PURESTORAGE®

SUSE RANCHER

REFERENCE ARCHITECTURE

# Rancher Kubernetes Engine 2 on VMware with Portworx

Gain a consistent experience across public cloud, on- premises, hybrid cloud, or edge architecture.

# Contents

## Executive Summary

Modern applications are built using containers and orchestrated by Kubernetes, but they still need a layer of persistence. SUSE Rancher is an industry-leading hybrid cloud application platform powered by Kubernetes, bringing together tested and trusted services to reduce the friction of developing, modernizing, deploying, running, and managing applications. It delivers a consistent experience across public cloud, on premises, hybrid cloud, or edge architecture.

Rancher Kubernetes Engine (RKE2) is SUSE Rancher's fully conformant Kubernetes distribution focused on security and compliance. To run stateful applications on RKE2, organizations need a robust data services platform like Portworx®, which provides features such as replication and high availability, security and encryption, capacity management, disaster recovery, and data protection to RKE2 deployments. By adopting RKE2 with Portworx, organizations can streamline their modernization journeys while avoiding the need to dedicate resources to designing and managing a custom Kubernetes storage layer.

---

## About This Document

This Portworx Reference Architecture provides a validated design and deployment model for running Portworx on RKE2 within a vSphere environment. It is intended for Kubernetes administrators and cloud architects who have experience with Portworx. We recommend that readers have a good understanding of RKE2 concepts, with familiarity in Kubernetes usage.

The document is organized into the following three technical sections:

- **Planning and architecture overview**

This section provides a high-level architecture overview of deploying Portworx on RKE2. It covers the requirements for storage and storageless nodes in RKE2, along with recommended configurations for vSphere datastores.

- **Design considerations**

This section provides detailed requirements and recommendations to be considered during the design phase. It covers RKE2 requirements, networking, capacity planning, high availability, resource and performance considerations, as well as security and monitoring.

This section concludes with a template recommended for Portworx installation.

- **Operations considerations**

This section focuses on “day 2” best practices following the deployment of Portworx. It includes guidance on validating the Portworx installation, observability, scaling Portworx, backup and recovery techniques, and best practices on upgrading Portworx and RKE2. Additionally, it provides instructions on checking Portworx logs.

## Value Proposition

### Benefits of Portworx

Traditional storage solutions provide a basic Container Storage Interface (CSI) driver connector to manage stateful applications in Kubernetes environments. However, CSI connectors have several limitations and lack the robustness required for high availability.

Portworx offers a superior alternative by accelerating time to revenue, delivering data resiliency, and agility at enterprise scale for Kubernetes storage and databases. This translates into improved productivity for platform engineering teams.

Portworx offers the following key advantages:

- Custom resources enables the storage-aware pod scheduling by co-locating the pod and volume together for optimal performance.
- Automatic volume or pool adjustment based on predefined rules, further enhance the efficiency.

Portworx storage services provide elastic scalability, industry-leading availability, and self-service access to storage for Kubernetes environments. With integrated storage management features such as rule-based automation, thin-provisioning allocation and flexibility for multicloud, hybrid-cloud, and on-premises deployments, Portworx delivers unmatched flexibility and reliability for modern infrastructures.

### Benefits of Running Portworx on RKE2 with vSphere

As part of their digital transformation efforts, organizations are modernizing their applications and infrastructure by adopting containers and Kubernetes for application management, while leveraging a solution like RKE2 for infrastructure. RKE2 allows organizations to benefit from fully automated operations, a consistent experience across all environments, and self-service provisioning for developers. This approach empowers teams to collaborate effectively and accelerate the transition from development to production.

RKE2 clusters can be provisioned using two methods:the SUSE Rancher UI or a cluster configuration file. For more details, refer to the RKE2 Cluster Configuration Reference | RKE2.

Portworx enhances RKE2 by adding a robust, secure, highly available, and scalable data management layer, which allows applications to seamlessly consume storage, simplifying data management, and boosting operational efficiency.

### Target Use Cases

This document provides guidelines and best practices to deploy Portworx on RKE2, specifically on workload clusters (deployment on the RKE2 admin cluster is not anticipated).

By following the recommendations in this document, RKE2 users can deploy any type of stateful application using Portworx. While the scope of this document does not include application-specific recommendations, the deployment approach described is designed to provide a stable and reliable storage solution suitable for any application requiring persistent storage.

## Planning and Architecture Overview

### Reference Architecture High Level Design

The diagram below illustrates the high level design of the Portworx reference architecture deployed on RKE2 within a vSphere environment:

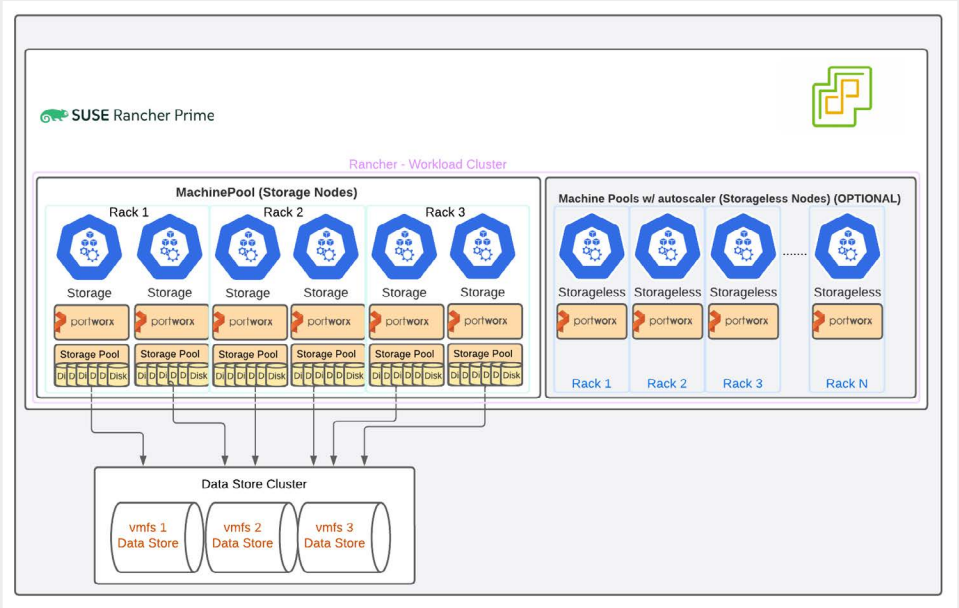



**FIGURE 1** High-level architecture overview

By implementing this design, platform engineering teams can automate the provisioning of a well-defined architecture following best practices. It incorporates key features such as high availability, operations management, observability, business continuity, performance, and security.

Portworx requires a minimum of three storage nodes in the cluster. However, for production environments, this reference architecture recommends starting with a cluster size of six storage nodes. Although a three-node Portworx cluster may perform adequately in certain environments, there are notable advantages to deploying six storage nodes initially.

- Cluster capacity

In a three-node storage cluster, the loss of a single node results in the loss of one-third of the cluster capacity, significantly increasing the load on the remaining two nodes until the lost node is restored.

In a six-node cluster, losing one node reduces only one-sixth of the cluster capacity, allowing the remaining five nodes to distribute the load more evenly and maintain better performance.

- I/O load distribution

A Portworx cluster with three storage nodes may face more frequent I/O latencies during peak usage periods.

In comparison, a six-node cluster can distribute the load across additional nodes and handle I/O requests more efficiently, reducing the risk of I/O latencies.

Based on the above recommendations, it is important to prepare RKE2 and vSphere before deploying Portworx.

Consider the following when planning your deployment:

### Storage Nodes

A group of six storage nodes is the minimum requirement for Portworx deployments. While fewer than six nodes may suffice for minimal workloads, it is not recommended for production environments. Storage nodes play a critical role in cluster quorum formation, which is achieved when ( n + 1) / 2 nodes join the cluster, where `n` is the total number of storage nodes.

