PURESTORAGE®

REFERENCE ARCHITECTURE

Deploying Portworx on Google Distributed Cloud Anthos with vSphere

Accelerate your modernization journey with Google Anthos and Portworx.

Contents

Executive Summary

Modern applications built using containers and orchestrated by Kubernetes still need a layer of persistence. Google Distributed Cloud (GDC) for VMware, also known as Anthos, is an industry-leading hybrid cloud application platform powered by Kubernetes. It brings together tested and trusted services to reduce the friction of developing, modernizing, deploying, running, and managing applications and delivers a consistent experience across public cloud, on-premises, hybrid-cloud, or edge architecture.

To run stateful applications on GDC, organizations need a robust data services platform like Portworx® from Pure Storage®. Portworx provides features like replication and high availability, security and encryption, capacity management, disaster recovery, and data protection to Google Anthos deployments. Instead of spending resources architecting and managing a custom Kubernetes storage layer, organizations can accelerate their modernization journeys by adopting a solution like Anthos with Portworx.

About This Document

This Portworx reference architecture contains a validated architecture and design model to deploy Portworx on Anthos running on vSphere. It is intended for Kubernetes administrators and cloud architects who are familiar with Portworx.

The audience must be familiar with Anthos concepts, and familiarity with how Kubernetes is used is helpful as well.

The document has three main technical areas as described below:

Planning and Architecture Overview

This section presents the high-level architecture overview on how Portworx is deployed on Anthos. It discusses the requirements related to Anthos for storage and storageless nodes and also vSphere Datastores configuration recommendations.

Design Considerations

This section provides more detailed requirements and recommendations that must be considered during the design phase. It covers the following areas:

- Anthos requirements
- Networking
- Capacity planning
- High availability
- Resource and performance considerations
- Security and monitoring

At the end of the section, a recommended Portworx installation template is presented.

Operations Considerations

This section covers “day 2” best practices after Portworx is deployed. It discusses the following topics:

- How to validate the Portworx installation
- Observability on the Portworx deployment
- How to scale Portworx
- Backup and recovery techniques
- Best practices on how to upgrade Portworx and Anthos
- How to check Portworx logs
- Application considerations

Value Proposition

Benefits of Portworx

Traditional storage solutions provide a simple Container Storage Interface (CSI) driver connector to handle stateful applications in Kubernetes environments. A CSI connector has several limitations and doesn't provide a robust solution for high availability.

Unlike these solutions with CSI drivers, Portworx accelerates time to revenue, delivers data resiliency, and agility at enterprise scale for Kubernetes storage and databases—leading to a boost in platform engineering productivity.

Portworx storage services provide scalability, industry-leading availability, and self-service access to storage for Kubernetes environments. Integrated storage management includes rule-based automation, thin-provisioning allocation and flexibility for multi-cloud, hybrid-cloud and on-premises environments.

Benefits of Running Portworx on Anthos with vSphere

As part of digital transformation efforts, organizations are modernizing their applications and infrastructure by adopting containers and Kubernetes for their applications and leveraging a solution like Google Anthos for their infrastructure. Anthos allows organizations to take advantage of full-stack automated operations, a consistent experience across all environments, and self-service provisioning for developers that lets teams work together to move ideas from development to production.

Portworx adds a robust, secure, highly available, and scalable data management layer to Google Anthos so applications can consume storage in an easy way.

Target Use Cases

This document provides guidelines and best practices to deploy Portworx on Anthos, specifically on workload clusters, since typically there would not be any stateful workloads on the Anthos admin cluster.

After deploying Portworx using the guidelines in this document, Anthos users can deploy any type of stateful application in Portworx. The scope of this document does not include any specific recommendations for particular applications, but it is meant to be a stable deployment suitable for any application that requires storage.

Planning and Architecture Overview

Reference Architecture High Level Design

The diagram below shows a high level design of the Portworx reference architecture deployed on Anthos running on vSphere.

By implementing this design, a platform engineering team can automate the provisioning of a well defined architecture following best practices that includes high availability, operations management, observability, business continuity, performance and security.

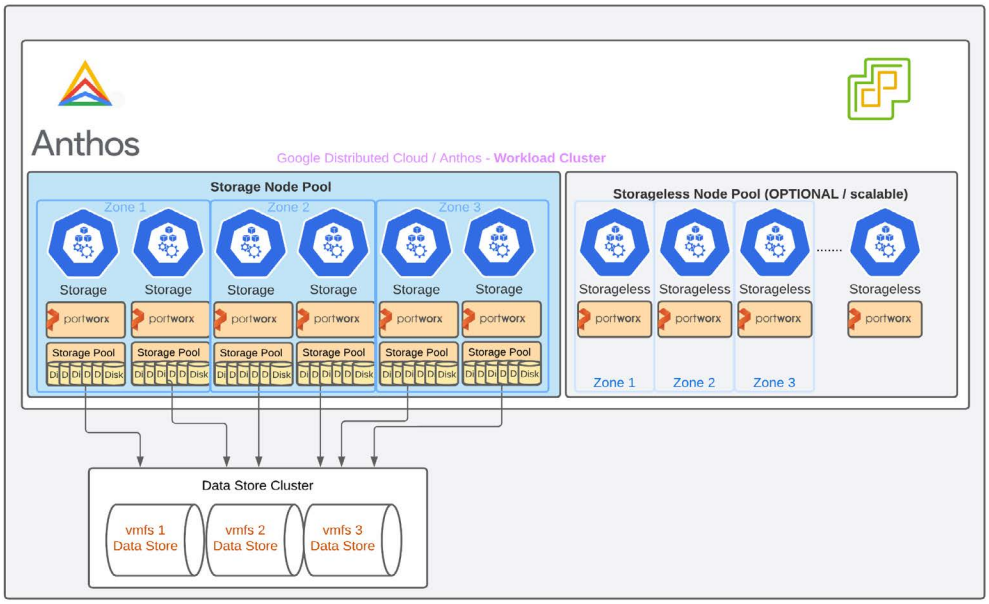

FIGURE 1 High level architecture overview

Portworx requires a minimum of three storage nodes in the cluster, but for a production environment this reference architecture recommends an initial cluster size with six storage nodes.

Storage nodes in Portworx handle both storage and compute tasks, so the term “storage node” does not imply exclusive storage functionality. Think of storage nodes as compute nodes with storage capabilities, while storageless nodes perform only compute tasks and access volumes over the network through storage nodes.

