KVM and QEMU. In OpenShift Virtualization, libvirt is used internally by KubeVirt components (such as the virt-launcher pod) to manage the lifecycle of virtual machines. However, unlike traditional virtualization environments where libvirt is directly exposed to administrators, OpenShift Virtualization leverages Kubernetes-native APIs and controllers for VM management, abstracting libvirt from end users.

To complement the virtualization capabilities provided by KVM, QEMU, and libvirt, Portworx Enterprise serves as an enterprise-grade storage solution that meets the needs of both containerized and virtualized workloads within OpenShift. Portworx offers robust management of persistent volumes for containers, as well as VM disks required for running VMs in OpenShift. Whether dealing with persistent data for container workloads or VM storage, Portworx delivers comprehensive data protection, business continuity and disaster recovery, capacity management, performance optimization, and security features including authorization and encryption. This unified storage solution ensures operational efficiency and consistency across both containerized and virtualized environments, supporting the demands of modern applications.

Design and Architecture

The overall architecture of running OpenShift Virtualization on a bare metal deployment with Portworx is designed to provide a unified platform where virtual machines (VMs) and containers run side by side, seamlessly integrated within the same Kubernetes cluster. This unification allows for greater flexibility and efficiency, enabling workloads to leverage the best of both worlds—virtual machines for traditional, stateful applications and containers for modern, cloud-native microservices. This document serves as an addendum to the Portworx on Red Hat OpenShift Bare Metal Reference Architecture and provides specific configuration and deployment guidance for environments utilizing Portworx version 3.6 , OpenShift 4.20 , and the OpenShift Migration Toolkit version 2.11.2

Note: *Portworx should be a minimum of version 3.5.2 to take advantage of capabilities designed for running virtual machines with OpenShift Virtualization.*

The architecture leverages the advanced storage capabilities of Portworx to ensure high availability, performance, and scalability for both VMs and containers, allowing for seamless migration, disaster recovery, and data management across a unified infrastructure.

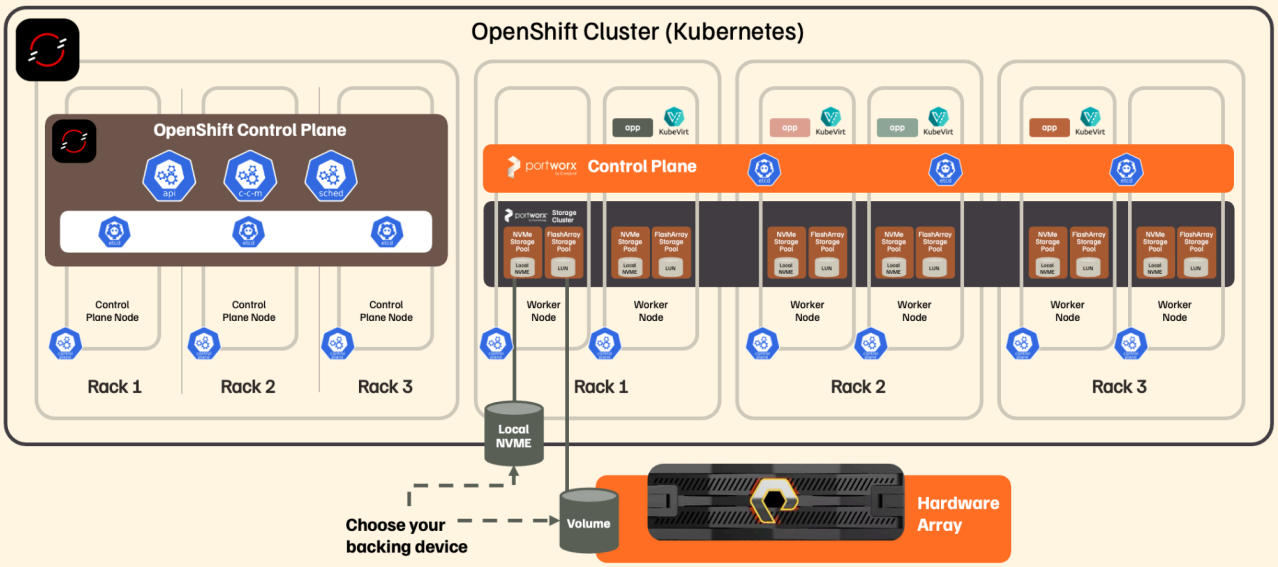

Figure 1. General Reference Architecture

Design Considerations

This section of the document provides considerations and situations that should be accounted for prior to deploying Red Hat OpenShift with Portworx Enterprise on bare metal nodes to operate OpenShift Virtualization virtual machines. Deploying such an advanced and integrated environment requires careful planning to ensure optimal performance, availability, reliability, and scalability. For tailored guidance and accurate sizing of your OpenShift Virtualization clusters with Portworx, contact your Portworx Account Executive to ensure all aspects are properly configured for a seamless deployment.

Properly sizing your infrastructure for the workloads it will run is a common but necessary consideration for bare metal workloads. This is no different for OpenShift Virtualization configured clusters since the physical infrastructure is shared by all virtual machines running in OpenShift. Basic sizing should include determining how much memory and compute can be used and oversubscribed, but also for storage. The storage considerations for sizing follow the instructions laid out in the Red Hat OpenShift on Bare Metal Reference Architecture as the considerations for virtual machine disks is a corollary to pods with persistent volumes. This section lays out additional considerations needed for sizing Portworx storage for virtual machines.

Block Size and bootloader compatibility

The VM disk configuration defaults to a 512-byte block size. The hypervisor makes it compatible so that 512-byte operations work over a provisioned Portworx storage disk with a 4096-byte block size. Therefore, no changes are needed because block size compatibility is ensured.

- Portworx block volumes always use a 4096-byte block size.
- VM disks default to a 512-byte block size unless otherwise specified. Specifying a logical block size of 512-byte and a physical block size of 4096-byte in the VM disk specification is an optional configuration detail within the VM that may help applications or file systems optimize performance, if supported.
- VM root disks also contain a bootloader, which can be either EFI or BIOS. BIOS supports booting only from disks with a 512-byte block size. EFI supports booting from disks with either a 4096-byte or 512-byte block size. They are independent bootloading mechanisms. The root disk configuration determines which bootloader is used.
    - You can identify the root disk configuration by examining the QCOW2 image or the disk partition table.
    - EFI requires an EFI system partition, a GPT partition table, and related components.
    - BIOS requires a partition marked as bootable.
    - For example, RHEL configures its QCOW2 cloud images to boot using both EFI and BIOS by creating two partitions that contain the required information. Note that not all distributions support this configuration.

Templates and Boot Images

Virtual machine templates are available by default when you enable OpenShift Virtualization and provide storage for the template catalog. The boot images used with those templates must also be accounted for when using Portworx storage.

Portworx recommends sizing boot image storage based on the total footprint of the OS/base images you plan to host (Red Hat-provided images plus any custom “golden” images) and the replication factor used for resiliency. Because image catalogs and custom image sets vary widely by customer, there is no single fixed capacity recommendation.

For example, *if you wanted to plan for roughly 500GB of image storage, Portworx recommends accounting for 1.5TB in disk space to store the default images from Red Hat and any other images up to this size with a replication factor of three for redundancy.*

Snapshots

When planning disk space for managing Portworx snapshots of virtual machines in your OpenShift Virtualization environment, it's important to understand how snapshots function and the storage they may require. Snapshots in Portworx are point-in-time captures of data volumes that efficiently allow for backup and restoration. However, they can consume a significant amount of disk space, depending on how much the data changes after each snapshot is taken.

The storage space used by snapshots largely depends on the rate at which your data changes. Snapshots only store the differences (deltas) between the current state and the previous snapshot, not full copies of the entire data volume. Therefore, if your virtual machines have a high rate of data change, snapshots will require more disk space.

To estimate the additional storage required for snapshots, you need to consider the rate of data change for your applications, the frequency of snapshots, how many snapshots will be retained and how long snapshots will be retained.

For example:

A virtual machine with a disk that is 30GB in size:

- with an hourly change rate of 10%,
- with an hourly snapshot schedule,
- storing a total of 7 snapshots

Would consume: 30GB(10% x 7) = 21GB

This means you would need an extra 21GB of disk space to store these snapshots.

Estimating these requirements can be challenging, especially for workloads with unpredictable or varying change rates. In such cases, leveraging the Portworx AutoPilot with Cloud Drive functionality for Everpure FlashArray is recommended. This approach allows you to start with an estimated amount of storage capacity for the cluster and automatically expand the storage as needed, ensuring you have sufficient space for snapshots and virtual machines without manual intervention.

