PURE VALIDATED DESIGN

VMware Tanzu with Portworx

Architecting a resilient, highly available Kubernetes data services platform for VMware Tanzu.

Pure Validated Design

Contents

Pure Validated Design

Executive Summary

As part of digital transformation efforts, organizations are modernizing their applications and the infrastructure they run on by adopting containers and Kubernetes for their applications and leveraging a solution like VMware Tanzu Kubernetes Grid for their infrastructure. A majority of enterprises already use some form of VMware solutions in their data centers, ranging from a basic vSphere cluster to a full-fledged VMware Cloud Foundation-based private cloud deployment. VMware Tanzu Kubernetes Grid allows organizations to use their existing investments in infrastructure to run enterprise-grade Kubernetes inside their own data centers. The VMware Tanzu portfolio enables organizations to build, run, and manage modern applications—and continuously deliver value to their customers. It enables development and operations teams to work together in new ways that deliver transformative business results.

Modern applications are built using containers and orchestrated by Kubernetes, but they still need a layer of persistence. To run stateful applications on VMware Tanzu Kubernetes Grid (TKG), organizations need a robust data services platform like Portworx® by Pure Storage®. Portworx provides features like replication and high availability, security and encryption, capacity management, disaster recovery, and data protection to TKG deployments. Instead of spending resources architecting and managing a custom Kubernetes storage layer, organizations can accelerate their modernization journeys by adopting a solution like Portworx and offload onto Portworx all the tasks that do not generate customer value.

When a solution is designated as a Pure Validated Design (PVD), it means that Pure has integrated and validated our leading-edge storage technology with an industry-leading application solution platform to simplify deployment, reduce risk, and free up IT resources for business-critical tasks. The PVD process validates a solution and provides design consideration and deployment best practices to accelerate deployment. The PVD process assures that the chosen technologies form an integrated solution to address critical business objectives. This document provides design consideration and deployment best practices for VMware Tanzu Kubernetes Grid and Portworx to provide a modern infrastructure platform to run Kubernetes.

Introduction

This document describes the benefits of using Portworx with VMware Tanzu Kubernetes Grid to run stateful containerized applications. It is a validated design that includes different use cases, design considerations, deployment specifics, and configuration best practices for a developer-ready environment.

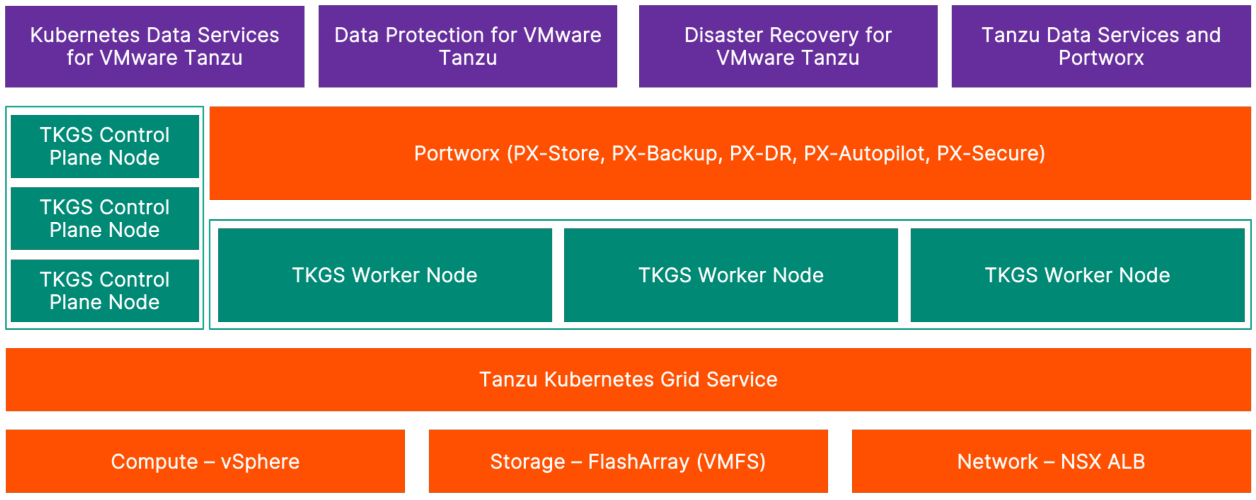
This document will cover four different use cases around VMware Tanzu Kubernetes Grid, which are described in the Solution Overview section below. To follow along with the deployment steps listed in this document, an administrator will need to deploy a production-ready VMware vSphere environment with Workload Management enabled. The vSphere environment must include a vCenter server appliance, an ESXi cluster of a minimum of four hosts, with a VMFS-backed datastore. For this validation, we have used a Pure FlashArray™ storage backend mounted as a VMFS datastore on our vSphere cluster.