Although a Portworx cluster with three storage nodes may work well in some environments, there are some advantages on using six storage nodes in the initial deployment:

- **Cluster capacity:** If a cluster with three storage nodes loses one, it loses one-third of its capacity, increasing the load on the remaining two nodes until the lost node recovers. In a six-node cluster, losing one node affects only one-sixth of its capacity, allowing the remaining five nodes to more evenly distribute the load.
- **I/O load distribution:** A Portworx cluster with three storage nodes may face more frequent I/O latencies during peak times. Conversely, a six-node cluster can distribute I/O requests more effectively, reducing the risk of I/O latencies. Based on the recommendations above, it is important to prepare Anthos and vSphere before deploying Portworx. Consider the following points:

Storage Nodes

A group of six storage nodes is recommended for a production environment, since less than six storage nodes is suboptimal, unless the workload is minimal or only for testing. Note that:

- The cluster should have six or more nodes that have access to storage.
- If the Anthos cluster spans multiple availability zones, equally distribute these storage nodes amongst them. If you deploy across multiple availability zones, at least three zones are recommended.
- At install time, Portworx will set the configuration parameter "MaxStorageNodesPerZone" to put an upper limit on the maximum number of storage nodes it creates. Based on how many storage nodes you need when scaling the cluster, you will need to update this field in the StorageCluster spec.
- You should not use cluster autoscaling (you can manually scale horizontally, i.e. add more nodes if needed to increase storage capacity, but cannot shrink).
- You should use the label portworx.io/node-type:storage on all nodes, so Portworx can automatically provision storage for new nodes.
- You need to run hyperconverged applications on these nodes, i.e. applications that need to achieve high performance levels.
- Each node should have at least 8 CPU cores and 16GB RAM (check the Resource considerations section for more details on CPU and memory requirements) along with what your end workloads may require.

Storageless Nodes (Optional)

An optional group of nodes for storageless nodes. Note that:

- There is no minimum number of nodes.
- You can have an associated cluster autoscaler.
- You should use the label "portworx.io/node-type:storageless" on all nodes, so Portworx can automatically add nodes as storageless
- It can be used in a very dynamic compute environment, where the number of compute nodes can elastically increase or decrease based on workload demand.
- Applications running on these nodes will access storage available in the storage nodes via the cluster network.
- It is crucial to monitor performance closely in this scenario, since increasing the number of storageless nodes can overload the storage nodes and affect application performance.
- Each node must have a minimum of 8 CPU cores and 16GB RAM (check the Resource considerations section for more details on CPU and memory requirements).

vSphere API

Portworx has integrations to communicate with the vSphere API and dynamically manage Portworx-provisioned block-storage automatically. Portworx will require a vCenter service-account to perform these operations.

DataStores

When Anthos is deployed on top of vSphere, along with planning for cluster and node capacity, you must also size the VMFS datastores Portworx will utilize accordingly. The combination of vSphere datastore sizes and Portworx pool sizes must be selected initially such that future growth of pool resizing is easily satisfied. A Portworx pool can be expanded by using two mechanisms: adding a new disk (PX creates a new VMDK) and adding it to the pool, or by resizing all the existing disks (VMDKs) within the pool.

Resize existing disks in a pool rather than adding new ones when increasing pool size. Resizing is quick and immediate as it doesn't require data redistribution, unlike adding new disks, which is an I/O intensive process that takes time proportional to the amount of data needing redistribution.

Here are the guidelines on planning and provisioning vSphere Datastores for Portworx to tailor to the resize disk recommendation:

- Provision multiple and large sized datastores which can be expanded in the future. This reduces the additional work of managing VMFS datastore sizing.
- Having larger datastores avoids limitations on the maximum size of disk Portworx can create on them.
- Choosing fewer datastores of larger sizes is preferred over multiple datastores of smaller sizes. For example, it is better to have 6 datastores of 16TiB than 12 datastores of 8TiB.
- By default Portworx allows for thin-provisioning, but to have a better insight into potentially consumed space, in this reference architecture we elect the explicitly provisioning type of the datastore volumes to be lazyzeroedthick
- Expanding existing datastores is preferred over adding new datastores, to avoid the need for performing a Storage DRS and rebalancing of VMDKs.
- Portworx recommends limiting Datastore usage to a single Portworx cluster. This helps in predicting the right starting datastore size. Do not share datastores across multiple Portworx clusters.

Storage Pools

Portworx recommends starting with a single pool per node with six drives in the pool.

- This can be achieved by having six different cloud drive spec entries in the StorageCluster spec where size and type of all the disks is the same. The example below will create a 3TiB storage pool per node where the pool has six disks.

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

- Dividing capacity across multiple drives in the pool will allow expanding the pool with just a drive resize in future instead of adding disks to the pool.
- 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, chances are that the datastore won't have enough space to resize that single disk to satisfy the pool expansion requirement.
- However when there are multiple disks in a pool, chances are that the datastore has the room for this smaller incremental resize required on the disk.
- The alternate expansion strategy (while it's available) of "Adding a disk" to the pool is a more expensive and time-consuming operation than resizing existing disks as addition leads to data movement.

Design Considerations

Anthos

The minimal version of Anthos required for this reference architecture at the time of this document is GDC v1.28 or newer. For currently supported versions of GDC (aka Anthos), please see the official google documentation.

Networking

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

By default Anthos will have all ports opened among the worker nodes, but in case you have any specific firewall in your networks please ensure that ports in the "East-to-West" section of the relevant Portworx documentation are opened for node to node communication. Additionally, the Telemetry-feature utilizes limited-outbound connectivity, the destinations of which are specified in the "Outbound" section of the same page and need to be reachable from nodes in the cluster.

Storage

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

It covers three aspects of the capacity planning:

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

Initial Cluster Capacity

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

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

Below are two examples on 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 and sizing for Portworx deployment

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

Storage Node Capacity Sizing

Once you have the cluster size, you can calculate the size of each storage node:

Cluster size / Number of storage nodes = node capacity + min(10%, 100 Gib) for pool recovery in case the pool becomes full

Following the example above:

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

TABLE 2 Storage node capacity sizing for Portworx deployment

vSphere Datastore Sizing

Finally, you can plan the number of vSphere Datastores in your environment. One key point to consider is that vSphere limits a Datastore capacity to 64TB, and Portworx recommends multiple datastores rather than a single large one.

Creating multiple datastores gives more room to expand these datastores in the future when the capacity needs increase.