Portworx Storage Classes

Portworx deploys several pre-configured StorageClasses after the cluster is created. Portworx recommends using the CSI provisioner (pxd.portworx.com) for persistent volume operations and OpenShift Virtualization integrations.

```
NAME                                PROVISIONER         RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
px-cdi-scratch                      pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-db                           pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-db-cloud-snapshot            pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-db-cloud-snapshot-encrypted  pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-db-encrypted                 pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-db-encrypted-pvc-k8s         pxd.portworx.com    Delete          Immediate              true                   13h
px-csi-db-local-snapshot            pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-db-local-snapshot-encrypted  pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-replicated                   pxd.portworx.com    Delete          Immediate              true                   45h
px-csi-replicated-encrypted         pxd.portworx.com    Delete          Immediate              true                   45h
px-rwx-block-kubevirt               pxd.portworx.com    Delete          WaitForFirstConsumer   true                   45h
px-rwx-file-kubevirt                pxd.portworx.com    Delete          WaitForFirstConsumer   true                   45h
stork-snapshot-sc                   stork-snapshot      Delete          Immediate              true                   45h
```

StorageClass	Primary Use
px-rwx-block-kubevirt	VM disks (block), high performance, live migration support **(Recommended)**
px-rwx-file-kubevirt	VM disks (file), shared access across VMs
px-cdi-scratch	Temporary storage for VM image import/clone (CDI)
px-csi-* (db, replicated, snapshot, encrypted variants)	General container workloads (databases, stateful apps)

Default Storage Class

A default StorageClass must be defined for either OpenShift Virtualization or OpenShift Container Platform to support VM provisioning. This ensures that VM workloads leverage storage optimized for performance, reliability, and overall user experience. When both defaults are configured, the OpenShift Virtualization StorageClass is given precedence for VM disk creation.

OpenShift Virtualization simplifies VM deployment by providing a catalog of preconfigured templates for operating systems such as CentOS, Red Hat Enterprise Linux, and Fedora. These templates include predefined settings and boot sources (OS disk images) required to initialize and run virtual machines.

To enable automatic download and availability of these boot sources, the platform relies on a default StorageClass that supports **ReadWriteMany (RWX) block access mode**. This allows multiple nodes to concurrently access the same persistent volume, which is required for managing shared VM boot images.

If the configured default StorageClass does not support RWX access, boot sources will not be automatically provisioned. In such cases, OpenShift Virtualization raises a **"CDIDefaultStorageClassDegraded"** alert, indicating that the storage configuration does not meet the requirements for automatic boot source management. In case of RWX Block volumes, you will not see this error.

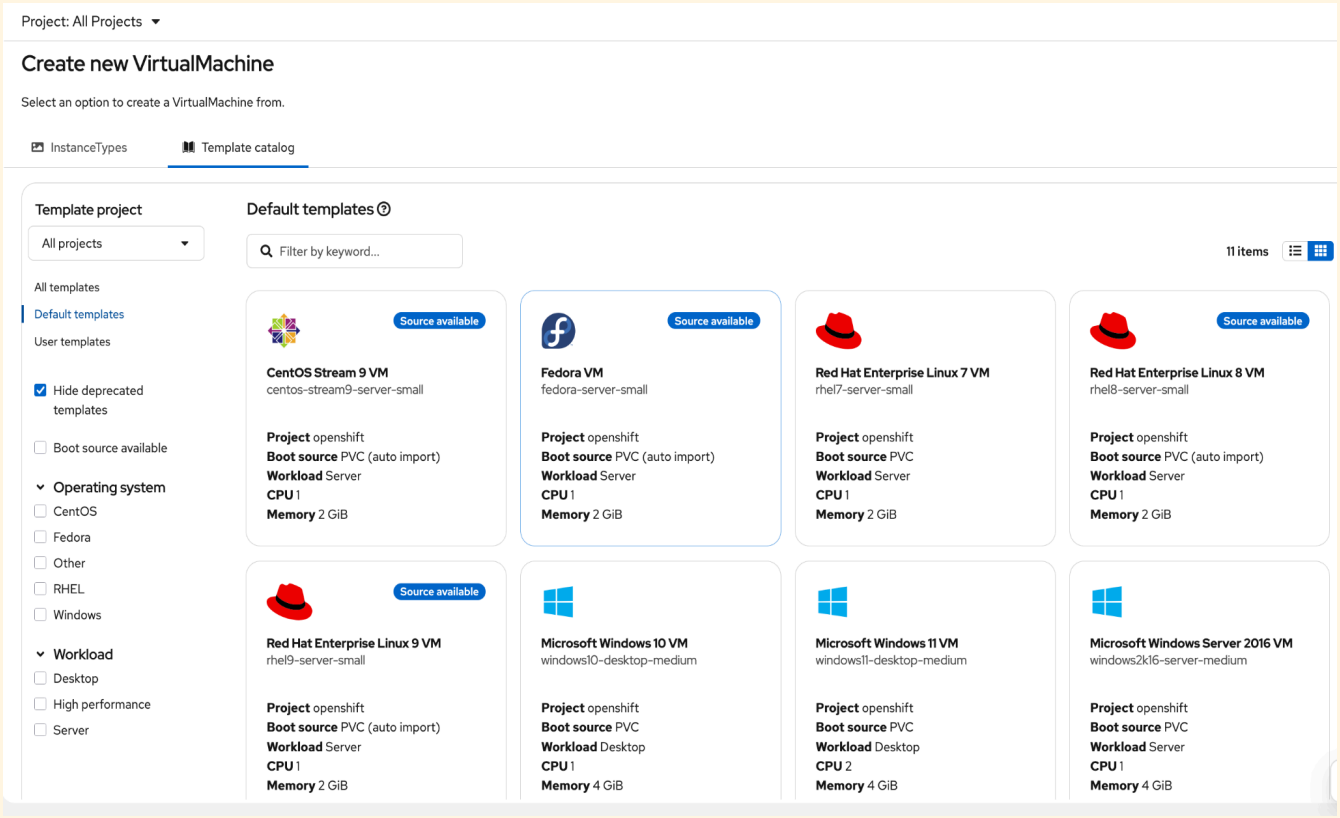

Figure 2. OpenShift Template Catalog

To update the default StorageClass via the command line, first remove the default annotation from the existing StorageClass:

```
oc patch storageclass <EXISTING-DEFAULT-STORAGECLASS-NAME> -p '{"metadata": {"annotations": {"
storageclass.kubevirt.io/is-default-virt-class":"false"}}}'
```

Next, assign the default role to a StorageClass that supports **RWX access modes**:

```
oc patch storageclass <NEW_RWX_STORAGECLASS_NAME> -p '{"metadata":
{"annotations":{"storageclass.kubevirt.io/is-default-virt-class":"true""}}}'
```

Portworx provides a preconfigured StorageClass for VM workloads, `px-rwx-block-kubevirt`, which offers RWX block storage. This StorageClass is automatically annotated as the default for KubeVirt when no other default is defined, simplifying VM provisioning and enabling features such as live migration.

As the boot volumes are automatically downloaded, they can be viewed in the OpenShift console under the Virtualization tab.

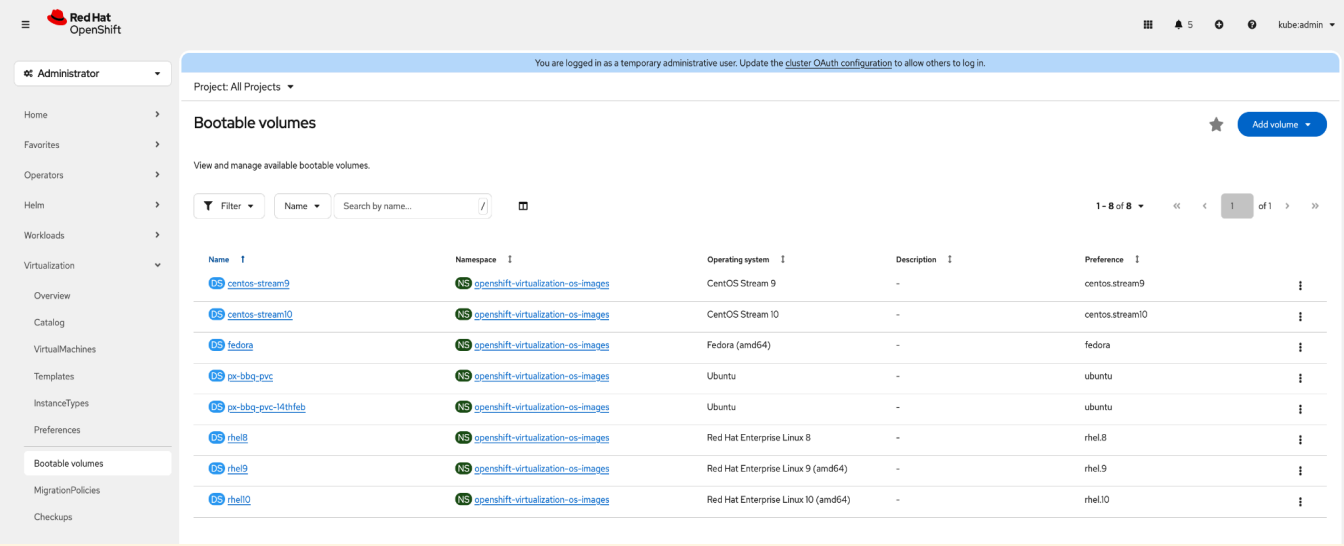

Figure 3. OpenShift Boot Volumes

Data Volumes

In OpenShift Virtualization, data volumes are objects that manage the initial lifecycle of persistent storage for virtual machines (VMs), including where the VM disk is sourced from, e.g. an existing PVC or registry image. PVCs serve as the storage backend for VM disks, providing the necessary data persistence for both operating systems and application data. Data volumes are an abstraction layer for VM disks, which are provided via Kubernetes PersistentVolumeClaims (PVCs).

Data volumes are integral to OpenShift Virtualization as they define how storage is allocated and populated when the VM disk is created. They provide a consistent way to handle storage for VMs, enabling integration with Kubernetes' storage orchestration capabilities. When a VM is created in OpenShift Virtualization, it relies on one or more PVCs to provide the necessary storage for its virtual disks.

