TECHNICAL WHITE PAPER

Portworx on SUSE Rancher Bare Metal

A validated architecture and design model to deploy Portworx® on SUSE Rancher Kubernetes Engine 2 running on bare metal hosts

Contents

Introduction 3

About This Document 3

Target Use Cases 3

Technology Overview 4

SUSE Linux Enterprise Server 4

SUSE Rancher Prime 4

SUSE Rancher Kubernetes Engine 2 4

Portworx by Pure Storage 4

Assumptions 5

Planning and Architectural Overview 5

High-level Design 5

Server Roles 6

Portworx Cluster Configuration 8

Software Versions Used in This Reference Architecture 9

Design Considerations 10

Storage Considerations 10

Network Considerations 19

SUSE Rancher Prime Considerations 26

SUSE Linux Enterprise Server Considerations 27

Performance Considerations 31

Security Considerations 34

Monitoring Considerations 40

Operational Considerations 41

Installation Methods and Procedures: Rancher Prime 41

Installation Methods and Procedures: SLES 43

Installation Methods and Procedures: RKE2 54

Installation Methods and Procedures: Portworx 56

Workload and Volume Considerations 67

Scaling 71

Backup and Disaster Recovery 74

Upgrading 74

Logging and Monitoring 79

Summary 85

Appendix: Portworx Configuration Example 86

Introduction

Modern applications rely on containers and Kubernetes for orchestration, but they still need persistent storage. SUSE Rancher Kubernetes Engine 2 (RKE2), a lightweight and robust Kubernetes distribution, simplifies deploying, running, and managing modern applications across diverse environments, including public cloud, on-premises, hybrid cloud, and edge. To support stateful applications on RKE2, organizations need a reliable data services platform like Portworx by Pure Storage®. Portworx provides features such as replication, high availability, encryption, capacity management, disaster recovery, and data protection. By using Portworx with RKE2 on bare metal servers, organizations can avoid the complexity of building a custom Kubernetes storage layer and speed up their application modernization efforts.

About This Document

This Portworx reference architecture provides a validated design for deploying Portworx on SUSE Rancher Kubernetes Engine 2 (RKE2), running on SUSE Linux Enterprise Server (SLES), which operates directly on bare metal hardware. It is designed for Kubernetes administrators and cloud architects who are familiar with Portworx. The audience should also understand basic RKE2 and SLES concepts, as well as hardware configurations like disk and network setups. The document focuses on three key technical areas:

1. **Planning and architecture overview**: This section outlines the high-level architecture for deploying Portworx on RKE2 with SLES. It includes requirements for configuring storage and storageless nodes, as well as recommendations for physical disk and network setups.
2. **Design considerations**: This section dives into detailed requirements and recommendations for the design phase. It covers RKE2 and SLES-specific needs, networking, capacity planning, high availability, resource performance, security, and monitoring.
3. **Operations considerations**: This section addresses operational tasks like installation methods, upgrades, data protection, scaling, logging, and monitoring. It also includes guidance for running stateful containerized applications in production.

Target Use Cases

This document provides detailed guidelines and best practices for deploying Portworx on SUSE Rancher Kubernetes Engine 2 (RKE2), running on SUSE Linux Enterprise Server (SLES) with bare metal servers as the underlying infrastructure. By following these recommendations, RKE2 users will be able to deploy Portworx in a way that ensures stability, reliability, and optimal performance.

Once Portworx is deployed using this framework, users can confidently run any type of stateful application within their RKE2 environment. The robust data management features of Portworx, combined with the efficient orchestration and management capabilities of RKE2, enable seamless and efficient storage operations for various applications that need persistent storage.

This document focuses on providing a stable and general deployment framework for Portworx on RKE2 with SLES. While it doesn't offer application-specific recommendations, the strategy outlined is designed to be widely applicable. This ensures reliable and scalable storage support for any application running within the RKE2 and Portworx ecosystem.

By following the best practices and guidelines provided, organizations can achieve a resilient and high-performing storage solution that meets the needs of diverse and demanding stateful applications.

Technology Overview

This section highlights the key features and benefits of the technologies used in this reference architecture (RA). Each technology plays a critical role in creating a robust, scalable, and reliable infrastructure for running stateful applications on SUSE Rancher Kubernetes Engine 2 (RKE2) with Portworx on SUSE Linux Enterprise Server (SLES) and bare metal hardware.

SUSE Linux Enterprise Server

SUSE Linux Enterprise Server (SLES) is a robust and versatile operating system designed to meet the demands of mission-critical workloads across a variety of environments, including on-premises data centers, cloud platforms, and edge deployments. Known for its reliability, SLES ensures consistent performance and uptime, making it a trusted choice for enterprise applications.

SUSE Rancher Prime

SUSE Rancher Prime is an open, interoperable, and extensible cloud native platform that delivers stress-free modern infrastructure management, enabling you to securely deploy, run, and manage container and VM workloads anywhere—from data center to cloud to edge.

SUSE Rancher Prime ensures consistency across your landscape with a unified interface, complete lifecycle management, and policy-driven automation. It supports your governance and compliance requirements with industry-leading container security and observability, centralized authentication, role-based access control (RBAC), and secure and trusted software delivery through the Application Collection. Additionally, you have access to a broad ecosystem of third-party applications, extensions, and integrations to streamline your operations and deliver the services and features you need.

SUSE Rancher Kubernetes Engine 2

SUSE Rancher Kubernetes Engine 2 (RKE2), also known as RKE Government, is SUSE's next-generation Kubernetes distribution.

It is a fully conformant Kubernetes distribution that focuses on security and compliance within the U.S. Federal Government sector.

To meet these goals, RKE2:

- Provides defaults and configuration options that allow clusters to pass the CIS Kubernetes Benchmark v1.6 or v1.23 with minimal operator intervention
- Enables FIPS 140-2 compliance
- Regularly scans components for CVEs using trivy in our build pipeline

Portworx by Pure Storage

Portworx is a software-defined, container-native storage solution designed for running highly available containerized applications across multiple nodes, cloud instances, regions, data centers, or even different cloud providers. It enables workload migration between clusters in hybrid or multi-cloud environments, supports hyperconverged deployments where data resides on the same host as applications, and provides programmatic control over storage resources. With enterprise-grade capabilities like disaster recovery, snapshots, encryption, and container-granular backup, Portworx ensures resilient, scalable, and efficient storage management for modern cloud-native applications.

Assumptions

This document includes a list of assumptions that outline the baseline configurations and settings used in this reference architecture. These assumptions are not recommendations or requirements for all deployments but rather reflect the specific setup we have implemented. They are intended to ensure that users following this guide exactly will understand the foundational configurations that influenced the build.

- **Static IP addresses**: All SLES nodes are configured to use static IP addresses to ensure consistent network connectivity and predictable behavior.
- **DNS configuration**: DNS is set up with entries for the SUSE Rancher Server hostnames and all SLES node hostnames, allowing for proper name resolution within the cluster.
- **Self-signed certificates**: SUSE Rancher Prime is configured to use self-signed certificates for secure communication.
- **Hostname configuration**: Each SLES node has a properly configured and unique hostname, aligned with the DNS entries.
- **Network interface mapping**: The administrator has manually mapped physical network interfaces to Linux devices, ensuring accurate configuration of network connectivity.
- **Disk device mapping**: Physical disk devices are mapped to Linux devices, enabling correct storage allocation and management.

By adhering to these assumptions, users can replicate the environment as described in this document and minimize unexpected variances during deployment.

Planning and Architectural Overview

This section provides an overview of various components of the design. This includes diagrams, definitions and high level information.

High-level Design

This section provides a planning and architecture overview for deploying SUSE Rancher Kubernetes Engine 2 (RKE2) with Portworx on SUSE Linux Enterprise Server (SLES) using bare metal infrastructure. The focus is on designing a high-level architecture that ensures scalability, reliability, security, and performance.

Key architectural components and their interactions are outlined, supported by logical diagrams. These visuals show the overall system structure, making it easier to understand how the components work together to support production workloads.

This section offers a summary of the architecture, giving you a clear picture of the system before diving into detailed explanations in the following sections.

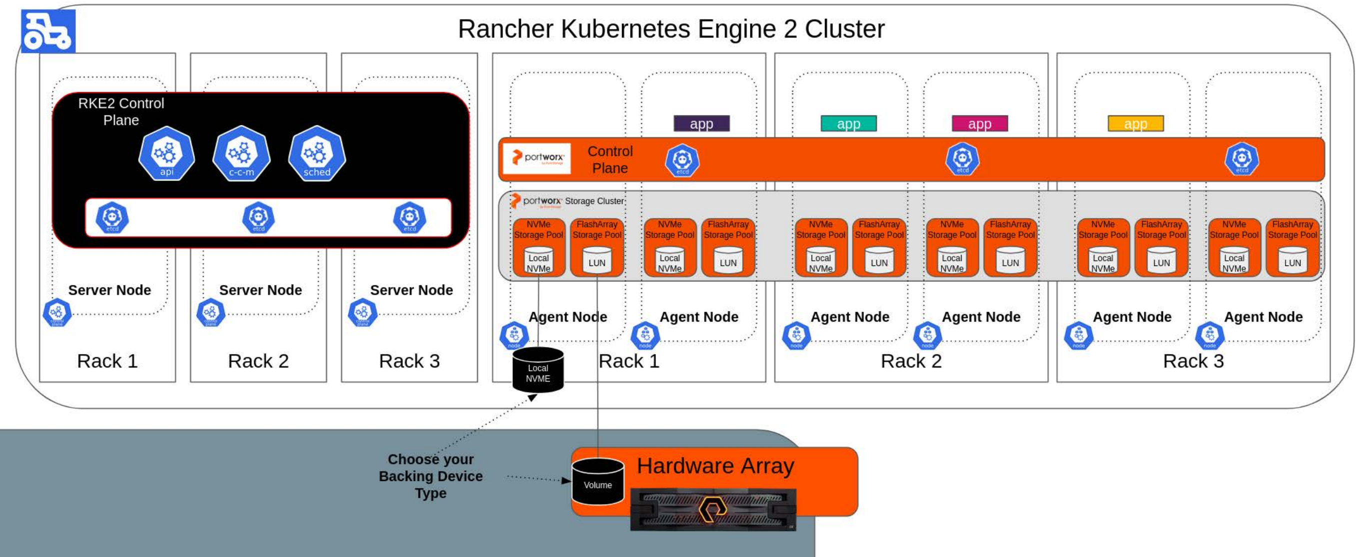

FIGURE 1 General reference architecture

In the above diagram, we have grouped control-plane and worker nodes into racks. Racks are used generically to define fault domains. Fault domains could be racks, rows, or sections of your datacenter. When grouping infrastructure into fault domains, it is helpful to think about what infrastructure is shared and what infrastructure is separate between our proposed fault domains. In general, fault domains should have the fewest shared infrastructure components (such as power, network, etc.) between them.

Portworx supports defining fault domains by using Kubernetes node labels. Portworx will ensure that data will be spread across different fault domains in your datacenter, so that a node or zone failure will not cause data loss.

It is ideal to have three fault domains for a Kubernetes cluster, both for Portworx, as well as control-plane redundancy. Portwork provides its own key-value store called KVDB that is deployed as pods using Portworx storage. Portworx will provision three KVDB pods. Three fault domains allows the cluster to maintain quorum if one domain goes offline.

Server Roles

This section covers the different roles that our Rancher Kubernetes cluster will use.

Server (Control-Plane) Nodes

In this architecture, the control-plane nodes are the backbone of the Kubernetes cluster, responsible for managing the state and orchestration of the cluster. This design includes three control-plane nodes to ensure high availability and fault tolerance.

Each control-plane node will have an integrated etcd database, which is co-located on these nodes to minimize latency and ensure consistency for critical cluster data. By co-locating etcd with the control-plane, the architecture reduces dependency on external systems, enhancing reliability and simplifying the overall design.

The control-plane nodes are dedicated solely to cluster management and will not host any application workloads or Portworx components. This separation of concerns improves cluster security and performance by isolating critical control-plane functions from potential resource contention caused by application workloads. This configuration ensures that the control-plane remains responsive and stable, even under heavy workloads or during maintenance operations, providing a solid foundation for the rest of the Kubernetes cluster.

Agent (Worker) Nodes

In this architecture, six worker nodes are configured to handle application workloads and provide storage capabilities for the Kubernetes cluster. These nodes are divided into two categories: **storage nodes** and **storageless nodes**, each serving specific roles within the environment.

Storage nodes are agent (worker) nodes equipped with local disks beyond the boot disk. These disks are utilized by Portworx to provide scalable and resilient storage for stateful applications. Portworx dynamically provisions and manages storage on these nodes, enabling features such as replication, encryption, and high availability. Storage nodes play a vital role in ensuring data durability and performance for applications requiring persistent storage.

Storageless nodes are agent (worker) nodes that do not have any Portworx storage configured on the node. Despite this, they can still consume storage resources via Portworx, leveraging the distributed nature of the storage platform. Storageless nodes allow applications to scale without being limited by local storage capacity, providing flexibility in resource allocation and workload distribution.

This combination of storage and storageless nodes creates a balanced and efficient setup, ensuring that the cluster can support diverse application requirements while maintaining high performance and scalability. The six-worker-node configuration offers sufficient capacity and redundancy to handle production workloads, delivering a robust foundation for both stateless and stateful applications.

Utility Infrastructure

In addition to the Kubernetes cluster, this architecture includes a separate utility infrastructure that is outside the scope of this document. This infrastructure is designed to support essential services required for the operation and management of the environment.

The utility infrastructure hosts services such as DHCP, DNS, and web services for tools like AutoYaST and SUSE Rancher Prime. DHCP and DNS services help with network management by automatically assigning IP addresses and ensuring seamless name resolution across the environment. The web services support AutoYaST for automated system deployment and configuration, as well as SUSE Rancher Prime for managing and orchestrating the Kubernetes cluster.

While this utility infrastructure is crucial for the overall functionality of the environment, its details are outside the scope of this reference architecture, which focuses on the Kubernetes and storage components. However, this infrastructure complements the cluster by providing the necessary network and management services to ensure smooth operations.

This infrastructure can be shared between multiple downstream clusters.

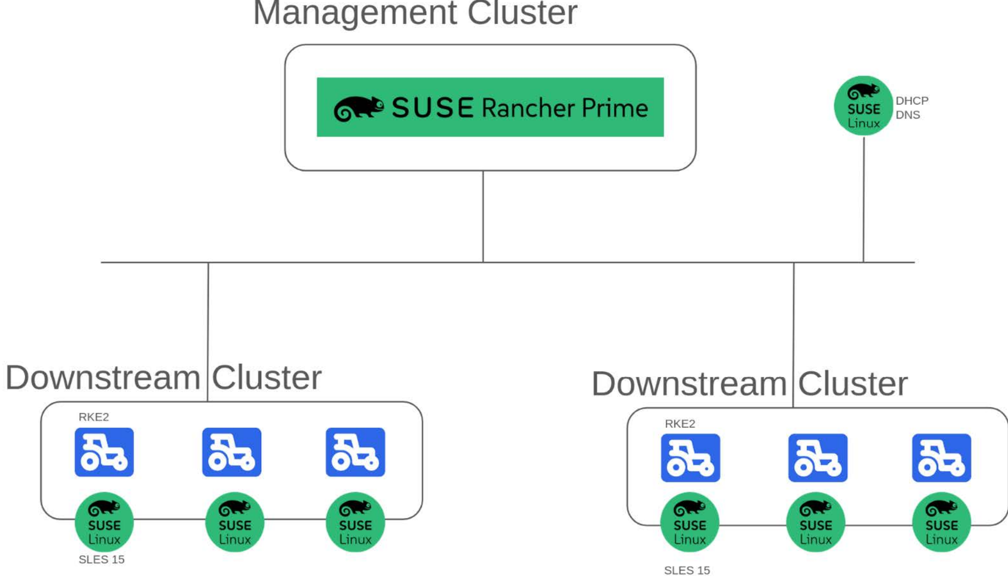

FIGURE 2 . Rancher diagram

Address Assignment and Name Resolution

Although DHCP is typically used for automatic configuration, in this architecture, we will be using static IP addresses for all nodes. Static IPs ensure consistency and simplify management, especially in production environments.

While DHCP can help with automatic IP assignment, we can further streamline the process using deterministic MAC addresses (such as those provided by Cisco UCS) or by pre-defining IP addresses and AutoYaST for each node. This approach allows for more control and predictability in the network setup.

We will be manually assigning host names to the nodes, but this process can be automated through DNS. By configuring DNS properly, host names can be automatically resolved, reducing manual intervention and ensuring efficient communication between the nodes in the cluster.

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:

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 SUSE Rancher Kubernetes Engine 2 (RKE2)

Before deploying Portworx, consider the following points to prepare RKE2 and the physical nodes:

Node configuration:

- At least six storage nodes is the minimum recommended for production environments. Using fewer nodes is not advised unless the load is minimal, such as in a development setup.
- Ensure the RKE2 cluster spans multiple availability zones or fault domains, and distribute storage nodes evenly across them. A minimum of three zones is recommended to enhance fault tolerance and availability.

Node labeling:

- Apply the label `portworx.io/node-type: storage` to all storage nodes. This label helps Portworx identify storage nodes and automatically provision storage for new nodes.
- If you plan to use a hyper-converged architecture where all worker nodes participate in the Portworx storage cluster (and there are no storageless nodes), this label is not required.

Resource recommendations:

- Each storage node should have a minimum of 8 CPU cores and 32 GB of RAM to meet Portworx's requirements. For further details, refer to the Hardware Resource Considerations section.

Software Versions Used in This Reference Architecture

Software	Version
Portworx Enterprise	3.2.1
STORK	24.3.3.1
SUSE Rancher Prime	v2.9.4
SUSE RKE2	v1.30.1 +rke2r1
SUSE Linux Enterprise Server	15 SP5
Kernel	5.14.21-150500.53-default

TABLE 1 Software versions used in this reference architecture

Design Considerations

This section covers the factors and decision points that go into this Rancher RKE2 and Portworx reference Architecture. It will explain the trade-offs of various design choices, as well as providing default recommendations.

Storage Considerations

In this section, we will be covering design considerations for our storage. Because Portworx is software defined it can support a variety of configurations. For example, some Portworx configurations will use local NVMe media, while others will use SAN appliance attached media.

The common requirement for Portworx storage is that it is presented as a block storage device to our worker nodes (e.g., /dev/sdb).

Defining Capacity Requirements

Proper storage sizing is essential for maintaining the performance and reliability of your Portworx deployment on SUSE Rancher Kubernetes Engine 2 (RKE2). An appropriately sized storage solution helps prevent bottlenecks, inefficiencies, and operational disruptions. To design a robust storage architecture, you need to assess key factors such as current and future data growth, I/O performance requirements, redundancy, high availability, and the ability to handle peak workloads. This ensures your storage infrastructure meets present demands while being scalable for future growth.

Portworx supports multiple storage device types within a cluster. For example, it is possible to have storage nodes with both SSDs and spinning hard drives in the same storage cluster. StorageClasses can then specify their preferred disk requirements.

Portwork organizes storage into storage pools. A storage pool is a group of one or more disks of the same class on a single node. Replicas are distributed between storage pools on different nodes within the same class of disk. For example, a PV will have its replicas on SSD pools on nodes 1 and 2, but will not use a magnetic replica on node 2.

It is possible for a node to have more than one storage pool.