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

In the first example above you can provide three datastores, each one initially with a 2.2TiB size, while in the second example you can provide three datastores with 4.6TiB size.

Implementing High Availability

To implement a highly available cluster, you can take advantage of Portworx topology awareness feature.

Additionally, Portworx can automatically identify topology node-labels, specifically topology.kubernetes.io/region and topology.kubernetes.io/zone

In the cloud, there are regions and zone values set by default, however in VMware (and on-prem) environments, these will have to be manually configured. Ensure that your Anthos nodes have the necessary topology labels by providing them in the node-pool configuration and that they are the same across nodes residing in the same Anthos node pool. The recommendation for the six storage nodes is to have two storage nodes in each zone. A zone in this case must be an isolated entity, that way if it fails, it does not affect other zones.

This scenario allows Portworx to automatically place volume replicas in separate failure-domains and if one domain becomes unavailable the application can run in a different location where another replica of the same volume resides.

The architecture diagram in section 3.1 illustrates how the different failure-domains are set up.

Deployment Model

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

As described in the reference architecture high level design section, we recommend two separate Anthos node groups; one for storage nodes, and one for the storageless nodes.

The node group for the storage nodes provides a static set of nodes for hyper converged applications. This node group can be manually scaled out if more storage nodes are needed, but cannot be scaled back, i.e. you cannot automatically remove storage nodes from the Portworx cluster (only manually).

On the other hand, with Anthos cluster autoscaler, it is easier to automate and manage Portworx storageless nodes in this deployment model. The optional node group for the storageless nodes can have an associated cluster autoscaler to automatically scale those nodes based on the cluster resources consumption. For details on how to configure Anthos for auto-scaling, please refer to the Google documentation.

Resource Considerations

When designing the cluster for a specific workload, make a note of the expected number of volumes in the cluster to be used at any given time, their average throughput or IOPS requirement, their HA level, and if snapshots are going to be used or not.

In the following example, we are utilizing averages, which will be sufficient for the majority of workloads, however workloads peaks will not be handled as efficiently due to these sizing decisions, but represent the best trade-off in terms of performance vs unused idle capacity resulting from sizing towards peaks (presuming they are less common).

Sum up these IOPS per volume, multiplied by their HA level, and add a snapshot and fragmentation overhead of 1.4. This will give you the approximate backend IOPS requirement, which should be less than the sum of the IOPS per pool across all the nodes in the cluster.

For vSphere, each pool could be aggregated out of many VMDKs across datastores, the total IOPS should be a sum of the IOPS per VMDK.

The table below shows two examples using sample numbers for illustrative purposes:

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

TABLE 3 Example IOPS Calculations for cluster sizing based on volume count, replication factor, and overhead

The minimum requirements for each Portworx node are 8 CPUs and 16 GB RAM, but depending on the expected I/O load on the system, we recommend providing enough RAM and CPU resources in a high performance setting. For high performance systems, we recommend at least 32 cores and 32 GB ram where Portworx is running along with the applications on the same virtual machine.

Performance Considerations

Portworx is set up with a default configuration to optimize cluster performance, but certain situations may benefit from additional parameters. This section outlines scenarios where adding these parameters can boost the overall performance of the Portworx cluster.

Journal Device

A dedicated journal device is recommended in all cases and newer versions of Portworx allow improved performance in certain scenarios when a journal device is available. To enable it, you can add the following in the StorageCluster yaml:

```
kind: StorageClusterspec:
...
 cloudStorage:
  ...
  journalDeviceSpec: type=lazyzeroedthick,size=3
...
```

When using a journal device you can also consider using the auto_journal I/O profile which can improve performance for volumes with replica 1. See this article for more details and use cases for this profile.

StorageClass Auto IO Profile

Portworx can auto-detect the optimal IO profile for an application when the StorageClass has the io_profile set to auto. In this case the replication factor must be set to 2 or 3. Here is an example of a custom StorageClass (in addition to what the operator already creates) using the following 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 for Leveraging More Resources

Portworx consumes minimum CPU and memory from the VMs, by default as outlined previously, using only up to 8 vCPUs and up to 16 GB of memory. However, if your nodes have a high number of CPUs (greater than 32 vCPU) and large memory (greater than 64GB), it is recommended to use the rt_opts_conf_high runtime option to allow Portworx to use more CPU threads and memory, which allows for further improving performance.

Below is a snippet of the StorageCluster yaml with this parameter:

```
kind: StorageCluster
...
spec:
...
  runtimeOptions:
    rt_opts_conf_high: "1"
...
```

'nodiscard' Option

Some applications, like Kafka and Elastic, perform a large number of discard/delete operations. This can affect the overall performance of the Portworx cluster. It is recommended when using the default ext4-based volume, to also consider including the nodiscard parameter.

When using the nodiscard parameter, you should also be sizing the volumes(s) closely to the amount of expected data, instead of overallocating the volume size upfront. Over-provisioning can result in inefficient allocation of blocks (potentially leaving them fragmented), and since volumes are by-default thin-provisioned, resizing in the future (which requires no downtime) is preferable rather than over-sizing volumes.

The example below has a StorageClass definition with this parameter:

```
kind: StorageClass
...
parameters:
 ...
 nodiscard: "true"
```

Additionally, when using the nodiscard option, as data is written/deleted from the filesystem, it won't immediately be removed from the block devices serving those volumes. Therefore, if the extra consumed space is a concern, a filesystem trim operation should run periodically to clear the deleted data from the block device. For instructions on enabling this feature, refer to the Maintain volumes using Filesystem Trim article in the Portworx documentation.

Security

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

Authorization

With authorization, Portworx adds an extra layer of security for the Portworx volumes. Using well known industry standards, Portworx protects the volumes from unauthorized access by adding a RBAC (Role Based Access Control) mechanism. Only authenticated and authorized users can access the volumes. (by default Portworx creates a user token for the 'kubernetes' user when security is enabled).

To enable authorization in Portworx, add the spec.security.enabled:true parameter in the StorageCluster:

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

Once enabled, only kubernetes users will be able to access the Portworx volumes if PVCs are created using StorageClasses 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 StorageClass, to disable 'guest access' see this Portworx documentation link.

Additionally, any future pxctl commands will require an admin token to be used. Details on how to set this up can be found in this Portworx documentation link.

More details and advanced configuration for multi-tenant clusters can be found in this Portworx documentation link.