Solution Overview

This Pure Validated Design is based on VMware Tanzu Kubernetes Grid using Portworx as the data services platform. This solution covers four use cases that should help administrators deploy and operate a robust Kubernetes stack for their developers.

Use Cases

- **Kubernetes storage and data management for VMware Tanzu:** In this use case, we will talk about how Portworx Enterprise provides cloud-native data services for VMware Tanzu Kubernetes Grid, including features like replication and high availability, encryption and role-based access control, IO profiles, Autopilot, and cloud drives.
- **Data protection for VMware Tanzu:** In this use case, we will talk about how PX-Backup from Portworx can provide a Kubernetes backup and restore solution for applications running on VMware Tanzu Kubernetes Grid.
- **Disaster recovery for VMware Tanzu:** In this use case, we will talk about building a disaster recovery solution for VMware Tanzu Kubernetes Grid. To enable customers to build DR solutions for their apps running on TKG, PX-DR can provide synchronous and asynchronous replication capabilities to ensure that even cloud-native applications have the resiliency needed to meet predefined service-level agreements (SLAs).
- **VMware Tanzu Data Services and Portworx:** In this use case, we will talk about how Portworx Enterprise, PX-DR, and PX-Backup can provide a consistent solution for different VMware Tanzu Data Services when it comes to Kubernetes storage, disaster recovery, and data protection, respectively.


Kubernetes Data Services for VMware Tanzu
Data Protection for VMware Tanzu
Disaster Recovery for VMware Tanzu
Tanzu Data Services and Portworx
TKGS Control Plane Node
TKGS Control Plane Node
TKGS Control Plane Node
Portworx (PX-Store, PX-Backup, PX-DR, PX-Autopilot, PX-Secure)
TKGS Worker Node
TKGS Worker Node
TKGS Worker Node
Tanzu Kubernetes Grid Service
Compute – vSphere
Storage – FlashArray (VMFS)
Network – NSX ALB

Solution Benefits

This solution enables organizations to accelerate their adoption of modern applications and Kubernetes by using a best-in-class Kubernetes offering from VMware and the gold standard of the Kubernetes data services platform from Portworx. Organizations that are getting started with their Kubernetes adoption journey will find this solution valuable, as it walks them through the basic use case of providing persistent storage for their containers and advanced use cases like data protection and disaster recovery. Organizations that are already running a couple of applications on Kubernetes will also find this solution valuable, as it helps them take the next step and adopt a platform that ensures business continuity, while also supporting their efforts to get the best performance and reliability from their Kubernetes storage layer.

In addition to the consistency provided by VMware Tanzu Kubernetes Grid across different on-premises and cloud environments, Portworx also provides consistency when it comes to Kubernetes storage. Organizations can choose to run their TKG clusters on-prem or in the public cloud, and they can still rely on Portworx to provide the same set of Kubernetes data services for their applications.

VMware Tanzu

VMware Tanzu enables organizations to build, run, and manage modern apps on any cloud—and continuously deliver value to their customers. With VMware Tanzu, organizations can simplify multi-cloud operations and free developers to move faster with easy access to the right resources. VMware Tanzu enables development and operations teams to work together in new ways that deliver transformative business results.

VMware Tanzu provides a full stack of capabilities for modernizing applications and infrastructure to continuously deliver better software to production. Organizations can transform their applications and infrastructure in their way and at their own pace to build new, cloud-native applications and run them consistently everywhere on a secure and scalable infrastructure with a conformant Kubernetes runtime.

The VMware Tanzu editions are:

- **VMware Tanzu Basic:** Simplify operation of Kubernetes on-premises as part of vSphere 7.
- **VMware Tanzu Standard:** Simplify operation of Kubernetes for multi-cloud deployments.
- **VMware Tanzu Advanced:** Simplify and secure the container lifecycle to speed the delivery of modern apps at scale.