Initial Cluster Capacity

Portworx supports both vertical and horizontal scaling, enabling your storage cluster to adapt dynamically to workload demands:

- **Vertical scaling**: Increases capacity on individual storage nodes.
- **Horizontal scaling**: Adds new nodes to the cluster to enhance capacity and redundancy.

These scaling features can be managed through automation tools like AutoPilot and Cloud Drives, which respond to workload demands in real time. While these capabilities reduce the strain of capacity planning, a well-planned initial sizing process remains crucial. Proper initial sizing ensures that your cluster is provisioned to handle expected workloads, providing a solid foundation for smooth scaling and efficient operation.

Factors for Initial Capacity Planning

When sizing your cluster, consider the following:

- **Number of volumes (PVCs):** Total Persistent Volume Claims in your cluster.
- **Average volume size**: Typical storage size per PVC.
- **Number of nodes**: The initial storage node count in your cluster.
- **Replication factor**: Portworx recommends a replication factor of 2 or 3 for high availability.

Calculating Total Cluster Capacity

Start by calculating the total storage required for your applications and their replicas to ensure high availability. Incorporate a growth factor to account for anticipated capacity increases. Use a table format to summarize total volume sizes, including replication and growth factors, for clear and organized planning.

This thorough approach to initial capacity planning sets the stage for efficient operations and ensures your cluster is prepared to handle workloads from day one.

Consider the below example for calculating the capacity requirements of your cluster.

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

TABLE 2 Volume sizing example

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 times 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)
18.85TiB	6	4TiB	24TiB

TABLE 3 Cluster sizing example

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

Care must be taken when planning for expansion beyond what was accounted for in configured hardware. It is important that a storage pool has adequate space. When a storage pool's utilization reaches 90% or 50GiB (whichever number is smaller) the storage pool will go offline. This will not cause a storage outage as other replicas will be marked as the active replica, but this should be avoided as it reduces the availability of the replicas in that pool as they are no longer available. Pods that are using a local volume mount will need to be restarted.

It is possible to add storage to a node in one of two ways:

1. A storage pool can be expanded by expanding disk(s) within a pool. This operation will take the pool offline when the expansion is in progress. This results in Pods that are using local volume mounts to be restarted.
2. An additional storage pool can be added to the same storage node. This operation is non-disruptive, but requires manual rebalancing of replicas to decrease storage utilization of the previous pool.

Defining Performance Requirements

When designing our storage devices for our anticipated workloads, it is important to understand the requirements of your applications, as well as the performance characteristics of the selected block devices. We will also need to understand how Portworx affects these numbers.

Storage performance involves the interaction of a number of performance metrics:

- Average and peak (95th percentile) input/output operations per second (IOPS)
- Latency expectations
- Average request size
- Write bias (how many of our I/Os are writes)

Table 4 shows an example of this.

Workload	Amount	Peak I/O	Average I/O	Latency Requirement	Average Request Size	Write Bias	Bandwidth
SQL	100 PVCs	500	300	1ms	16KB	80%	468MB/s
Workload 2	50 PVCs	1000	500	5ms	32KB	20%	781MB/s
TOTAL	150 PVCs	100,000	55,000				

TABLE 4 Performance sizing example

Input/output operations (IO) operations can have different impacts on the underlying storage media depending on the size of the I/O. A large request has a larger impact on the storage media as the read or write operation needs to be translated to the underlying sectors.

It can be helpful to normalize the above I/O for comparison purposes. By thinking of all I/O operations as a 4k request size, we can ensure that the impact of larger request sizes are reflected in our performance calculation. For Instance, we can think of our average SQL I/O as 1200 4k IOPS.

It is an oversimplification to say that a 16KB request size actually results in four 4KB I/Os. Other factors such as the filesystem, kernel, and device drivers affect how I/O will be read and written to the actual storage sectors.

Portworx adds overhead and resiliency to our I/Os. Remember that Portworx provides resiliency, so we need to amplify the I/O that will make it to our underlying storage devices. For example, if we are using a replication factor of 3, we will need to write 3 copies.

Additionally, Portworx has overhead for copy-on-write snapshots and other operations. When using a copy-on-write snapshot, the data is locked and a new active data set is used for all changes to preserve the original data set. When a write happens, the original data is copied to the new active data location (a read, then a write) and then the data is modified in that new area. This overhead further increases the write I/O requirements by a factor of 1.4 as a conservative estimate, but this number is variable depending on a number of factors such as write bias, and the number of snapshots in use.

Continuing the above example:

Profile	4k Read I/O	4k Write I/O	Overhead	Replication Factor	Total 4k I/O to Disk
Peak	360,000	240,000	1.4	3	1,368,000
Average	184,000	136,000	1.4	3	755,200

TABLE 5 Cluster performance example

The above performance requirements are for the entire cluster's disk subsystem and can be useful for selecting disks. This calculation does not account for I/O that is fragmented (such as all of your peak reads coming from a few devices) so it is prudent to have more performance than you need.

Storage Pools

By following these guidelines, organizations can deploy a robust and scalable Portworx architecture on RKE2, 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 methods available to increase 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). Using a single drive allows for the storage pool to be resized by expanding the backing disks. When the Portworx storage cluster is expanded by vertically scaling the backing drives, it does not require a data rebalance. For this reason, Portworx recommends using a single disk as the backing disk for the storage cluster in a bare metal environment for the storage pool where possible.

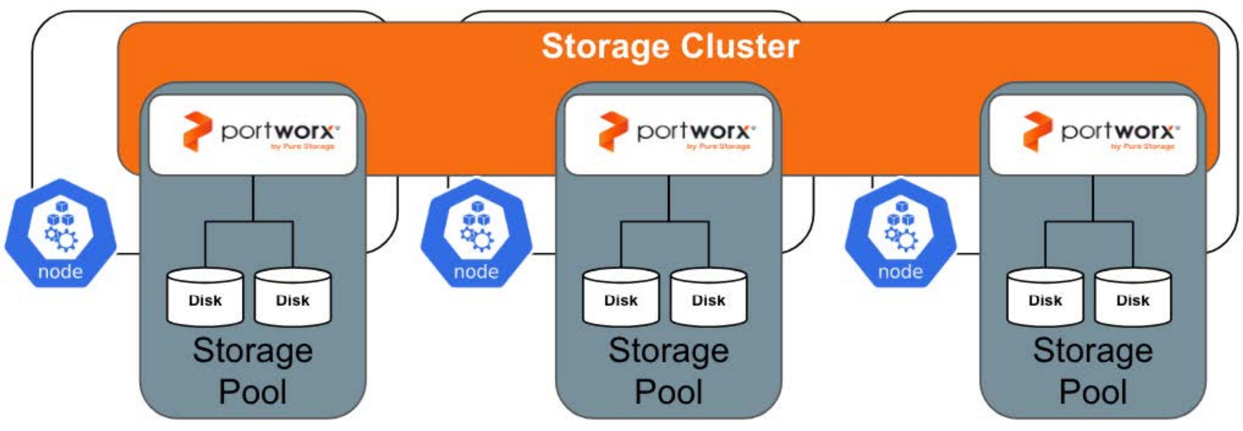

FIGURE 3 Portworx storage pools

Journal Devices

Portworx recommends using a journal device to handle metadata writes. Journal writes are small and frequent, so an SSD or NVMe drive is recommended for this purpose. 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. 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.

If using local drives for backing disks, it is recommended to have the journal device automatically created from the existing disks instead of dedicating a large disk for a 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.

KVDB Considerations

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 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 Synchronous Disaster Recovery (SyncDR) solution.

When deploying the internal KVDB, Portworx recommends using a dedicated 64GB disk for the metadata disk if available. The metadata requires flash media for consistent operation. 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 storage cluster. The metadata disk only requires three of the RKE2 worker nodes for proper operation so this 64GB disk only needs to be available on the nodes where the KVDB will live. In the event that there is a KVDB node failure, Portworx will promote another node to become a metadata node. For this reason, Portworx recommends specifying six nodes that are capable of running the KVDB and labeling them with the px/metadata-node=true key-value pair. These nodes should be available across the fault domains to ensure that there are always two KVDB nodes available to maintain quorum in the event of a host failure.

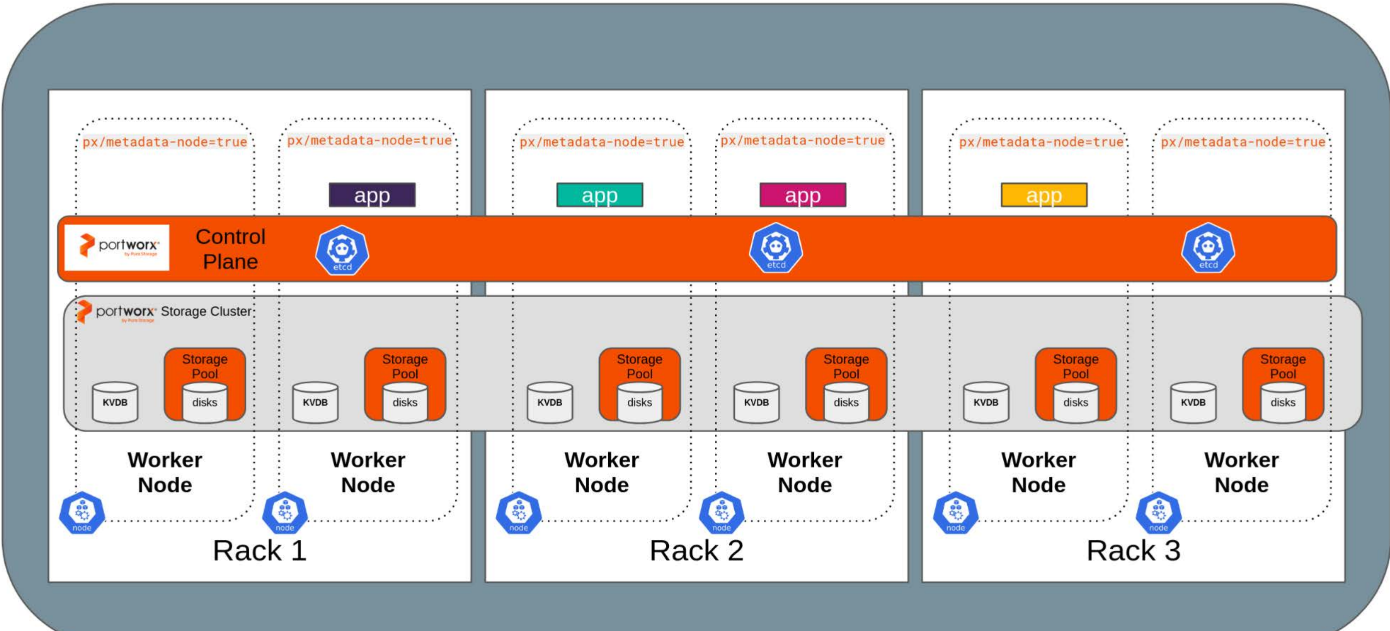

FIGURE 4 KVDB reference

In the event that the etcd 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.

Backing Disk Selection

When configuring a Portworx storage cluster on RKE2, 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 Pure Storage 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 HDDs.
- **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. RAID configurations are only applicable to local disks.

- **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 to operate as a storage node, 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 SUSE Rancher Kubernetes Engine 2 is well-equipped to handle the demands of your applications, providing high performance, reliability, and scalability.

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 SAN for Portworx backing disks.

Block devices are typically presented to physical hosts (RKE2 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 and iSCSI connections:

- **Fibre-Channel**: If using Fibre-Channel, ensure proper zoning in the SAN Fabric for the nodes.
- **iSCSI**: When using iSCSI, it is crucial to dedicate Ethernet interfaces solely for storage purposes, rather than sharing them with RKE2 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.

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

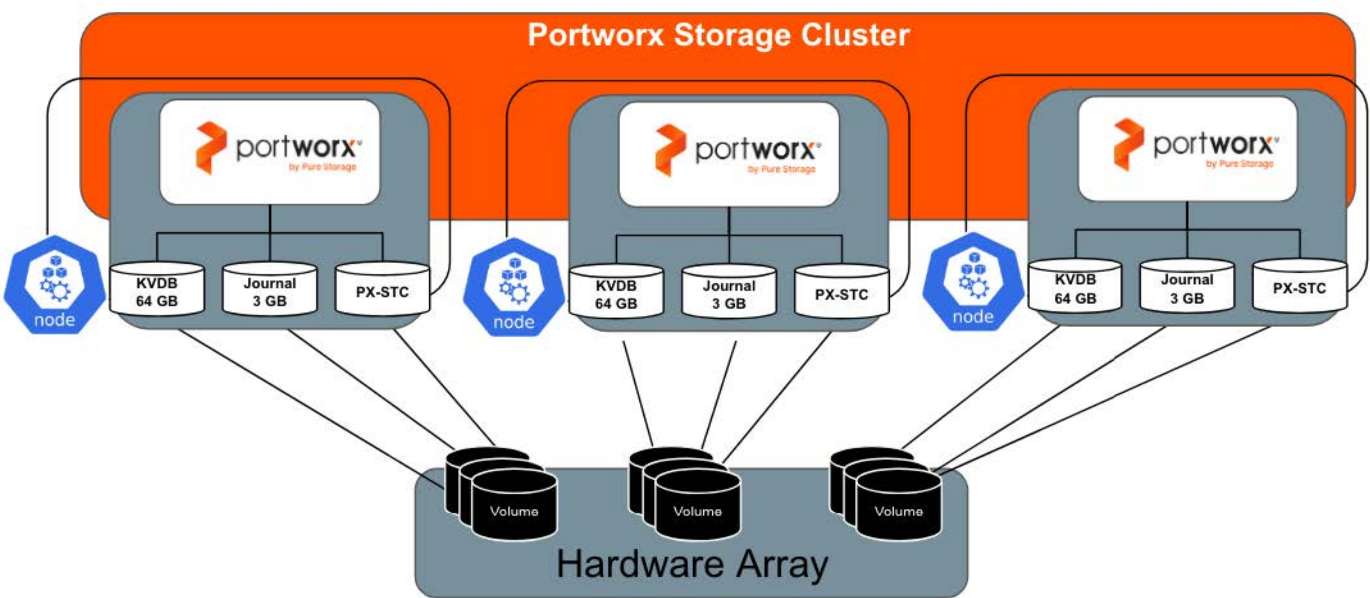

FIGURE 5 Third-party array diagram

FlashArray Considerations

For Pure Storage FlashArray users, the process of managing the backing disks on the storage is similar to a generic array, but adds Portworx cloud drive functionality to simplify provisioning and scaling of backing disks. Similar to any hardware array, the Fibre-Channel or iSCSI interfaces should be configured with proper zoning and masking, as well as configuring devices for Jumbo Frames.

Portworx Enterprise has cloud drives which is a way for the Portworx control plane to manage Pure Storage disk arrays and persistent disks on Amazon Web Services (AWS), Azure, Google Compute Platform (GCP), IBM Cloud, and VMware vSphere. To set up Cloud Drives the RKE2 worker nodes must be configured to disable Secure Boot, and to apply a multipathd configuration file with the appropriate settings.

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 Pure Flash Array to create volumes, expand volumes, etc. providing automation between Portworx and the Flash Array.

When deploying Portworx, specify the desired configuration values as usual but choose theFlash Array as the provider. When it comes to the capacity configuration section of the config, specify a 3GB journal volume, and the desired capacity for your Portworx storage pool. When you apply the storage cluster configuration to the RKE2 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.

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 KVDB 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.

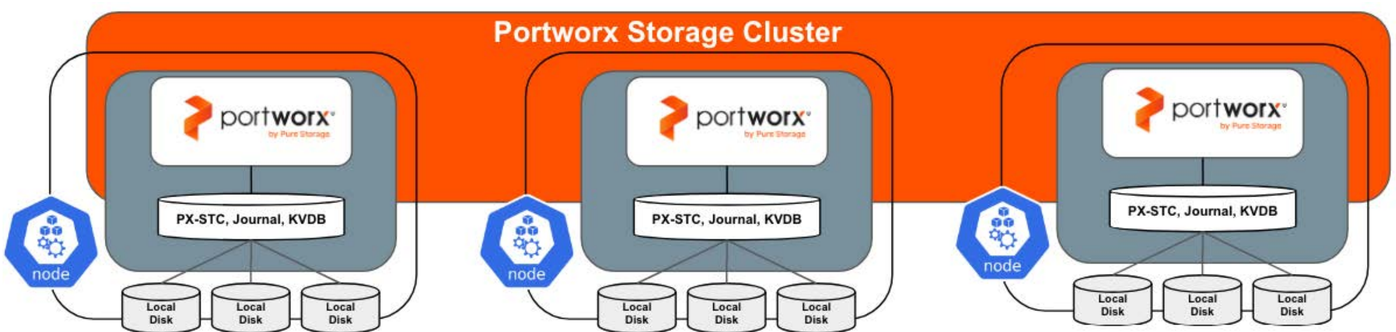

FIGURE 6 Local disk configuration diagram

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 to add redundancy, such as RAID 1, 5, 6, or 10, while not necessary for maintaining high availability, can prevent a single disk failure from taking a Portworx node offline. Replication traffic is configured to use a dedicated back-end network as outlined in the Network Considerations section.

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 Pure Storage 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 supports two storage configurations: PX-StoreV1 and PX-StoreV2. We have selected PX-StoreV2 for our reference architecture.

PX-StoreV1 Considerations

PX-StoreV1 is the default store for Portworx Enterprise. It is suitable for general I/O use.

PX-StoreV1 offers a few features above PX-StoreV2 implementations:

- XFS volumes
- Aggregated volumes
- PX-Cache

PX-StoreV1 does not support PX-Fast, which is used for high intensity write workloads with consistent latency.

In the future, PX-StoreV1 will be replaced by PX-StoreV2.

PX-StoreV2 Considerations

PX-StoreV2 is a Portworx datastore optimized for supporting I/O intensive workloads for configurations utilizing high performance NVMe class devices. It's the platform that enables the PX-Fast feature that targets workloads requiring high data ingestion rates with consistent latency.

PX-Fast is a Portworx feature that enables an accelerated I/O path for the volumes that meet certain prerequisites. It is optimized for workloads requiring consistent low latencies. PX-Fast is built on top of PX-StoreV2 datastore.

PX-StoreV2 should be used in most scenarios, but it is worth noting that the following Portworx features do not work with PX-StoreV2:

- XFS volumes
- Aggregated volumes
- PX-Cache
- Adding disks to an existing storage pool
- Online disk expansion of the backing disk in a storage pool

Because our reference architecture runs on bare metal, we will be using PX-StoreV2.

Network Considerations

This section provides architectural detail to configuring network interfaces on our SUSE Linux Enterprise Server nodes.

Bonding Overview

SUSE Linux Enterprise Server (SLES) offers several options for bonding Ethernet interfaces, allowing you to combine multiple physical network interfaces into a single logical interface. This setup improves performance, fault tolerance, or both. For additional information about bonding modes, see Managing Network Bonding Devices

For our reference architecture, we will be using bond mode 1 - active-backup. Only one network interface is active. If it fails, a different interface becomes active, which provides fault tolerance. This is the default mode, and no specific switch support is required.