Encryption

Portworx recommends protecting your volumes with encryption. This can happen at the Portworx volume level, or further down in the stack (such as on a FlashArray).

All PX-encrypted volumes use a passphrase for protection 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 of the public cloud provider secret stores. The complete list of supported secret store management can be found in the Secret Store Management article of the Portworx documentation.

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

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

Then when deploying Portworx, you can set Vault as the secret store provider:

```
kind: StorageCluster
…
 spec:
 …
 secretsProvider: vault
 …
```

After Portworx is successfully deployed, you must define a cluster-wide passphrase for Portworx to encrypt the volumes.

Follow the steps in the Encrypting Kubernetes PVCs with Vault article of the Portworx documentation to create the passphrase and start using encrypted volumes with Portworx. This document recommends using at least the cluster-wide encryption feature provided by Portworx. For environments requiring additional security, such as multi-tenant clusters, refer to the documentation in the link above, which details more advanced per-volume encryption options.

Monitoring

Portworx can be configured to permit general observability by creating several numeric-insights (metrics) into the various aspects of the system (performance, state-changes, resource-usage, etc).

Portworx has support for automatically deploying an open-source monitoring tool to collect and store metrics to help with monitoring purposes. The widely used Prometheus-project is available to be spun up during the initial Portworx deployment (or later if needed) by use of the Portworx Operator. It is enabled using the StorageCluster object. More information is available here.

Prometheus Information Collected

You can find a list of Portworx metrics generated in the Portworx Metrics Reference article of the Portworx documentation.

AlertManager Deployment

AlertManager is a component of Prometheus deployed alongside, that can act on the metrics available and send alerts when certain conditions are met. Portworx provides an initial set of rules to capture common metrics-based alerts. It can be viewed as per this documentation page.

You can additionally set up user-defined alerts in Anthos to receive standard Prometheus alerts provided by Portworx.

Note the following:

- Any fired (active) alerts will be displayed in the “Alerts” tab of the web interface of prometheus.
- You can see details on Portworx Prometheus rules by running the following command:

```
kubectl -n portworx get prometheusrules portworx -o yaml
```

Portworx also has its own internal implementation of other alerts based on Cluster/Node/Drive/Volume state changes and conditions, the full list is here, however they are also made available via the metrics Prometheus makes available.

Grafana Dashboards

Portworx provides five out-of-the-box Grafana dashboards to help monitor its status and performance. To deploy these dashboards in your own Grafana instance on Anthos, follow the steps outlined in the Configure the monitoring solution article of the Portworx documentation. If Grafana is not already installed on your Anthos cluster, refer to the Grafana documentation for installation and configuration instructions.

List of Portworx Grafana resources available:

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

Installation Methods and Tooling

Before installing Portworx on Anthos, complete the prerequisite steps outlined in the Portworx on Anthos section of the Portworx documentation. Here is a summary:

- Follow the steps on the relevant section of the Portworx documentation.
- Once you complete the PX-Central's installation-wizard, a zip file will be generated containing two YAML files, one to install the Portworx Operator, and one to create the StorageCluster object, which will install Portworx itself.
- First Install the Portworx operator using the YAML generated.
- Create the StorageCluster object generated in step 2.

We suggest using its StorageCluster spec generator referenced above, to create the initial StorageCluster spec template. Once generated, you can incorporate this template into an existing CI/CD pipeline as needed.

The initial generated StorageCluster from the installation-wizard needs to be modified in the following ways to follow the best-practices guidelines in this document:

- cloudStorage.deviceSpecs: contains 6 disks in the storage pool as discussed in the Reference architecture high level design section, all of of provision-type lazyzeroedthick
- cloudStorage.journalDeviceSpec: create a 3GB journal device per best practices recommendation, also lazyzeroedthick
- Security: enabled. Enable RBAC authorization for Portworx volumes
- SecretsProvider: vault. Use Vault for encryption and security
- vSphere details: Use the credentials of the required vCenter service-account stored in the px-vsphere-secret, as well as details for the vCenter URL and port, Datastore prefix.

Below is an example StorageCluster YAML definition, created using the Portworx installation wizard in accordance with the best-practices guidelines (these additions/changes are in ***italics***), which are provided in this document.

Note: Some annotations have been omitted for brevity.

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
 name: px-cluster-refarch
 namespace: portworx
spec:
 image: portworx/oci-monitor:3.2.0
 imagePullPolicy: Always
 security:
  enabled: true
 kvdb:
  internal: true
 cloudStorage:
  deviceSpecs:
  - type=lazyzeroedthick,size=150
  - type=lazyzeroedthick,size=150
  - type=lazyzeroedthick,size=150
  - type=lazyzeroedthick,size=150
  - type=lazyzeroedthick,size=150
  - type=lazyzeroedthick,size=150
  journalDeviceSpec: type=lazyzeroedthick,size=3
  provider: vsphere
 secretsProvider: vault
 stork:
  enabled: true
  args:
   webhook-controller: "true"
 autopilot:
  enabled: true
 runtimeOptions:
  default-io-profile: "6"
  rt_opts_conf_high: "1" # use only on high-performance nodes as discussed in section 4.7
 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, you can perform the following steps to ensure that all components are functioning correctly. Verify:

- If all pods are running
- The Portworx cluster status
- The Portworx internal KVDB status
- The Portworx cluster provisioning status (more details on the pools status)

To ensure your Portworx installation is successful, consult the Verify your Portworx Installation article in the Portworx documentation for detailed command instructions.

Once you confirm that Portworx is installed correctly, you can proceed to create your first Persistent Volume Claim (PVC). For step-by-step instructions, visit the Create your First PVC article in the Portworx documentation.

Additionally, you may wish to consider setting up a defragmentation schedule in order to prevent fragmentation of blocks (as discussed in section 4.7 on the topic of nodiscard parameter).

Scaling Portworx

There are several reasons to scale Portworx in your environment. Depending on these reasons, various methods can be employed to effectively scale your deployment.

Adding Storage

Adding more storage on current Portworx nodes is commonly referred to as “vertical scaling up” the cluster. We recommend you use the Portworx Autopilot feature to accomplish this task.

You can create Autopilot rules to automatically increase the size of Portworx storage pools. Autopilot rules help in managing storage pools by expanding every Portworx storage pool in your cluster and expanding individual nodes’ Portworx storage pools.

Adding Storage Nodes

In certain cases, you may need to add more storage nodes to your cluster. This task is commonly referred to as “horizontal scaling out” the cluster.