VMware Tanzu editions build on each other to simplify Kubernetes adoption and enable organizations to run modern apps at scale.

NOTE: To learn more, see this VMware blog about Tanzu editions.

When using VMware Tanzu, you need to understand the following terminology:

- **VMware Tanzu Kubernetes Grid (TKG):** VMware Tanzu Kubernetes Grid, informally known as TKG, is a multi-cloud Kubernetes footprint that you can run both on-premises in vSphere and the public cloud on Amazon EC2 and Microsoft Azure. In addition to Kubernetes binaries that are tested, signed, and supported by VMware, Tanzu Kubernetes Grid includes signed and supported versions of open-source applications to provide the registry, networking, monitoring, authentication, ingress control, and logging services that a production Kubernetes environment requires.

- **VMware Tanzu Kubernetes Grid Service:** Tanzu Kubernetes Grid Service, informally known as TKGS, lets you create and operate TKG clusters natively in vSphere with Tanzu. You use the Kubernetes CLI to invoke the Tanzu Kubernetes Grid Service and provision and manage TKG clusters. The Kubernetes clusters provisioned by the service are fully conformant, so you can deploy all the types of Kubernetes workloads that you would expect. vSphere with Tanzu leverages many reliable vSphere features to improve the Kubernetes experience, including vCenter Single Sign-On (SSO), the Content Library for Kubernetes software distributions, vSphere networking, vSphere storage, vSphere HA and Distributed Resource Scheduler (DRS), and vSphere security.
- **VMware Tanzu Kubernetes Grid Integrated Edition:** VMware Tanzu Kubernetes Grid Integrated Edition, informally known as TKGI, is a Kubernetes-based container solution that is integrated with Cloud Foundry BOSH and Ops Manager. Formerly known as VMware Enterprise PKS, Tanzu Kubernetes Grid Integrated Edition allows you to provision, operate, and manage Kubernetes clusters. It provides advanced networking, a private container registry, and lifecycle management so that you can run and manage containers at scale on private and public clouds.

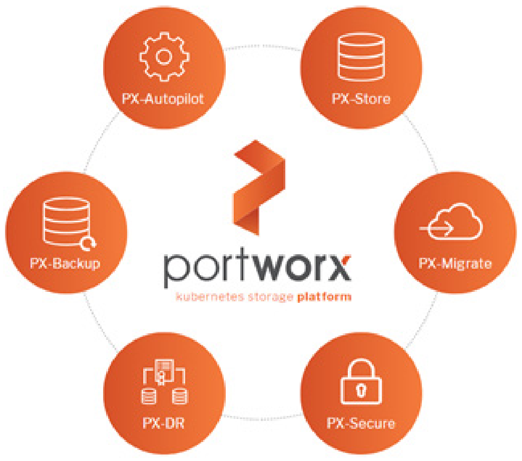
VMware Tanzu Kubernetes Grid clusters support three different networking options based on the infrastructure of your choice:

- **vSphere Networking and HAProxy Load Balancer**: If you are using vSphere Distributed Switch networking for your vSphere with Tanzu environment, you can install and configure the open-source HAProxy load balancer. VMware provides an implementation of HAProxy that you can deploy from an Open Virtual Appliance (OVA) file.
- **vSphere Networking and NSX Advanced Load Balancer**: vSphere with Tanzu supports the NSX Advanced Load Balancer, also known as Avi Load Balancer, Essentials Edition. If you are using vSphere Distributed Switch (vDS) networking for Workload Management, you can install and configure the NSX Advanced Load Balancer in your vSphere with Tanzu environment.
- **NSX-T Data Center**: vSphere with Tanzu requires a specific networking configuration to enable connectivity to the Supervisor Clusters, vSphere Namespaces, and all objects that run inside the namespaces, such as vSphere Pods, VMs, and Tanzu Kubernetes Grid clusters. As a vSphere administrator, install and configure NSX-T Data Center for vSphere with Tanzu.

For this validated design document, we have used vSphere with Tanzu with vSphere Networking and NSX Advanced Load Balancer for networking and a Pure FlashArray mounted VMFS datastore for VM storage.

Portworx

Portworx is a data management solution that serves applications and deployments in Kubernetes clusters. Portworx is deployed natively within Kubernetes and extends the automation capabilities down into the infrastructure to reduce the complexities of managing data. Portworx provides simple and easy-to-consume storage classes that are usable by stateful applications in a Kubernetes cluster.