When creating virtual machines through the OpenShift UI or by using the dataVolumeTemplate field in the VM specification, the data volumes share the same lifecycle as the virtual machine itself. Meaning that if you were to delete the virtual machine, the data volume would also be deleted. If your use case requires virtual machines and data volume lifecycles to be independent of each other, the data volume should be created separately, either through the storage API (preferred, see storage profile section in this document) or through the Kubernetes PVC API.

The source specification within a data volume defines the origin or the method by which the data for the volume is populated. Essentially, the source spec determines how the initial content of the VM disk is created or imported, providing flexibility in managing VM storage. For example the source spec can reference different types of sources for different use cases. These data volumes can include an existing disk image, a volume to clone, an image from a URL or a blank disk.

For virtual machines that are built from the default OpenShift templates, the boot source will be a PVC for that specific virtual machine template that has the operating system pre-installed. Creating a data volume from this instance works like a clone operation where all the data in the template's PVC will be used as the starting point for this data volume as seen from an example below.

```
spec:
  source:
    pvc:
      name: rhel9-d23c02e124d3
      namespace: openshift-virtualization-os-images
  storage:
    resources:
      requests:
        storage: 30Gi
```

To dive deeper into this example, the template itself has a data volume, and that source comes from the URL for the image in the image registry. See below.

```
spec:
  source:
    registry:
      pullMethod: node
      url:
docker://quay.io/containerdisks/centos-stream@sha256:a358a97caa12326de6281e864e599bf65894c8a81d9a4cc9ae789
592e7e81bb0
  storage:
    resources:
      requests:
        storage: 30Gi
```

When deploying a data volume, it is possible to preallocate disk space within the storage system. Preallocation can improve write performance for applications by reserving and initializing the entire storage space for the volume at time of creation, sometimes referred to as thick-provisioning. Reserving this space at the data volume creation, removes the need to reserve space before each write operation of the virtual machine using the volume.

With RWX Block volumes, preallocation is more likely to deliver noticeable performance gains than it does with filesystem-backed storage. By reserving capacity at creation time, preallocation can help keep write performance more consistent by avoiding on-demand allocation work during steady-state I/O and reducing fragmentation as the volume grows.

Preallocation consumes the full requested capacity up front, so use it primarily for performance-sensitive VM disks. Prefer thin provisioning with monitoring and automated expansion for everything else.

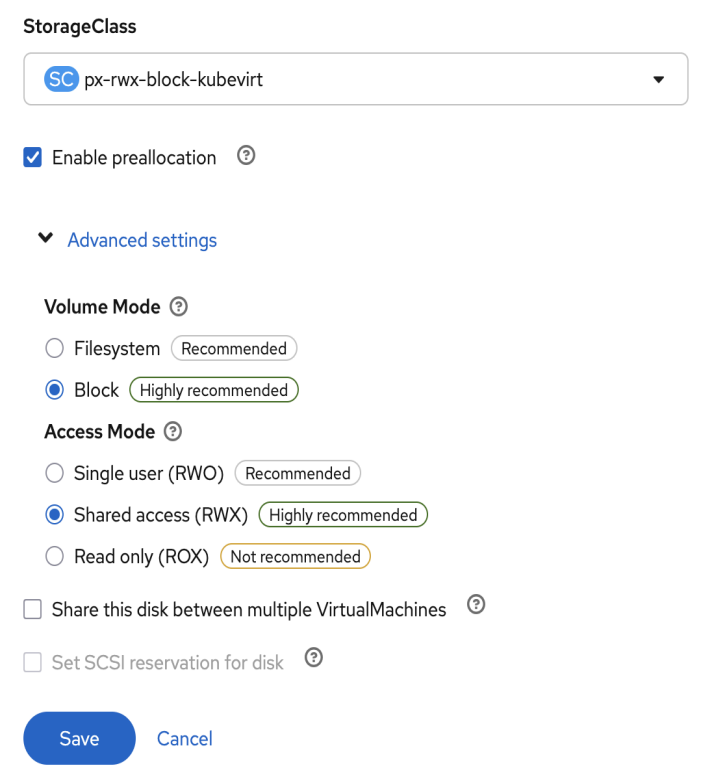

Figure 4. Preallocation

```
spec:
  preallocation: true
```

Preallocating virtual machine disk space offers a trade-off: improved performance via upfront disk reservation and faster I/O, versus immediate use of all allocated storage capacity. In RWX Block mode, preallocation significantly boosts performance and latency consistency by reserving capacity and minimizing allocation overhead, making it ideal for performance-sensitive VMs.

For environments prioritizing capacity efficiency, thin provisioning allows unused capacity to be shared across virtual machines, improving overall utilization. However, this introduces the risk of capacity exhaustion if multiple VMs grow simultaneously. To mitigate this risk, organizations should implement robust capacity monitoring, alerting, and automated expansion strategies.

This risk can be further addressed using Portworx Cloud Drives with Autopilot rules, which automatically manage storage scaling and help maintain sufficient capacity headroom.

Therefore, the choice between preallocation and thin provisioning should be driven by workload requirements, use preallocation for predictable performance and latency-sensitive workloads, and thin provisioning for efficient capacity utilization, supported by strong monitoring and automation.

Data volumes can also use annotations to change their functionality. For example, the annotation v1.multus-cni.io/default-network: bridge-network can be used to specify which network importer pods use to copy data from a source, into the volume. For more information on the annotations available, see the Red Hat OpenShift Virtualization documentation.

Storage Profiles

A storage profile in OpenShift Virtualization is a custom resource that provides optimized storage settings based on a specific Kubernetes storage class. Each storage class in the cluster has a corresponding storage profile that helps streamline the creation of data volumes, minimizing configuration errors and simplifying the process for users. When you create a data volume through the storage API, the storage profile automatically selects the best storage options for a virtual machine based on the underlying storage class.

For example, a Kubernetes storage class might support multiple access modes, such as ReadWriteMany (RWX) and ReadWriteOnce (RWO), as well as different volume modes, such as Block or Filesystem. A storage profile associated with that class will define the most suitable access and volume modes for a VM running in OpenShift Virtualization. To enable features like live migration, the storage profile could ensure that a data volume is created with RWX access mode and volumeMode as Block, assuming the storage class supports these options. This approach reduces the number of decisions users need to make when deploying virtual machines, ensuring consistency and reliability.

To view the specific settings defined in a storage profile, you can query the Kubernetes API using the following command:

```
oc describe storageprofile <Storage_Profile_Name>
```

Placement considerations for templates and cloning

When virtual machines are created from templates (or when DataVolumes are cloned), the resulting PVCs can inherit placement characteristics from the source (See Workload Consideration for more details). If many VMs are repeatedly created from the same base image or template, replicas and VM placement can concentrate on the same subset of nodes over time. This can lead to storage and compute hot spots, uneven pool utilization, and inconsistent performance.

To minimize imbalance:

- Ensure the VM StorageClass uses volumeBindingMode: WaitForFirstConsumer so PVC binding and placement are influenced by the VM scheduling decision.
- Use STORK-aware scheduling/data locality so VMs are preferentially placed on nodes where replicas exist (and avoid unnecessary cross-node I/O).
- Monitor pool utilization and replica distribution regularly, and use Portworx Autopilot rebalancing to correct drift when hotspots develop.
- Validate behavior with representative clone/scale tests (for example, repeatedly creating VMs from the same template) before production rollout.

The status section of the output will display the available access and volume mode combinations based on the associated storage class. For example:

```
Status:
  Claim Property Sets:
    Access Modes:
      ReadWriteMany
    Volume Mode:  Block
    Access Modes:
      ReadWriteOnce
    Volume Mode:  Block
    Access Modes:
      ReadWriteOnce
    Volume Mode:  Filesystem
```

When using storage profiles, an important consideration is the CloneStrategy. The CloneStrategy defines how data is replicated from an existing PersistentVolumeClaim (PVC) to a new data volume, which is particularly useful when creating new virtual machines from existing images or snapshots.

OpenShift Virtualization offers different CloneStrategy options to optimize cloning operations based on storage performance, efficiency, and backend capabilities:

- **CSI-Clone**: Uses the storage backend's native cloning capabilities to create a fast, efficient duplicate of the source volume, with minimal overhead. This is the preferred method for environments with CSI-based storage backends.
- **Snapshot**: Creates a point-in-time snapshot of the source volume and then provisions a new volume from this snapshot, ensuring data consistency but potentially adding overhead and time due to the snapshot process.
- **Copy**: Performs a complete data copy from the source volume to a new volume using standard read and write operations. This method is universally supported but is generally slower and more resource-intensive compared to CSI-Clone or Snapshot.

Storage classes using the pxd.portworx.com provisioner utilize the CSI-Clone strategy for creating new data volumes, making it the most efficient option for deploying VMs from templates due to its speed and low overhead.

You can influence the cloning behavior by updating the spec.cloneStrategy field in the StorageProfile configuration. By default, CDI automatically selects the cloning method based on storage provider capabilities, such as snapshot or host-assisted cloning. The csi-clone strategy is used only when explicitly configured and supported by the underlying storage.