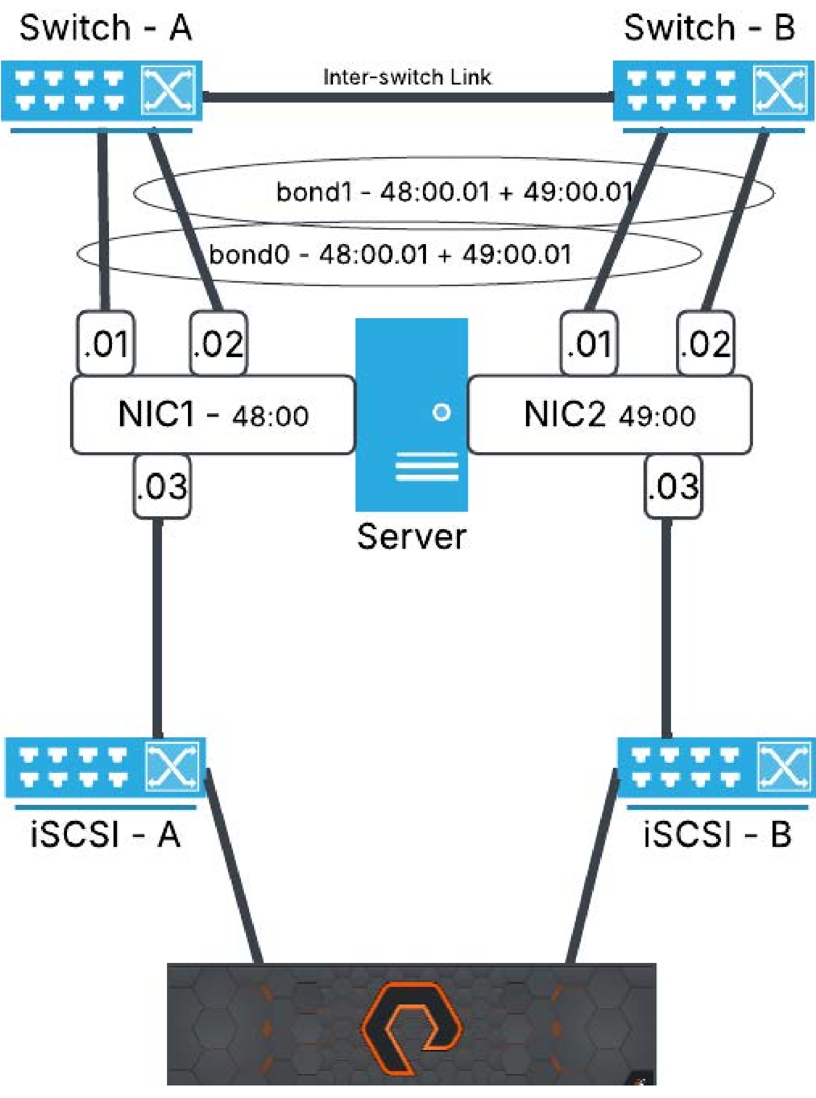

FIGURE 7 Networking diagram

Front-end Network: Bond0

The front-end network in our reference architecture is designed to handle both management traffic and container communication. This network bond is configured to carry the default gateway, ensuring seamless connectivity for both control plane operations and application communication. To achieve reliability and simplicity, we use an active-passive bond (mode 1), which is switch-independent and provides failover protection.

Physical Interface and Bonding Requirements

The front-end bond, named bond0, is composed of at least two physical network interfaces connected to separate physical switches. This redundancy ensures that a failure in one switch or interface does not disrupt traffic flow. In this active-backup configuration, one interface actively handles traffic, while the other remains on standby to take over in case of failure.

To maintain proper failover functionality, it is critical that the switches connected to the bonded interfaces act as a single logical switch. This can be achieved through appropriate networking configurations, even if the switches do not use advanced protocols like The Link Aggregation Control Protocol (LACP).

Switch Port Configuration

Each physical interface of the bond must connect to switch ports configured with the following settings:

- **Access VLAN:** The ports should be assigned to the designated VLAN (VLAN 13 in this case). Note that non-Cisco switches may refer to this as an untagged port.
- **Spanning-tree protocol (STP):** Spanning-tree should be disabled to prevent potential blocking of bonded traffic.
- **Port security:** This should be disabled to avoid unintended disruptions caused by strict MAC address enforcement.

Below is an example of the switch port configuration of a Cisco Nexus switch for the front-end network:

```
interface Ethernet1/1

  switchport

  switchport mode access

  switchport access vlan 13

  no switchport port-security

  spanning-tree port type edge

  spanning-tree bpdufilter enable

  spanning-tree bpduguard disable

  spanning-tree guard none
  no shutdown
```

Isolating this VLAN from other network traffic ensures both security and optimal performance for management and container communication.

Example Bonding Configuration

The following configuration represents an active-passive bonding setup for bond0. It demonstrates how two physical interfaces (eth1 and eth3) are configured to operate in fault-tolerance mode, with one interface actively managing traffic and the other on standby.

```
sles4:/etc/wicked # cat /proc/net/bonding/bond0

Ethernet Channel Bonding Driver: v5.14.21-150500.53-default

Bonding Mode: fault-tolerance (active-backup)

Primary Slave: None

Currently Active Slave: eth1

MII Status: up

MII Polling Interval (ms): 100

Up Delay (ms): 0

Down Delay (ms): 0

Peer Notification Delay (ms): 0

Slave Interface: eth1

MII Status: up

Speed: 10000 Mbps

Duplex: full

Link Failure Count: 0

Permanent HW addr: 00:50:56:b5:a2:58

Slave queue ID: 0

Slave Interface: eth3

MII Status: up

Speed: 10000 Mbps

Duplex: full

Link Failure Count: 0

Permanent HW addr: 00:50:56:b5:55:5f

Slave queue ID: 0
```

Back-end Network: Bond1

Our back-end network will handle data communication for Portworx, requiring a reliable and efficient configuration. We have selected an active-passive bonding mode (mode 1) for its simplicity and switch-independent setup. This configuration ensures failover protection, with one interface actively handling traffic and the other on standby to take over in case of a failure.

Physical Interface Requirements

The network bond should consist of at least two physical network interfaces connected to separate physical switches. This redundancy enhances fault tolerance and ensures continuous data flow even in the event of a hardware or switch failure.

While the switches do not need to support advanced configurations like link aggregation (LACP), they must act as a single logical entity. This can typically be achieved through proper network topology and switch configurations that support independent link failover.

Switch Port Configuration

The switch ports connected to the bonded interfaces must be set to the correct access VLAN and configured to disable spanning-tree for simplicity and to prevent unexpected blocking of traffic. Port security should also be disabled to avoid unintended disruptions. The VLAN designated for back-end traffic should be isolated from other network traffic to ensure security and performance.

Below is an example switch configuration of a Cisco Nexus switch for a back-end VLAN (VLAN 14):

```
interface Ethernet1/2
  switchport
  switchport mode access
  switchport access vlan 14
  no switchport port-security
  spanning-tree port type edge
  spanning-tree bpdufilter enable
  spanning-tree bpduguard disable
  spanning-tree guard none
  no shutdown
```

Although it is possible to use the same switches as the front-end network, dedicated switches for back-end traffic are preferred. This separation minimizes contention and ensures optimal performance for Portworx communication.

Example Bonding Configuration

Below is an example configuration for an active-backup bond. It demonstrates how the two interfaces (`eth0` and `eth5`) operate in active-passive mode, providing fault tolerance with one interface actively managing traffic and the other ready to take over if needed:

```
Ethernet Channel Bonding Driver: v5.14.21-150500.53-default

Bonding Mode: fault-tolerance (active-backup)

Primary Slave: None

Currently Active Slave: eth0

MII Status: up

MII Polling Interval (ms): 100

Up Delay (ms): 0

Down Delay (ms): 0

Peer Notification Delay (ms): 0

Slave Interface: eth0

MII Status: up

Speed: 10000 Mbps

Duplex: full

Link Failure Count: 0

Permanent HW addr: 00:50:56:b5:98:2d

Slave queue ID: 0

Slave Interface: eth5

MII Status: up

Speed: 10000 Mbps

Duplex: full

Link Failure Count: 0

Permanent HW addr: 00:50:56:b5:f3:ff

Slave queue ID: 0
```

This configuration provides a robust and reliable foundation for Portworx back-end communication, ensuring high availability and consistent performance. By isolating this traffic on a dedicated VLAN and using redundant switches, the architecture supports both reliability and security for critical storage operations.

iSCSI Networks

In this reference architecture, our iSCSI network interfaces will not use bonded interfaces. This is because iSCSI includes built-in failover mechanisms, which provide redundancy and reliability without requiring bonding at the network interface level.

For this design, the iSCSI traffic will use the network interfaces `eth2` and `eth4`. These interfaces should connect to two separate physical and logical switches to ensure redundancy and high availability. Each interface will have its own unique IP address and subnet configuration, allowing the network traffic to remain isolated and manageable.

Below is an example configuration of the `eth2` and `eth4` interfaces:

```
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
   link/ether 00:50:56:b5:f2:dc brd ff:ff:ff:ff:ff:ff
   inet 192.168.1.84/24 brd 192.168.1.255 scope global eth2
        valid_lft forever preferred_lft forever
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
   link/ether 00:50:56:b5:44:45 brd ff:ff:ff:ff:ff:ff
   inet 192.168.2.84/24 brd 192.168.2.255 scope global eth4
        valid_lft forever preferred_lft forever
```

The paired switch configuration should be set as access ports on the desired VLAN. This VLAN should be isolated from all other network traffic and should be considered confidential for security reasons.

See below for our switchport configuration of a Cisco Nexus switch examples:

```
interface Ethernet2/3
  description iSCSI-B
  switchport
  switchport mode access
  switchport access vlan 15
  no switchport port-security
  spanning-tree port type edge
  spanning-tree bpdufilter enable
  spanning-tree bpduguard disable
  spanning-tree guard none
  no shutdown
```

This setup ensures redundancy at both the network and storage levels by leveraging separate physical interfaces and switches. The isolation of iSCSI traffic to its own VLAN enhances security and reliability, while the avoidance of bonded interfaces allows the inherent failover capabilities of iSCSI to handle interface-level redundancy effectively. This configuration is well-suited for enterprise environments where reliability and performance are critical.

SUSE Rancher Prime Considerations

SUSE Rancher Prime serves as a centralized management layer for RKE2 clusters and other Kubernetes distributions, simplifying multi-cluster management. In this architecture, Rancher Prime is deployed via a Helm chart in the utility infrastructure. While the specifics of the SUSE Rancher Prime installation are outside the scope of this document, it is critical to note that SUSE Rancher Prime supports managing multiple Kubernetes clusters.

For details on installation and upgrades, refer to the SUSE Rancher Installation/Upgrade Documentation.

Required Privileges and Access Configuration

Setting up SUSE Rancher Prime and RKE2 clusters requires specific permissions and configurations:

- A user with Standard User or higher global permissions is necessary for cluster creation.
- Authentication to SUSE Rancher Prime's API will rely on a bearer token. This token can be generated and stored in environment variables to simplify subsequent operations. Refer to the Environment Variable Example section for implementation details.

Creating a New RKE2 Cluster

Creating a custom cluster in SUSE Rancher Prime generates installation commands to configure RKE2 on provisioned SLES nodes. SUSE Rancher Prime automates many cluster configuration parameters, streamlining the process. However, users should review the production readiness checklist to ensure compliance with RKE2 requirements.

Key requirements for Portworx integration:

- **Attached storage:** Worker nodes must have additional attached storage.
- **Minimum worker nodes:** At least three worker nodes with local storage are required.

Follow the SUSE Rancher documentation for configuring Kubernetes on existing nodes and specifying cluster parameters.

Pod Security Admission (PSA) Configuration

To enhance cluster security, SUSE recommends enabling Pod Security Admission with the `rancher-restricted` template. This template must be configured to exempt the `portworx` namespace to ensure compatibility. Updates to the PSA template apply across all downstream clusters using it. Ensure cluster updates are triggered after modifying the template.

Monitoring Configuration

SUSE Rancher recommends deploying the `rancher-monitoring` application chart on downstream clusters for Prometheus-based monitoring and alerting. This architecture has monitoring enabled without SSL using the default installation options. Note that the Portworx-provided Prometheus package cannot be used concurrently with Rancher monitoring.

Portworx Extension for Rancher UI

The Portworx extension integrates storage monitoring directly into the SUSE Rancher dashboard, providing actionable insights and simplifying cluster management. However, this extension is incompatible with PX-Security-enabled clusters, which are used in this reference architecture.

Reference Architecture Example

The cluster is configured with the following parameters:

Configuration Item	Value
Kubernetes Version	v1.30.1+rke2
Container Network	calico
CIS Profile	cis
Pod Security Admission Template	rancher-restricted
System Services	CoreDNS, NGINX Ingress, Metrics Server

TABLE 6 RKE2 cluster configuration values

Registration commands generated by SUSE Rancher Prime will be used to install RKE2 on SUSE Linux Enterprise Server nodes. These commands include the necessary parameters for control-plane and worker node configuration, ensuring a consistent deployment process.

This overview provides a high-level understanding of the components and configurations, allowing users to navigate the detailed procedures with clarity.

SUSE Linux Enterprise Server Considerations

Our reference design contains two server specifications.
First, control-plane nodes run ETCD and the RKE2 Control-Plane components. These nodes do not run any Portworx software and do not schedule workloads.
Worker nodes run Portworx containers, as well as user workloads. All of our worker nodes are configured identically.
For this reason, we will be using the Hyper-converged Architecture found in this document.

SUSE Linux Enterprise Server Overview

SUSE Linux Enterprise Server (SLES) is a robust, secure, and scalable operating system designed to meet the needs of modern IT infrastructure. Built on open-source foundations, it provides a reliable platform for mission-critical workloads, cloud deployments, containerized environments, and edge computing. SUSE is recognized for its focus on enterprise-grade stability, performance, and support for diverse hardware architectures.

Key Features

SUSE Linux Enterprise Server (SLES) offers enterprise-grade reliability with long-term support, ensuring stability and performance for mission-critical workloads. It supports diverse hardware architectures, including x86_64, ARM, IBM Power, IBM Z, and RISC-V, making it highly adaptable to various deployment environments. Security is a core focus, with features like Secure Boot, AppArmor, SELinux, and encrypted filesystems, providing robust protection against vulnerabilities. Additionally, SLES is certified to meet regulatory standards such as FIPS, Common Criteria, and GDPR compliance.

Modern IT environments benefit from SLES's seamless integration with cloud and container platforms. It supports Kubernetes and Docker for containerized workloads and is optimized for deployment in public cloud services like AWS, Azure, and Google Cloud. Management and automation are streamlined through tools such as YaST for system configuration and AutoYaST for automated deployments, along with SUSE Manager for centralized system management, patching, and compliance.

SLES is designed for high availability and scalability, featuring clustering capabilities and support for distributed storage solutions to ensure uptime and balanced workloads. Its rich software ecosystem is certified to work with popular enterprise applications, including SAP, Oracle, and Microsoft solutions. This makes SLES a versatile and powerful choice for businesses needing a reliable and adaptable operating system.

Hardware Resource Considerations

This section documents the configuration we used for this reference architecture, as well as the minimum and recommended requirements for Portworx.

Table 7 shows the server (control-plane) node reference configuration:

Hardware	Reference Configuration
CPU	4 cores for PX-StoreV2
RAM	16 GB
OS Drive	/dev/sda 100GB for / (root)
Network Connectivity	2 10Gbps Network Interface Cards
Operating System	SUSE Linux Enterprise Server 15 SP5
Secure Boot	Enabled

TABLE 7 Control-plane node resources

Table 8 shows the agent (worker) node reference configuration:

Hardware	Required Resources	Recommended Resources	Reference Configuration
CPU	4 cores for PX-StoreV1 8 cores for PX-StoreV2	8 cores	8 cores for PX-StoreV2
RAM	4 GB	8GB	32 GB
OS Drive	64GB / (root)	128GB / (root)	/dev/sda100GB for /(root)
KVDB/ Metadata Path	32 GB for KVDB (PX-StoreV1) 64GB for Metadata Path (PX-StoreV2)	32 GB for KVDB (PX-StoreV1) 64GB for Metadata Path (PX-StoreV2)	/dev/nvme0n1100GB
PX Store Backing Drives	8GB	128 GB	NVMe for metadata (4) 512GB NVMe for PX Store - /dev/nvme0n2 - /dev/nvme0n3 - /dev/nvme0n4 - /dev/nvme0n5
Network Connectivity			6 10Gbps Network Interface Cards
Operating System			SUSE Linux Enterprise Server 15 SP5
Secure Boot			Disabled

TABLE 8 Worker node resources

Note: It is important to size hardware for anticipated workloads above the minimum requirements of Portworx, which can be found in the Rancher Documentation

For specifics on the installation and configuration of SLES, see the Installation Methods and Procedures - SLES section of this document.

Performance Tuning

SUSE Linux Enterprise Server supports TuneD profiles.

TuneD optimizes SUSE Linux Enterprise Server systems using predefined profiles with settings tailored for different use cases. These profiles adjust several system settings, including CPU governor policies, disk I/O scheduling, network parameters, and kernel parameters, to enhance performance, energy efficiency, or other system characteristics. Each profile is designed for specific scenarios, such as high throughput, low latency, power saving, or virtualized environments.

Which tuning profile to select is outside of the scope of this document, but it is recommended to avoid “balanced,” “virtual-guest,” and “powersave” profiles to ensure that additional latency is not added to operations. This added latency can have a negative effect on the Portworx components.

Rancher Kubernetes Engine 2 Considerations

Although RKE2 can be configured manually, our reference architecture uses central configurations from SUSE Rancher Prime. For RKE2 architectural considerations, see the SUSE Rancher Prime Considerations section. For specific RKE2 installation procedures, see the Installation Methods and Procedures - RKE2.

High Availability Considerations

High availability (HA) is a critical component in the design of any resilient storage and compute infrastructure. While Portworx and RKE2 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 RKE2 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.

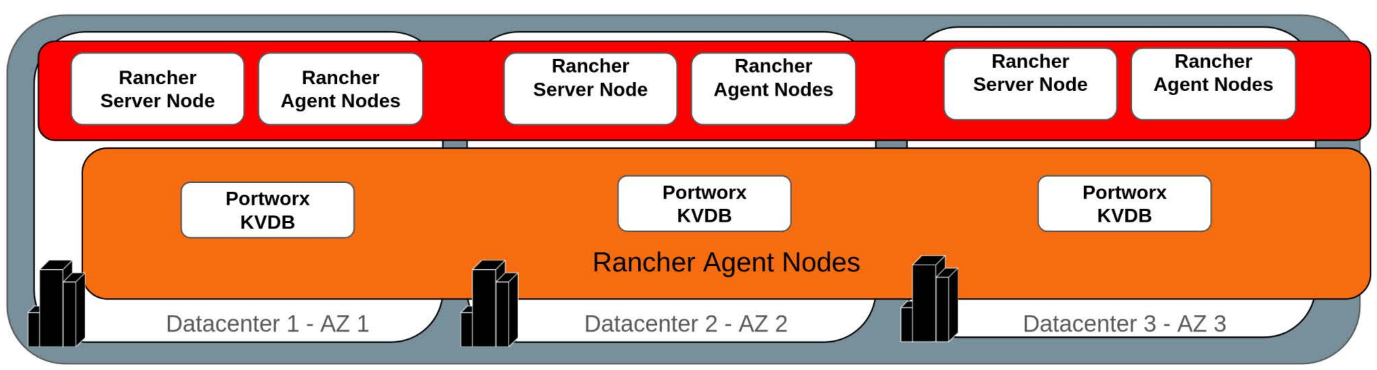

FIGURE 8 Data center fault domains

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 RKE2 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.