Key considerations for storage nodes:

- Ensure at least six nodes are present for production environments.
- If the RKE2 cluster spans multiple availability zones, distribute storage nodes evenly across zones. A minimum of three availability zones is recommended for deployments across regions in cloud environments. For on-premises deployments, zones or rack labels can be added manually to achieve fault tolerance.
- During installation, Portworx will set the `MaxStorageNodesPerZone` configuration parameter to limit the maximum number of storage nodes it creates per zone. If additional storage nodes are needed while scaling the cluster, you should update this field in the `StorageCluster` specification.
- Portworx does not support cluster autoscaling. However, horizontal scaling can be performed manually by adding more nodes to increase storage capacity. Reducing the number of nodes is not supported.
- Apply the label `portworx.io/node-type:storage` to all storage nodes to enable Portworx to automatically provision storage for newly added nodes.
- Run hyperconverged applications on these nodes to achieve high performance levels.
- Each node must have a minimum of 8 CPU cores and 16 GB of RAM. For more information about the CPU and memory requirements, refer to the Resource considerations section.

### Storageless Nodes (optional)

Storageless nodes are an optional group of nodes that can be added to a Portworx cluster for specific use cases.

- There is no defined minimum number of storageless nodes required.
- They can be associated with a cluster autoscaler, allowing dynamic scaling to meet fluctuating workload demands.
- Apply the label `portworx.io/node-type:storageless` to all nodes so that Portworx can automatically recognize and classify them as storageless.
- They can be used in a dynamic compute environment, where the number of compute nodes can elastically increase or decrease based on workload requirements.
- Applications running on these nodes access storage through the cluster network, utilizing storage available on the storage nodes.
- In environments with a growing number of storageless nodes, it's critical to monitor performance closely. A high number of storageless nodes can place excessive load on storage nodes, potentially impacting application performance.
- Each node must have at least eight CPU cores and 16 GB of RAM. For more details on the resource requirements, refer to the Resource considerations section.

### vSphere API

Portworx has integrations with the vSphere API to dynamically manage and provision block-storage automatically. This functionality requires a vCenter service account for Portworx to perform these operations.

### Datastores

When RKE2 is deployed on VMware vSphere, it's important to plan not only for cluster and node capacity but also to appropriately size the shared VMFS datastores that Portworx will use. The initial selection of vSphere datastore sizes and Portworx pool sizes should allow for future pool expansions with ease. A Portworx pool can be expanded using two methods:

- Adding a new disk (Portworx creates a new VMDK) and adding it to the pool
- Resizing all the existing disks (VMDKs) within the pool

Portworx recommends resizing existing disks within a pool for expansion, as this method is faster and efficient. Unlike adding new disks, resizing doesn't require data redistribution, which is an I/O intensive process that takes time proportional to the amount of data needing redistribution.

Following are the guidelines on planning and provisioning vSphere datastores for Portworx to align with the resizing disk strategy:

- Provision multiple and large-sized datastores which can be expanded in the future. This approach minimizes the effort involved in managing VMFS datastore sizing.
- Having larger datastores avoids limitations on the maximum size of disk that Portworx can create on them.
- Choose fewer large datastores, such as six datastores of 16TiB, instead of many smaller datastores like 12 datastores of 8TiB.
- By default, Portworx recommends thin-provisioning for cloud drives. However, customers can also use lazy zeroed thick or eager zeroed thick provisioning for better insight into the provisioned space.
- Expanding existing datastores is preferred over adding new ones, as it avoids the need to perform a Storage DRS and rebalance VMDKs.
- Portworx recommends limiting datastore usage to a single Portworx cluster to predict the appropriate initial datastore size. Do not share datastores across multiple Portworx clusters.
- Portworx supports both Compute vMotion and Storage vMotion.

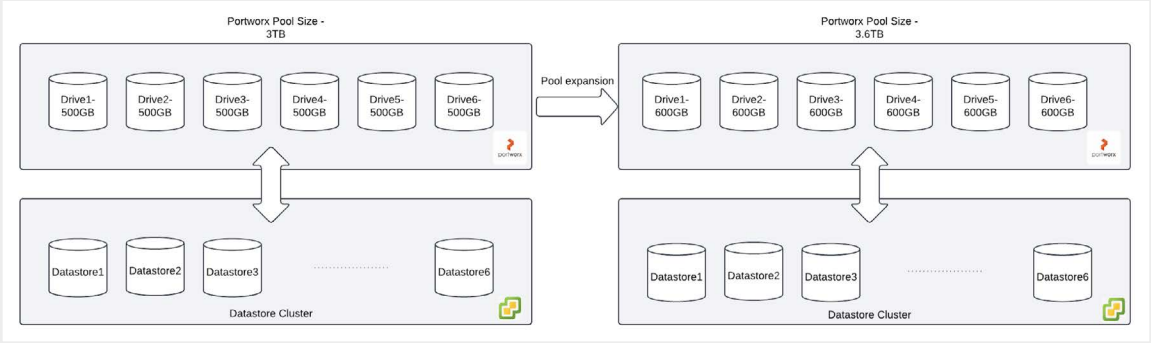
### Storage Pools

Portworx recommends configuring a single pool per node, consisting of six drives.

- This can be achieved by specifying six different cloud drive entries in the StorageCluster specification, ensuring all disks have the same size and type. The  example demonstrates creating a 3TiB storage pool per node, with each pool containing six disks:

```
cloudStorage:
    deviceSpecs:
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
  journalDevice: type=thin,size=3
```

- Dividing capacity across multiple drives in a pool enables future pool expansions through simple disk resizing, instead of adding disks.
- This approach ensures a non-disruptive “resize” operation, which will be the only available method in the next-generation px-storev2 backend.
- With multiple disks in a single pool, while expanding a pool, smaller incremental resize is required on the VMDKs which are spread out across datastores.
- If a datastore hosts a large single disk, there is a high risk that the datastore may lack enough space to resize that single disk to meet the pool expansion requirement.
- However, multiple smaller disks in a pool increases the likelihood that the datastore can accommodate these incremental resizes required on the disk.
- The alternative expansion strategy–adding a new disk to the pool–is a more expensive and time-consuming operation than resizing existing disks, due to the data movement it requires.
- It is important to note that a Portworx pool supports a maximum of six disks. Therefore, dividing capacity among multiple disks not only facilitates smoother expansions but also optimizes resource utilization.



## Design Considerations

### RKE2

The minimum required version of RKE2 for this reference architecture is RKE2 1.28.8 or newer. For information on the currently supported versions of RKE2, refer to the SUSE Rancher Manager documentation.

Ensure that the RKE2 version is compatible with the Portworx version by reviewing the Prerequisites for RKE2 - Portworx Documentation and verifying kernel compatibility in the Supported Kernels - Portworx Documentation.

### Prerequisites

Before installing Portworx on RKE2, ensure that all prerequisite steps outlined in the above section are completed.

### Networking

Portworx recommends a network bandwidth of 10 Gbps, with a minimum requirement of 1Gbps and latency under 10 ms between nodes.

By default, RKE2 keeps all ports open among worker nodes. However, if your network uses a specific firewall configuration, ensure that the ports listed in the “East-to-West” section of the Prerequisites for RKE2 - Portworx Documentation are open for node-to-node communication.

Additionally, the telemetry feature utilizes limited-outbound connectivity. The destinations for this connectivity are specified in the “Outbound” section of the Prerequisites for RKE2 - Portworx Documentation and must be reachable from all cluster nodes.

### Storage

This section provides guidelines for capacity planning in your Portworx cluster.