It is a good practice to validate that each StorageProfile used for virtual machines has the intended default access mode, volume mode, and (where applicable) cloneStrategy. This helps ensure consistent VM disk provisioning in the UI and improves the success rate of validation checks such as Storage Checkup.

```
oc edit storageprofile <Storage_Profile_Name>
```

You can then set the desired access and volume modes:

```
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteMany
    volumeMode: Block
  cloneStrategy: csi-clone
```

Note: *In most deployments, the StorageProfile defaults derived from the StorageClass are sufficient. However, explicitly setting the defaults can be helpful in environments with multiple StorageClasses or where standardized behavior is required across clusters.By understanding and configuring storage profiles appropriately, administrators can ensure that virtual machine storage is optimized for performance, efficiency, and operational needs within OpenShift Virtualization.*

Container Data Import Scratch Space

The Container Data Importer (CDI) is a key component in KubeVirt that facilitates the import, upload, and conversion of virtual machine disk images to persistent volumes (PVs) within a Kubernetes cluster. It streamlines the process of getting external data—such as raw disk images or QCOW2—or ISO files into the cluster for use by virtual machines. CDI is particularly useful in scenarios where virtual machines need to be initialized with existing data or when migrating workloads from other environments.

CDI requires temporary storage, known as scratch space, during the import process. This scratch space is used to store intermediate data or perform transformations on disk images before they are written to the final Persistent Volume. The scratch space typically resides on a separate Persistent Volume that CDI dynamically provisions and manages. The following operations require scratch space:

- Registry imports
- Uploading image
- HTTP imports of archived images
- HTTP imports of authenticated images
- HTTP imports of custom certificates

When creating the HyperConverged Configuration you may specify a scratchSpaceStorageClass specification that defines which storage class will be used for creating these temporary volumes. This storage class requires a file volume mode regardless of the PVC backing the origin data volume.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "px-cdi-scratch"
```

Since these volumes are ephemeral and only used temporarily during data import tasks, it is not necessary to protect these volumes with a data protection solution. However, Portworx recommends using a replication factor of one to reduce storage usage. Since these volumes are temporary, there is no need to add additional replicas which use valuable storage space.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-cdi-scratch
provisioner: pxd.portworx.com
parameters:
  repl: "1"
  sharedv4: "true"
  sharedv4_mount_options: vers=3.0,nolock
volumeBindingMode: Immediate
reclaimPolicy: Delete
allowVolumeExpansion: true
```

Virtual Machine State Storage Classes

In some circumstances, virtual machines need to store sensitive data outside of the virtual machine itself. One example of this is for a virtual trusted platform module (vTPM). Virtual TPMs require persistent storage to securely store sensitive data such as cryptographic keys and certificates. This storage must be reliable and isolated to ensure that security guarantees provided by the vTPM are maintained throughout the lifecycle of the virtual machine.

Persistent storage can be provided for solutions like vTPM through the VMStateStorageClass configuration in the hyperconverged configuration of OpenShift Virtualization. The storage class used must support ReadWriteMany (RWX) or ReadWriteOnce with Filesystem volume mode.

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  vmStateStorageClass: "<STORAGE_CLASS_HERE>"
```

Portworx strongly recommends defining a VMStateStorageClass during the provisioning of the hyperconverged configuration, especially since certain virtual machines, such as those running Windows 11, require a Trusted Platform Module for operation. To ensure high availability, this storage class should be configured with a replication factor of two or higher, providing resilience in the event of an outage. The reclaimPolicy should also be set to “retain” to ensure that the accidental deletion of the persistent volume claim does not remove the data in this volume.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: vmstate
provisioner: pxd.portworx.com
parameters:
  repl: "3"
reclaimPolicy: Retain
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

File System Overhead

Filesystem overhead refers to the portion of storage space on a filesystem that is reserved for the filesystem’s internal management and metadata. This overhead is not directly usable for storing application data because it is used for maintaining the structure and integrity of the file system itself. The amount of overhead can vary depending on the type of filesystem and its configuration.

During a migration process using the OpenShift Migration Toolkit for Virtualization (MTV), it’s crucial to account for filesystem overhead to ensure that the destination environment has sufficient storage capacity to accommodate both the application data and the filesystem's internal needs. If filesystem overhead is not considered, the migration might fail due to insufficient storage space, leading to interruptions and potential data loss.

The amount of filesystem overhead to provide during a migration can be configured in the filesystemOverhead.global spec of the hyperconverged object:

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  filesystemOverhead:
    global: "0.08"
```

Portworx recommends setting this value to "0.08" or higher to ensure there is sufficient room in the persistent volumes to perform a migration with the Migration Toolkit for Virtualization.

Considerations for Portworx 3.6 and Block Volumes

With Portworx 3.6, support for raw block volumes (`volumeMode: Block`) allows workloads in **OpenShift** to consume storage without an OpenShift-managed filesystem layer. In such cases, filesystem overhead at the storage layer is not applicable.

- For **Filesystem volumes** (**`volumeMode: Filesystem`**):
  The overhead setting must be retained, and a value of **0.08 or higher** is recommended to account for filesystem metadata.
- For **Block volumes** (**`volumeMode: Block`**):
  The filesystem overhead setting is **generally not required**, as no filesystem is created on the volume by OpenShift.

Storage Checkup

Red Hat OpenShift provides a cluster checkup framework to ensure the cluster is optimally configured for OpenShift Virtualization. Portworx recommends running the storage checkup either from the command line or through the OpenShift UI prior to deploying production workloads.

As noted earlier in the document, each storage profile needs to have the appropriate claimPropertySets, and cloneStrategy in order to complete successfully. The storage checkup is a good way to check to ensure these values are set for each storage profile in the cluster.

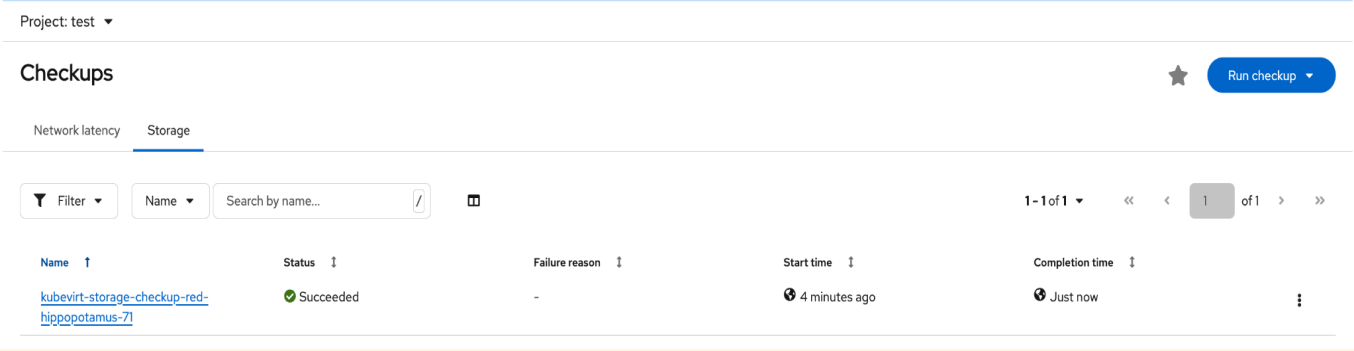

Figure 5. OpenShift Storage Checkup

High Availability

Virtual machines, like any other application, are susceptible to outages caused by failures in the underlying physical hardware. These failures can result from a variety of issues, including power outages, hardware malfunctions, or human error. "High availability" (HA) refers to the system's ability to automatically and quickly recover from such failures, ensuring minimal disruption to services.

In Kubernetes and Red Hat OpenShift, orchestration is built into the control plane, enabling the automatic restart of failed applications based on health checks and desired state configurations. Specifically, the OpenShift control plane includes logic to restart container-based workloads on alternative nodes in the event of a node failure.

However, when dealing with workloads that contain persistent data, such as virtual machines running on OpenShift Virtualization, there is an additional requirement: the virtual machine's disks must be accessible by all nodes in the cluster to allow the VM to be restarted on another node. For instance, if a virtual machine is using local storage directly attached to a single worker node, and that node fails, the VM cannot be restarted on another node unless those nodes have access to the same storage. This requirement, known as "shared storage," is critical for ensuring the high availability of virtual machines in OpenShift. Without shared storage, the HA capabilities of OpenShift cannot fully protect against node failures for virtualized workloads.

Additionally, all the considerations outlined in the Portworx Enterprise on Red Hat OpenShift bare metal base reference architecture should be strictly adhered to when designing for OpenShift Virtualization. This includes key strategies such as distributing clusters over multiple fault domains or racks to enhance fault tolerance and reduce the risk of simultaneous failures affecting multiple nodes, and using multiple replicas for persistent volumes.

When running virtual machines in Kubernetes clusters, it is essential to maintain sufficient spare capacity, particularly in terms of CPU and memory, to handle node or fault domain failures. In the event of a failure, both pods and virtual machines will need available resources to be rescheduled and restarted on the remaining nodes. This becomes critical in environments where the Kubernetes Out of Memory (OOM) manager may begin terminating processes if nodes are overwhelmed, potentially disrupting workloads. Proper capacity planning ensures stability and prevents resource exhaustion during pod or virtual machine restart scenarios.

When using some storage solutions that comply with the CSI specification, failover can take several minutes to complete. This delay occurs because before a container-based workload with attached persistent volumes can fail over to another node, the storage connection between the failed node and the storage solution must be fully disconnected. Many solutions impose a timeout period—often up to five minutes—before this disconnection occurs, resulting in lengthy failover times.

In contrast, with Portworx Enterprise, the storage connection is severed immediately after the node has been detected as unhealthy. This rapid disconnection allows virtual machines to fail over to another node much more quickly, significantly reducing downtime and improving overall availability. By following the guidelines provided in the reference architecture and leveraging Portworx's capabilities, you can ensure a robust, high-availability environment for your virtualized workloads on OpenShift.