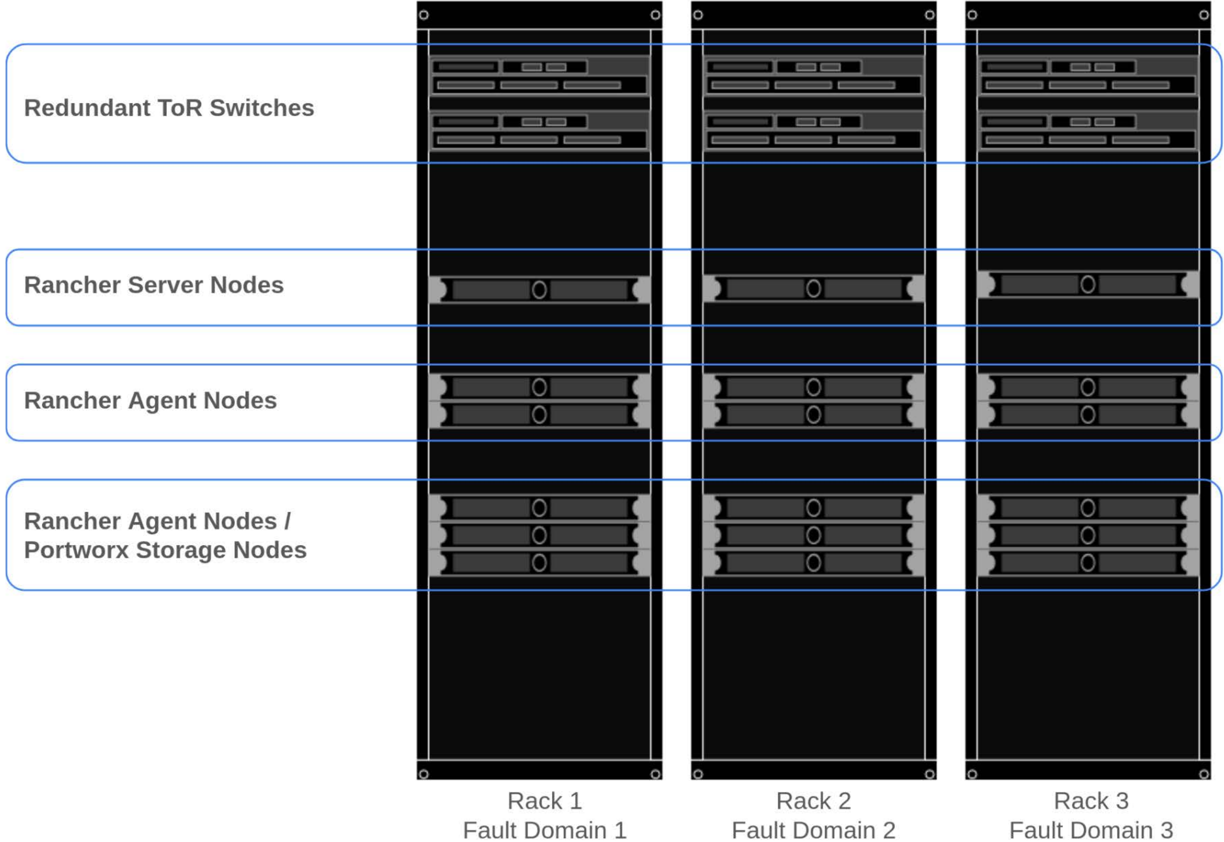

FIGURE 9 Rack fault domains

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

In cloud environments, the `topology.portworx.io/region` and `topology.portworx.io/zone` labels are automatically applied. In an on-premesis environment, it is required to specify labels manually.

Portworx uses a label called `topology.portworx.io/rack`. This label provides granular control over replica placement decisions:

`topology.portworx.io/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: kubernetes.io/portworx-volume
parameters:
  repl: "2"
  shared: "true"
  racks: "rack1,rack2"
```

Using the `topology.portworx.io/rack` labels allows you to implement specific placement logic tailored to your environment's needs.

Performance Considerations

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 RKE2 workloads.

Disaggregated Architecture

In a disaggregated deployment, where dedicated storage nodes are used, you can enable higher resource consumption by specifying the rtopsconf_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: portworx/oci-monitor:2.7.0

        ...

  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:

```
kind: StorageCluster

...

storage:

  devices:

        - /dev/sdb

  journalDevice: auto

...
```

Note: If you don't specify a journal device, Portworx will carve out 3GB out of the storage cluster to use for the journal device automatically. It is recommended to use auto when using local storage devices to avoid the cost of provisioning a dedicated journal device.

I/O Profiles

Portworx utilizes I/O profiles to optimize performance of different workloads. In previous versions it was required to specify an I/O profile for the best performance, however Portworx introduced an “auto” profile that will detect the optimal I/O profile based on the observed workload, and the configuration of the storage class. It is still possible to specify an I/O profile, but it is not recommended. You can read more about I/O profiles in the Portworx Documentation.

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: kubernetes.io/portworx-volume

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 autofstrim in the cluster to periodically delete unused data blocks.

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

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 RKE2, "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.

Note: Application I/O control may require reconfiguring the cgroups version on RKE2 nodes running on SLES 15 SP5 in the future. At the time of this writing, SLES15 SP5 runs in hybrid mode. See the SLES 15 SP5 documentation on Kernel Control Groups for details. CgroupsV2 support is planned for an upcoming Portworx release. For detailed information on configuring cgroups and implementing Application I/O Control, refer to the RKE2 and Portworx documentation.

Security Considerations

Securing your Portworx cluster involves two key areas:

1. **Authorization**: Protects Portworx volumes from unauthorized access.
2. **Encryption**: 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

        auth:

        guestAccess: 'Disabled'

...
```

Once authorization is enabled, Portworx recommends enabling authorization and disabling guest access.

Configuring pxctl For Authorization

Because our Portworx configuration is using authorization, we must ensure that `pxctl` is authorized.

Note: This procedure assumes that you have followed the steps in the Portworx CLI (pxctl) section to install `pxctl` locally, and that you have an active port-forward.

The admin token can be found in the secret `px-admin-token` in the portworx namespace. Let's load this into a variable:

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

We can now update our `pxctl` context to include the token:

```
pxctl context create admin --token=$ADMIN_TOKEN
```

This context's configuration will be stored in `${HOME}/.pxctl/contextconfig.yaml` and should be kept confidential. If we do not wish to store the token locally, we can configure `pxctl` for authorization in one of our cluster's pods by using the procedure found in the Portworx Documentation.

Configuring StorageClass For Authorization

We must configure our StorageClass objects to reference the authorization tokens to be able to create persistent volumes.

We can now update our `pxctl` context to include the token:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-repl3
parameters:
  csi.storage.k8s.io/controller-expand-secret-name: px-user-token
  csi.storage.k8s.io/controller-expand-secret-namespace: portworx
  csi.storage.k8s.io/node-publish-secret-name: px-user-token
  csi.storage.k8s.io/node-publish-secret-namespace: portworx
  csi.storage.k8s.io/provisioner-secret-name: px-user-token
  csi.storage.k8s.io/provisioner-secret-namespace: portworx
  io_profile: db_remote
  repl: "3"
provisioner: pxd.portworx.com
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

The above tokens are used for their associated CSI function. The location of these secrets can also be referenced with a variable which allows for multi-tenant configurations:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: px-repl3
parameters:
  csi.storage.k8s.io/controller-expand-secret-name: px-user-token
  csi.storage.k8s.io/controller-expand-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: px-user-token
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/provisioner-secret-name: px-user-token
  csi.storage.k8s.io/provisioner-secret-namespace: ${pvc.namespace}
  io_profile: db_remote
  repl: "3"
provisioner: pxd.portworx.com
allowVolumeExpansion: true
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

This allows namespaces to have their own authentication tokens, and would disallow a tenant in one namespace from being able to access a volume from a different namespace. Tenant tokens can be generated using 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. See the Portworx Documentation for further details.

The passphrases can be used in one of two ways for encrypting volumes:

1. **Per-volume secret**: uses a different secret passphrase for each encrypted volume
2. **Cluster-wide secret**: 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:

```
kubectl 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.

```
kubectl create secret generic mysql-pvc-1 -n csi-test-demo --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}

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 cluster-wide encryption.

```
kubectl -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=$(kubectl get pods -l name=portworx -n portworx -o jsonpath='{.items[0].metadata.name}')
kubectl 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: kubernetes.io/portworx-volume
parameters:
  secure: "true"
  repl: "3"
```

Additional Encryption Options

It is also possible to encrypt volumes using annotations on the PVC itself. See the Portworx Documentation for details.

Reference Architecture Example

Authorization

In this reference architecture, we have enabled security and disabled guest access:

```
kind: StorageCluster

...

spec:

  ...

  security:

        enabled: true

        auth:

        guestAccess: 'Disabled'

...
```

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

Volumes are naked as public if they are created without a token. The admin may also explicitly mark a volume as public to expose it to the guest role.

Because this role allows operations to volumes without an authentication token, it is recommended to disable guest access as shown above.

Monitoring Considerations

Monitoring is a vital aspect of managing and maintaining both RKE2 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 Rancher and RKE2

Rancher Prime provides a monitoring stack based on Prometheus. This application must be installed in downstream clusters as it is not included by default, but highly recommended by SUSE.

The `rancher-monitoring` package includes default dashboards and alerting that integrates with the Rancher Server UI.

Installation of this monitoring stack is documented in the Monitoring Configuration for Rancher Kubernetes Engine 2 section of this document.

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.

Portworx typically deploys its own version of Prometheus for monitoring activities of the storage cluster. It is possible to use this included version, but SUSE and Portworx recommend using the monitoring stack provided by Rancher Prime. This will require some configuration changes that are documented in the Working with the Rancher provided Monitoring application section of this document. Portworx integrates with this Prometheus stack and provides several key metrics that can be used to monitor the health and performance of the Portworx cluster.

Operational Considerations

This section provides procedures for installations and configurations, as well as additional considerations for operating a SUSE Rancher Kubernetes Engine 2 and Portworx Enterprise environment.

Installation Methods and Procedures: Rancher Prime

Rancher Prime provides a multi-cluster management layer for RKE2 clusters, as well as other Kubernetes distributions. Rancher Prime is currently deployed via a helm chart inside our utility infrastructure onto a dedicated local RKE2 cluster. The details are outside the scope of this document. Our Rancher Prime installation can manage multiple CNCF certified Kubernetes clusters.

More details on installing Rancher Prime can be found in the Rancher Installation/Upgrade Documentation.

Required Pivileges and Access Configuration

Cluster admins will require permissions to create downstream clusters. They will require the `Standard User` or better global permission to be able to create a cluster.

We will also require authentication to our Rancher Prime API. We will use a bearer token at several points in our examples.

Not only that, but we can generate a token by following the instructions here.

Furthermore, we can load our token into environment variables to make code snippets throughout this document easier.

Create a New RKE2 Cluster

We need to create a new custom cluster in Rancher Prime. Creating a new cluster will simply generate the commands that we will be running on our provisioned SLES nodes. We will be using these generated commands to install RKE2 on our provided servers. Rancher Prime can generate an installation command that can be run on any supported Linux server to streamline the installation process. Rancher Prime will manage a number of configuration parameters on our RKE2 cluster.

It is helpful to review the Checklist for Production-Ready Clusters to understand RKE2 specific requirements.

Pod Security Admission (PSA) Configuration

SUSE recommends configuring Pod Security Admission on RKE2 clusters. In accordance with best practices we have set our Pod Security Admission Template to rancher-restricted.

It is important that we exempt the portworx namespace from this PSA by doing the following:

1. In the upper left corner, click > `Cluster Management`.
2. Click `Advanced` to open the dropdown menu.
3. Select `Pod Security Admissions`.
4. Find the `rancher-restricted` template, and click the .
5. Select `Edit Config`.
6. Click the `Namespaces` checkbox under `Exemptions` to edit the `Namespaces` field.
7. Add the `portworx` namespace to the end of the list.
8. When you're done exempting namespaces, click `Save`.

NOTE: This updates the rancher-restricted template for ALL downstream clusters using this PSA Template.

NOTE: You need to update the target cluster to make the new template take effect in that cluster. An update can be triggered by editing and saving the cluster without changing values.

Monitoring Configuration for Rancher Kubernetes Engine 2

SUSE recommends installing the `rancher-monitoring` application chart to downstream clusters, which provides Prometheus alerting and monitoring to downstream Kubernetes clusters. For an overview, the Monitoring and Alerting documentation.

To install rancher-monitoring, follow these steps.

1. Click > `Cluster Management`.
2. Go to the cluster that you created and click `Explore`.
3. Click `Cluster Tools` (bottom left corner).
4. Click `Install` by Monitoring.

We used the default options for the installation.

Installing the above will conflict with the Portworx provided Prometheus package. See the SUSE Rancher Monitoring Service Monitor section for details.

For installation details, see the Enable Monitoring reference document.

Portworx Extension for Rancher UI

Use the Portworx extension in the Rancher UI to monitor your cluster directly within the Rancher platform. This extension simplifies cluster management and provides actionable insights by integrating Portworx storage monitoring directly into the Rancher dashboard.

NOTE: Portworx Rancher extension is not supported on PX-Security enabled clusters, and we have enabled security on this reference cluster.

For installation details, see the Portworx Documentation.

Reference Architecture Example

Create a new cluster using the referenced configuration parameters from the Software Versions Used in this RA section of this document. This will generate Registration Commands.

Example registration commands are below:

```
## Control-plane host example

curl -fL https://rancher.tmelab.local/system-agent-install.sh | sudo  sh -s - \
--server https://rancher.tmelab.local --label 'cattle.io/os=linux' \
--token <TOKEN> --ca-checksum <CHECKSUM> --etcd --controlplane

## Worker host example

curl -fL https://rancher.tmelab.local/system-agent-install.sh | sudo  sh -s - \
--server https://rancher.tmelab.local --label 'cattle.io/os=linux' \
--token <TOKEN> --ca-checksum <CHECKSUM> --worker
```

We will use these commands later when installing RKE2 on our SLES nodes.

Installation Methods and Procedures: SLES

In this section, we will cover the installation specifics of Suse Linux Enterprise Server.

Installation Overview

Details on installing SLES 15 SP5 can be found on SUSE's documentation site.

SUSE Linux Enterprise Server provides a variety of deployment options. Our reference architecture used a combination of ISO media, manual installation configurations, and AutoYaST configurations.

Manual Installation

For a manual installation, we booted our physical servers from an ISO. Procedures will vary on how to connect an ISO depending on our hardware vendor.

Although it is possible to manually install our SLES nodes, our recommendation is to use AutoYaST. Details can be found in the Automated installations with AutoYaST section of this document.

The below procedure is not meant to be a complete step-by-step guide, but rather to document the installation options used in this reference architecture.

1. Start by booting the installation media. You will need to select your language and keyboard layout. Be sure to select the `SUSE Linux Enterprise Server 15 SP5` under `Product to Install`.

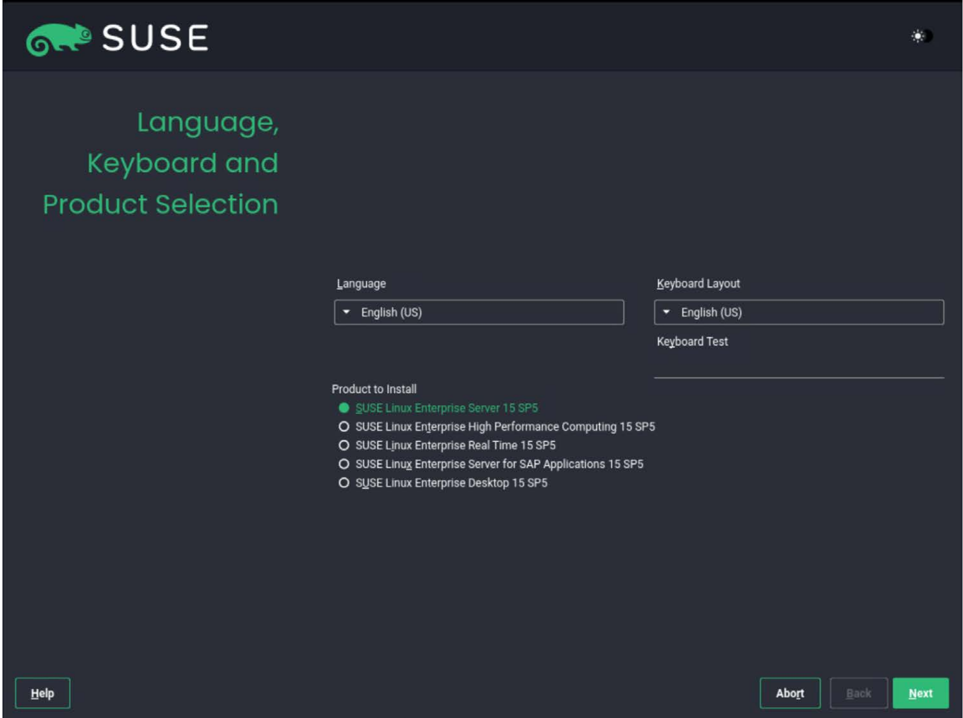

FIGURE 10 SLES installation 1

2. Accept the EULA and configure your SLES Software Subscription settings.
3. Select both the `Basesystem Module` and `Server Applications Module`.