It covers three key aspects:

- Initial cluster capacity
- Storage node capacity sizing
- vSphere datastore sizing

#### Initial Cluster Capacity

Consider the following factors when planning the initial capacity for your Portworx cluster:

- Number of volumes (PVCs) in the cluster
- Average size of volumes
- Number of nodes
- Replication factor: Portworx recommends a replication factor of two or three. This setting provides redundancy to protect against single volume or pool failures, and includes I/O optimizations that enhance performance.
- Buffer for local snapshots

Below are two examples demonstrating how to calculate the initial cluster capacity:

| Volume Size | Volumes | Replication | Cluster Size (1.3 x Repl x Volumes x Size) |
|---|---|---|---|
| 50GiB | 30 | 3 | 5.85 TiB |
| 100GiB | 50 | 2 | 13 TiB |

**TABLE 1** Initial Cluster capacity

The initial cluster size is calculated by multiplying the average volume size, the number of volumes, and the replication factor, and then adding a 30% buffer to account for local snapshots.

**Storage Node Capacity Sizing**

After determining the cluster size, you can calculate the capacity required for each storage node using the formula:

(Cluster size ÷ Number of storage nodes) + min (10%, 100 GiB)

The additional buffer accounts for pool recovery in case the pool becomes full.

Using the above example:

| Cluster Size | Number of Storage Nodes | Node Capacity | Number of Datastores |
|---|---|---|---|
| 5.85 TiB | 6 | 1.075 TiB | 3 (2.2 TiB each) |
| 13 TiB | 6 | 2.26 TiB | 3 (4.6 TiB each) |

**TABLE 2** Node Capacity Sizing

**vSphere Datastore Sizing**

Finally, when planning the number of vSphere datastores for your environment, keep in mind that vSphere limits the capacity of a single datastore to 64 TB. Portworx recommends using multiple datastores rather than relying on a single large one.

Creating multiple datastores allows for easier future expansion of existing datastores when the capacity requirements increase.

Expanding existing datastores is preferred over adding new datastores to avoid the need for Storage DRS and rebalancing VMDKs.

For example, in the first scenario above, you can allocate three datastores, each initially sized at 2.2 TiB. In the second scenario, you can allocate three datastores with an initial size of 4.6 TiB.

## Implementing High Availability

To implement a highly available cluster, you can use the Portworx topology awareness feature. In an RKE2 cluster running on a vSphere environment, Portworx can automatically identify topology node labels, specifically topology.kubernetes.io/region and topology.kubernetes.io/zone. While regions and availability zones are typically used in cloud deployment, for on-premises deployments, you can manually assign either rack or zone labels to nodes.

Ensure that topology labels are added to each node in your RKE2 cluster. Portworx recommends placing all six storage nodes in the same region, with two nodes in each zone. A zone in this case must be an isolated entity such that a failure in one zone does not affect other zones.

In this scenario, Portworx will automatically distribute volume replicas across separate failure-domains. If one domain becomes unavailable, the application can continue running in another domain where a replica of the same volume resides.

## Deployment Model

This reference architecture uses a deployment model where applications can run both on both storage and storageless nodes.

As described in the Reference architecture high level design section, the recommended approach involves creating two separate RKE2 “machine pools”: one for storage nodes and another for storageless nodes.

**Storage nodes**: The machine group for storage nodes provides a static set of nodes for hyperconverged applications. While you can manually scale out this machine group to add more storage nodes as needed,scaling back is not automated. Storage nodes can only be removed manually from the Portworx cluster.

**Storageless nodes**: On the other hand, with RKE2 cluster autoscaler, it is easier to automate and manage Portworx storageless nodes in this deployment model. The optional machine group for storageless nodes benefits from the RKE2 cluster autoscaler, enabling automated scaling based on resource consumption. This allows for easier management and flexibility in handling cluster workloads.

For more information on configuring RKE2 for auto-scaling, refer to Cluster Autoscaler | RKE2.

## Resource Considerations

When designing a cluster for a specific workload, consider the following factors:

- The expected number of volumes in use at any given time
- The average throughput or IOPS requirement of these volumes
- The desired high-availability level
- Whether snapshots will be used

To estimate the backend IOPS requirement, sum up the IOPS per volume, multiply by the HA level, and add a snapshot and fragmentation overhead factor of 1.4. This total should remain below the aggregate IOPS capacity of all storage pools across all the cluster nodes.

For vSphere deployments, each storage pool can consist of multiple VMDKs across datastores. The total IOPS capacity is the sum of the IOPS for all VMDKs in the pool.

The table below provides two examples for reference. These sample numbers and the associated overhead focus primarily on Write operations, as Read operations do not experience amplification.

**NOTE:** Different workloads may exhibit varying read/write I/O patterns, which should be considered when planning resources.

| Number of Volumes | Repl Factor | Average IOPS | Overhead | Total IOPS |
|---|---|---|---|---|
| 240 | 2 | 200 | 1.4 | 134400 |
| 300 | 3 | 200 | 1.4 | 252000 |

**TABLE 3** Calculating Total IOPS based on Replica factor and Average IOPS

The minimum requirements for each Portworx node are eight CPUs and 16 GB RAM. However, for high-performance workloads, Portworx recommends allocating sufficient enough RAM and CPU resources based on the expected I/O load. For optimal performance, especially when Portworx is running alongside applications on the same virtual machine, it is advisable to provide at least 32 CPU cores and 32 GB of RAM.

## Performance Considerations

Portworx is configured by default to optimize cluster performance. However, in specific scenarios (outlined in this section), additional parameters can further enhance the overall performance of the Portworx cluster.

- Journal device

Using a dedicated journal device is recommended in all cases. Newer versions of Portworx allow improved performance in certain scenarios when a journal device is configured. To enable this feature, include the following configuration in the `StorageCluster` yaml:

```
kind: StorageCluster

…
 cloudStorage:
    …

    journalDeviceSpec: type=thin,size=3
…
```

When using a journal device, consider enabling the `auto_journal` I/O profile to enhance performance for volumes with a replication factor of one. For more details and specific use cases, refer to this article.

- Storage class auto profile

Portworx can automatically detect the optimal I/O profile for an application when you set the `io_profile` parameter in the storage class to **auto**. In this case, you must set the replication factor to two or three. Below is an example of a storage class configuration using this parameter:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-storage-class
provisioner: pxd.portworx.com
allowVolumeExpansion: true
parameters:
  repl: "2"
  priority_io: "high"
  io_profile: auto
```

- Runtime options

By default, Portworx consumes minimal CPU and memory from the virtual machines, with a maximum of eight vCPUs and eight GB of memory. If your nodes have a high number of CPUs (more than 32 vCPU) and large memory (greater than 64GB), it's recommended to enable the `rt_opts_conf_high` runtime option for performance.

Below is an example of a `StorageCluster` YAML with this parameter:

```
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  name: px-cluster
  namespace: portworx

spec:
   image: portworx/oci-monitor:3.2.0
      ...
   runtimeOptions:
          rt_opts_conf_high: "1"
  …
```

- `'nodiscard'` option

Certain applications, such as Kafka and Elastic, generate a high volume of discard and delete operations, which can impact the overall performance of the Portworx cluster. To mitigate this, it's recommended to use the `nodiscard` parameter.

**NOTE**: When you enable the nodiscard parameter, data written or deleted from the filesystem is not immediately removed from the block device. Therefore, a periodic filesystem trim operation must be run to clear the deleted data from the block device. For more information, refer to the relevant documentation.

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

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-storage-class
provisioner: pxd.portworx.com
allowVolumeExpansion: true
parameters:
  repl: "2"
  nodiscard: "true"
  io_profile: auto
```

When using the `nodiscard` option, Portworx recommends enabling the `autofstrim` option in the cluster to periodically delete unused data blocks. For detailed instructions, refer to the Maintain volumes using Filesystem Trim article in the Portworx documentation.