Host Failures

As mentioned earlier in this document, the Kubernetes control plane is designed to automatically restart failed containers or virtual machines on healthy nodes in the event of a node failure. When a host failure occurs, STORK will help to schedule any virtual machines that were running on that node, to a node where a replica exists. The virtual machine can then access the virtual disks through a bind mount since data is local to the VM. Using bind mounts here reduced the amount of time needed for failovers compared to using a hardware storage array since iSCSI or Fibre Channel connections don't need to be disconnected and reconnected on additional nodes.

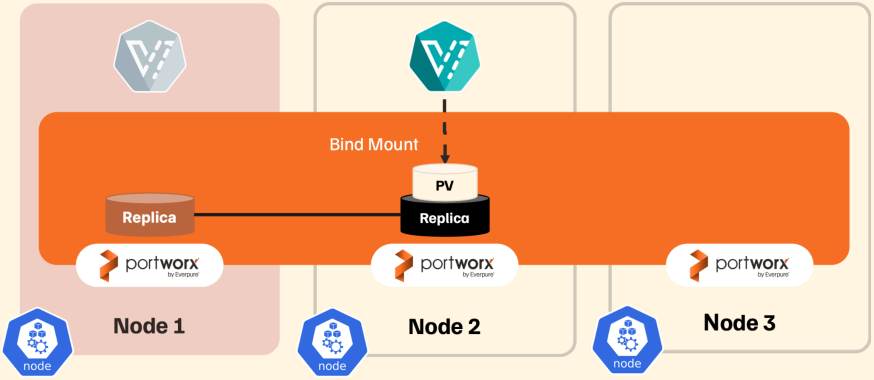

Figure 6. Host Failures Example

Boot Storms

When a node fails, any virtual machines running on that node are automatically restarted on another node. If a large number of VMs were running on the failed node, they would all attempt to boot simultaneously, creating what is commonly referred to as a boot storm. This sudden spike in I/O demand, caused by all the virtual machines attempting to load their operating systems at once, can place significant strain on the storage system.

When designing for mitigation of boot storm impacts, you must take into consideration VM density on surviving nodes and the impact on CPU and memory allocation. In addition, the impact on storage must be taken into consideration due to the high number of IOPS issued during a boot storm event.

Portworx recommends that you test boot storm impacts due to a failed worker node in a test or development environment before deploying into Production to observe the impact these situations can create.

Note: *Template-driven provisioning can concentrate VM disks and I/O on a subset of nodes over time, which can amplify boot storm impact. See Workload Distribution for mitigation guidance.*

Live Migration

Live migration is a critical feature for managing virtual machine (VM) workloads in OpenShift Virtualization, playing a vital role in maintaining high application availability and operational efficiency. This capability allows running VMs to be seamlessly moved from one host to another without downtime, ensuring that applications continue to operate smoothly even during infrastructure changes.

The benefits of live migration are particularly evident in several key scenarios:

- **Host upgrades and planned maintenance**: During scheduled maintenance or host upgrades, live migration allows administrators to move VMs to other nodes in the cluster without interrupting services. This enables routine maintenance to be performed without impacting application uptime, reducing the risk of downtime and ensuring business continuity.
- **Cluster rebalancing**: As workloads evolve, certain nodes in a cluster may become overloaded while others are underutilized. Live migration facilitates cluster rebalancing by enabling administrators to redistribute VMs across the available nodes, optimizing resource utilization, and maintaining performance levels across the environment.
- **Improved application availability**: By enabling VMs to be moved without interruption, live migration directly enhances application availability. In the event of a planned host outage or other critical issues, VMs can be quickly relocated to healthy nodes, minimizing the impact on end-users and ensuring that services remain available.

Live migration is an essential tool for modern IT operations, enabling greater flexibility, improved resource management, and enhanced resilience within OpenShift Virtualization environments. By leveraging this capability, organizations can achieve a higher level of operational efficiency while maintaining the availability and performance of their critical applications.

Shared Storage for Live Migration (RWX)

The live migration capability in OpenShift Virtualization relies on a key feature of the underlying storage system. For a virtual machine to be moved seamlessly between hosts, both the source and destination hosts must have access to the same storage device where the virtual machine's disks are stored. In a Kubernetes environment, this is accomplished by using storage with a ReadWriteMany (RWX) access mode. Portworx supports persistent volumes with multiple access modes, including RWX. Customers looking to leverage live migration functionality must ensure that their virtual machine workloads are backed by RWX persistent volumes to facilitate this capability.

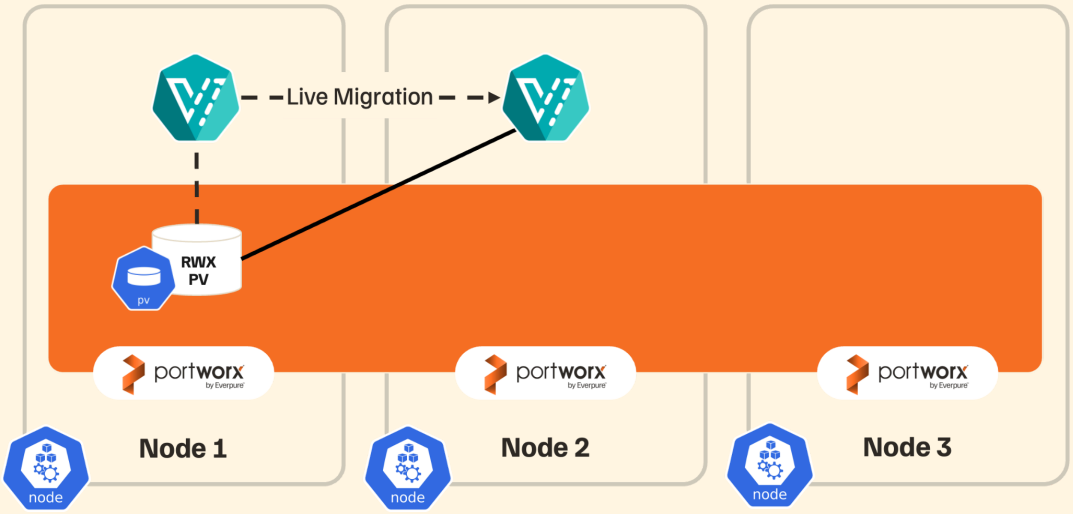

Figure 7. Live Migrations

Note: *With RWX Block storage, VM disks remain concurrently accessible from multiple nodes, which enables live migration without requiring storage to be "moved" between nodes. Data resiliency continues to be handled by the storage layer according to the volume's protection policyPortworx can provide access to RWX volumes through a storage class configured for RWX access modes. The snippet below is an example of a RWX storage class that can facilitate live migrations.*

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-rwx-block-kubevirt
provisioner: pxd.portworx.com
parameters:
  repl: "3"
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

To improve the manageability of the environment, Portworx recommends making your virtual machine storage class the default storage class for the cluster. This prevents users from needing to understand and specify which storage class should be used for virtual machines.

Mixed Processor Types

An OpenShift cluster can be built with mixed processor types across nodes, but should be discouraged. For example some worker nodes use an AMD processor type while others use an Intel processor. This is a valid configuration but prevents live migrations from occurring between nodes with different processor types. If using mixed processor types, please create an OpenShift machineset for each processor type.

Portworx recommends using a similar processor type for each node participating in the OpenShift cluster to provide flexibility with virtual machine workloads. However, if a mixed processor type environment is necessary, Portworx recommends using node affinity with the portworx volumes to "pin" Portworx replicas to specific nodes. Using volume affinity rules, can ensure that replicas will be available on multiple nodes where virtual machines may be migrated.

Performance

Storage performance for virtual machines in OpenShift Virtualization is primarily influenced by the volume mode, access mode, and the ability to consistently deliver predictable latency under mixed workloads. For VM disks, this reference architecture uses ReadWriteMany (RWX) Block to provide shared access across nodes while retaining the performance characteristics of block storage.

Access modes and volume modes

RWX Block enables multiple nodes to access the same VM disk concurrently, supporting capabilities such as live migration and improving operational flexibility without relying on filesystem-based sharing. For specialized workloads that do not require shared access across nodes, RWO Block can be used to enforce single-node attachment semantics.

Note: *RWO volumes restrict access to one node at a time and may limit VM mobility features that depend on shared access.*

Capabilities	RWX Filesystem	RWO Block	RWX Block
Live Migrations	Yes	No	Yes
Best Fit	For all purposes RWX Block volumes are recommended for VM disks	Yes	Recommended for all VM data disks. General purpose VM disks, mobility Stateful clustered apps requiring shared block access (e.g., databases)
Operational flexibility	High (shared access across nodes)	Limited (single-node access)	High

Table 1. VM disk access mode capabilities

Application I/O Control

Portworx provides Application I/O Control to enforce predictable performance and reduce “noisy neighbor” effects across mixed VM workloads. I/O policies can be applied to VM StorageClasses to:

- Cap maximum IOPS and/or bandwidth for non-critical VMs
- Reserve performance for latency-sensitive workloads
- Reduce the impact of boot storms, large-scale provisioning, or maintenance windows on other tenants

Use Application I/O Control when multiple VM tenants share the same cluster or when you need to enforce service-level objectives for performance-critical applications.