At the core of Portworx is PX-Store, a software-defined storage platform that works on practically any infrastructure, regardless of whether it is in a public cloud or on-premises. PX-Store is complemented by these modules:

- **PX-Migrate:** Allows applications to be easily migrated across clusters, racks, and clouds
- **PX-Secure:** Provides access controls and enables data encryption at a cluster, namespace, or persistent volume level
- **PX-DR:** Allows applications to have a zero recovery point objective (RPO) failover across data centers in a metro area as well as continuous backups across the WAN for even greater protection
- **PX-Backup:** Allows enterprises to back up and restore the entire Kubernetes application, including data, app configuration, and Kubernetes objects, to any backup location—including S3, Azure Blob, etc.—with the click of a button
- **PX-Autopilot:** Provides rules-based auto-scaling for persistent volumes and storage pools

PX-Store

PX-Store is a 100% software-defined storage solution that provides high levels of persistent volume density per block device per worker node. The key features of PX-Store include:

- **Storage virtualization:** The storage made available to each worker node is effectively virtualized such that each worker node can host pods that use up to hundreds of thousands of persistent volumes per Kubernetes cluster. This benefits Kubernetes clusters deployed to the cloud, in that larger volumes or disks are often conducive to better performance.
- **Storage-aware scheduling:** STORK, a storage-aware scheduler, co-locates pods on worker nodes that host the persistent volume replicas associated with the same pods, resulting in reduced storage access latency.
- **Storage pooling for performance-based quality-of-service:** PX-Store segregates storage into three distinct pools of storage based on performance: low, medium, and high. Applications can select the storage based on performance by specifying one of these pools at the storage class level.
- **Persistent volume replicas:** You can specify a persistent volume replication factor at the storage class level. This enables the state to be highly available across the cluster, cloud regions, and Kubernetes-as-a-service platforms.

- **Cloud volumes:** Cloud volumes enable storage to be provisioned from the underlying platform without the need to present storage to worker nodes. PX-Store running on most public cloud providers and VMware Tanzu have cloud volume capability.
- **Automatic I/O path tuning:** Portworx provides different I/O profiles for storage optimization based on the I/O traffic pattern. By default, Portworx automatically applies the most appropriate I/O profile for the data patterns it sees. It does this by continuously analyzing the I/O pattern of traffic in the background.
- **Metadata caching:** High-performance devices can be assigned the role of journal devices to lower I/O latency when accessing metadata.
- **Read- and write-through caching:** PX-Cache-enabled high-performance devices can be used for read- and write-through caching to enhance performance.

PX-Backup

Backup is essential for enterprise applications, serving as a core requirement for mission-critical production workloads. The risk to the enterprise is magnified for applications on Kubernetes where traditional, virtual machine (VM)-optimized data protection solutions simply don’t work. Protecting stateful applications like databases in highly dynamic environments calls for a purpose-built, Kubernetes-native backup solution.

Portworx PX-Backup solves these shortfalls and protects your applications’ data, application configuration, and Kubernetes objects with a single click at the Kubernetes pod, namespace, or cluster level. Enabling application-aware backup and fast recovery for even complex distributed applications, PX-Backup delivers true multi-cloud availability with key features, including:

- **App-consistent backup and restore:** Easily protect and recover applications regardless of how they are initially deployed on, or rescheduled by, Kubernetes.
- **Seamless migration:** Move a single Kubernetes application or an entire namespace between clusters.
- **Compliance management:** Manage and enforce compliance and governance responsibilities with a single pane of glass for all your containerized applications.
- **Streamlined storage integration:** Back up and recover cloud volumes with storage providers, including Amazon EBS, Google Persistent Disk, Azure Managed Disks, and CSI-enabled storage.

PX-DR

PX-DR extends the data protection included in PX-Store with zero RPO disaster recovery for data centers in a metropolitan area as well as continuous backups across the WAN for an even greater level of protection. PX-DR provides both synchronous and asynchronous replication, delivering key benefits, including:

- **Zero data loss disaster recovery:** PX-DR delivers zero RPO failover across data centers in metropolitan areas in addition to high availability (HA) within a single data center. You can deploy applications between clouds in the same region and ensure application survivability.
- **Continuous global backup:** For applications that span a country—or the entire world—PX-DR also offers constant incremental backups to protect your mission-critical applications.