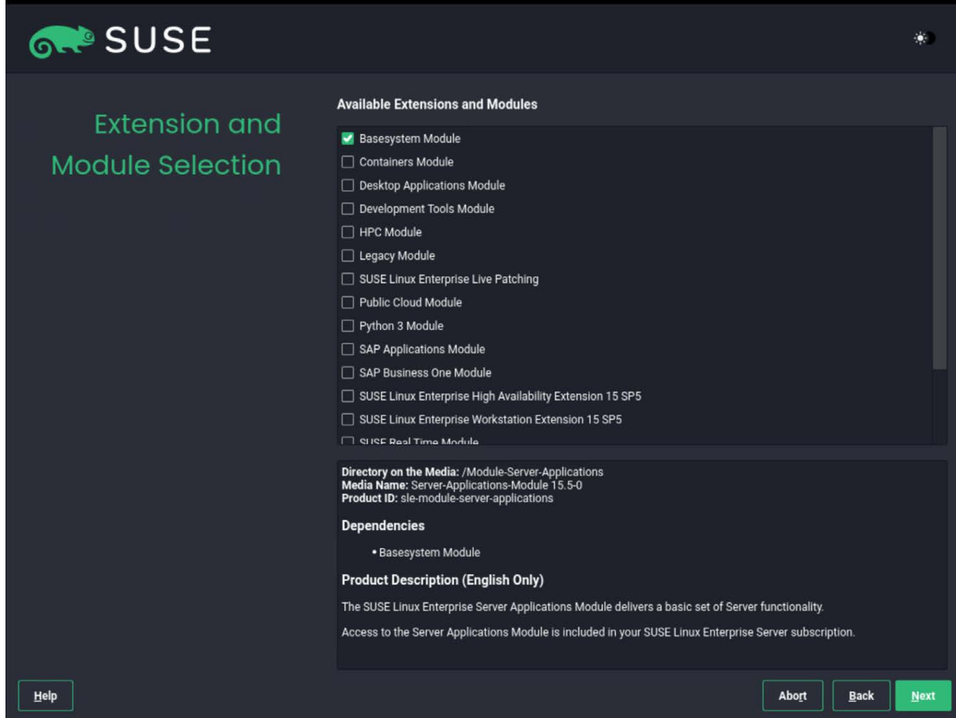

FIGURE 11 SLES installation 2

4. Select `Text Mode` for the `System Role`.

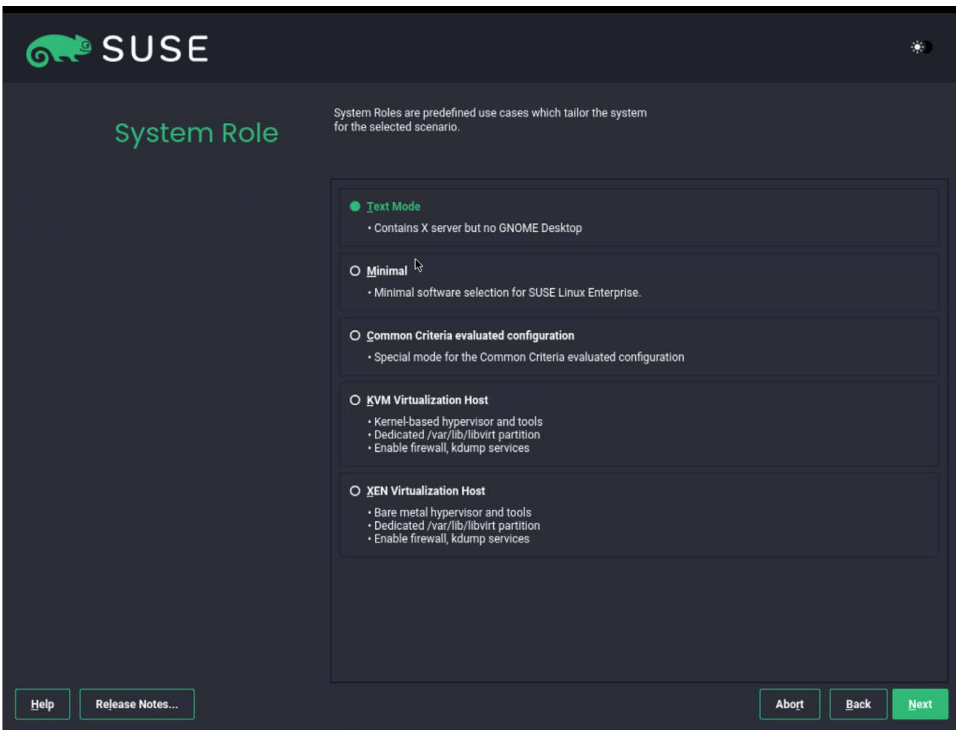

FIGURE 12 SLES installation 3

Note: It is important to verify the partition setup as SLES will try to auto-detect the partition configuration. It is important to select 'Guided Setup'

5. Select `Guided Setup`.

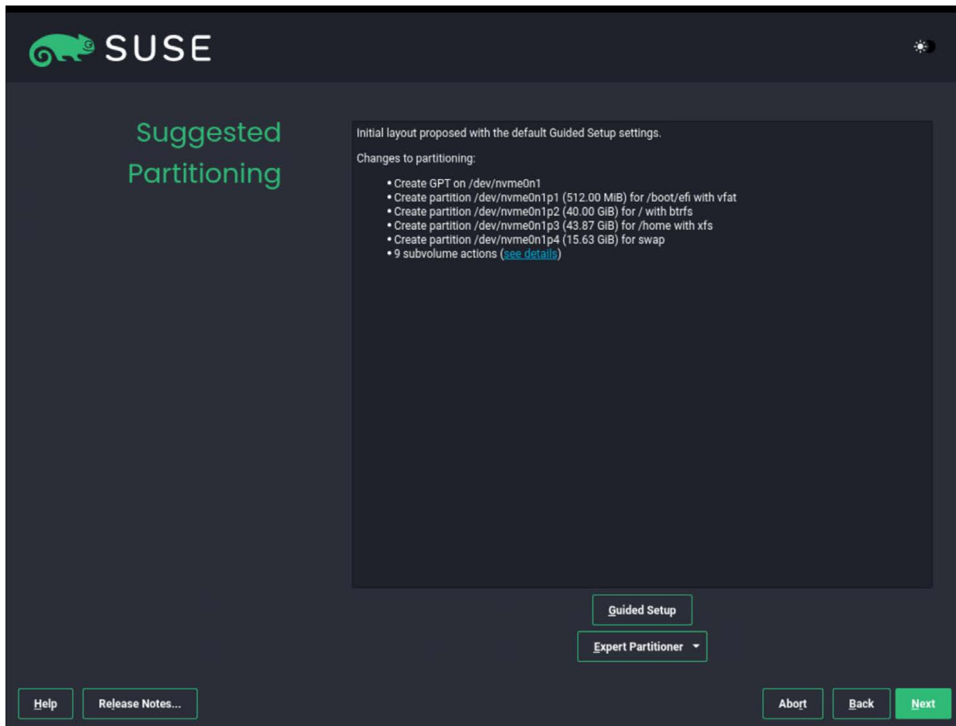

FIGURE 13 SLES installation 4

6. Select the disk you have designated as the boot drive.

Note: In our hardware configuration, we are booting from a SATA DOM on (/dev/sda).

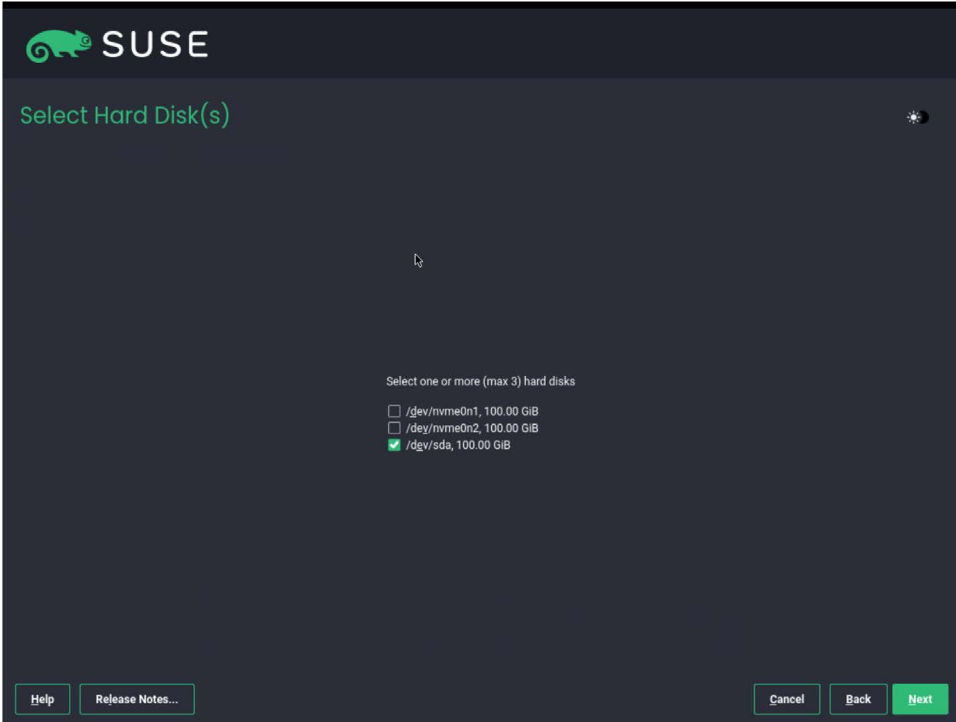

FIGURE 14 SLES installation 5

7. Enable logical volume management (LVM) and disable disk encryption.

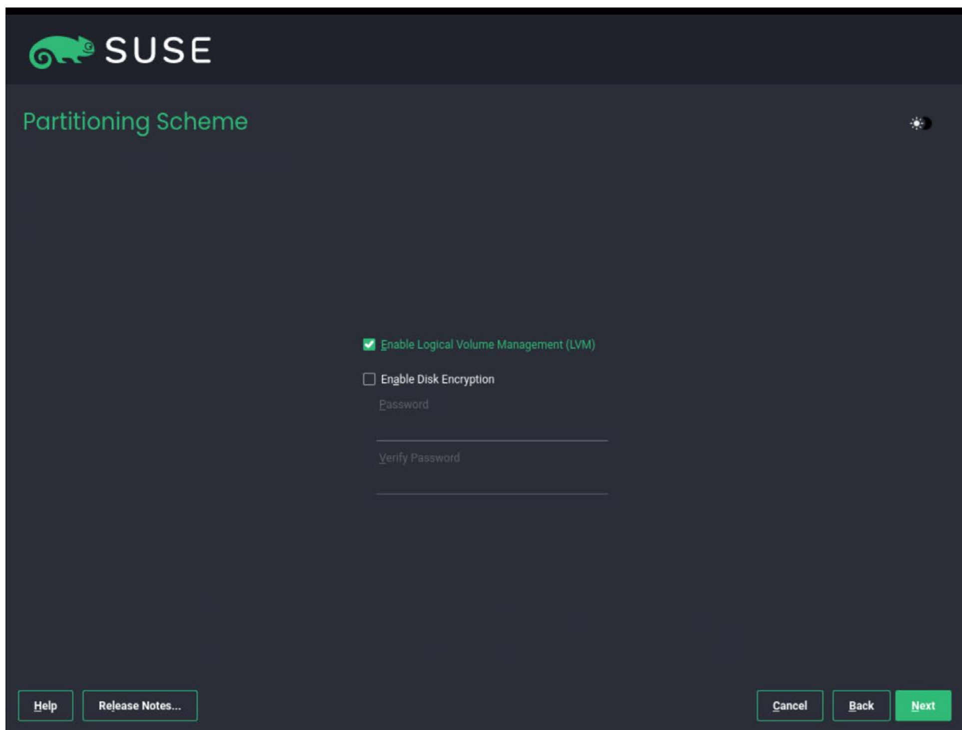

FIGURE 15 SLES installation 6

8. Select `BTRFS` as the File System Type and Enable Snapshots.
   - Do not `Propose Separate Home LVM Logical Volume.`
   - Do not `Enlarge to RAM size for Suspend.`

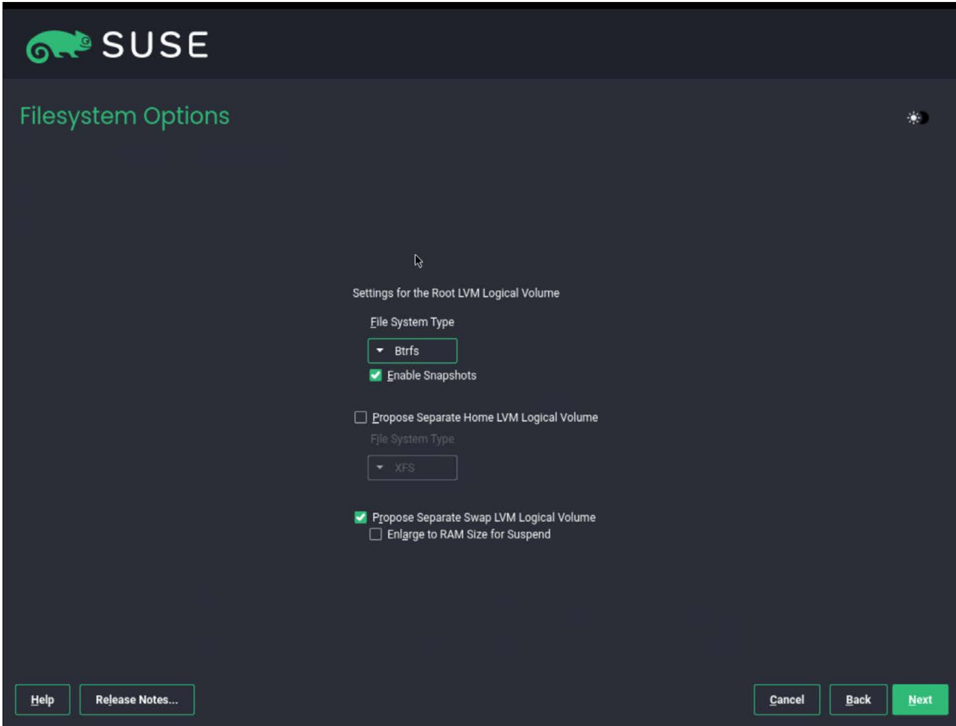

FIGURE 16 SLES installation 7

Note: Swap is not supported on Kubernetes environments, we will disable swap in our AutoYaST profile, or manually after install.

9. Select your Clock and Time Zone
10. Create a user account and set the `root` password.

YaST Overview

YaST is a SUSE Linux Enterprise Server tool that provides a graphical interface for all essential installation and system configuration tasks. Whether you need to update packages, configure a printer, modify firewall settings, set up an FTP server, or partition a hard disk—you can do it using YaST. Written in Ruby, YaST features an extensible architecture that makes it possible to add new functionality via modules.

Additional information about YaST can be found in SUSE's documentation.

Network Configuration

For an overview of our reference network configuration, see the Network Considerations section of this document.

The below configurations are just snippets of the configuration profile that is used by AutoYaST. See the Automated installations with AutoYaST section of this document for a full example and information on applying the configuration.

Front-end Network Configuration

Our front-end network configuration snippet:

```
<interfaces t="list">
    <interface t="map">
        <bonding_module_opts>mode=active-backup miimon=100</bonding_module_opts>
        <bonding_slave0>eth1</bonding_slave0>
        <bonding_slave1>eth3</bonding_slave1>
        <bootproto>static</bootproto>
        <ipaddr>10.13.200.81</ipaddr>
        <name>bond0</name>
        <prefixlen>21</prefixlen>
        <startmode>auto</startmode>
        <zone>public</zone>
    </interface>
</interfaces>
```

Back-end Network Configuration

Our back-end network configuration snippet:

```
<interfaces t="list">
    <interface t="map">
        <bonding_module_opts>mode=active-backup miimon=100</bonding_module_opts>
        <bonding_slave0>eth0</bonding_slave0>
        <bonding_slave1>eth5</bonding_slave1>
        <bootproto>static</bootproto>
        <ipaddr>192.168.0.81</ipaddr>
        <name>bond1</name>
        <prefixlen>24</prefixlen>
        <startmode>auto</startmode>
        <zone>public</zone>
    </interface>
</interfaces>
```

iSCSI Network Configuration

Our iSCSI network configuration snippet:

```
<interface t="list">
    <interface t="map">
        <bootproto>static</bootproto>
        <ipaddr>192.168.1.81</ipaddr>
        <name>eth2</name>
        <prefixlen>24</prefixlen>
        <startmode>auto</startmode>
        <zone>public</zone>
    </interface>
    <interface t="map">
        <bootproto>none</bootproto>
        <name>eth3</name>
        <startmode>hotplug</startmode>
        <zone>public</zone>
    </interface>
    <interface t="map">
        <bootproto>static</bootproto>
        <ipaddr>192.168.2.81</ipaddr>
        <name>eth4</name>
        <prefixlen>24</prefixlen>
        <startmode>auto</startmode>
        <zone>public</zone>
     </interface>
</interface>
```

Firewall Configuration

We disabled the firewall for this reference architecture. This task can be accomplished by using YaST.

The below configuration is just a snippet of the configuration profile that is used by AutoYaST. See the Automated installations with AutoYaST section of this document for a full example and information on applying the configuration.

YaST Configuration Snippet:

```
<firewall t="map">
     <default_zone>public</default_zone>
     <enable_firewall t="boolean">false</enable_firewall>
     <log_denied_packets>off</log_denied_packets>
     <start_firewall t="boolean">false</start_firewall>
</firewall>
```

SSH and Account Configuration

Although outside the scope of this document, Portworx highly recommends installing SSH keys on your SLES system to make authentication easier and more secure.

Authorized keys can be added to the `users` section of the configuration profile:

```
<users t="list">
       <user t="map">
       <encrypted t="boolean">true</encrypted>
       <fullname>root</fullname>
       <gid>0</gid>
       <home>/root</home>
       <home_btrfs_subvolume t="boolean">false</home_btrfs_subvolume>
       <password_settings t="map">
       <expire/>
       <flag/>
       <inact/>
       <max/>
       <min/>
       <warn/>
       </password_settings>
       <shell>/bin/bash</shell>
       <uid>0</uid>
       <user_password>******PASSWORD_HASH********</user_password>
       <authorized_keys config:type="list">
       <listentry>**SSHRSA-PUBLIC_KEY**</listentry>
       </authorized_keys>
       <username>root</username>
       </user>
</users>
```

Cgroup Configuration

Portworx only supports cgroupsv1 currently, although support for cgroups v2 is comming in a future version.

The kernel cgroup API comes in two variants, v1 and v2. Additionally, there can be multiple cgroup hierarchies exposing different APIs. From the numerous possible combinations, there are two practical choices: - hybrid: v2 hierarchy without controllers, and the controllers are on v1 hierarchies - unified: v2 hierarchy with controllers.

See the SLES 15 SP5 documentation for more details.

Swap Configuration

It is important that we disable our swap partition for our installation. This has already been done in the reference profile below.