Workload Distribution

A well-balanced cluster is essential for evenly distributing virtual machines and storage utilization across worker nodes and storage pools. If VM disks and related storage activity concentrate on a subset of nodes, hot spots can form—creating uneven pool utilization, localized performance bottlenecks, and slower recovery behavior during restart or maintenance events.

Template-driven provisioning and clone skew

This becomes especially important when virtual machines are repeatedly created from the same template or base image. Depending on how VM disks are provisioned and cloned, newly created VMs may tend to follow similar placement patterns over time, which can gradually concentrate storage utilization on the same subset of nodes and pools. In large environments, this "clone skew" can lead to:

- Uneven storage pool consumption
- Localized latency increases during burst events (provisioning, patching, restarts)
- Increased sensitivity to boot storms or node maintenance on the affected nodes

To reduce the risk of hot spots:

- Monitor replica distribution and pool utilization trends
- Periodically validate VM placement behavior under scale-out operations
- Use automated rebalancing mechanisms to correct skew as it develops

Storage pool rebalancing with Portworx Autopilot

To mitigate storage hot spots over time, Portworx Autopilot can rebalance storage pools across the cluster on a best-effort basis. An Autopilot rule like the following helps distribute pool usage more evenly:

> A well-balanced cluster is essential for evenly distributing workloads across all worker nodes and storage devices. When virtual machines and volume replicas are concentrated on a few nodes, "hot spots" can form, where certain resources are overused while others remain idle. To prevent performance bottlenecks, it's important to distribute workloads uniformly across the cluster, making efficient use of both compute and storage resources.
>
> This becomes especially important during virtual machine cloning operations. For example, when you clone an existing virtual machine that has volume replicas on nodes 1, 3, and 5, the resultant cloned VM and its volumes will be placed on the same nodes. Over time, this repetition can lead to hot spots, where nodes 1, 3, and 5 become overburdened, impacting cluster performance. Regularly monitoring and adjusting the distribution of workloads will help prevent these imbalances.

To mitigate storage hotspots, use Portworx Autopilot to rebalance storage pools across the cluster. An Autopilot rule like the one below helps distribute volume replicas across the Kubernetes cluster on a best effort basis.

```
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: pool-rebalance
spec:
  conditions:
    requiredMatches: 1
    expressions:
    - keyAlias: PoolProvDeviationPerc
      operator: NotInRange
      values:
      - "-20"
      - "20"
    - keyAlias: PoolUsageDeviationPerc
      operator: NotInRange
      values:
      - "-20"
      - "20"
  actions:
  - name: openstorage.io.action.storagepool/rebalance
```

Networking

Networking is an important design consideration for any virtual machine design, but is outside the scope of this document focused on the storage considerations for a Red Hat OpenShift Virtualization cluster.

The guidance from the Portworx found in the Red Hat OpenShift Bare Metal reference architecture linked above should be followed for networking considerations here as well. The Portworx data network should be placed on different NICs to ensure the storage network and virtual machine or Kubernetes networks aren't in contention for resources. As a good design practice management, live migration, storage, and application networks should be placed on different physical devices to prevent resource contention within the cluster.

Persistent Volume Claims

There may be times when you pre-create a persistent volume for use with KubeVirt virtual machines. Portworx can support this method of deploying virtual machine disks in this way but requires an annotation in the PVC specification.

portworx.io/app: kubevirt

This annotation allows Portworx to apply KubeVirt-specific logic when processing the volume.

Virtual Machines

In a Kubernetes cluster, pods serve as the fundamental deployment units, encapsulating containers and their associated resources. Similarly, in a virtualization environment, virtual machines (VMs) act as the primary deployment units, encapsulating an operating system, applications, and the necessary resources to function independently. However, unlike pods, VMs bring a unique set of storage requirements and challenges due to their need for persistent, high-performance storage that can support complex workloads and maintain state across reboots and migrations.

In this section, we will delve into the crucial relationship between VMs and storage within an OpenShift Virtualization environment. We'll examine how storage underpins the reliability, performance, and scalability of VMs, and explore the various storage options and best practices available to ensure your virtualized workloads operate smoothly. Understanding this relationship is key to designing a robust and efficient infrastructure that meets the demands of modern applications running in virtualized environments.

Virtual Machine Templates

Red Hat OpenShift provides default templates for quickly deploying virtual machines within a cluster. However, these templates do not include the specific configurations necessary for Portworx. To ensure compatibility and optimal performance, Portworx recommends creating custom virtual machine templates that incorporate the configuration guidelines outlined in this section.

Portworx utilizes a solution called Storage Orchestrator Runtime for Kubernetes (STORK). STORK is designed to optimize pod scheduling by ensuring workloads are placed on nodes where their data is locally available. By co-locating data with running containers on the same node, STORK enhances storage performance, enabling faster and more consistent data access.

Given how STORK manages placement decisions, Portworx recommends adding the following annotation to your virtual machine templates when running OpenShift 4.14 or higher:

```
dataVolumeTemplates:
 - metadata:
     annotations:
       cdi.kubevirt.io/storage.usePopulator: "false"
```

This annotation allows Portworx to either place virtual machines on the same node as a Portworx replica or to live migrate the VM to a node with a Portworx replica. Without this annotation, virtual machines may not be co-located with their data, requiring them to use the physical network to access their VM disks, which can degrade performance.

Additionally, it is crucial to ensure that the following annotation is NOT included in the virtual machine template:

```
dataVolumeTemplates:
  - metadata:
      annotations:
          cdi.kubevirt.io/storage.bind.immediate.requested=true
```

This annotation overrides the WaitForFirstConsumer binding mode in the virtual machine storage class and should be avoided to maintain optimal performance and proper data locality.

An eviction strategy should also be set on the virtual machines if they are to be live migrated during a maintenance operation. This is set with an eviction strategy on the virtual machine.

```
spec:
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible
```

Day Two Disk Operations

Day two operations encompass the ongoing management and maintenance tasks that occur after the initial installation and deployment of your OpenShift Virtualization (OSV) cluster. These operations typically involve managing virtual machines that are already running workloads, ensuring their performance, reliability, and scalability in a production environment.

Disk Expansion

In OpenShift Virtualization, virtual machine (VM) disks are backed by PersistentVolumeClaims (PVCs). When you need to expand a virtual machine's disk, you can do so by increasing the size of the underlying PVC. Portworx supports dynamic resizing of PVCs, allowing you to expand the storage capacity without downtime, provided that the storage class and the filesystem in use support resizing.

To expand a virtual machine disk:

1. **Verify support for resizing**: Ensure that the storage class used by the PVC supports volume expansion. In Portworx, this is typically enabled by default, but you should confirm that the storage class configuration includes `allowVolumeExpansion: true`
2. **Update the PVC size**: Modify the size of the PVC associated with the VM disk. This can be done by editing the PVC YAML to specify a larger storage size.
3. **Resize the filesystem (if necessary)**: After the PVC size is updated and the underlying storage is expanded, the filesystem on the VM may also need to be resized to utilize the additional space. This step might require VM-specific commands depending on the filesystem used by the VM.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-pvc-name
  namespace: virtualmachinenamespace
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 20Gi  # Updated size from 10Gi to 20Gi
```

As mentioned in the Portworx OpenShift Bare Metal Reference Architecture, Autopilot can also be used to automatically expand virtual machine disks that match certain capacity rules. These autopilot rules also apply to virtual machines and can be used to reduce manual day two tasks related to disk capacity. See the Portworx on OpenShift Bare Metal Design or Portworx documentation for more details.

Adding Disks

In OpenShift Virtualization, adding secondary disks to a virtual machine (VM) can provide additional storage capacity for applications, data, or backup purposes. Secondary disks can be added to a VM either dynamically (hot-plugged) or statically (requiring the VM to be powered off), depending on the disk bus type and the underlying capabilities of the VM's operating system.

The disk interface determines how a disk is connected to the VM. In OpenShift Virtualization, common interface types include VirtIO, SATA, and SCSI. The choice of bus type affects whether the disk can be hot-plugged (added or removed while the VM is running) or if the VM needs to be powered off to attach the disk. Portworx recommends using the SCSI interface to take advantage of hot-plugging disks.

Interface Type	Hot-Plugging Supported	Description
VirtIO	Yes	A high-performance paravirtualized interface designed for virtual environments. Supports hot-plugging, making it ideal for dynamic workloads requiring additional storage without downtime. Note: Windows requires additional drivers to use this type.
SCSI	Yes	Provides support for advanced storage features like multipathing. Supports hot-plugging, allowing disks to be added or removed without powering off the VM.
SATA	No	Offers compatibility with a wide range of operating systems but does not support hot-plugging. Disks require the VM to be powered off before they can be added or removed.

Table 2. OpenShift Disk Interface Type

The disks can be added from the VirtualMachines tab of the OpenShift console or by using the virtctl command line utility.

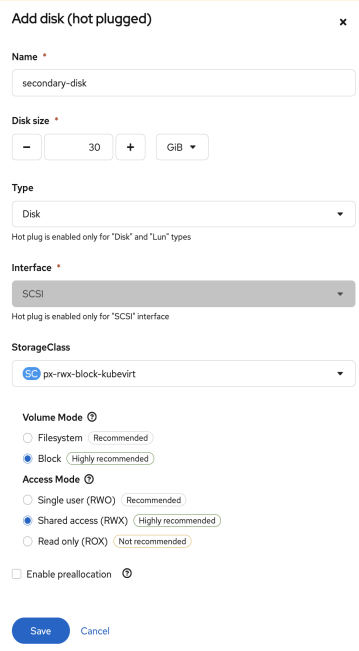

Figure 8. OpenShift Add Disks Menu

After adding additional disks to the virtual machines, verify that the disk is visible from the virtual machine and that the filesystem is available.

Configuring Shared Disks

RWX block volumes in Portworx are supported for KubeVirt virtual machine use cases, particularly to enable capabilities such as live migration and high availability. These volumes provide shared block access across nodes, allowing a VM disk to be accessible during controlled transitions like migration.

However, this shared access model is not designed to meet the requirements of traditional clustering solutions such as Windows Server Failover Clustering (WSFC). Specifically, Portworx RWX block volumes do not implement SCSI-3 Persistent Reservations (SCSI-PR), which are essential for coordinating access to shared disks in cluster environments.

While RWX block volumes are suitable for VM mobility and platform-level operations in KubeVirt, they cannot be used for cluster-aware shared disk use cases that rely on SCSI-PR semantics.

Snapshots

Snapshots are a powerful feature in OpenShift Virtualization that allow you to capture the state of a virtual machine's data volume at a specific point in time. This functionality is particularly useful for scenarios such as testing, development, and quickly recovering from user errors by rolling back to a known good state. However, it is important to understand the proper configuration and limitations of snapshots to use them effectively in your virtualization environment.

To use snapshots in OpenShift Virtualization, you first need to deploy a VolumeSnapshotClass. A VolumeSnapshotClass provides the necessary configurations and parameters to create snapshots of PersistentVolumeClaims (PVCs) in your Kubernetes cluster. It defines the driver and the snapshotter backend used to create the snapshots, ensuring that snapshots are handled consistently across the cluster.

Create a VolumeSnapshotClass like the YAML example below:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
 name: px-csi-snapclass
 annotations:
   snapshot.storage.kubernetes.io/is-default-class: "true"
driver: pxd.portworx.com
deletionPolicy: Delete
parameters:
 csi.storage.k8s.io/snapshotter-secret-name: px-user-token
 csi.storage.k8s.io/snapshotter-secret-namespace: portworx
 csi.openstorage.org/snapshot-type: local
```