PX-Autopilot

PX-Autopilot allows enterprises to automate storage management to intelligently provision cloud storage only when needed and eliminate the problem of paying for storage when over-provisioned. PX-Autopilot delivers many benefits:

- **Grow storage capacity on-demand:** Automate your applications' growing storage demands while also minimizing disruptions. Set growth policies to automate cloud drive and Kubernetes integration to ensure each application's storage needs are met without performance or availability degradations.
- **Slash storage costs by half:** Intelligently provision cloud storage only when needed and eliminate the problem of paying for storage when over-provisioned instead of consumed. Scaling at the individual volume or entire cluster level saves money and avoids application outages.
- **Integrate with all major clouds and VMware:** PX-Autopilot natively integrates with AWS, Azure, and Google Cloud ,as well as VMware Tanzu, enabling you to achieve savings and increase automated agility across all your clouds.

Deployment Options

When creating a specification for the deployment of Portworx, you have several options to consider:

- **Existing KVDB:** For most deployments, you can create a deployment specification with the option of storing Portworx metadata in a separate etcd cluster. There are two exceptions to this:
  - The first scenario is when PX-DR is used for Kubernetes clusters that are not within the same metro area, meaning that the network round-trip latency between the primary and disaster recovery sites is greater than 10ms.
  - The second scenario is when a dedicated etcd cluster should be used for large-scale deployment, with 10 or more worker nodes, in which a heavy dynamic provisioning activity takes place.
- **Dedicated journal device:** A dedicated journal device can be specified to buffer metadata writes.
- **Dedicated cache device:** A dedicated cache device can be specified to improve performance by acting as a read/write-through cache.
- **Container storage interface (CSI) API compatibility:** You can choose the option to deploy Portworx with CSI enabled if PX-Secure is to be used.
- **STORK:** STORK is a storage-aware scheduler that attempts to co-locate application pods onto the same nodes as the persistent volumes and persistent volume replicas that it uses. Use STORK if your underlying infrastructure uses either servers with dedicated internal storage or servers with dedicated network-attached storage appliances.
- **Dedicated network:** Consider using a dedicated network for storage cluster traffic if the existing network infrastructure does not support quality-of-service.

Planning, Design, and Prework

This section of the document covers the detailed setup used for Portworx deployment and testing on VMware Tanzu. The network requirements must be carefully planned for a successful deployment, and you must carefully review the prework for the vSphere cluster, vCenter, and Workload Management.:

- vSphere Hosts: This solution is based on four ESXi 7.0.2 hosts.
- vCenter Server: vCenter Server Appliance 7.0.2.
- FlashArray//X VMFS Datastore for running all the management components and VMware Tanzu Kubernetes Grid clusters.

- HA and DRS must be enabled for the cluster. Storage DRS should not be enabled for the VMFS datastore.
- vSphere Distributed Switch version 7.0.2 with all ESXi hosts connected.
- vMotion Network. The best practice is a dedicated vLAN for production workloads.
- NTP configuration must be the same across all ESXi hosts and the vCenter server.
- NSX Advanced Load Balancer version 20.1.4.

  NOTE: For detailed prerequisites needed for deploying VMware Tanzu on an ESXi cluster with a FlashArray backend, as well as a step-by-step guide to enable Workload Management, refer to this Pure Validated Design Guide.

The above document uses a VMware vSphere Distributed Virtual Switch + HAProxy networking option. To set up NSX Advanced Load Balancer with your vSphere Distributed Virtual Switch, follow the steps in VMware's documentation. Once we had the NSX ALB configured, we enabled Workload Management and configured a namespace that will be used for all our Portworx use cases in this document.