The easiest way to interactively disable swap is by using `yast`:

1. Navigate to the `System` > `Partitioner module.`

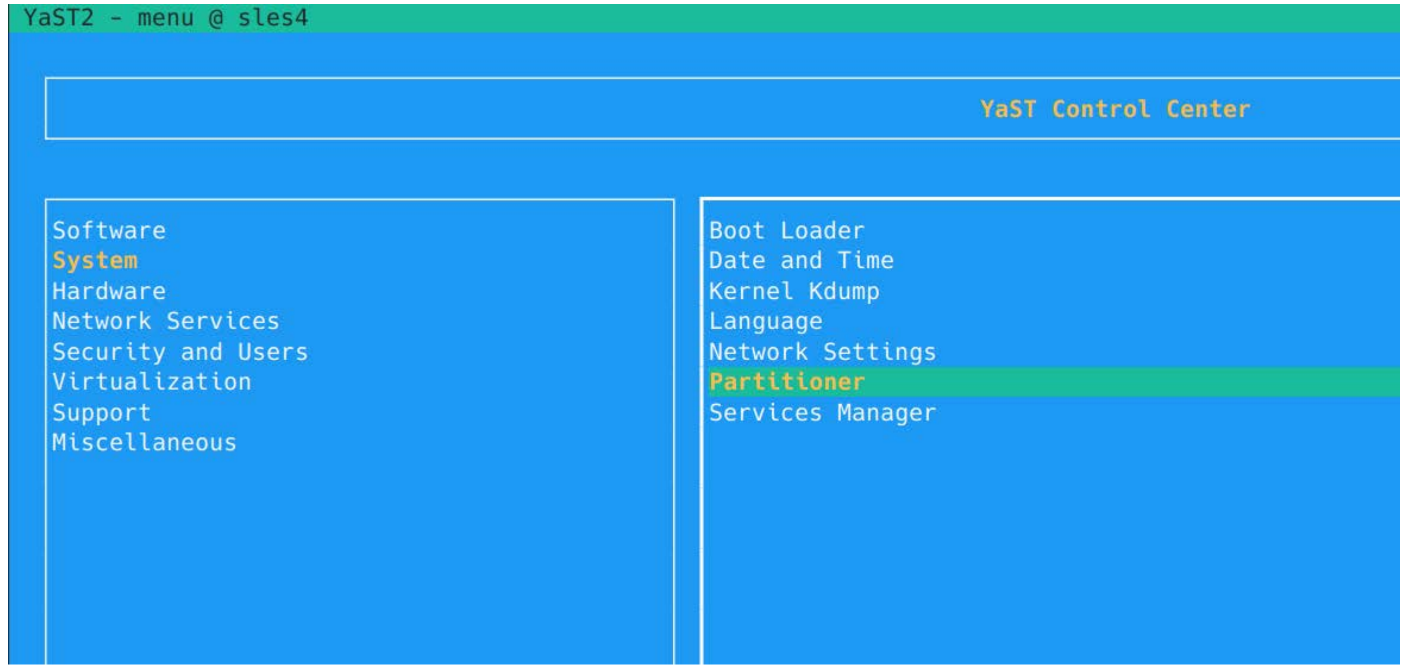

FIGURE 17 SLES Swap configuration 1

2. Proceed past the warning.
3. Select the `swap` partition and edit it with by pressing `alt-e.`

```
YaST2 - partitioner @ sles4

System  Add  Device  View

──All Devices          Device              Size  F Enc Type                          Label          Mount Point
─┬Hard Disks           ──/dev/md/pwx0     99.94 GiB     PV of pwx0
 ├─nvme0n1             ──/dev/nvme0n1    100.00 GiB     XFS Disk                      mdvol
 ├─nvme0n2             ──/dev/nvme0n2    100.00 GiB     Part of pwx0
 └─sda                 ┬─/dev/pwx0        99.93 GiB     LVM
─┬RAID                 ├┬─pxpool          85.93 GiB     Thin Pool
 └─pwx0                │└──pxMetaFS       64.00 GiB     XFS Thin LV                   pxpool=0,u=5
─┬LVM Volume Groups    └──pxreserve       10.00 GiB     LV
 ├─pwx0                ┬─/dev/sda        100.00 GiB     VMware Virtual disk
 └─system              ├──sda1             0.50 GiB     EFI System Partition Partition               /boot/efi
──Bcache Devices       └──sda2            99.50 GiB     PV of system
─┬Btrfs                ┬─/dev/system      99.50 GiB     LVM
 └─root                ├──root            83.87 GiB     Btrfs LV                                     /
──Tmpfs                └──swap            15.63 GiB     Swap LV
──NFS

[Edit...][Add Logical Volume...][Delete]
```

FIGURE 18 SLES Swap configuration 2

4. Select `Do not mount device.`

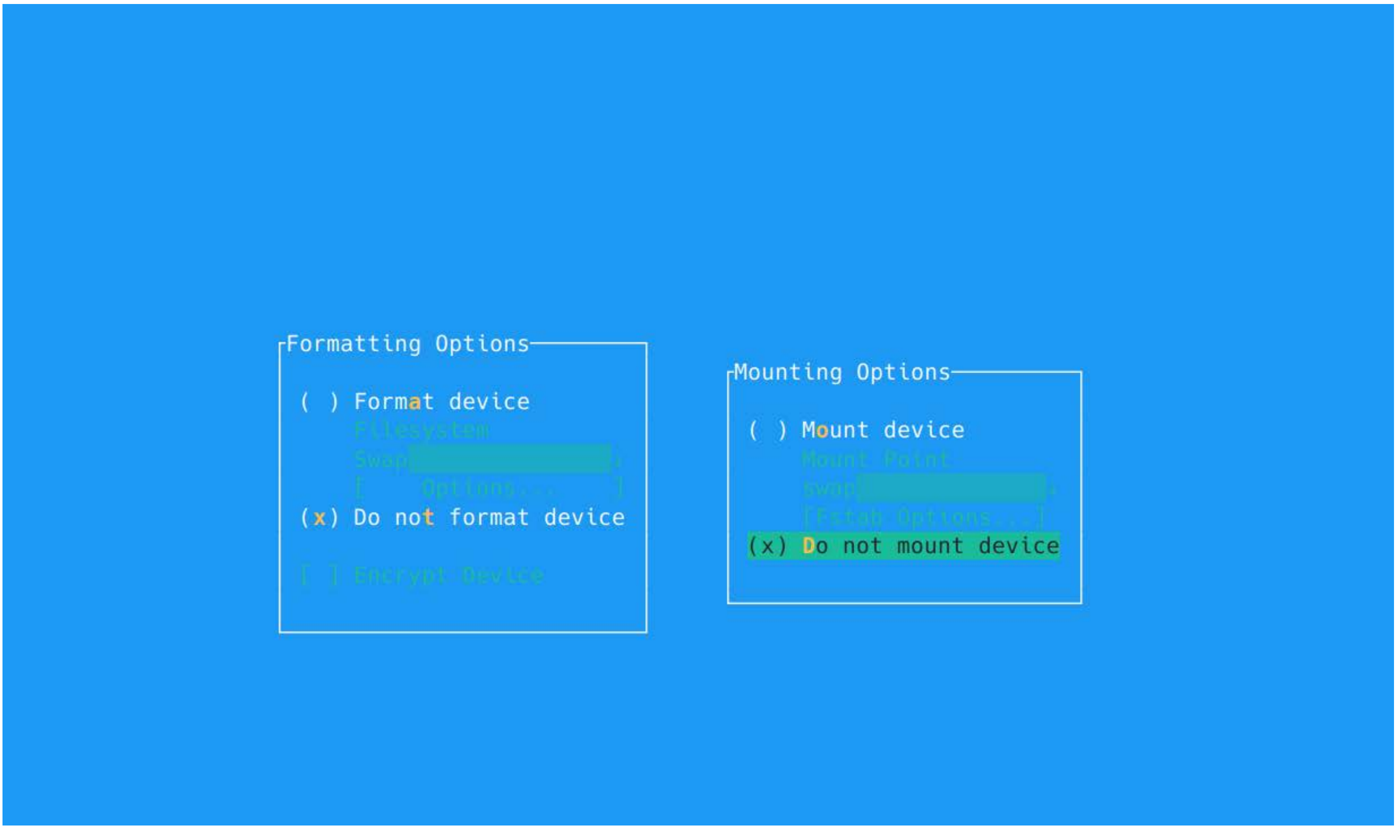

FIGURE 19 SLES Swap configuration 3

5. Select `Next` and save your changes.

Automated installations with AutoYaST

AutoYaST is a system for unattended mass deployment of SUSE Linux Enterprise Server systems. It uses an AutoYaST profile that contains installation and configuration data. The book guides you through the basic steps of auto-installation: preparation, installation, and configuration.

We can find additional information about AutoYaST in the AutoYaST Guide.

Automatically Cloning a System

The easiest way to create an AutoYaST profile is to use an existing SUSE Linux Enterprise Server system as a template. On an already installed system, launch YaST › Miscellaneous › Autoinstallation Configuration. Then select Tools › Create Reference Profile from the menu. Choose the system components you want to include in the profile. Alternatively, create a profile containing the complete system configuration by launching YaST › Miscellaneous › Autoinstallation Cloning System or running `sudo yast clone_system` from the command line.

For this reference architecture, we manually configured a reference host, and then cloned the system.

Both methods will create the file `/root/autoinst.xml`. The cloned profile can be used to set up an identical clone of the system it was created from. However, you will usually want to adjust the file to allow for installing multiple machines that are very similar, but not identical. This can be done by adjusting the profile with your favorite text/XML editor.

Warning: The created profile will contain sensitive information.

Using Our AutoYaST Profile with a SLES ISO

Adding the command line variable `autoyast` causes `linuxrc` to start in automated mode. In our architecture, we are hosting these profiles on an http server:

```
autoyast=http://[user:password@]_SERVER_/_PATH_
```

This retrieves the control file from a Web server using the HTTP protocol. Specifying a user name and a password is optional.

Installation Methods and Procedures: RKE2

Rancher Prime significantly simplifies the process of deploying and managing RKE2 clusters by providing standardized configurations. These predefined templates ensure consistency and reduce the complexity of setting up and configuring RKE2 nodes, streamlining the entire installation process. By leveraging Rancher Prime, administrators can deploy clusters with confidence, knowing that best practices are built into the configuration templates.

Note: The installation of the cluster will not complete unless we have at least 1 control-plane node and 1 worker node installed.

Install Control Plane Nodes

We can install our control plane node with the Agent Install command we retrieved from the Rancher Prime installation.

Example installation command:

```
bash curl -fL https://<RANCHER_PRIME_SERVER>/system-agent-install.sh | sudo sh -s - \
--server https://rancher.tmelab.local --label 'cattle.io/os=linux' \
--token <TOKEN> --ca-checksum <CHECKSUM> --etcd --controlplane
```

Install Worker Nodes

We can install our control plane node with the Agent Install command we retrieved from the Rancher Prime installation.

```
bash curl -fL https://<RANCHER_PRIME_SERVER>/system-agent-install.sh | sudo sh -s -\
 --server https://rancher.tmelab.local --label 'cattle.io/os=linux' \
--token <TOKEN> --ca-checksum <CHECKSUM> --etcd --controlplane
```

Cluster Access

We can access our RKE2 cluster in one of the following methods:

Cluster Dashboard

On the Clusters page, select the Explore button at the end of each row to view that cluster's Cluster Dashboard. You can also view the dashboard by clicking the name of a cluster in the table, then clicking the Explore button on the Cluster page.

The Cluster Dashboard is also accessible from the Rancher UI Home page, by clicking on the name of a cluster.

You can also access the Cluster Dashboard from the in the top navigation bar:

1. Click .
2. Select the name of a cluster from the Explore Cluster menu option.

The Cluster Dashboard lists information about a specific cluster, such as number of nodes, memory usage, events, and resources.

Accessing Clusters with kubectl Shell in the Rancher UI

You can access and manage your clusters by logging into Rancher and opening the kubectl shell in the UI. No further configuration necessary.

1. Click > Cluster Management.
2. Go to the cluster you want to access with kubectl and click Explore.
3. In the top navigation menu, click the Kubectl Shell button. Use the window that opens to interact with your Kubernetes cluster.

Accessing Clusters with kubectl from Your Workstation

This section describes how to download your cluster's kubeconfig file, launch kubectl from your workstation, and access your downstream cluster.

This alternative method of accessing the cluster allows you to authenticate with Rancher and manage your cluster without using the Rancher UI.

As a prerequisite, hese instructions assume that you have already created a Kubernetes cluster, and that kubectl is installed on your workstation. For help installing kubectl, refer to the official Kubernetes Documentation.

1. Click in the top left corner.
2. Select Cluster Management.
3. Find the cluster whose kubeconfig you want to download, and select ⋮ at the end of the row.
4. Select Download KubeConfig from the submenu.
5. Save the YAML file on your local computer. Move the file to `~/.kube/config`. Note: The default location that kubectl uses for the kubeconfig file is `~/.kube/config`, but you can use any directory and specify it using the `--kubeconfig` flag, as in this command:

```
kubectl --kubeconfig /custom/path/kube.config get pods
```

From your workstation, launch kubectl. Use it to interact with your kubernetes cluster.

See the Rancher Prime Documentation for more details.

Installation Methods and Procedures: Portworx

This section of the reference architecture explains instructions for proper installation of a Portworx Enterprise storage cluster on RKE2 bare metal nodes. This section includes installation procedures as well as monitoring and troubleshooting information for the installation process.

Portworx supports both a manifest based installation as well as a Rancher Prime specific Helm based installation. We have chosen a manifest based installation to conform with the Portworx documentation.

Portworx Configuration Builder

To start the installation of Portworx on your deployed RKE2 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 RKE2 cluster.

Generate Spec

Step 1: Select Your Platform

Portworx Version * 3.2

Platform * DAS/SAN

Portworx will automatically scan for already provisioned disks. To modify this, click the "Customize" button in Step 3.

Step 2: Select Kubernetes Distribution

Distribution Name * None

Namespace * portworx

Step 3: Summary

K8s Version	1.30.1
Cluster Name Prefix	px-cluster
License Type	Enterprise
Platform	DAS/SAN
K8s Distribution	None
Portworx Operator	Enabled

Customize   Show More

Cancel   Save & Download

FIGURE 20 Portworx central installation 1

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. Our reference architecture requires some configuration customization. Click the Customize button.

Select the Built-in etcd option and click Next.

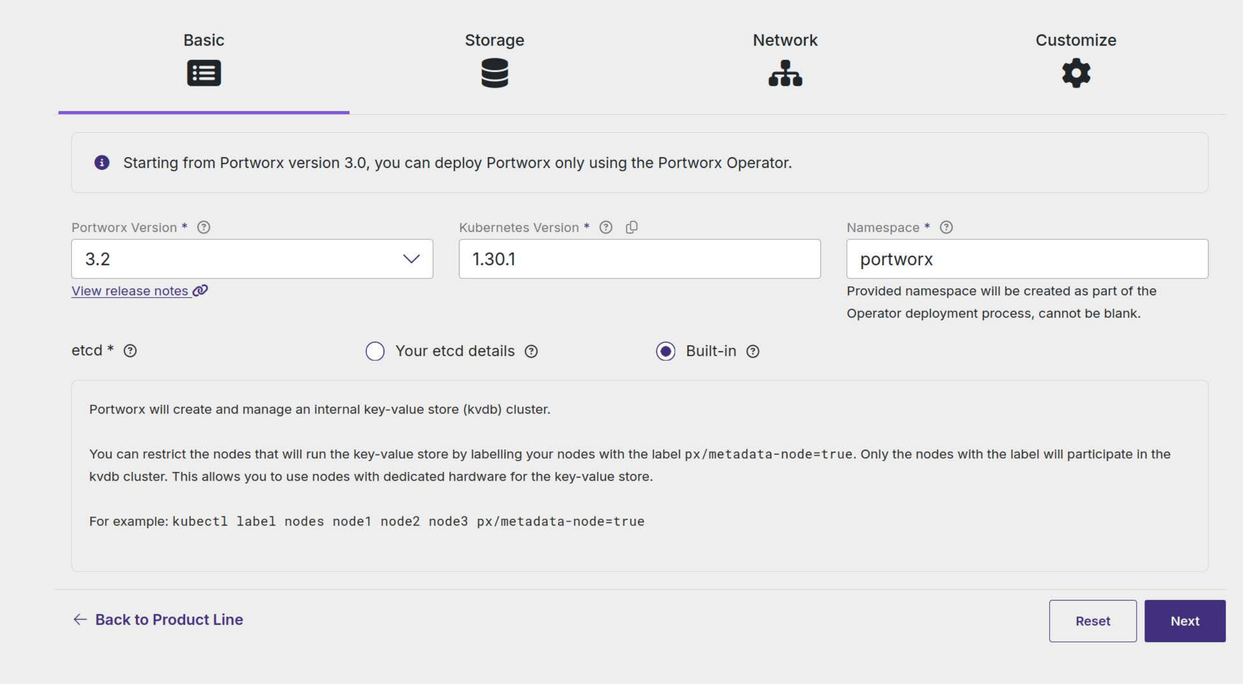

FIGURE 21 Portworx central installation 2

On our Storage configuration page, specify the disk configuration used in your environment. Our hardware device configuration is documented in our Hardware Resource Considerations section.

Select PX-StoreV2, enter the Drive/Device information, and press Next.

Note: We are not going to Automatically scan disks as we want to specify our disk manually to ensure that they are assigned to the appropriate task.

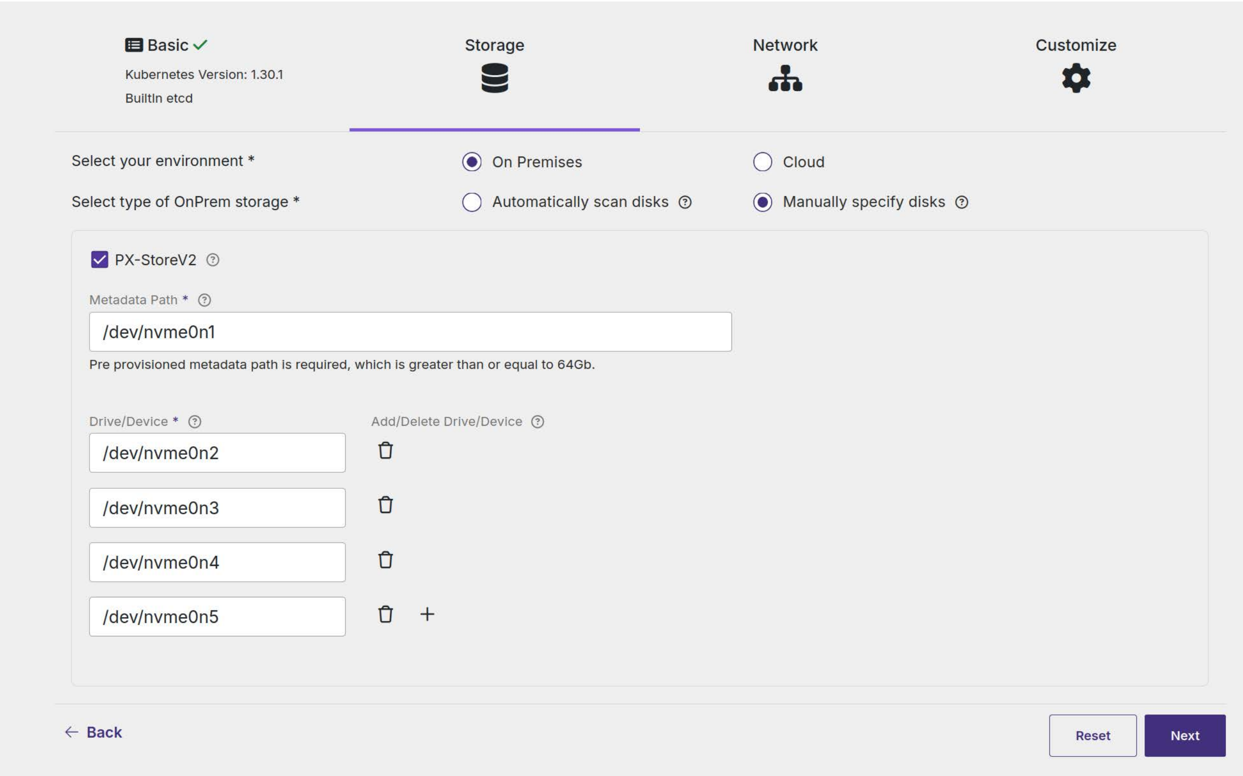

FIGURE 22 Portworx central installation 3

Specify our networking configuration by entering our `bond1` interface for the Data Network and `bond0` as the Management Network. A detailed overview can be found in the Network Considerations section. Specific SLES network configuration options can be found in the Network Configuration section.