Note: *The csi.storage.k8s.io/snapshotter-secret-namespace parameter should specify the namespace where the portworx storage cluster was created. If you are not using PX-Security the secret name and secret namespace parameters are not necessary.*

After deploying the VolumeSnapshotClass, you can create snapshots of your PVCs as needed. These snapshots provide a quick way to capture the state of a data volume, which can be useful for various operational scenarios such as test environments, development workflows, or before applying potentially disruptive changes.

While snapshots provide a convenient method to preserve the state of a data volume at a particular moment, they should not be considered a backup solution.

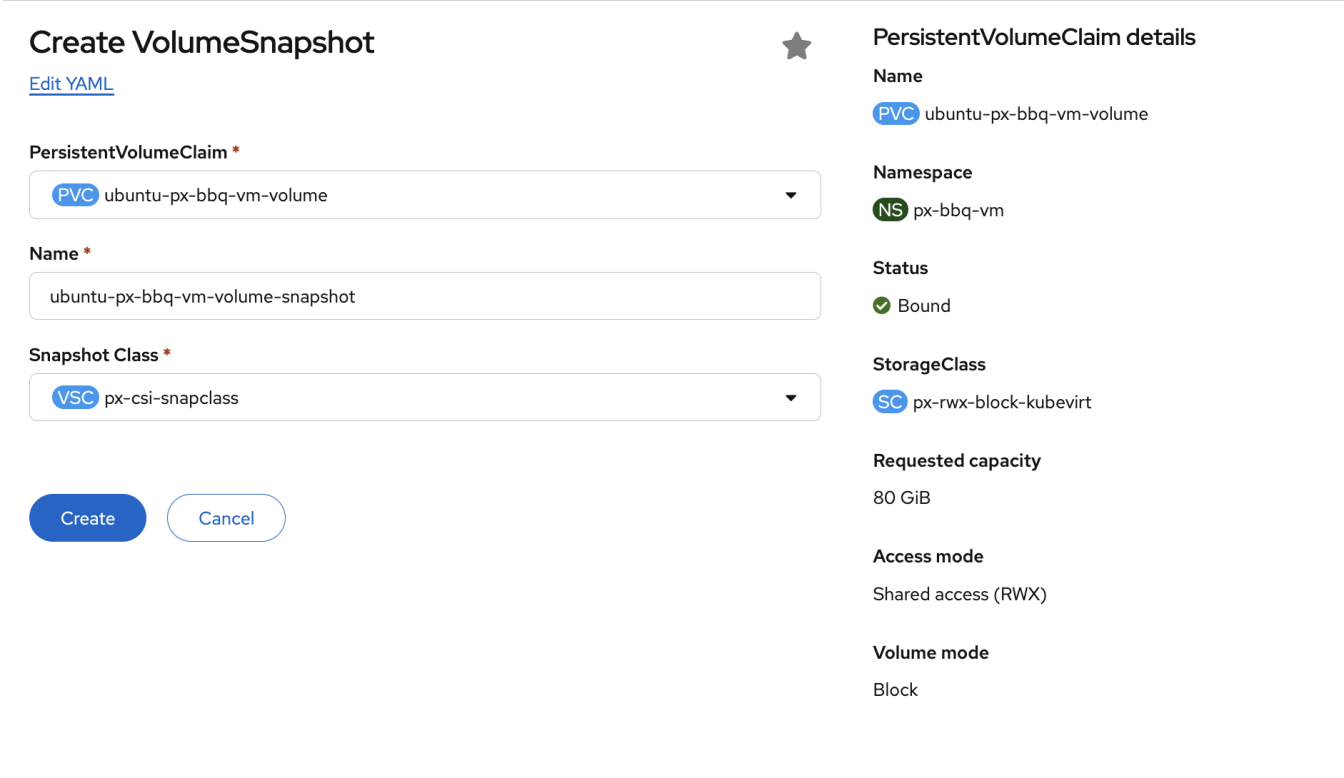

Figure 9. OpenShift Create Volume Snapshots

For a complete data protection and disaster recovery strategy, you must implement a backup solution that provides off-site storage, redundancy, and long-term data retention. Further details on designing and implementing a comprehensive backup and disaster recovery plan will be available in the Data Protection and Disaster Recovery Addendum.

By understanding the role and limitations of snapshots, you can effectively integrate them into your OpenShift Virtualization environment as a valuable tool for quick recovery and operational efficiency, while also recognizing the need for a more robust backup strategy to protect against data loss and ensure business continuity.

Upgrades

Upgrading a Red Hat OpenShift cluster on bare metal with Portworx as the storage layer requires planning and execution to maintain data availability and prevent disruptions. Portworx, being a distributed storage solution, depends on multiple nodes within the cluster to deliver redundancy, high availability, and data resilience. Consequently, it is crucial to ensure the storage cluster's integrity during upgrades to avoid any loss of quorum or data availability issues.

Always refer to the official Red Hat OpenShift and Portworx documentation before performing upgrades. This includes verifying version compatibility between OpenShift and Portworx, as well as reviewing any relevant release notes or upgrade guidelines.

When upgrading OpenShift clusters running OpenShift Virtualization, it's essential to verify that Live Migration is functioning correctly to keep virtual machines running smoothly during node upgrades. When a node maintenance custom resource is added by the Node Maintenance Operator, any virtual machines that have the eviction strategy set to live migrate will be migrated to other nodes in the cluster. Without the configuration below, set on a virtual machine, you may encounter a virtual machine outage when taking a node offline for maintenance.

```
spec:
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible
```

Additionally, you should upgrade worker nodes one at a time rather than all at once. This is critical because Portworx can only maintain replicas on a maximum of three nodes. Simultaneously upgrading all three nodes would result in virtual machine downtime.

For additional details on upgrade considerations, please refer to the Red Hat OpenShift Bare Metal Reference Architecture or official documentation from Red Hat or Portworx.

Migration Toolkit for Virtualization

The OpenShift Migration Toolkit for Virtualization (MTV) is a tool designed to simplify the migration of virtual machine workloads to Red Hat OpenShift. It provides a set of tools and capabilities that enable organizations to seamlessly transition their virtual environments from traditional virtualization platforms, such as VMware vSphere or Red Hat Virtualization (RHV), to a modern, container-based infrastructure. By leveraging the OpenShift Migration Toolkit for Virtualization, users can automate and orchestrate the migration process, minimizing downtime and ensuring data integrity. This toolkit is helpful for organizations looking to modernize their IT infrastructure, consolidate workloads, and fully embrace the agility and scalability offered by OpenShift's Kubernetes-based architecture.

Rapid VM Migration with Everpure FlashArray

The migration is orchestrated using the **Migration Toolkit for Virtualization** (MTV, also known as Forklift), while **Portworx Enterprise** provides the persistent storage layer for migrated workloads.

To accelerate migration and reduce compute overhead, this solution integrates **Pure Storage FlashArray**, enabling storage-level data transfer and minimizing reliance on host-based data copy operations.