Once you have Workload Management enabled, you can log into the namespace by using the following command:

```
kubectl vsphere login --server 10.21.132.25 --vsphere-username administrator@vsphere.local --
insecure-skip-tls-verify
```

Next, you can create TKG clusters for each of the use cases that we will discuss in this document. TKG clusters can be deployed by simply using a YAML file and specifying all the configuration parameters in it. Below is a sample YAML file with comments on what each field represents:

```
apiVersion: run.tanzu.vmware.com/v1alpha1
kind: TanzuKubernetesCluster
metadata:
  name: px-demo-cluster # Name of the TKG Cluster
  namespace: tkc-px # Namespace configured in Workload Management
spec:
  distribution:
    version: v1.19.7 # Version of Kubernetes we want to run on the cluster
  topology:
    controlPlane:
      count: 3 # Number of control plane nodes. For Production clusters, use at least 3 nodes
      class: guaranteed-medium # Size of the VM to be used for control plane nodes
      storageClass: vmfs-cns # VMware CSI backed StorageClass
    workers:
      count: 3 # Number of worker nodes. These nodes will be used for deploying Portworx.
      class: best-effort-large # Size of the VM to be used for worker nodes
      storageClass: vmfs-cns # VMware CSI backed StorageClass
  settings: #all spec.settings are optional
    storage: #optional storage settings
      defaultClass: vmfs-cns # VMware CSI backed StorageClass
```

```
        network: #optional network settings
          cni: #override default cni set in the tkgservicesonfiguration spec
            name: antrea # Antrea and Calico are two CNI options
          pods: #custom pod network
            cidrBlocks: [100.96.0.0/11] # CIDR to be used for pod to pod communication
          services: #custom service network
            cidrBlocks: [100.64.0.0/13] # CIDR to be used for service to service communication
```

The above YAML file can be applied against your supervisor cluster to deploy a VMware Tanzu Kubernetes Grid cluster with:

```
kubectl apply -f tkc.yaml
```

Cluster deployment can be monitored by using the following commands:

```
kubectl get tkc -w
kubectl describe tkc px-demo-cluster
```

Once your cluster is up and running, use the following command to log out of the supervisor cluster and log into the TKG cluster:

```
kubectl vsphere logout
kubectl vsphere login --server 10.21.132.25 --vsphere-username administrator@vsphere.local --
insecure-skip-tls-verify --tanzu-kubernetes-cluster-namespace=tkc-px --tanzu-kubernetes-cluster-
name=px-demo-cluster

kubectl config use-context px-demo-cluster
```

Kubernetes Storage and Data Management for VMware Tanzu

Portworx is built on PX-Store, which aggregates and pools storage capacity and provides a series of advanced data management components that are part of the Portworx Data Services platform. PX-Store provides modern, distributed, container-optimized cloud-native storage with elastic scaling, storage-aware class of service, multi-writer shared volumes, local snapshot capabilities, and multiple failover options (node aware, rack aware, availability zone aware). Portworx v2.8 added support for VMware Tanzu, which means that Portworx can now provide all the data services capabilities to your VMware Tanzu Kubernetes Grid cluster. Once you have a TKG cluster running with at least three (3) worker nodes, you can deploy Portworx on it using a specification generated from PX-Central. After applying this configuration, Portworx leverages the underlying VMware CSI-backed StorageClass object to provision VMDK files on your VMFS datastore and mounts or attaches those VMDKs to your TKC worker nodes. Portworx aggregates these virtual disks (VMDK files) into a single storage pool that can be leveraged by your stateful applications. A Portworx storage pool is composed of a collection of drives of the same capacity and type. So, if you have VMDKs provisioned from multiple storage backends, Portworx will categorize the drives according to their latency and performance in random and sequential input/output operations per second (IOPS) into low, medium, and high categories.