We recommend adding a new node in your cluster if one of the current nodes reaches a consistent 80% IOPS or 80% CPU utilization. These can be checked in the monitoring system discussed previously (specifically the Grafana dashboards).

Monitoring the latency on the pools is also important and high latency can also indicate a need for a new node.

To scale out a cluster you would modify the node-pool to increase the number of nodes. Portworx will automatically be deployed in the new node that is created.

It is worth considering whether you may also want to redistribute volume replicas onto the new node(s), which can be accomplished using autopilot as well.

While the best results come when leveraging Portworx’s ability to perform hyperconvergence of workloads on storage nodes, there may be additional (non-IO) based reasons you also may wish to consider when deciding to scale out the set of Storage Nodes (like needing more compute resources for those workloads). These considerations are left up to the reader to decide when is appropriate to scale for those other (non-IO based) reasons.

Adding Compute-only (Storageless) Nodes

As mentioned in the Reference architecture high level design section if you plan to run stateful applications in compute nodes, i.e. nodes without storage, Portworx recommends you create a separate node pool in your Anthos cluster and automatically add the label:

```
portworx.io/node-type: storageless
```

Storageless nodes can be created and removed as needed without impacting the overall status of the Portworx cluster. You may want to consider evenly scaling of the storageless nodes across any topology-informed failure-domains.

Portworx will automatically clean up and remove any storageless node from being in the storage cluster's membership after 20 minutes of the VM being unavailable.

Backup and Recovery

Portworx recommends PX-Backup for backup and recovery of your cluster.

PX-Backup is a complete Kubernetes backup solution fully integrated with Portworx, and it is Kubernetes-aware, i.e. understands all Kubernetes resources like statefulsets, secrets, configmap, PVC, etc. and Portworx volumes, so you can have granular backups and restores if needed.

Portworx recommends creating different backup schedule policies for each namespace in your cluster and space out those schedules throughout the day. This minimizes backups to interfere and compete with regular I/O loads in the cluster.

For more details on PX-Backup please check its documentation in this link.

Upgrading Portworx

Pre-upgrade Checks

Before upgrading Portworx, it is important to check if its current deployment is healthy by ensuring all pods in the portworx namespace are running:

```
$ kubectl -n portworx get pods
NAME                                            READY   STATUS    RESTARTS
autopilot-dbb5448c8-5jhpk                       1/1     Running   0
portworx-api-bdh9l                              2/2     Running   0
portworx-api-jkhmw                              2/2     Running   0
portworx-api-mfpvt                              2/2     Running   0
portworx-kvdb-czm9p                             1/1     Running   0
portworx-kvdb-fbpm5                             1/1     Running   0
portworx-kvdb-txggs                             1/1     Running   0
portworx-operator-5868cfb59b-fg6qq              1/1     Running   2
portworx-pvc-controller-57b8ff658b-qmlcd        1/1     Running   1
portworx-pvc-controller-57b8ff658b-z656z        1/1     Running   1
portworx-pvc-controller-57b8ff658b-zcwbp        1/1     Running   1
prometheus-px-prometheus-0                      2/2     Running   0
px-cluster-refarch-2ca0db39-a1a2-47b5-9048-7261zv  1/1     Running   0
px-cluster-refarch-2ca0db39-a1a2-47b5-9048-72gzjs  1/1     Running   0
px-cluster-refarch-2ca0db39-a1a2-47b5-9048-72gs4n  1/1     Running   0
px-csi-ext-749ddcb98d-8rwdw                     3/3     Running   3
px-csi-ext-749ddcb98d-mz5qv                     3/3     Running   1
px-csi-ext-749ddcb98d-wtg85                     3/3     Running   4
px-prometheus-operator-cb976559-6qh7m           1/1     Running   0
px-telemetry-phonehome-5xx2z                    2/2     Running   0
px-telemetry-phonehome-8bdck                    2/2     Running   0
px-telemetry-phonehome-fz2j5                    2/2     Running   0
px-telemetry-registration-7f8485b8b-xkkkf       2/2     Running   0
stork-776979dc7b-fht84                          1/1     Running   1
stork-776979dc7b-l2pgk                          1/1     Running   1
stork-776979dc7b-ldvv9                          1/1     Running   1
stork-scheduler-5bc5576dcc-8rzpz                1/1     Running   1
stork-scheduler-5bc5576dcc-bbjxq                1/1     Running   1
stork-scheduler-5bc5576dcc-nx5vt                1/1     Running   0
```

Ensure all pods are in `Ready` state and all numbers (such as `1/1, 2/2, or 3/3`) have matching numbers, i.e. running and ready. If any pod is not in this state please fix the pod(s) before starting the upgrade. Check the Troubleshooting section in this document or contact Portworx support for further assistance.

Ensure All Anthos Nodes Are Ready

```
$ kubectl get nodes
NAME                     STATUS ROLES                  AGE  VERSION
aleksanthos-uc-vm1       Ready  control-plane,master 31d  v1.28.12-gke.1100
pool-1-75f796558d-c7cxm Ready  <none>                31d  v1.28.12-gke.1100
pool-1-75f796558d-h42mr Ready  <none>                31d  v1.28.12-gke.1100
pool-1-75f796558d-hfsb4 Ready  <none>                31d  v1.28.12-gke.1100
pool-1-75f796558d-jt2h2 Ready  <none>                31d  v1.28.12-gke.1100
pool-1-75f796558d-l2fn2 Ready  <none>                31d  v1.28.12-gke.1100
pool-1-75f796558d-v4crk Ready  <none>                31d  v1.28.12-gke.1100
```

Ensure that all nodes are in Ready state before starting the Portworx upgrade. If any node is not in the 'Ready' state, please address the issue before proceeding. For assistance, check Anthos documentation or contact Google support.