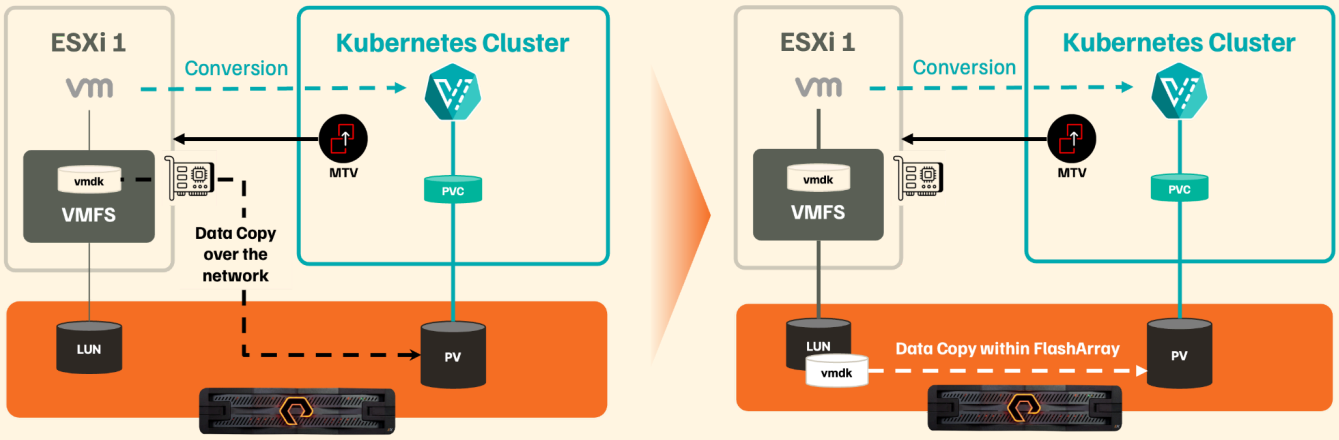

Figure 10: Rapid VM Migration with Portworx and FlashArray

With Rapid VM Migration:

- Pure uses array-native XCOPY to move data internally within FlashArray.
- Kubernetes volumes are populated without network-intensive host copies.

Red Hat Container Registry

The OpenShift Container Registry is a critical component for supporting the Migration Toolkit for Virtualization. MTV relies on containerized dependencies, including the VMware Virtual Disk Development Kit (VDDK), to facilitate the migration of VM workloads from VMware environments into OpenShift. The VDDK and other MTV dependencies must be stored in an image registry to ensure compatibility and accessibility during migrations.

In an OpenShift environment, the OpenShift Container Registry is typically deployed by default during installation, provided that object storage is available. However, since Portworx does not natively offer object storage, the OpenShift Container Registry may initialize in a "Removed" state during bare metal installations, allowing the OpenShift installer to complete successfully without configuring the registry.

To deploy the image registry without relying on object storage, you can provision a ReadWriteMany (RWX) persistent volume and configure the image registry to a "Managed" state, using this RWX volume as the backend storage.

Portworx recommends using an RWX persistent volume to allow multiple replicas of the image registry to be deployed. This approach ensures high availability and fault tolerance for the image registry, which is crucial for production environments. In contrast, using ReadWriteOnce (RWO) volumes would restrict the image registry to a single replica, potentially leading to issues with scalability and reliability, and should be avoided in production clusters. This volume should have a minimum of two replicas to protect from a host failure.

To set the OpenShift Container Registry to "managed" run:

```
oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch
'{"spec":{"managementState":"Managed"}}'
```

Deploy a persistent volume with an appropriate amount of storage to house virtual machine boot images for your environment:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: registry-storage-pvc
namespace: openshift-image-registry
spec:
accessModes:
- ReadWriteMany
resources:
  requests:
    storage: 300Gi
storageClassName:   px-rwx-block-kubevirt
```

Patch the container registry to use the persistent volume claim deployed in the previous step.

```
oc patch configs.imageregistry.operator.openshift.io/cluster --type=merge \
  -p
'{"spec":{"managementState":"Managed","replicas":2,"storage":{"pvc":{"claim":"registry-storage-pvc"}}}}'
```

Then patch the default route to the image registry.

```
oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec":{"defaultRoute":true}}'
--type=merge
```

For the latest instructions see the official Red Hat documentation for configuring the OpenShift Container Registry on bare metal.

In a production deployment, it is also crucial to consider that virtual machine container disks can be stored in the image registry. This storage capability ensures that VM disks are readily accessible for efficient scaling, backup, and migration processes within the OpenShift environment.

Storage Plans

The Migration Toolkit for Virtualization operator requires the creation of migration plans that define how resources are mapped from a source environment, such as VMware vSphere, Red Hat OpenStack Platform (RHOSP), or Red Hat Virtualization, to a destination like Red Hat OpenShift. These migration plans need to include mappings between the source and destination networks, as well as storage configurations. For instance, a storage mapping involves specifying the source storage location, like a VMware vSphere datastore, and pairing it with a corresponding Red Hat OpenShift storage class. These mappings are essential to facilitate the seamless transfer of virtual machines from the source to the target clusters.

Portworx recommends creating a storage class specifically for the migration of virtual machines. This storage class matches the vm-storage storage class found earlier in this document with the exception of the volumeBindingMode parameter which should be changed to “Immediate.”

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: mtv-storage
provisioner: pxd.portworx.com
parameters:
 repl: "3"
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

When creating a storage mapping, choose this storage class shown above or a similar storage class with volumeBindingMode set to Immediate. This storagemap can be created through the OpenShift Console or with the YAML found below and customized for your environment.

```
apiVersion: forklift.konveyor.io/v1beta1
kind: StorageMap
metadata:
 name: storagemap-name
 namespace: openshift-mtv
spec:
 map:
   - destination:
       storageClass: mtv-storage # Or your custom storage class
     source:
       id: [DATASTORE_ID]
 provider:
   destination:
     apiVersion: forklift.konveyor.io/v1beta1
     kind: Provider
     name: host
     namespace: openshift-mtv
   source:
     apiVersion: forklift.konveyor.io/v1beta1
     kind: Provider
     name: [PROVIDER_NAME]
     namespace: openshift-mtv
```

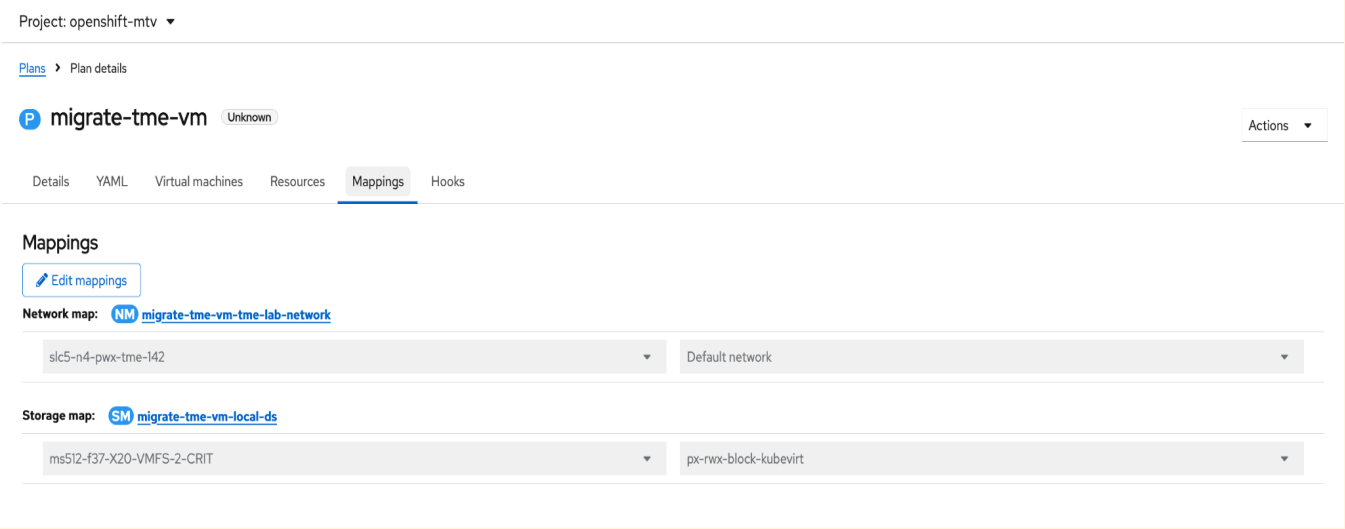

Figure 11. OpenShift Migration Plans

Summary

This Portworx with Red Hat OpenShift Virtualization Reference Architecture Addendum provides supplemental design guidance for integrating Red Hat OpenShift Virtualization with Portworx Enterprise in bare metal environments. Expanding on the foundational architecture outlined in the "Red Hat OpenShift with Portworx on Bare Metal" reference architecture, this document offers an approach to managing both virtualized and containerized workloads in a unified Kubernetes platform.

The architecture enables organizations to modernize existing virtualization infrastructures by migrating traditional workloads to a cloud-native environment without compromising on performance or flexibility. By leveraging Portworx enterprise-grade storage solutions, this design delivers scalability, security, and high availability, ensuring that virtual machines (VMs) and containers can coexist seamlessly within a single infrastructure.

Key features of this architecture include support for live migration of virtual machines, automated storage provisioning, and integration with Portworx's advanced storage capabilities for high-performance workloads. This document outlines critical considerations for storage configuration, performance optimization, and capacity planning, offering best practices for setting up ReadWriteMany (RWX) storage for live migrations, managing persistent volumes, and scaling the infrastructure. Additionally, the addendum addresses operational factors such as snapshot management, upgrades, and the use of the OpenShift Migration Toolkit for Virtualization (MTV) for efficient migration from legacy platforms.

By adopting this architecture, organizations can future-proof their infrastructure, drive digital transformation, and achieve consistent operational efficiency across both virtualized and containerized environments.