Once you have Portworx PX-Store deployed on your TKG cluster, you can go ahead and create Kubernetes storage classes using the template below:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: px-storage-class # Name of the storage class
provisioner: kubernetes.io/portworx-volume
allowVolumeExpansion: true
parameters:
  repl: "2"
  priority_io: "high"
  io_profile: "db_remote"
```

You can customize a Portworx storage class based on the type of workload and the underlying storage layer. Here are a few settings that you can use to get the best out of Portworx:

- **fs (xfs|ext4)**: Filesystem to be laid out.
- **priority_io (low|medium|high)**: I/O priority for the volume. Use high for IOPS optimized volumes and use medium for throughput optimized volumes.
- **shared_v4 (true)**: Flag to create a globally shared namespace volume that can be used by multiple pods over NFS with POSIX-compliant semantics.
- **repl (1|2|3)**: Replication factor for the volume. This represents the number of copies stored on Portworx.
- **io_profile (auto|db|db_remote|sequential|random)**: IO profiles change how a Portworx volume interacts with the underlying storage disks to improve traffic for different workloads. If you don't provide an IO profile, Portworx will set it to auto, and it will automatically apply an IO profile that is most appropriate to the data pattern it sees.

These are the most used parameters; for additional options, you can check out our Portworx documentation.

STORK: Storage Operator Runtime for Kubernetes

STORK is Portworx's storage scheduler for Kubernetes that helps create an even tighter integration of Portworx and VMware Tanzu. It allows users to co-locate pods with their data, provides seamless migration of data in case of errors, and makes it easier to create and restore snapshots of Portworx volumes. This is achieved by using a Kubernetes scheduler extender. So, every time a new pod is being scheduled on your TKC, STORK will work with Portworx to ensure that the pod is being deployed on a worker node that has a local copy of the persistent volume. In case of a node failure, STORK also works with Kubernetes to ensure that the new pod is deployed on a host with a replica of its persistent volume. All of this is automated for you, so as long as you specify *schedulerName: stork* in your pod specification, you will get the best performance because of data locality using STORK.

Here is an example YAML file that deploys a mysql instance using a Kubernetes deployment object and STORK:

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

High Availability and Replication

One of the key benefits that Portworx provides is Kubernetes-native high availability and replication. Portworx can store multiple copies of your volume on the same storage pool spread across different nodes in the TKG cluster, so when a node or a pod goes down, Kubernetes will quickly spin up another instance of the pod, and it can be attached to a replica of the volume hosted on another node. This decreases the amount of time it takes for applications to fail over and come back online. To test the high availability and replication feature of Portworx with VMware Tanzu, we will use a Cassandra distributed NoSQL database instance and run it on a VMware Tanzu Kubernetes Grid cluster with at least three (3) worker nodes running Portworx. We are using the k8ssandra project for our Cassandra deployment.

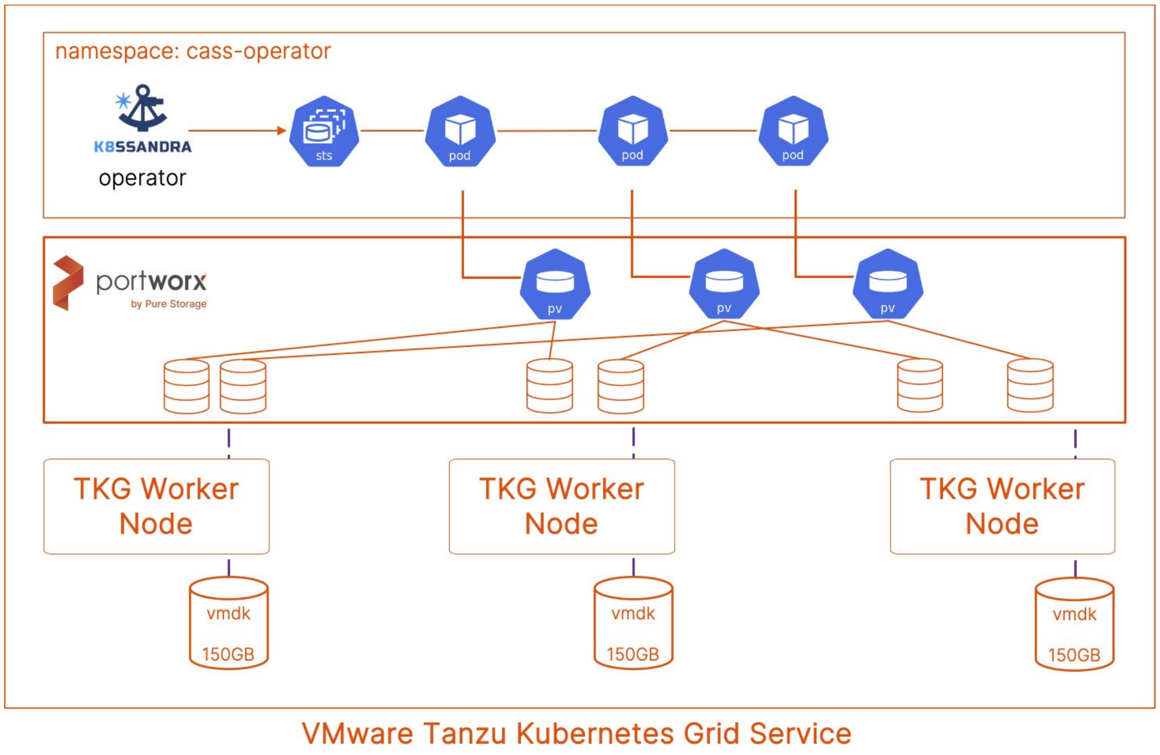


Figure 2. VMware Tanzu Kubernetes Grid Service

1. Install the K8ssandra operator.