Ensure all Portworx nodes are up and running (in the ***Ready*** state). You can then run the following commands below to check Portworx status:

```
$ 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: Healthy
Metering: Disabled or Unhealthy
License: Trial (expires in 31 days)
Node ID: 5bef844f-f9e0-4126-aebd-3a12bb3cfaa2
    IP: 10.13.237.179
    Local Storage Pool: 1 pool
    POOL  IO_PRIORITY   RAID_LEVEL   USABLE  USED   STATUS ZONE    REGION
    0     HIGH          raid0        900 GiB 16 GiB Online default default
    Local Storage Devices: 6 devices
    Device Path        Media Type           Size        Last-Scan
    0:1    /dev/sds    STORAGE_MEDIUM_SSD   150 GiB     08 Nov 24 17:22 UTC
    0:2    /dev/sdv    STORAGE_MEDIUM_SSD   150 GiB     08 Nov 24 17:22 UTC
    0:3    /dev/sdr    STORAGE_MEDIUM_SSD   150 GiB     08 Nov 24 17:22 UTC
    0:4    /dev/sdu    STORAGE_MEDIUM_SSD   150 GiB     08 Nov 24 17:22 UTC
    0:5    /dev/sdt    STORAGE_MEDIUM_SSD   150 GiB     08 Nov 24 17:22 UTC
    0:6    /dev/sdx    STORAGE_MEDIUM_SSD   150 GiB     08 Nov 24 17:22 UTC
    total          -                        900 GiB
    Cache Devices:
     * No cache devices
    Journal Device:
    1      /dev/sdw1   STORAGE_MEDIUM_SSD   3.0 GiB
    Kvdb Device:
    Device Path   Size
    /dev/sdh      32 GiB
     * Internal kvdb on this node is using this dedicated kvdb device to store its data.
Cluster Summary
    Cluster ID: px-cluster-8e9eaead-3f4d-4aaf-8f9d-e34f2f8ae88b
    Cluster UUID: e5b5a686-c269-4f97-be9e-4ceb3922ff2b
    Scheduler: kubernetes
    Total Nodes: 6 node(s) with storage (6 online)
    IP           ID          SchedulerNodeName Auth    StNd Used   Capacity Status StStat Version     Kernel                OS
    10.13.226.39 f1cf2...74  pool-1-75f7...rk  Enabled Yes  16 GiB 900 GiB  Online Up     3.2.0.0-2ded0fe 5.15.0-1051-gkeop Ubuntu 22.04.4
    10.13.229.226 b3ea1...14 pool-1-75f7...b4  Enabled Yes  16 GiB 900 GiB  Online Up     3.2.0.0-2ded0fe 5.15.0-1051-gkeop Ubuntu 22.04.4
    10.13.229.18 b0a75...fb  pool-1-75f7...mr  Enabled Yes  16 GiB 900 GiB  Online Up     3.2.0.0-2ded0fe 5.15.0-1051-gkeop Ubuntu 22.04.4
    10.13.237.179 5bef8...a2 pool-1-75f7...h2  Enabled Yes  16 GiB 900 GiB  Online Up.    3.2.0.0-2ded0fe 5.15.0-1051-gkeop Ubuntu 22.04.4
    10.13.227.252 40011...6e pool-1-75f7...n2  Enabled Yes  16 GiB 900 GiB  Online Up     3.2.0.0-2ded0fe 5.15.0-1051-gkeop Ubuntu 22.04.4
    10.13.239.37 277e8...d1  pool-1-75f7...xm  Enabled Yes  16 GiB 900 GiB  Online Up     3.2.0.0-2ded0fe 5.15.0-1051-gkeop Ubuntu 22.04.4
Global Storage Pool
    Total Used     : 96 GiB
    Total Capacity : 5.3 TiB
Telemetry: Healthy
```

Similar to previous steps, ensure all Portworx nodes are online and errors or warnings are displayed in the command above. If you encounter any errors, check the Troubleshooting section in this document or contact Portworx support for further assistance.

Ensure all Portworx KVDB instances are running by running the command below:

```
$ 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 service kvdb members
Kvdb Cluster Members:
ID          PEER URLs                         CLIENT URLs                 LEADER HEALTHY DBSIZE
40011...6e  [http://px-1.internal.kvdb:9018] [http://10.13.227.252:9019] true   true    548 KiB
277e8...d1  [http://px-2.internal.kvdb:9018] [http://10.13.239.37:9019]  false  true    536 KiB
f1cf2...74  [http://px-3.internal.kvdb:9018] [http://10.13.226.39:9019]  false  true    536 KiB
```

Your output **must**:

- Show three KVDB members.
- One of the members must be the leader.
- Members must be all healthy.

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

Operator Upgrade

After all prerequisites above have been completed the first component to upgrade is the Portworx operator.

You will need to upgrade the operator to the latest version (see the release notes to see the latest version available):

1. Change the px-operator deployment image: version-tag, if necessary.
2. Then run the following command to confirm Portwox operator is running after the upgrade:

```
$ kubectl -n portworx get pod -l name=portworx-operator
NAME                               READY   STATUS    RESTARTS   AGE
portworx-operator-86dff4955-12fsd  1/1     Running   0          25m
```

Portworx 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. For instance, to upgrade to release 3.2.0, modify the image entry as follows:

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

Besides Portworx, other components are automatically upgraded as well to the versions compatible with the Portworx release you are upgrading to, like Autopilot, Stork and CSI.

For more detailed instructions on upgrading Portworx, visit the Upgrade Portworx using the Operator article of the Portworx documentation.

Upgrading Anthos

Before upgrading Anthos, you must check if the new version of Anthos is compatible with Portworx to ensure Portworx will work properly after the upgrade. In some cases, you may need to upgrade Portworx first, prior to upgrading the Anthos cluster.

To ensure the smooth operation of your Anthos cluster with Portworx during and after an Anthos upgrade follow these best practices:

- If the new Anthos version has a new kernel, make sure the current deployed version of Portworx is compatible with this new kernel version.
- If the new Anthos version has a new version of Kubernetes, make sure the currently deployed version of Portworx is compatible with it, otherwise you'll need to update Portworx to support the newer version of Anthos as outlined here.
- Make sure the deployed version of Portworx is compatible with the new version of Anthos you are planning to use
- Make sure the Portworx cluster is healthy.
- It is preferable to have the Anthos nodes use static IP allocation (along with only one extra IP in reserve) rather than dynamic (which may not have any limits on allocatable IP addresses), since this can help serialize the Anthos upgrades which (as discussed in the next point) should be more stable from the Portworx perspective. The main downside is more time needs to be allocated to the upgrade if there is a high number of nodes to process sequentially.
- Parallel upgrades of Anthos nodes are not advised, as this can lead to a situation where more than one node can go offline that has services Portworx runs, that are designed for single failures only, and this can affect your Portworx cluster's availability during Anthos upgrades. If parallel-node Anthos upgrades are a requirement, please perform them during maintenance windows so as not to impact IO-requiring applications (and Portworx should auto-recover if the nodes successfully come back up) .
- Make sure Portworx internal KVDB is healthy and all three instances of KVDB are up and running. Portworx internal KVDB has an associated PodDisruptionBudget (PDB) that requires at least two KVDB pods be up and running, so you need all three KVDB pods running before starting the Anthos upgrade, otherwise a node draining operation could fail and block the upgrade.