**Large Clusters Considerations**

For large clusters with 100 to 200 nodes, Portworx recommends additional configuration to ensure optimal performance and stability:

1. Configure dedicated KVDB nodes:

- Reserve nodes with 32 vCPUs and 48 GB of memory for KVDB.
- Use a minimum of three nodes (five nodes recommended for high availability).
- Apply the following label to the reserved nodes:
- `kubectl label nodes <node-name> px/metadata-node=true`

2. Increase vCenter session limits:

- Adjust the session limits in vCenter to allow more connections for handling large numbers of nodes. Ensure the vCenter Server has adequate resources (at least 16 CPUs and 32 GB of RAM).
- Update the vCenter configuration:
- maxSessionCount = 6000
- maxSessionsPerUser = 3000
- For detailed instructions, refer to the VMware article: vCenter Cloud Account Datacollection Fails.

3. Increase quorum timeout:

- Update the quorum timeout parameter to enhance cluster stability:
- `pxctl cluster options update --runtime-options "quorum_timeout_in_seconds=3600"`

**NOTE:** For clusters larger than 200 nodes, consult your Portworx Sales team for additional guidance.

## Security

Securing your Portworx cluster involves two primary factors:

1. **Authorization**: Protects Portworx volumes from unauthorized access
2. **Encryption**: Secures data within the volumes by encrypting it

### Authorization

Authorization adds an additional layer of security to Portworx volumes. By leveraging industry standards, Portworx implements Role-Based Access Control (RBAC) to safeguard volumes against unauthorized access. Only authenticated and authorized users can access volumes. When security is enabled, Portworx automatically generates a user token for the kubernetes user by default.

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

```
kind: StorageCluster

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

Once authorization is enabled, only `kubernetes` users can access Portworx volumes if PVCs are created using storage classes with the authentication token (see example here). By default, `guest access` is allowed for pre-existing (unowned) volumes if no authentication token is included in the storage class. To disable `guest access`, refer to the relevant Portworx documentation.

Additionally, all future `pxctl` commands will require an admin token for execution. Instructions for setting this up can be found in the Portworx documentation.

For more details and advanced configurations in multi-tenant clusters, refer to the Portworx documentation.

### Encryption

Portworx recommends securing your volumes with encryption. You can apply this at the Portworx volume level or further down the stack (for example, FlashArray™).

All Portworx encrypted volumes are secured using a passphrase and are encrypted both at rest and in transit.

Portworx supports several secret stores, such as Hashicorp Vault and Vault Transit, Kubernetes Secrets, and most public cloud provider secret stores. The complete list of supported secret store management can be found in the Secret Store Management article in the Portworx documentation.

For this reference architecture, Portworx uses the Hashicorp Vault secret store.

### Set Up Hashicorp Vault

Before deploying Portworx, you must configure your RKE2 cluster to access the Vault server using the Vault Kubernetes authentication method. Follow the instructions in the Vault article in the Portworx documentation to complete this configuration.

When deploying Portworx, you can add Vault as the secret store provider by including the following in the `StorageCluster` YAML:

```
kind: StorageCluster

…
 spec:
  …

  secretsProvider: vault
  …
```

After deploying Portworx, you must define a cluster-wide passphrase to enable volume encryption.

Refer to the Encrypting Kubernetes PVCs with Vault article in the Portworx documentation to create the passphrase and start using encrypted volumes with Portworx. At a minimum, this document recommends using Portworx's cluster-wide encryption feature. For environments requiring additional security, such as multi-tenant clusters, refer to the same article for advanced per-volume encryption options.

## Monitoring

Portworx seamlessly integrates with the RKE2 Prometheus, offering a range of metrics to monitor the health and performance of your Portworx cluster.

Before deploying Portworx in your RKE2 cluster, ensure that monitoring for RKE2 is enabled. For instruction, refer to Enable Monitoring | RKE2.

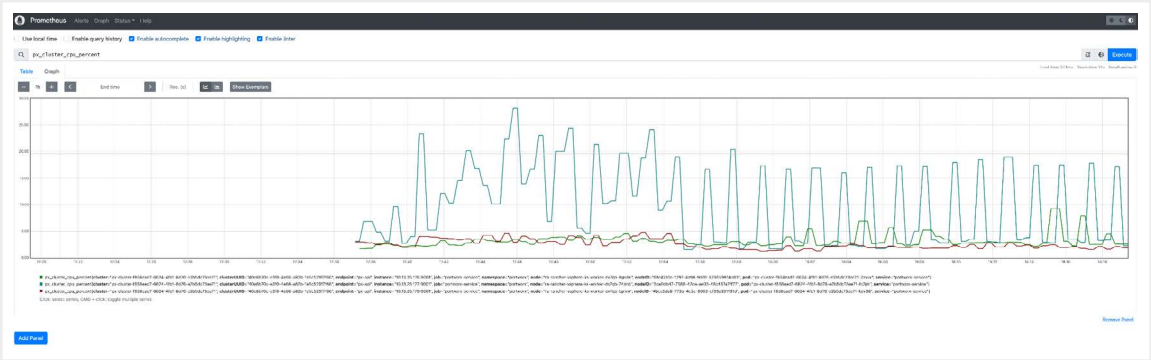
Once RKE2 monitoring is enabled and Portworx is deployed, update the `exportMetrics` option to `true` in the `StorageCluster` object to enable metric support:

```
  monitoring:
    prometheus:
      exportMetrics: true
```

Additionally, you must create a ServiceMonitor. For example:

```
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
```

To query Portworx metrics in the RKE2 Monitoring console, navigate to the Metrics page and enter any Portworx metric, all of which start with '`px_`'. For example, to monitor CPU usage in the Portworx cluster, use the metric '`px_cluster_cpu_percent`'.



### Prometheus Information Collected

A comprehensive list of generated Portworx metrics is documented in the Portworx Metrics Reference article.

### AlertManager Deployment

AlertManager (a component of Prometheus) deployed alongside, processes metrics and sends alerts when specified conditions are met. Portworx provides an initial set of rules to capture common metrics-based alerts, which can be viewed in this documentation page.

You can also configure user-defined alerts in RKE2 to receive standard Prometheus alerts provided by Portworx.

**NOTE**: Active (any fired) alerts are displayed under the "Alerts" tab. To view details of Portworx Prometheus rules, run the following command: `kubectl -n portworx get prometheusrules portworx -o yaml`

Portworx includes its own internal alerts based on cluster, node, drive, or volume state changes and conditions. A complete list of alerts is available here, and they are also accessible via Prometheus metrics.

### Grafana Dashboards

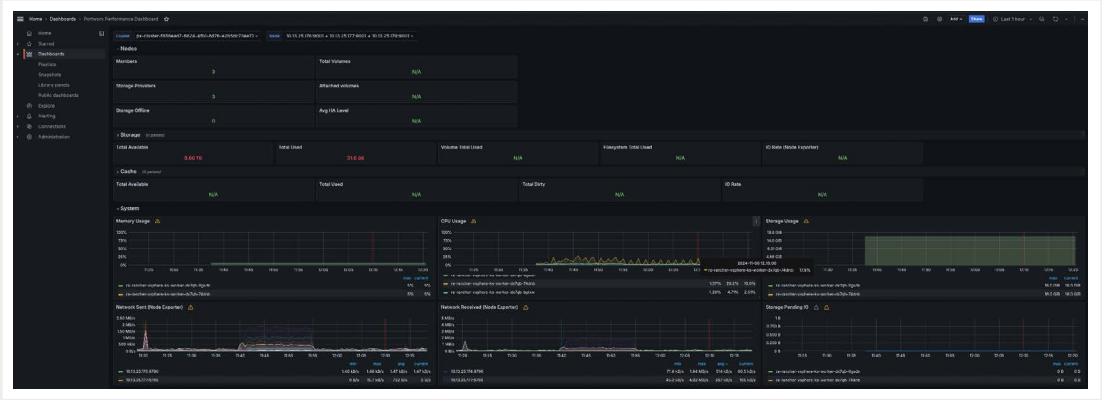
Portworx provides five preconfigured Grafana dashboards to monitor cluster status and performance. Portworx recommends using the latest GA version of Grafana, with a minimum supported version of 8.5.x.

To deploy these dashboards in your Grafana instance on RKE2:

1. Use the JSON files listed in this Portworx documentation article Configure the monitoring solution.
2. Import the dashboards into your RKE2 Grafana instance.
3. If Grafana is not already installed on your RKE2 cluster, refer to the Customizing Grafana Dashboards | RKE2 article for installation and configuration instructions.

Available Portworx Grafana dashboards:

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



## Installation Methods and Tooling

Before installing Portworx on RKE2, complete the prerequisite steps in the Design Considerations section.

Installation steps summary:

- Follow the installation guide for Portworx on RKE2.
- Once you complete the installation wizard, a zip file containing two YAML files will be generated
- One file to install the Portworx Operator
- One to create the StorageCluster object, which installs Portworx.
- Apply the YAML file generated for the Portworx operator installation.
- Before deploying Portworx, create the Kubernetes secret named `px-vsphere-secret` in the “portworx” namespace. Follow the steps outlined in this section.

Portworx dynamically creates and manages vSphere disks using the VMware API. The `px-vsphere-secret` should include the vSphere API username and password credentials. Ensure that the supplied credentials meet the minimum requirements specified in the Portworx documentation (step 1 of this page).

- Use the YAML file generated by the Portworx Central installation wizard to create the `StorageCluster` object.

Portworx recommends using its StorageCluster spec generator to create the initial StorageCluster spec template. Once generated, you can incorporate this template into existing CI/CD pipelines, as needed.

To align with best practices, the initial StorageCluster YAML generated by the installation-wizard should be modified as follows:

- `cloudStorage.deviceSpecs:`

contains six disks in the storage pool (as discussed in the Reference architecture high level design section), all set to provision-type thin.

- `cloudStorage.journalDeviceSpec`

create a 3GB journal device, also set to thin, per best practices.

- `security: enabled`

Enable RBAC authorization for Portworx volumes.

- `secretsProvider: vault`

Set to `vault` for encryption and security.

- **vSphere credentials:**

Portworx will use the px-vsphere-secret to access vSphere APi.

- **vSphere configuration:**

Define details for vCenter URL, port, datastore prefix, and any additional details. If the vsphere certificate signing-authority (CA) is not available on the nodes, then set the `VSPHERE_INSECURE` environment variable to skip the verification.

Below is an example of a StorageCluster YAML created using the Portworx spec generator and updated according to the guidelines provided in this document. *Some annotations have been omitted for brevity.*

```
apiVersion: core.libopenstorage.org/v1
metadata:
  name: px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71
  namespace: portworx
  annotations:
    portworx.io/install-source: “https://install.portworx.
com/?operator=true&mc=false&kbver=1.30.5%2Brke2r1&b=true&iop=6&vsp=true&vc=dosavc.pwx.dev.
purestorage.com&vcp=443&ds=PXDS01&s=%22type%3Dthin%2Csize%3D150%22&j=auto&ce=vsphere&c=
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71&stork=true&csi=true&tel=true&st=k8s”
spec:
  image: portworx/oci-monitor:3.2.0
  imagePullPolicy: Always
  security:
    enabled: true
  kvdb:
    internal: true
  cloudStorage:
    deviceSpecs:
    - type=thin,size=150
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
    - type=thin,size=500
    journalDeviceSpec: type=thin,size=3
secretsProvider: k8s
  stork:
    enabled: true
    args:
      webhook-controller: “true”
  autopilot:
    enabled: true
  runtimeOptions:
    default-io-profile: “6”
  csi:
 enabled: true
  monitoring:
    telemetry:
      enabled: true
    prometheus:
      exportMetrics: true
    env:
  - name: VSPHERE_INSECURE
    value: “”“true””
  - name: VSPHERE_USER
    valueFrom:
      secretKeyRef:
        name: px-vsphere-secret
        key: VSPHERE_USER
```

```
- name: VSPHERE_PASSWORD
  valueFrom:
    secretKeyRef:
      name: px-vsphere-secret
      key: VSPHERE_PASSWORD
- name: VSPHERE_VCENTER
  value: """<vcenter_endpoint>""
- name: VSPHERE_VCENTER_PORT
  value: """443""
- name: VSPHERE_DATASTORE_PREFIX
  value: """<vcenter_datastore_prefix>""
- name: VSPHERE_INSTALL_MODE
  value: """shared""
```

## Operational Considerations

### Post-installation Validation

After deploying Portworx, perform the following steps to ensure that all components are functioning as expected:

- Ensure all Portworx pods are running.
- Confirm that the Portworx cluster is healthy and operational.
- Validate the status of the Portworx internal KVDB.
- Review the status of storage pools for the Portworx cluster provisioning.

To ensure your Portworx installation is successful, refer to the detailed command instructions in the Verify your Portworx Installation article.

Once the Portworx installation is successful, proceed to create your first PVC. For step-by-step instructions, refer to the Create your First PVC article in the Portworx documentation.

### Scaling Portworx

There are various reasons to scale Portworx in your environment, and different methods can be employed to scale your deployment.

#### Adding Storage

Expanding storage on existing Portworx nodes, also known as **vertical scaling**, increases the cluster's capacity. Portworx recommends using the Portworx Autopilot feature to accomplish this task.

You can create Autopilot rules to automatically increase the size of Portworx storage pools. The following types of Autopilot rules help on managing storage pools:

- Expand every Portworx storage pool in your cluster.
- Expand Portworx storage pools.

As Rancher has inbuilt prometheus, for Autopilot to pull the metrics from Rancher one

```
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"}]'
```

### Adding Storage Nodes

In certain scenarios, you may need to add more storage nodes to your cluster. This is also known as **horizontal scaling out** the cluster.

Portworx recommends adding a new node in your cluster when any existing node consistently reaches 80% IOPS or 80% CPU utilization. Additionally, monitoring the latency of storage pools is important, as high latency can also indicate the need for a new node.

To scale out a cluster, add more storage nodes. Portworx will automatically deploy any newly added nodes, integrating them seamlessly into the cluster.

### Adding Compute (Storageless) Nodes

As mentioned in the Reference architecture high level design section, if you plan to run stateful applications on compute nodes (nodes without storage), Portworx recommends creating a separate machine group in your RKE2 cluster, and automatically adding the label portworx.io/node-type: `storageless`.

Storageless nodes can be created or removed as needed without impacting the overall health of the Portworx cluster.

Portworx will automatically remove a storageless node from the cluster 20 minutes after its corresponding VM is deleted.

## Backup and Recovery

For backup and recovery of your cluster, Portworx recommends PX-Backup, which is a complete Kubernetes backup solution fully integrated with Portworx.

PX-Backup is Kubernetes-aware, meaning it understands Kubernetes resources such as statefulsets, secrets, configmaps, PVCs, and Portworx volumes, enabling granular backups and restores, as needed.

To optimize backups, Portworx recommends creating separate backup schedule policies for each namespace in your cluster and space out these schedules throughout the day. This minimizes the chances of backups interfering with regular I/O loads in the cluster.

For more information, refer to the PX-Backup documentation.

## Upgrading Portworx

### Pre-upgrade Checks

Before upgrading Portworx, it's important to verify the health of the current deployment:

- Ensure all pods in the `portworx` namespace are in a running state:

```
> kubectl -n portworx get pods
NAME                                                READY   STATUS    RESTARTS   AGE
autopilot-67664dbf87-fxxkh                          1/1     Running   0          87m
portworx-api-bk6p2                                  2/2     Running   0          36m
portworx-api-nn657                                  2/2     Running   0          36m
portworx-api-px78w                                  2/2     Running   0          36m
portworx-kvdb-cw8pc                                 1/1     Running   0          84m
portworx-kvdb-mf8gt                                 1/1     Running   0          84m
portworx-kvdb-zklhb                                 1/1     Running   0          84m
portworx-operator-755f7cf75c-452g9                  1/1     Running   0          37m
portworx-pvc-controller-759968569d-79qxv            1/1     Running   0          87m
portworx-pvc-controller-759968569d-fptkm            1/1     Running   0          87m
portworx-pvc-controller-759968569d-xt2hd            1/1     Running   0          87m
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71-2zxvx   1/1     Running   0          87m
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71-fc2jm   1/1     Running   0          87m
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71-kdv98   1/1     Running   0          87m
px-csi-ext-79c84ff97b-45t4x                         4/4     Running   0          36m
px-csi-ext-79c84ff97b-fr2gj                         4/4     Running   0          36m
px-csi-ext-79c84ff97b-wrbgd                         4/4     Running   0          36m
px-telemetry-phonehome-hqn2g                        2/2     Running   0          84m
px-telemetry-phonehome-lgwmq                        2/2     Running   0          84m
px-telemetry-phonehome-s25p8                        2/2     Running   0          84m
px-telemetry-registration-7fc9d684bc-7818f          2/2     Running   0          84m
stork-6b86b584bd-jtxhg                              1/1     Running   0          87m
stork-6b86b584bd-nnn98                              1/1     Running   0          87m
stork-6b86b584bd-rklm5                              1/1     Running   0          87m
stork-scheduler-656b6486bd-dsv2b                    1/1     Running   0          87m
stork-scheduler-656b6486bd-kxw2k                    1/1     Running   0          87m
stork-scheduler-656b6486bd-thhsf                    1/1     Running   0          87m
```

Ensure all pods are in a **1/1** state, meaning they are running and ready. If any pod is not in this state, fix the pod(s) before proceeding with the upgrade. Refer to the Troubleshooting section in this document or contact Portworx support for assistance.

- Ensure all RKE2 nodes are in a Ready state:

```
>kubectl get no
NAME                                      STATUS   ROLES                       AGE     VERSION
ra-RKE2-vsphere-k8s-master-pwcgn-56fsw    Ready    control-plane,etcd,master   11m     v1.30.5+rke2r1
ra-RKE2-vsphere-k8s-master-pwcgn-8sbz8    Ready    control-plane,etcd,master   11m     v1.30.5+rke2r1
ra-RKE2-vsphere-k8s-master-pwcgn-xnw7c    Ready    control-plane,etcd,master   15m     v1.30.5+rke2r1
ra-RKE2-vsphere-ks-worker-dx7qb-74dnb     Ready    worker                      7m58s   v1.30.5+rke2r1
ra-RKE2-vsphere-ks-worker-dx7qb-8gsdn     Ready    worker                      9m47s   v1.30.5+rke2r1
ra-RKE2-vsphere-ks-worker-dx7qb-bplxw     Ready    worker                      9m41s   v1.30.5+rke2r1
```

Ensure that all nodes are in `Ready` state before starting the Portworx upgrade. If any node is not in the Ready state, resolve the issue before proceeding. For assistance, refer to the RKE2 documentation or contact SUSE support.

- Ensure all Portworx nodes are up and running, and in the `Ready` state.
- You can then run the following commands to check Portworx status. There is no need to pass the `ADMIN_TOKEN`, if security is not enabled.

```
> ADMIN_TOKEN=$(kubectl -n portworx get secret px-admin-token \
      --template='{{index .data "auth-token" | base64decode}}')
> PX_POD=$(kubectl get pods -l name=portworx -n portworx \
      -o jsonpath='{.items[0].metadata.name}')

> kubectl -n portworx exec -it $PX_POD -- /opt/pwx/bin/pxctl context create \
      admin --token=$ADMIN_TOKEN
Context created.
> kubectl -n portworx exec $PX_POD -- /opt/pwx/bin/pxctl status
Status: PX is operational
Telemetry: Disabled or Unhealthy
Metering: Disabled or Unhealthy
License: Trial (expires in 31 days)
Node ID: 3ba0db47-7588-42ce-ae03-48c457a7ff77
        IP: 10.13.25.177
        Local Storage Pool: 1 pool
        POOL    IO_PRIORITY    RAID_LEVEL     USABLE  USED    STATUS  ZONE    REGION
        0       HIGH           raid0          2.9 TiB 16 GiB  Online  default default
        Local Storage Devices: 6 devices
        Device  Path           Media Type             Size           Last-Scan
        0:1     /dev/sdg       STORAGE_MEDIUM_SSD     500 GiB        06 Nov 24 15:35 UTC
        0:2     /dev/sdh       STORAGE_MEDIUM_SSD     500 GiB        06 Nov 24 15:35 UTC
        0:3     /dev/sdi       STORAGE_MEDIUM_SSD     500 GiB        06 Nov 24 15:35 UTC
        0:4     /dev/sdj       STORAGE_MEDIUM_SSD     500 GiB        06 Nov 24 15:35 UTC
        0:5     /dev/sde       STORAGE_MEDIUM_SSD     500 GiB        06 Nov 24 15:35 UTC
        0:6     /dev/sdf       STORAGE_MEDIUM_SSD     500 GiB        06 Nov 24 15:35 UTC
        total                  -                      2.9 TiB
        Cache Devices:
         * No cache devices
        Kvdb Device:
        Device Path    Size
        /dev/sdk       32 GiB
         * Internal kvdb on this node is using this dedicated kvdb device to store its data.
        Journal Device:
        1       /dev/sdd1      STORAGE_MEDIUM_SSD     3.0 GiB
Cluster Summary
        Cluster ID: px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71
        Cluster UUID: 40e8b70c-e3f0-4e66-a82b-1e5c529f7f66
        Scheduler: kubernetes
        Total Nodes: 3 node(s) with storage (3 online)
```

```
        IP              ID                                       SchedulerNodeName
Auth            StorageNode     Used    Capacity        Status  StorageStatus   Version         Kernel
        OS
        10.13.25.179    4bcc5da8-773b-4c5c-8063-cf99a331191d  ra-RKE2-vsphere-ks-worker-dx7qb-
bplxw   Disabled        Yes             16 GiB  2.9 TiB         Online  Up              3.2.0.0-2ded0fe
5.15.0-100-generic      Ubuntu 22.04.3 LTS
        10.13.25.177    3ba0db47-7588-42ce-ae03-48c457a7ff77  ra-RKE2-vsphere-ks-worker-dx7qb-
74dnb   Disabled        Yes             16 GiB  2.9 TiB         Online  Up (This node)  3.2.0.0-2ded0fe
5.15.0-100-generic      Ubuntu 22.04.3 LTS
        10.13.25.176    01e42b1e-1297-4a98-9d01-323659814b03  ra-RKE2-vsphere-ks-worker-dx7qb-
8gsdn   Disabled        Yes             16 GiB  2.9 TiB         Online  Up              3.2.0.0-2ded0fe
5.15.0-100-generic      Ubuntu 22.04.3 LTS
        Warnings:
Global Storage Pool
        Total Used      :  48 GiB
        Total Capacity  :  8.8 TiB
```

Similar to previous steps, ensure all Portworx nodes are online and verify that no errors or warnings are displayed in the output of the above command. If you encounter any errors, refer to the Troubleshooting section in this document or contact Portworx support for assistance.

- Run the following command to ensure all Portworx KVDB instances are in a running state:

```
> kubectl -n portworx exec $PX_POD -- /opt/pwx/bin/pxctl service kvdb members