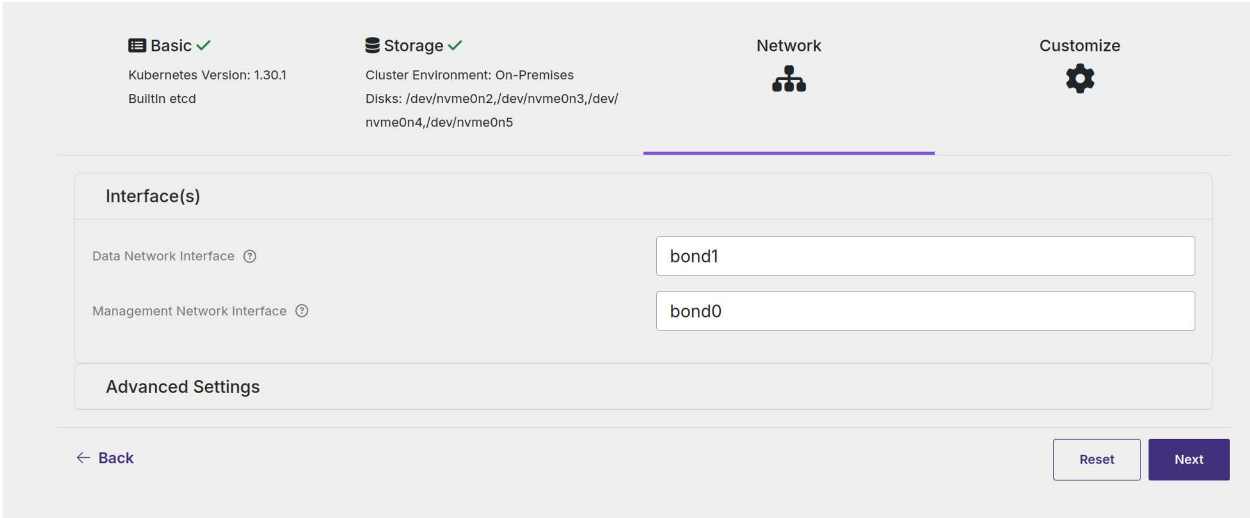

FIGURE 23 Portworx central installation 4

After adding any additional customizations, we can click next and view our installation commands.

Basic ✓
Kubernetes Version: 1.30.1
BuiltIn etcd

Storage ✓
Cluster Environment: On-Premises
Disks: /dev/nvme0n2,/dev/nvme0n3,/dev/nvme0n4,/dev/nvme0n5

Network ✓
Port Range Start: 9001
Data Interface: bond1
Mgmt Interface: bond0

Customize ✓
Running on

Portworx Operator

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

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

```
kubectl apply -f 'https://install.portworx.com/3.2?comp=pxoperator&kbver=1.30.1&ns=portworx'
```

```
kubectl apply -f 'https://install.portworx.com/3.2?operator=true&mc=false&kbver=1.30.1&ns=portworx&b=true&dmthin=true&md=%2Fdev%2Fnvme0n1&s=%2Fdev%2Fnvme0n2%2C%2Fdev%2Fnvme0n3%2C%2Fdev%2Fnvme0n4%2C%2Fdev%2Fnvme0n5&m=bond0&d=bond1&c=px-cluster-eede58f0-c75a-4731-b462-78664b7a2bba&stork=true&csi=true&mon=true&tel=true&st=k8s&promop=true'
```

Save Spec

Spec Name *

Spec Tags *

Comma separated Tags

← Back

Download .yaml

Save Spec

FIGURE 24 Portworx central installation 5

It is recommended that we copy the Operator deployment command and download the `StorageCluster` specification YAML. We will need to provide additional changes to work with the `rancher-monitoring` application.

Specifics on this customization can be found in the Monitoring Configuration for Rancher Kubernetes Engine 2 section of this document.

A complete reference of the Operator deployment command and `StorageCluster` specification YAML can be found in the Reference Architecture Example - Portworx section of this document. The YAML has been embedded here for reference.

Portworx StorageCluster Object

In addition to values provided by the configuration builder, we can edit the StorageCluster CR to make additional configurations, as well as review the configurations we specified in the configuration builder.

Note the following changes that we have made to our specification that match the reference architecture recommendations:

This annotation specifies that we are using PX-StoreV2. See the PX-StoreV2 Considerations for details.

```
metadata:
  annotations:
    portworx.io/misc-args: " -T px-storev2 "
```

This section of the cluster config enables Security and Authorization. It also disables guest access. This requires all connections to the Portworx API to be authorized.

```
spec:
  security:
   enabled: true
   auth:
     guestAccess: 'Disabled'
```

Our 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 KVDB Device is used, and the 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.

```
spec:
  storage:
    devices:
        - /dev/nvme0n2
        - /dev/nvme0n3
        - /dev/nvme0n4
        - /dev/nvme0n5
    systemMetadataDevice: /dev/nvme0n1
```

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: bond1
    mgmtInterface: bond0
```

Telemetry is enabled by default. We should disable the portworx prometheus installation in favor of the `rancher-monitoring` application.

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

We must also configure autopilot to point at the Rancher Prime provided endpoint:

```
spec:
  autopilot:
    enabled: true
    providers:
      - name: default
      params:
        url: http://rancher-monitoring-prometheus.cattle-monitoring-system.svc.cluster.local:9090
      type: prometheus
```

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.

```
spec:
  image: portworx/oci-monitor:3.1.0
  storage:
    devices:
      - /dev/nvme0n2
      - /dev/nvme0n3
      - /dev/nvme0n4
      - /dev/nvme0n5
  nodes:
    - selector:
      labelSelector:
        matchLabels:
          portworx.io/node-type: "storage"
      storage:
        devices:
        - /dev/nvme0n2
        - /dev/nvme0n3
        - /dev/nvme0n4
        - /dev/nvme0n5
    - selector:
        labelSelector:
          matchLabels:
            portworx.io/node-type: "storageless"
      storage:
        devices: []
```

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:

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

Monitoring During Installation

Ensuring a smooth and successful installation of Portworx on RKE2 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.

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.

```
kubectl get pods -n portworx
```

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

```
NAME                                          READY   STATUS    RESTARTS       AGE
autopilot-67f89c4877-rm8xl                    1/1     Running   12 (21h ago)   21h
portworx-api-9n9zn                            2/2     Running   0              148m
portworx-api-grlh4                            2/2     Running   0              151m
portworx-api-pgjnc                            2/2     Running   0              146m
portworx-api-r4kzm                            2/2     Running   0              149m
portworx-kvdb-27ptz                           1/1     Running   0              21h
portworx-kvdb-4pqmw                           1/1     Running   0              21h
portworx-kvdb-jgmk4                           1/1     Running   0              21h
portworx-operator-8dc75bdd5-kc9hg             1/1     Running   2 (21h ago)    21h
portworx-pvc-controller-68f469844b-c8ks6      1/1     Running   2 (21h ago)    21h
portworx-pvc-controller-68f469844b-r95sd      1/1     Running   2 (21h ago)    21h
portworx-pvc-controller-68f469844b-wtcm6      1/1     Running   2 (21h ago)    21h
px-cluster-jjzvq                              1/1     Running   0              151m
px-cluster-kznp5                              1/1     Running   0              146m
px-cluster-l7z8z                              1/1     Running   0              149m
px-cluster-m74gs                              1/1     Running   0              148m
px-csi-ext-7bcf767777-4z4nw                   4/4     Running   20 (21h ago)   21h
px-csi-ext-7bcf767777-87r6x                   4/4     running   20 (0s ago)    21h
px-csi-ext-7bcf767777-d7gm5                   4/4     Running   19 (21h ago)   21h
px-telemetry-phonehome-8svtp                  2/2     Running   0              21h
px-telemetry-phonehome-bhjnl                  2/2     Running   0              21h
px-telemetry-phonehome-fds8s                  2/2     Running   0              21h
px-telemetry-phonehome-kttnc                  2/2     Running   0              21h
px-telemetry-registration-7f5f574d8d-kttjl    2/2     Running   0              21h
stork-59d74b9c7d-jw74l                        1/1     Running   2 (21h ago)    21h
stork-59d74b9c7d-mtxbm                        1/1     Running   1 (21h ago)    21h
stork-59d74b9c7d-svjwr                        1/1     Running   2 (21h ago)    21h
stork-scheduler-54c998b564-2l9vx              1/1     Running   2 (21h ago)    21h
stork-scheduler-54c998b564-fvh84              1/1     Running   2 (21h ago)    21h
stork-scheduler-54c998b564-md9ct              1/1     Running   2 (21h ago)    21h
```

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.

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

Portworx CLI (pxctl)

Many functions in Portworx can be accessed and automated through the command line using `pxctl`. There are a few ways to access `pxctl` from a Portworx installation.

The simplest way to use `pxctl` is by running it directly from one of the Portworx cluster pods:

```
kubectl exec $(kubectl get pods -l name=portworx -n portworx \
-o jsonpath='{.items[0].metadata.name}') -n portworx -- /opt/pwx/bin/pxctl
```

The above can be streamlined into an alias with:

```
alias pxctl='kubectl exec $(kubectl get pods -l name=portworx -n portworx \
-o jsonpath='{.items[0].metadata.name}') -n portworx -- /opt/pwx/bin/pxctl'
```

This will allow the use of `pxctl` cli commands.

We can also download `pxctl` to our workstation or jumphost. Running `pxctl` from our localhost allows us to store configuration and authentication information locally, instead of inside of the pod. It will also allow us to use interactive options (e.g., answering confirmation questions).

We can install `pxctl` using the following:

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

kubectl -n portworx \
cp $PX_POD:/opt/pwx/bin/pxctl ./pxctl chmod +x pxctl sudo mv pxctl /usr/local/bin
```

`pxctl` assumes that Portworx is running on the localhost. In order to connect to the Portworx service, use `kubectl` to proxy a connection:

```
kubectl port-forward service/portworx-api -n portworx 9001 9020 9021
```

NOTE: The above command will run interactively unless we send it to the background by appending & to the command.

We can now use our pxctl command from our local workstation/jumphost

Post Installation Validation

After successfully installing Portworx on your RKE2 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

Ensure that Portworx has entered the `Running` phase:

```
kubectl -n portworx get stc -o jsonpath='{.items[0].status.phase}'
```

This will return `Running` if the installation is complete.

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.

Ensure that `pxctl` is configured correctly by following the procedure outlined above in the Portworx CLI (pxctl) section.

We can now run the `pxctl status` command.

```
ccrow@ccrow-ubuntu:~$ pxctl status
Handling connection for 9001
Status: PX is operational
License: Trial (expires in 30 days)
Node ID: 9d3a3b8b-bbe6-4235-974b-f102b4326e03
        IP: 10.13.200.86
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY     RAID_LEVEL      USABLE  USED    STATUS  ZONE    REGION
        0       HIGH            raid0           86 GiB  35 MiB  Online  default default
        Local Storage Devices: 1 device
        Device  Path            Media Type              Size            Last-Scan
        0:0     /dev/nvme0n2    STORAGE_MEDIUM_NVME     100 GiB         31 Dec 24 08:42 PST
        total                   -                       100 GiB
        Cache Devices:
        * No cache devices
        Metadata Device:
        1       /dev/nvme0n1    STORAGE_MEDIUM_NVME     100 GiB
...
Global Storage Pool
        Total Used      :  141 MiB
        Total Capacity  :  344 GiB
Collected at: 2024-12-31 11:19:00 PST
```

NOTE: The above output has been truncated for brevity.

Installation is expected to take about 10 minutes.

Portworx creates a number of default storage classes as part of the installation. We have disabled creation of default storage classes as part of this reference architecture. This is due to the default storage classes being incompatible with our authorization configuration. Instead, create a storage class to fit the needs of your environment. A reference storage class can be found in the Configuring StorageClass for Authorization section of this document.

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. With Portworx as your storage platform this comes with options.

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 Pure Flash Array, deduplication can be enabled to remove this concern.

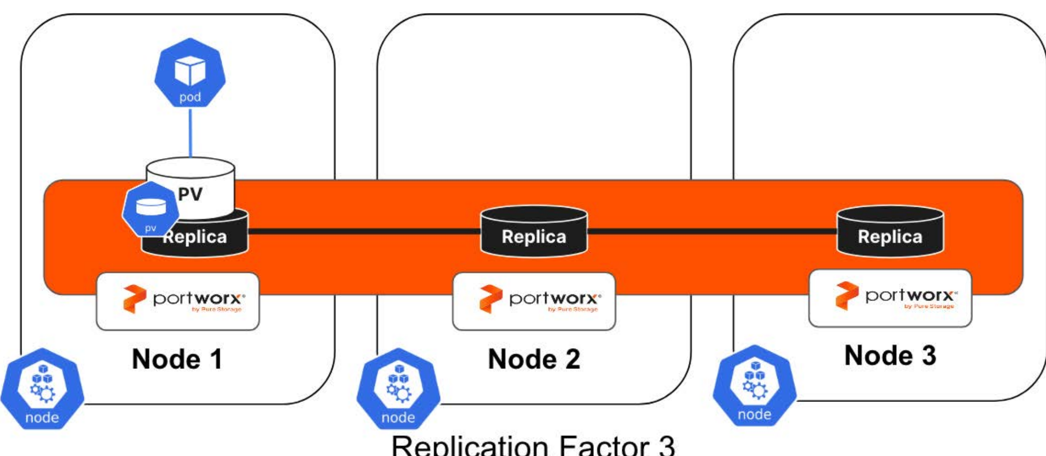

FIGURE 25 Replication factor 3

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 `repl2` or `repl3` with app based replication, but you're increasing the number of copies of your data that exist.

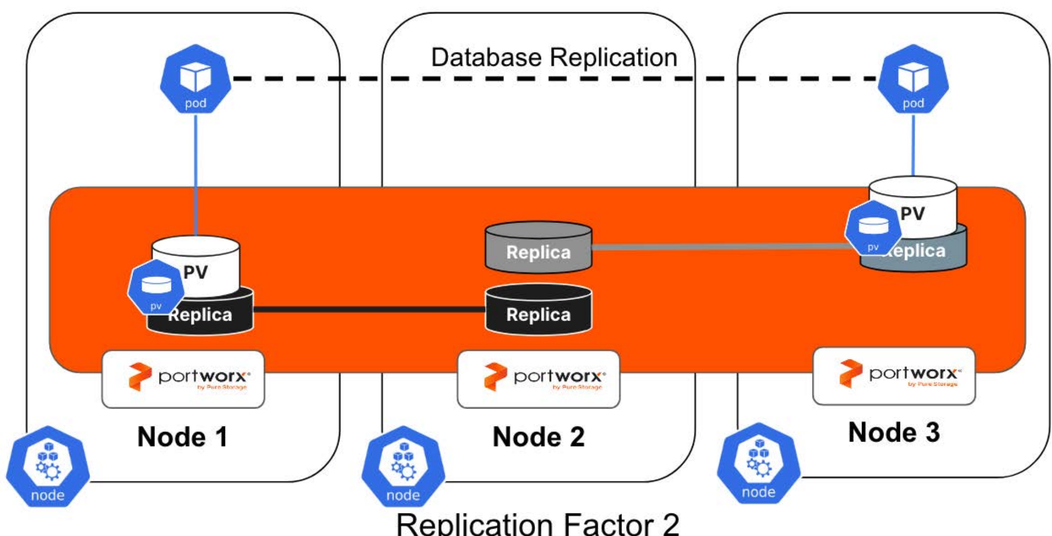

FIGURE 26 Replication factor 2

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.

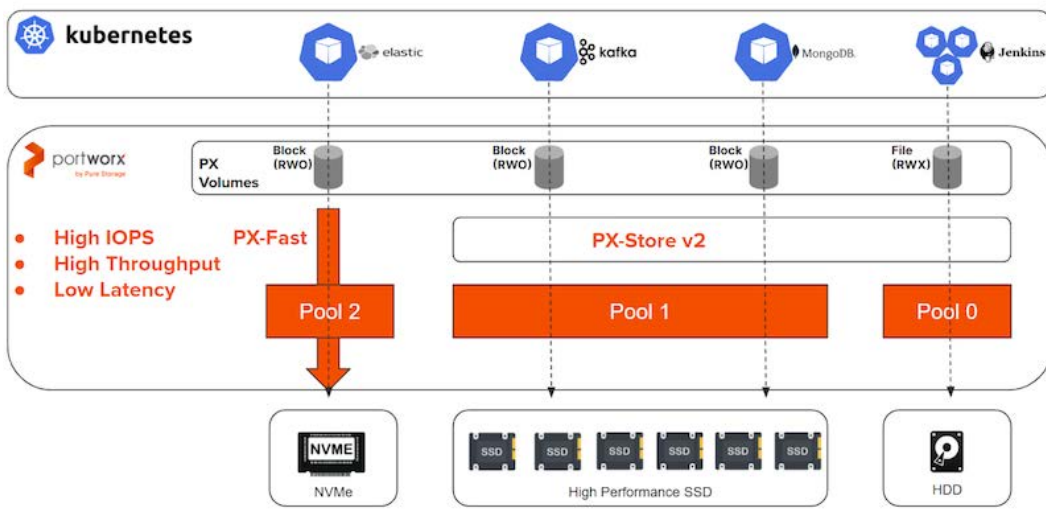

FIGURE 27 PX-Fast

PX-Fast Requirements

PX-Fast has a few requirements and design considerations:

- **PX-StoreV2:** PX-Fast requires that PX-StoreV2 is deployed. See the PX Store Configuration section for more details and considerations.
- **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

Configuring PX-Fast

First, let's create a StorageClass for PX-Fast:

```
cat << EOF | kubectl apply -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: px-fast
provisioner: pxd.portworx.com
parameters:
 repl: "1"
 fastpath: "true"
allowVolumeExpansion: true
EOF
```

We can now create a PVC for our application:

```
cat << EOF | kubectl apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: px-fast-pvc
spec:
 storageClassName: px-fast
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
        storage: 1Gi
EOF
```

PX-Fast requires that the pod accessing the volume resides on the same node as the volume. This is the default preferred behavior of STORK, but we can enforce this by using the `stork.libopenstorage.org/preferLocalNodeOnly` annotation:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
        matchLabels:
        app: mysql
  strategy:
        rollingUpdate:
        maxSurge: 1
        maxUnavailable: 1
        type: RollingUpdate
  replicas: 1
  template:
        metadata:
        annotations:
                stork.libopenstorage.org/preferLocalNodeOnly: "true"
        labels:
                app: mysql
        spec:
        schedulerName: stork
        containers:
        - image: mysql:5.6
                name: mysql
                env:
                - name: MYSQL_ROOT_PASSWORD
                value: password
                ports:
                - containerPort: 3306
                volumeMounts:
                - name: mysql-persistent-storage
                mountPath: /var/lib/mysql
        volumes:
        - name: mysql-persistent-storage
                persistentVolumeClaim:
                claimName: px-fast-pvc
```

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 the open-source project 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 uses two rules:

- **Affinity rules**: 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.
- **Anti-affinity rules**: Similarly, the ability to keep volumes on different nodes is also possible. 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 using volume placement strategies. Details can be found in the Portworx Documentation.

Scaling

Scaling a RKE2 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 RKE2 and Portworx clusters remain responsive and resilient, even during peak usage times.

RKE2 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.

RKE2 worker nodes can be added using the same procedure we used during our initial install which is documented in the Install Worker Nodes section.

Portworx

There are several objects that might need to be scaled during the normal course of operating Portworx on RKE2. 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 RKE2 administrators aren't available immediately to resize a disk that is quickly running out of space, and reduces the operational overhead of bespoke update to the cluster.

Note: To deploy AutoPilot in an air gapped environment, the autopilot images must be downloaded and placed in a local image 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 9 Autopilot sections

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` label, in a namespace with a resize: true label, 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 RKE2 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. This option requires no downtime for the storage cluster.
- **Add backing disks**: Horizontally scaling the number of backing drives per storage nodes is an alternative method to expand the storage cluster. The new drives should match the existing drives in terms of size and IOPS. This operation requires no downtime for the Portworx storage cluster but may involve a significant amount of data movement since the existing data in the cluster must be re-striped. During this re-striping period, the pool runs in a degraded mode.
- **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 RKE2 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 RKE2 cluster involves two distinct components: the RKE2 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.

SUSE Enterprise Linux Server (SLES)

Before upgrading SLES, it is essential to verify that the new version is compatible with both Portworx and RKE2. In some instances, you may need to upgrade Portworx and/or RKE2 before proceeding with the SLES upgrade. To ensure seamless operation of your RKE2 cluster with Portworx during and after the upgrade, please follow these best practices:

- Kernel Compatibility: If the new SLES version includes a new kernel, ensure that the current version of Portworx is compatible with this kernel version. Details can be found in Portworx documentation.
- Ensure RKE Compatibility: Details on RKE2 compatibility can be found in SUSE's documentation.

For details on upgrading SUSE Enterprise Linux Server, consult the SLES Upgrade Guide for the desired version.
For example, if upgrading to SLES 15 SP5, consult this guide.

Rancher Prime

It may be necessary to upgrade Rancher Prime to support newer versions of RKE2. Details are outside the scope of this document. Consult the Rancher Prime documentation for details.

Rancher Kubernetes Engine 2 (RKE2)

Before upgrading RKE2, 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 RKE2 upgrade. To ensure seamless operation of your RKE2 cluster with Portworx during and after the upgrade, please follow these best practices:

- **Kernel compatibility**: If the new RKE2 version includes a new kernel, ensure that the current version of Portworx is compatible with this kernel version.
- **Kubernetes compatibility**: Verify that the currently deployed version of Portworx is compatible with the new version of RKE2. Details can be found in the Portworx Documentation.
- **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 instructions on upgrading RKE2, consult the Rancher Prime documentation.

Portworx

As with an RKE2 upgrade, it's important to check version compatibility with the underlying RKE2 cluster. Ensure that the Kubernetes version, RKE2 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:

```
kubectl get pods -n portworx
```

```
NAME                                        READY   STATUS    RESTARTS       AGE
autopilot-67f89c4877-rm8xl                  1/1     Running   12 (21h ago)   21h
portworx-api-9n9zn                          2/2     Running   0              148m
portworx-api-grlh4                          2/2     Running   0              151m
portworx-api-pgjnc                          2/2     Running   0              146m
portworx-api-r4kzm                          2/2     Running   0              149m
portworx-kvdb-27ptz                         1/1     Running   0              21h
portworx-kvdb-4pqmw                         1/1     Running   0              21h
portworx-kvdb-jgmk4                         1/1     Running   0              21h
portworx-operator-8dc75bdd5-kc9hg           1/1     Running   2 (21h ago)    21h
portworx-pvc-controller-68f469844b-c8ks6    1/1     Running   2 (21h ago)    21h
portworx-pvc-controller-68f469844b-r95sd    1/1     Running   2 (21h ago)    21h
portworx-pvc-controller-68f469844b-wtcm6    1/1     Running   2 (21h ago)    21h
px-cluster-jjzvq                            1/1     Running   0              151m
px-cluster-kznp5                            1/1     Running   0              146m
px-cluster-l7z8z                            1/1     Running   0              149m
px-cluster-m74gs                            1/1     Running   0              148m
px-csi-ext-7bcf767777-4z4nw                 4/4     Running   20 (21h ago)   21h
px-csi-ext-7bcf767777-87r6x                 4/4     running   20 (0s ago)   21h
px-csi-ext-7bcf767777-d7gm5                 4/4     Running   19 (21h ago)   21h
px-telemetry-phonehome-8svtp                2/2     Running   0              21h
px-telemetry-phonehome-bhjnl                2/2     Running   0              21h
px-telemetry-phonehome-fds8s                2/2     Running   0              21h
px-telemetry-phonehome-kttnc                2/2     Running   0              21h
px-telemetry-registration-7f5f574d8d-kttjl 2/2     Running   0              21h
stork-59d74b9c7d-jw74l                      1/1     Running   2 (21h ago)    21h
stork-59d74b9c7d-mtxbm                      1/1     Running   1 (21h ago)    21h
stork-59d74b9c7d-svjwr                      1/1     Running   2 (21h ago)    21h
stork-scheduler-54c998b564-2l9vx            1/1     Running   2 (21h ago)    21h
stork-scheduler-54c998b564-fvh84            1/1     Running   2 (21h ago)    21h
stork-scheduler-54c998b564-md9ct            1/1     Running   2 (21h ago)    21h
```

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. Refer to the Portworx CLI (pxctl) section for details.

Run the pxctl command line utility to query the Portworx Storage Cluster with a `pxctl status` command.

```
ccrow@ccrow-ubuntu:~$ pxctl status
Handling connection for 9001
Status: PX is operational
License: Trial (expires in 30 days)
Node ID: 9d3a3b8b-bbe6-4235-974b-f102b4326e03
         IP: 10.13.200.86
         Local Storage Pool: 1 pool
         POOL     IO_PRIORITY       RAID_LEVEL        USABLE  USED      STATUS  ZONE      REGION
         0        HIGH              raid0             86 GiB  35 MiB  Online  default default
         Local Storage Devices: 1 device
         Device  Path               Media Type                 Size              Last-Scan
         0:0      /dev/nvme0n2      STORAGE_MEDIUM_NVME        100 GiB           31 Dec 24 08:42 PST
         total                      -                          100 GiB
         Cache Devices:
         * No cache devices
         Metadata Device:
         1        /dev/nvme0n1      STORAGE_MEDIUM_NVME        100 GiB
         * Internal kvdb on this node is using this dedicated metadata device to store its data.
Cluster Summary
         Cluster ID: px-cluster
         Cluster UUID: 5450566e-ae8c-4293-ba7a-0506183bd78d
         Scheduler: kubernetes
         Total Nodes: 4 node(s) with storage (4 online)
         IP                 ID                                    SchedulerNodeName           Auth      StorageNode        Used
Capacity         Status  StorageStatu
s        Version            Kernel                         OS
         192.168.0.84       cb505e46-1953-4253-b538-fa38c1bccfd2  sles4                       Enabled Yes(PX-StoreV2) 35 MiB  86
GiB      Online  Up         3
.2.1.0-b298102  5.14.21-150500.53-default      SUSE Linux Enterprise Server 15 SP5
         192.168.0.85       aaf20c82-b40f-407c-b984-cfaca856923b  sles5                       Enabled Yes(PX-StoreV2) 35 MiB  86
GiB      Online  Up         3
.2.1.0-b298102  5.14.21-150500.53-default      SUSE Linux Enterprise Server 15 SP5
         192.168.0.86       9d3a3b8b-bbe6-4235-974b-f102b4326e03  sles6                       Enabled Yes(PX-StoreV2) 35 MiB  86
GiB      Online  Up (This nod
e)       3.2.1.0-b298102 5.14.21-150500.53-default      SUSE Linux Enterprise Server 15 SP5
         192.168.0.87       92ac6c9b-2038-49f1-b1f5-6ea0fc84bd81  sles7                       Enabled Yes(PX-StoreV2) 35 MiB  86
GiB      Online  Up         3
.2.1.0-b298102  5.14.21-150500.53-default      SUSE Linux Enterprise Server 15 SP5
Global Storage Pool
         Total Used         :  141 MiB
         Total Capacity  :  344 GiB
Collected at: 2024-12-31 11:19:00 PST
```

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.

```
pxctl sv kvdb members
```

```
ccrow@ccrow-ubuntu:~$ pxctl sv kvdb members

Handling connection for 9020

Kvdb Cluster Members:

ID                                      PEER URLs                               CLIENT URLs
LEADER  HEALTHY DBSIZE

92ac6c9b-2038-49f1-b1f5-6ea0fc84bd81  [http://portworx-1.internal.kvdb:9018]  [http://192.168.0.87:9019]
false   true    620 KiB

aaf20c82-b40f-407c-b984-cfaca856923b  [http://portworx-2.internal.kvdb:9018]  [http://192.168.0.85:9019]
false   true    612 KiB

9d3a3b8b-bbe6-4235-974b-f102b4326e03  [http://portworx-3.internal.kvdb:9018]  [http://192.168.0.86:9019]
true    true    616 KiB
```

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.

We can view the current version of the operator by running:

```
kubectl -n portworx get deployments.apps portworx-operator \
-o jsonpath='{.spec.template.spec.containers[0].image}'
```

You can get a list of operator versions from the Portworx documentation.

We can now upgrade the operator:

```
kubectl -n portworx set image deployment/portworx-operator \
portworx-operator=portworx/px-operator:24.2.0
```

Replace the version at the end of the command with the desired operator version.

We can now verify that the operator is running:

```
kubectl -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.

```
ccrow@ccrow-ubuntu:~$ kubectl -n portworx get pod -l name=portworx-operator

NAME                                    READY   STATUS  RESTARTS       AGE

portworx-operator-8dc75bdd5-kc9hg   1/1       Running   2 (22h ago)   22h
```

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.

```
kubectl edit -n portworx storagecluster
```

For example: `image: portworx/oci-monitor:3.2.1` would be changed to `image: portworx/oci-monitor:3.2.2` if you were attempting to upgrade to Portworx 3.2.2.

```
spec:

...

  csi:

        enabled: true

        installSnapshotController: true

  image: portworx/oci-monitor:3.1.2

  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

Prometheus is an open-source monitoring and alerting toolkit designed specifically for reliability and scalability in dynamic environments like Kubernetes. In a RKE2 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. It is possible to use this included version, but SUSE and Portworx recommend using the monitoring stack provided by SUSE Rancher Prime. This will require some configuration changes that are documented in the Working with the SUSE Rancher provided Monitoring application section of this document. Portworx integrates with this Prometheus stack and provides several key metrics that can be used to monitor the health and performance of the Portworx cluster.

SUSE Rancher Monitoring Service Monitor

In order for Rancher Monitoring to access the Portworx metrics, we will need to configure a new ServiceMonitor. If we used the installation option documented in Monitoring Configuration for Rancher Kubernetes Engine 2 section, we can simply use the following definition:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  namespace: cattle-monitoring-system
  name: portworx-prometheus-sm
  labels:
        name: portworx-prometheus-sm
        k8s-app: prometheus-operator
spec:
  selector:
        matchLabels:
        name: portworx
  namespaceSelector:
        matchNames:
        - portworx
  endpoints:
  - port: px-api
        targetPort: 9001
```

StorageCluster Configuration Changes

By default, Portworx installs a version of Prometheus. This will conflict with the Rancher provided version of prometheus. For this reason, we need to disable the Portworx provided prometheus instance. We will still need to export metrics for the Rancher provided monitoring package.

Update your StorageCluster configuration with the following:

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

We also must export metrics so that the Rancher supplied Prometheus installation can import them. This is required for Autopilot to operate.

We will need to change the autopilot URL to reference the rancher-monitoring application. Update your StorageCluster configuration with the following:

```
spec:
  autopilot:
        providers:
        - params:
          url: http://rancher-monitoring-prometheus.cattle-monitoring-system.svc.cluster.local:9090
```

All of these configuration changes to support monitoring have been captured in our Reference Architecture Example.

Alternatively, the above changes can be applied to an existing StorageCluster by running a kubectl patch command:

```
 kubectl patch storagecluster px-cluster \
 -n portworx \
 --type='json' \
 -p='[ {"op": "replace", "path": "/spec/autopilot/providers/0/params/url", "value": "http://rancher-
monitoring-prometheus.cattle-monitoring-system.svc.cluster.local:9090"}, {"op": "replace", "path": "/
spec/monitoring/prometheus/enabled", "value": false} ]'
```

See the Portworx Documentation for further details.

Querying Portworx Metrics in Prometheus

We can use the prometheus web interface to query Portworx metrics:

1. Click on the RKE2 cluster from the navigation menu
2. Select the `monitoring` section
3. Click on the `Prometheus Graph` panel from the dashboard
4. Type in the metric you wish to query (e.g. `px_cluster_cpu_percent`)
5. Click execute

FIGURE 28 Portworx metrics in Prometheus

In this way you can view Portworx metrics from the SUSE Rancher Prime provided Prometheus Instance.

Note: Portworx metrics will always begin with a px_ prefix.

AlertManager

Alertmanager is a critical component of the Prometheus ecosystem, responsible for handling alerts generated by Prometheus. In a RKE2 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.

Portworx installs a number of Portworx specific rules by default. We can view these rules from the Prometheus Rules UI found here:

1. Click on the RKE2 cluster from the navigation menu
2. Select the `monitoring` section
3. Click on the `PrometheusRules` panel from the dashboard

It is also possible to get a list of rules by running the following command:

```
kubectl get prometheusrules.monitoring.coreos.com  -n portworx portworx -o yaml
```

Grafana Dashboards

Portworx provides five out-of-the-box Grafana dashboards to help monitor its status and performance.

We can deploy these dashboard with the following steps:

Download the Portworx Grafana dashboards:

```
curl "https://docs.portworx.com/samples/portworx-enterprise/k8s/pxc/portworx-cluster-dashboard.json" -o portworx-cluster-dashboard.json && \

curl "https://docs.portworx.com/samples/portworx-enterprise/k8s/pxc/portworx-node-dashboard.json" -o portworx-node-dashboard.json && \

curl "https://docs.portworx.com/samples/portworx-enterprise/k8s/pxc/portworx-volume-dashboard.json" -o portworx-volume-dashboard.json && \

curl "https://docs.portworx.com/samples/portworx-enterprise/k8s/pxc/portworx-performance-dashboard.json" -o portworx-performance-dashboard.json && \

curl "https://docs.portworx.com/samples/portworx-enterprise/k8s/pxc/portworx-etcd-dashboard.json" -o portworx-etcd-dashboard.json
```

Install the Portworx Grafana dashboards:

```
kubectl -n cattle-dashboards create configmap grafana-dashboards \
--from-file=portworx-cluster-dashboard.json \
--from-file=portworx-performance-dashboard.json \
--from-file=portworx-node-dashboard.json \
--from-file=portworx-volume-dashboard.json \
--from-file=portworx-etcd-dashboard.json
```

Label our new configmap so that the Rancher Prime Grafana instance will use it:

```
kubectl -n cattle-dashboards label configmaps portworx-grafana-dashboards grafana_dashboard=1
```

We can then access Grafana from the Rancher Prime interface:

1. Click on the RKE2 cluster from the navigation menu
2. Select the monitoring section
3. Click on the Grafana panel from the dashboard

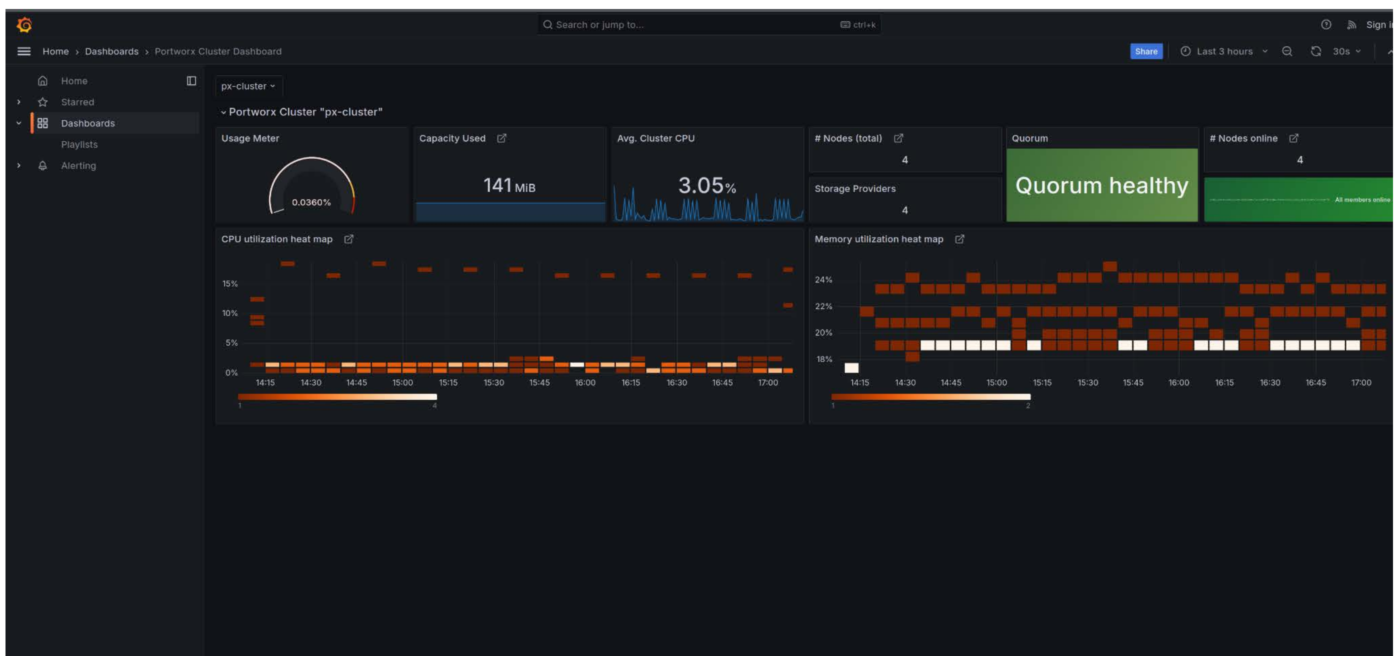

FIGURE 29 Example Grafana dashboard

Summary

This reference architecture provides a comprehensive guide for deploying and managing RKE2 clusters on bare metal with Portworx Enterprise, ensuring a robust, scalable, and high-performing infrastructure. The architecture emphasizes critical considerations such as high availability, data locality, and efficient resource utilization. It outlines best practices for installation, configuration, and operational management, covering essential areas like backup and recovery, monitoring, and logging. By following this reference architecture, organizations can confidently deploy RKE2 and Portworx Enterprise on bare metal, achieving optimal performance, resilience, and reliability for their production workloads.

Appendix: Portworx Configuration Example

Throughout this document we have referenced our Portworx StorageCluster specification. This specification can be found in its entirety here.

Operator Install Command

```
kubectl apply -f 'https://install.portworx.
com/3.2?comp=pxoperator&kbver=v1.30.1+rke2r1&ns=portworx'
```

Portworx Storage Cluster Specification


```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
  name: px-cluster
  namespace: portworx
  annotations:
        portworx.io/misc-args: " -T px-storev2 "
        portworx.io/disable-storage-class: "true"
spec:
  autopilot:
        enabled: true
        providers:
        - name: default
        params:
        url: http://rancher-monitoring-prometheus.cattle-monitoring-system.svc.cluster.local:9090
        type: prometheus
        env:
        - name: PX_SHARED_SECRET
        valueFrom:
        secretKeyRef:
        key: apps-secret
        name: px-system-secrets
  image: portworx/oci-monitor:3.2.1
  imagePullPolicy: Always
  kvdb:
        internal: true
  storage:
        devices:
        - /dev/nvme0n2
        - /dev/nvme0n3
        - /dev/nvme0n4
        - /dev/nvme0n5
        systemMetadataDevice: /dev/nvme0n1
  secretsProvider: k8s
  security:
   enabled: true
   auth:
        guestAccess: 'Disabled'
  network:
        dataInterface: bond1
        mgmtInterface: bond0
  stork:
        enabled: true
        args:
        webhook-controller: "true"
  autopilot:
        enabled: true
  csi:
        enabled: true
  monitoring:
        telemetry:
        enabled: true
        prometheus:
        enabled: false
        exportMetrics: true
```


©2025 Pure Storage, the Pure Storage P Logo, FlashArray, FlashBlade, Portworx, and the marks in the Pure Storage Trademark List are trademarks or registered trademarks of Pure Storage Inc. in the U.S. and/or other countries. The Trademark List can be found at purestorage.com/trademarks. Other names may be trademarks of their respective owners.

PS2745-01-en 05/25