Logging and Monitoring

All Portworx pods will generate logs that can be viewed or retrieved using standard Anthos command lines (`kubectl logs`). If necessary (or required by Portworx support personnel) you can increase log levels by updating the StorageCluster for each component, for example to enable `debug` level for the Stork component you can add the `spec.stork.args.verbose: true` stanza to the StorageCluster resource:

```
kind: StorageCluster
...
spec:
 ...
 stork:
  args:
   verbose: true
   ...
  enabled: true
...
```

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

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

To create a diagnostic-bundle for troubleshooting Portworx, you initiate the collection by running the following command on a specific Portworx-running node:

```
$ ADMIN_TOKEN=$(kubectl -n portworx get secret px-admin-token \
   --template='{{index .data "auth-token" | base64decode}}')
$ PX_POD=$(kubectl -n portworx get po -lname=portworx -o wide | awk '$7 ~ /^yournodename$/ {print $1}')

$ 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 service diags -a
Running PX diagnostics inside OCI-Monitor container - forwarding request to 127.0.0.1
Running Diagnostics on remote node 127.0.0.1....
Generated diags for node 127.0.0.1
```

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

To enable Telemetry follow the instructions in the documentation link here.

Application Considerations

All stateful applications will consume one or many PersistentVolumeClaims (PVC) which are provided by Portworx. These PVCs should be created using one of the default StorageClasses created by Portworx Operator. By default all StorageClasses have volume-replication enabled and Portworx recommends enabling volume replication if other StorageClasses are created.

Application HA

Intra-cluster

Portworx ensures data availability through storage replication, allowing an application to access its data from a replica node should the original node become unavailable. However, if the node hosting an application pod fails, the application will experience downtime until Kubernetes moves this pod to another healthy node within the cluster. Thus, while storage remains accessible, the application itself may be temporarily offline. Depending on your requirements and the nature of your application, you may opt for either HA Mode or non-HA mode.

HA mode is for applications that support HA mode and require zero downtime. it is advisable to run multiple replicas of the application pods. This configuration ensures continuous service availability, as the failure of a node hosting one of the pods will lead the other replicas to seamlessly continue handling traffic.

Non-HA mode: For applications that can withstand temporary downtime during Kubernetes' failover process or those that do not support HA mode, deploying a single replica of the application pod is sufficient.

Portworx Images

To get list of Portworx images, you can point your browser to an URL similar to this:

https://install.portworx.com/3.2/images?kbver=1.28.12-gke.1100

The example above will display images for the latest PX version 3.2.x for the kubernetes version 1.28.12-gke.1100.

You can also use the curl command, for example:

```
$ curl -s 'https://install.portworx.com/3.2/images?kbver=1.28.12-gke.1100' | sort
docker.io/nginxinc/nginx-unprivileged:1.25
docker.io/openstorage/cmdexecutor:24.3.2
docker.io/openstorage/stork:24.3.2
docker.io/portworx/autopilot:1.3.15
docker.io/portworx/oci-monitor:3.2.0
docker.io/portworx/portworx-dynamic-plugin:1.1.1
docker.io/portworx/px-enterprise:3.2.0
docker.io/portworx/px-operator:24.1.3
docker.io/purestorage/ccm-go:1.2.2
docker.io/purestorage/log-upload:px-1.1.29
docker.io/purestorage/realtime-metrics:1.0.29
docker.io/purestorage/telemetry-envoy:1.1.16
quay.io/prometheus-operator/prometheus-config-reloader:v0.75.0
quay.io/prometheus-operator/prometheus-operator:v0.75.0
quay.io/prometheus/alertmanager:v0.27.0
quay.io/prometheus/prometheus:v2.54.1
registry.k8s.io/kube-controller-manager-amd64:v1.28.12
registry.k8s.io/kube-scheduler-amd64:v1.21.4
registry.k8s.io/kube-scheduler-amd64:v1.28.12
registry.k8s.io/pause:3.1
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
```

In order to pull down the images and push them to your private registry, please consult this section of the docs, which describes the procedure we have developed for how you can upload those images to a local repository.

Monitoring during the Installation

Once Portworx is deployed, you can follow the progress by checking status of the pods:

```
$ kubectl -n portworx get pods
NAME                                    READY STATUS   RESTARTS    AGE
autopilot-858b769dbc-4qg86              1/1   Running  0           25m
portworx-api-4k7w2                      2/2   Running  3 (23m ago) 24m
portworx-api-hkqh8                      2/2   Running  3 (23m ago) 24m
portworx-api-pqdtr                      2/2   Running  3 (23m ago) 24m
portworx-api-qfbct                      2/2   Running  3 (23m ago) 24m
portworx-api-snp9b                      2/2   Running  4 (22m ago) 25m
portworx-api-xkt84                      2/2   Running  3 (23m ago) 24m
portworx-kvdb-d57ps                     1/1   Running  0           22m
portworx-kvdb-kdhsz                     1/1   Running  0           22m
portworx-kvdb-v4wm2                     1/1   Running  0           22m
portworx-operator-86dff4955-l2fsd       1/1   Running  0           38m
portworx-pvc-controller-68fcdc9fdc-c2tpb 1/1   Running  0           24m
portworx-pvc-controller-68fcdc9fdc-jgh7m 1/1   Running  0           24m
portworx-pvc-controller-68fcdc9fdc-n4jp9 1/1   Running  0           25m
prometheus-px-prometheus-0              2/2   Running  0           24m
px-cluster-8e9ea...mv                   1/1   Running  0           24m
px-cluster-8e9ea...sg                   1/1   Running  0           24m
px-cluster-8e9ea...65                   1/1   Running  0           24m
px-cluster-8e9ea...cs                   1/1   Running  0           24m
px-cluster-8e9ea...tp                   1/1   Running  0           24m
px-cluster-8e9ea...kq                   1/1   Running  0           24m
px-csi-ext-5cc967d5-2s2bl               3/3   Running  0           24m
px-csi-ext-5cc967d5-lfcb7               3/3   Running  0           24m
px-csi-ext-5cc967d5-qjndw               3/3   Running  0           24m
px-prometheus-operator-7fd768bcff-bhgbt 1/1   Running  0           25m
px-telemetry-phonehome-54bxc            2/2   Running  0           22m
px-telemetry-phonehome-dk75p            2/2   Running  0           22m
px-telemetry-phonehome-fr974            2/2   Running  0           22m
px-telemetry-phonehome-hlwmm            2/2   Running  0           22m
px-telemetry-phonehome-s628m            2/2   Running  0           22m
px-telemetry-phonehome-xq26r            2/2   Running  0           22m
px-telemetry-registration-c5bfdf4f-kpknc 2/2   Running  0           22m
stork-674c5d4bf5-454q8                  1/1   Running  0           24m
stork-674c5d4bf5-6kmwn                  1/1   Running  0           25m
stork-674c5d4bf5-wd8nj                  1/1   Running  0           24m
stork-scheduler-6dcdc656d8-5jvl9        1/1   Running  0           24m
stork-scheduler-6dcdc656d8-5zqpn        1/1   Running  0           25m
stork-scheduler-6dcdc656d8-xkszj        1/1   Running  0           24m
```