Kvdb Cluster Members:

ID                                      PEER URLs                               CLIENT URLs
LEADER  HEALTHY DBSIZE

4bcc5da8-773b-4c5c-8063-cf99a331191d  [http://portworx-1.internal.kvdb:9018]
[http://10.13.25.179:9019]     true    true    308 KiB

01e42b1e-1297-4a98-9d01-323659814b03  [http://portworx-2.internal.kvdb:9018]
[http://10.13.25.176:9019]     false   true    316 KiB

3ba0db47-7588-42ce-ae03-48c457a7ff77  [http://portworx-3.internal.kvdb:9018]
[http://10.13.25.177:9019]     false   true    304 KiB
```

The output must meet the following criteria:

- Display three KVDB members.
- One member must be designated as the leader.
- All members must be in a healthy state.

If you encounter any errors, check the Troubleshooting section in this document or contact Portworx support for assistance.

### Operator Upgrade

After completing all the above prerequisites, the first component to upgrade is the Portworx operator.

You should manually upgrade the operator to the latest version using `kubectl`:

1. After completing the upgrade, run the following command to confirm if Portwox operator is in a running state:

```
> kubectl -n portworx get pod -l name=portworx-operator
NAME                                READY   STATUS    RESTARTS   AGE

portworx-operator-755f7cf75c-452g9   1/1     Running   0          39m
```

### Portworx Upgrade

After upgrading the Operator, proceed to upgrading Portworx and all its associated components. Portworx utilizes a rolling upgrade process, upgrading one node at a time, ensuring each node is upgraded before proceeding to the next.

To upgrade Portworx:

1. Edit the `StorageCluster` resource and update the Portworx image.
   For example, to upgrade to release 3.2.1, modify the image entry:

```
image: portworx/oci-monitor:3.2.0
```

2. During this upgrade process, other components such as Autopilot, Stork, and CSI will also be automatically upgraded to versions compatible with the Portworx release.

For detailed upgrade instructions, refer to the Upgrade Portworx using the Operator article in the Portworx documentation.

## Upgrading RKE2

Before upgrading RKE2, ensure the new RKE2 version is compatible with the currently deployed Portworx version. In some cases, you may need to upgrade Portworx before upgrading your RKE2 cluster. Refer to the supported Kubernetes versions for details.

To ensure smooth operation during and after the RKE2 upgrade, follow these best practices:

- If the new RKE2 version introduces a new kernel, then ensure the current Portworx version is compatible with this new kernel version.
- Ensure that the current Portworx version is compatible with the new Kubernetes version in RKE2.
- Make sure the deployed Portworx version is compatible with the new RKE2 version you plan to use.
- Verify that the Portworx cluster is healthy.
- Ensure all three Portworx internal KVDB instances are healthy and in a running state. Portworx internal KVDB has an associated PodDisruptionBudget (PDB) that requires at least two KVDB pods to be running. All three KVDB pods must be running before starting the RKE2 upgrade to avoid node draining issues that could block the upgrade.

If Portworx is not compatible with any of these points, upgrade Portworx to a supported version before upgrading RKE2.

## Logging and Monitoring

All Portworx pods generate logs that can be viewed or retrieved using standard RKE2 commands, such as kubectl logs. If required by Portworx support, you can increase log levels by updating the `StorageCluster` for each component. For example, to enable debug-level logging for the Stork component, add the following to the `StorageCluster` resource:

```
…
  stork:
    args:
      verbose: true
      webhook-controller: "true"
    enabled: true
…
```

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

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

To collect more details for troubleshooting Portworx, you can generate diagnostics by running the following command on a specific node where Portworx is running:

```
> pxctl sv diags -a
```

This will generate a tar.gz file that can be shared with Portworx support for issue investigation. If Telemetry is enabled, the diagnostics file is automatically uploaded to Pure1®.

To enable Telemetry, follow the instructions outlined in the Portworx documentation.

## Application Considerations

All stateful applications consume one or many PVCs provided by Portworx. These PVCs should be created using one of the default StorageClasses provided by the Portworx Operator. By default, all StorageClasses have replication enabled. For custom StorageClasses, Portworx recommends enabling volume replication to ensure data redundancy.

### Application HA

#### Intra-cluster

Portworx ensures data availability through storage replication, allowing an application to access its data from a replica node if the original node becomes unavailable. However, in the event of a node failure hosting an application pod, the application will experience downtime until Kubernetes reschedules the pod to another healthy node within the cluster. Thus, while storage remains accessible, the application itself may be temporarily offline. Depending on your application's requirements, you may choose one of the following strategies:

1. **HA mode**: For applications that support HA mode and require zero downtime, run multiple replicas of the application pods. This configuration ensures continuous service availability, as other replicas can handle traffic if anode hosting one pod fails.
2. **Non-HA mode**: For applications that can withstand temporary downtime during the Kubernetes failover process or do not support HA mode, deploying a single replica of the application pod is sufficient.

### Portworx Images

To view the list of Portworx images, visit a URL similar to the following:

https://install.portworx.com/3.2/images?kbver=1.30.5+rke2r1

The example above displays images for Portworx version 3.2.x compatible with the kubernetes version 1.30.5+rke2r1.

You can also retrieve the image list using the `curl` command, for example:

```
> curl -s 'https://install.portworx.com/3.2/images?kbver=1.30.5+rke2r1'
docker.io/portworx/px-enterprise:3.2.0
docker.io/portworx/oci-monitor:3.2.0
docker.io/openstorage/stork:24.3.2
docker.io/openstorage/cmdexecutor:24.3.2
docker.io/portworx/autopilot:1.3.15
docker.io/purestorage/ccm-go:1.2.2
docker.io/purestorage/realtime-metrics:1.0.29
docker.io/purestorage/telemetry-envoy:1.1.16
docker.io/purestorage/log-upload:px-1.1.29
docker.io/portworx/px-operator:24.1.3
registry.k8s.io/pause:3.1
registry.k8s.io/kube-controller-manager-amd64:v1.30.5
registry.k8s.io/kube-scheduler-amd64:v1.30.5
docker.io/portworx/portworx-dynamic-plugin:1.1.1
docker.io/nginxinc/nginx-unprivileged:1.25
registry.k8s.io/kube-scheduler-amd64:v1.21.4
registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.12.0
registry.k8s.io/sig-storage/csi-provisioner:v3.6.1
registry.k8s.io/sig-storage/csi-resizer:v1.12.0
registry.k8s.io/sig-storage/csi-snapshotter:v8.1.0
registry.k8s.io/sig-storage/snapshot-controller:v8.1.0
quay.io/prometheus/prometheus:v2.54.1
quay.io/prometheus-operator/prometheus-operator:v0.75.0
quay.io/prometheus-operator/prometheus-config-reloader:v0.75.0
quay.io/prometheus/alertmanager:v0.27.0%
# the following images may also be needed
docker.io/openstorage/csi-attacher:v1.2.1-1
```

For more information about the list of images and instructions on uploading them to a local repository, refer to the Install Portworx on bare metal air-gapped Kubernetes cluster article in the Portworx documentation.

## Monitoring During the Installation

After deploying Portworx, you can monitor the installation progress by checking status of the pods:

```
> kubectl -n portworx get pods
NAME                                                READY   STATUS    RESTARTS   AGE
autopilot-67664dbf87-fxxkh                          1/1     Running   0          92m
portworx-api-bk6p2                                  2/2     Running   0          42m
portworx-api-nn657                                  2/2     Running   0          42m
portworx-api-px78w                                  2/2     Running   0          42m
portworx-kvdb-cw8pc                                 1/1     Running   0          90m
portworx-kvdb-mf8gt                                 1/1     Running   0          89m
portworx-kvdb-zklhb                                 1/1     Running   0          89m
portworx-operator-755f7cf75c-452g9                  1/1     Running   0          42m
portworx-pvc-controller-759968569d-79qxv            1/1     Running   0          92m
portworx-pvc-controller-759968569d-fptkm            1/1     Running   0          92m
portworx-pvc-controller-759968569d-xt2hd            1/1     Running   0          92m
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71-2zxvx   1/1     Running   0          92m
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71-fc2jm   1/1     Running   0          92m
px-cluster-f858ead7-6624-4fb1-8d76-e2b5dc73ee71-kdv98   1/1     Running   0          92m
px-csi-ext-79c84ff97b-45t4x                         4/4     Running   0          42m
px-csi-ext-79c84ff97b-fr2gj                         4/4     Running   0          42m
px-csi-ext-79c84ff97b-wrbgd                         4/4     Running   0          42m
px-telemetry-phonehome-hqn2g                        2/2     Running   0          89m
px-telemetry-phonehome-lgwmq                        2/2     Running   0          89m
px-telemetry-phonehome-s25p8                        2/2     Running   0          89m
px-telemetry-registration-7fc9d684bc-7818f          2/2     Running   0          89m
stork-6b86b584bd-jtxhg                              1/1     Running   0          92m
stork-6b86b584bd-nnn98                              1/1     Running   0          92m
stork-6b86b584bd-rklm5                              1/1     Running   0          92m
stork-scheduler-656b6486bd-dsv2b                    1/1     Running   0          92m
stork-scheduler-656b6486bd-kxw2k                    1/1     Running   0          92m
stork-scheduler-656b6486bd-thhsf                    1/1     Running   0          92m
```

Another method to monitor the installation progress is by checking the status of the StorageNodes resources:

```
> kubectl -n portworx get storagenodes
NAME                                        ID                                     STATUS   VERSION
AGE

ra-RKE2-vsphere-ks-worker-dx7qb-74dnb   3ba0db47-7588-42ce-ae03-48c457a7ff77   Online   3.2.0.0-2ded0fe
92m

ra-RKE2-vsphere-ks-worker-dx7qb-8gsdn   01e42b1e-1297-4a98-9d01-323659814b03   Online   3.2.0.0-2ded0fe
92m

ra-RKE2-vsphere-ks-worker-dx7qb-bplxw   4bcc5da8-773b-4c5c-8063-cf99a331191d   Online   3.2.0.0-2ded0fe
92m
```

At the end of the deployment process, when StorageNodes transition from **Initializing to Online**, all pods should be running and in the Ready state.IIf any problems arise, refer to the troubleshooting section below.

## Validating Post-installation: Checking Cluster Operators, Cluster Version and Nodes

After deploying Portworx, verify the installation by checking the cluster operators, cluster version, and node status. Follow the steps provided in this link to verify the installation and create your first PVC using a Portworx storage class.

## Troubleshooting Commands

To troubleshoot installation issues, check the logs of any failing pods for errors. For example, to troubleshoot a specific Portworx pod in a cluster named `px-cluster-refarch-2ca0d...149a`, run the following command:

```
> kubectl -n portworx logs px-cluster-refarch-2ca0xxx4n
```

**Other Useful Commands:**

- Describe a pod to check the Events section for errors. The following example shows a failed image pull for the stork component:

```
> kubectl -n portworx describe pod stork-776979dc7b-tmq4d
Events:

  Type    Reason     Age   From               Message

  ----    ------     ----  ----               -------

  Normal  Scheduled  55s   default-scheduler  Successfully assigned portworx/stork-776979dc7b-tmq4d to
pool-0-556f55b84d-6rkv7

  Normal  Pulling    55s   kubelet            Pulling image "docker.io/openstorage/stork:24.3.2"

  Normal  Pulled     25s   kubelet            Successfully pulled image "docker.io/openstorage/
stork:24.3.2" in 29.848s (29.848s including waiting)

  Normal  Created    25s   kubelet            Created container stork
```

- Retrieve logs from the Portworx operator pod:

```
> kubectl -n portworx logs -l name=portworx-operator --tail=9999

time="06-11-2024 17:07:10" level=info msg="Using version extracted from image name: 3.2.0" file="util.
go:591"

time="06-11-2024 17:07:10" level=info msg="shouldUseQuorumMember for node ra-RKE2-vsphere-ks-worker-
dx7qb-8gsdn: true , isQuorumMember? true" file="storagenode.go:566"

time="06-11-2024 17:07:10" level=info msg="Reconciling StorageNode" file="storagenode.go:214"
storagenode=ra-RKE2-vsphere-ks-worker-dx7qb-bplxw

time="06-11-2024 17:07:10" level=info msg="Using version extracted from image name: 3.2.0" file="util.
go:591"

time="06-11-2024 17:07:10" level=info msg="shouldUseQuorumMember for node time="06-11-2024 17:07:15"
level=info msg="Using version extracted from image name: 3.2.0" file="util.go:591"

time="06-11-2024 17:07:15" level=info msg="Using version extracted from image name: 3.2.0" file="util.
go:591"

time="06-11-2024 17:07:15" level=info msg="Using version extracted from image name: 3.2.0" file="util.
go:591"
```

## Conclusion

This document can be used as reference architecture to deploy Portworx on Rancher Kubernetes Engine 2.

## Additional Resources

Refer to the troubleshooting section in the Portworx documentation for more information.

For further assistance, refer to the Portworx support contact details.

## Appendix A

### How a Customer Setup and Use of RKE2 Differs

The following lists ways that the above document does not match one of our primary FSI-based customers:

**Storage**

- **Cloud-drives in a pool**
  - The customer started with one cloud-drive. It has been recommended they update all their pools to be backed by the maximum six (6) cloud-drives. This has been an ongoing endeavor and to my knowledge is not yet complete.
  - Reason: This was an initial selection based on FA-CSI deployment prior to involving Portworx architects.
- **Datastores**
  - The customer has only one FlashArray, shared between multiple Portworx clusters (as well as other apps), and currently they have at least four or more datastores dedicated for each cluster. The datastores have a fixed (max) size of 16TiB, per their storage-team limitations.
  - Reason: This is a limitation of the environment, the customer has not considered having more than one FlashArray per DC.
- **High availability**
  - Not set up completely as their workload clusters exist and are connected via a single vCenter (e.g., it's shared by multiple Portworx clusters in a given data center, which inevitably becomes a choke point leading to low vCenter api query performance).
  - As all nodes in the cluster are in the same data center, topology awareness cannot be implemented unless rack and hypervisor-based locality is utilized.
  - Reason: The customer wished to localize clusters on a per-site basis.
- **Scaling**
  - The customer manually scales clusters during maintenance windows and does not utilize auto-scaling.
  - Reason: customer decision
- **Resources**
  - I am not aware of any resource and I/O resource measurements being performed to determine what adequate I/O behavior would be made available to Portworx, and thus end-user applications.
  - Reason: The customer's initial deployment was around the CSI "free" version of Portworx and it was not considered as things scaled several years ago, prior to the development of these recommendations.
- **Performance**
  - The customer **does not** utilize journal devices for Portworx use anywhere in their environment.
  - Reason: The recommendation on the necessity of journal devices was not yet in place or available at the time of cluster formation.
- **Security**
  - The customer does not have PX-Security enabled and I don't believe they're using encrypted volumes either.
  - Reason: The customer is aware of the limitation and has been researching implementing this.
- **Observability**
  - The customer utilizes external third-party (datadog) tools for ingesting metrics and performance data generated by Portworx, and has their own dashboards and alerting systems configured.
  - Only recently have they expressed interest in deploying and utilizing our pre-created Grafana dashboards.
  - Reason: A pre-existing monitoring solution (used by other apps) was chosen to fit into the environment.





PS2725-01-en 02/25