Another monitoring option is to get the status of the StorageNodes resources:

```
$ kubectl -n portworx get storagenodes
NAME                     ID          STATUS  VERSION          AGE
pool-1-75f796558d-c7cxm  277e8...d1  Online  3.2.0.0-2ded0fe  27m
pool-1-75f796558d-h42mr  b0a75...fb  Online  3.2.0.0-2ded0fe  26m
pool-1-75f796558d-hfsb4  b3ea1...14  Online  3.2.0.0-2ded0fe  26m
pool-1-75f796558d-jt2h2  5bef8...a2  Online  3.2.0.0-2ded0fe  26m
pool-1-75f796558d-l2fn2  40011...6e  Online  3.2.0.0-2ded0fe  26m
pool-1-75f796558d-v4crk  f1cf2...74  Online  3.2.0.0-2ded0fe  26m
```

At the end of deployment (when the StorageNodes transition from `Initializing` to `Online`), all pods should be running and in the `Ready` state, if any problems happen please check the troubleshooting section below.

Post-installation Validation: Checking Cluster Operators, Cluster Version, and Nodes

After deploying Portworx you can follow the steps in this link to verify the installation and create your PVC with a Portwox StorageClass.

Troubleshooting Commands

To troubleshoot installation issues you can check the logs from the pods that may be failing and look for some errors. For example to troubleshoot a specific Portworx pod from a cluster deployed with the name `px-cluster-refarch` you can run this command:

```
$ kubectl -n portworx logs px-cluster-refarch-pod-name-xxxxxx
```

Other helpful commands include:

- Describe a pod to check for any errors in the Events section. The example below shows that the `stork` image pull is failing:

```
$ kubectl -n portworx describe pod stork-674c5d4bf5-6kmwn
Events:
 Type    Reason    Age  From              Message
 ----    ------    ---- ----              -------
 Normal  Scheduled 55s  default-scheduler Successfully assigned portworx/stork-674c5d4bf5-6kmwn to pool-1-
75f796558d-jt2h2
 Normal  Pulling   55s  kubelet           Pulling image "docker.io/openstorage/stork:24.3.2"
 Warning Failed    14m  kubelet           Failed to pull image "docker.io/openstorage/stork:24.3.2": rpc
error: code = Unknown desc = writing blob: storing blob to file "/var/tmp/storage63576889/3": happened
during read: read tcp [2620:125:9006:1324:bb4c:9a1:72cf:488b]:55724→[2606:4700::6810:62d7]:443: read: con-
nection reset by peer
```

- Retrieve logs from the Portworx operator pod:

```
$ kubectl -n portworx logs -l name=portworx-operator --tail=9999
time="08-11-2024 17:06:14" level=info msg="Starting openstorage operator version 24.1.3-d831f9cc" file="op-
erator.go:125"
time="08-11-2024 17:06:14" level=info msg="Registering components" file="operator.go:167"
time="08-11-2024 17:06:14" level=info msg="Found namespaceNamespaceportworx" file="k8sutil.go:80"
time="08-11-2024 17:06:14" level=info msg="Found podnamePod.Nameportworx-operator-86dff4955-12fsd" file="k-
8sutil.go:127"
time="08-11-2024 17:06:14" level=info msg="Found PodPod.NamespaceportworxPod.Nameportworx-opera-
tor-86dff4955-12fsd" file="k8sutil.go:142"
time="08-11-2024 17:06:14" level=info msg="Pods owner foundKindDeploymentNameportworx-operatorNamespace-
portworx" file="metrics.go:174"
time="08-11-2024 17:06:14" level=info msg="Metrics Service object updatedService.Nameportworx-operator-met-
ricsService.Namespaceportworx" file="metrics.go:94"
time="08-11-2024 17:06:14" level=info msg="cluster is running k8s distribution v1.28.12-gke.1100"
```

For more information on how to troubleshoot Portworx, refer to the Troubleshooting section of the Portworx Documentation.

Details on how to contact Portworx support are available here.

Summary

This document provides a comprehensive reference architecture for deploying Portworx on Google Distributed Cloud (GDC) Anthos running on vSphere. It is intended for Kubernetes administrators and cloud architects who are familiar with Portworx and Anthos concepts. The document is divided into three main sections: planning and architecture overview, design considerations, and operations considerations.

The planning and architecture overview section presents a high-level design of how Portworx is deployed on Anthos, including requirements for storage and storageless nodes, and recommendations for configuring vSphere Datastores. The design considerations section provides detailed requirements and recommendations for the design phase, covering areas such as Anthos requirements, networking, capacity planning, high availability, resource and performance considerations, security, and monitoring. It also includes a recommended Portworx installation template. The operations considerations section offers best practices for day 2 operations after Portworx is deployed, including validating the Portworx installation, observability, scaling, backup and recovery techniques, upgrading Portworx and Anthos, and checking Portworx logs.

Portworx enhances Google Anthos by providing a robust, secure, highly available, and scalable data management layer, enabling organizations to run stateful applications on GDC with features like replication, high availability, security, encryption, capacity management, disaster recovery, and data protection. This integration simplifies the modernization journey for organizations by reducing the need for custom Kubernetes storage layer management and allowing for more efficient and resilient application deployment and management.

©2025 Pure Storage, the Pure Storage P Logo, Portworx, Pure1, 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.

PS2716-01-en 01/25