```
kubectl apply -f https://raw.githubusercontent.com/k8ssandra/cass-operator/v1.7.0/docs/user/cass-
operator-manifests.yaml
kubectl get all -n cass-operator
```

2. Next, create a StorageClass object that will be used to dynamically provision persistent volumes needed by the Cassandra instance. Any persistent volumes provisioned using this storage class will have two replicas, so even if one node goes down, our Cassandra instance should be still up and running with no data loss.

```
cat > px-cassandra-sc.yaml << EOF
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: portworx-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "2"
  priority_io: "high"
  group: "cassandra_vg"
EOF
kubectl apply -f px-cassandra-sc.yaml
kubectl get sc
```

3. Next, deploy a Cassandra database instance using the following YAML file:

```
cat > cass-instance.yaml << EOF
apiVersion: cassandra.datastax.com/v1beta1
kind: CassandraDatacenter
metadata:
  name: dc1
spec:
  clusterName: cluster1
  serverType: cassandra
  serverVersion: "3.11.7"
  managementApiAuth:
    insecure: {}
  size: 3
  storageConfig:
    cassandraDataVolumeClaimSpec:
      storageClassName: portworx-sc
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
  config:
    cassandra-yaml:
      authenticator: org.apache.cassandra.auth.PasswordAuthenticator
      authorizer: org.apache.cassandra.auth.CassandraAuthorizer
      role_manager: org.apache.cassandra.auth.CassandraRoleManager
    jvm-options:
      initial_heap_size: "800M"
      max_heap_size: "800M"
      max_direct_memory: "800M"
      additional-jvm-opts:
        - "-Ddse.system_distributed_replication_dc_names=dc1"
        - "-Ddse.system_distributed_replication_per_dc=3"
EOF
kubectl apply -f cass-instance.yaml -n cass-operator
```

4. To monitor Cassandra deployment, you can use the following commands:

```
kubectl get pods -n cass-operator
kubectl get pvc -n cass-operator
kubectl -n cass-operator get pods --selector cassandra.datastax.com/cluster=cluster1
kubectl -n cass-operator get cassdc/dc1 -o "jsonpath={.status.cassandraOperatorProgress}"
kubectl -n cass-operator exec -it -c cassandra cluster1-dc1-default-sts-0 -- nodetool status
kubectl get pvc -n cass-operator
```

5. Next, let's write some data to our database:

```
CASS_USER=$(kubectl -n cass-operator get secret cluster1-superuser -o json | jq -r '.data.username'
| base64 --decode)
CASS_PASS=$(kubectl -n cass-operator get secret cluster1-superuser -o json | jq -r '.data.password'
| base64 --decode)
```

```
kubectl -n cass-operator exec -ti cluster1-dc1-default-sts-0 -c cassandra -- sh -c "cqlsh -u
'$CASS_USER' -p '$CASS_PASS'"
CREATE KEYSPACE demodb WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 2 };
use demodb;
CREATE TABLE emp(emp_id int PRIMARY KEY, emp_name text, emp_city text, emp_sal varint,emp_phone
varint);
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(123423445,'Steve', 'Denver',
5910234452, 50000);
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(123423446,'Brian', 'San
Jose', 2525672346, 100000);
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(123423447,'Eric', 'New
York', 2123579895, 95000);
select * from emp;
```

6. Next, let's cordon off the node that is running one of the Cassandra pods so that, when we delete the pod, it can't be deployed on the same node.

```
kubectl cordon <<nodename>>
kubectl delete pods <<Cassandra-pod-name>> -n cass-operator
```

7. This will force the pod to be deployed on a different worker node on the cluster. Use the following command to log into the database instance and list all the entries in the EMP table:

```
kubectl -n cass-operator exec -ti <<podname>> -c cassandra -- sh -c "cqlsh -u '$CASS_USER' -p
'$CASS_PASS'"
use demodb;
select * from emp;
```

8. In addition to Kubernetes-native high availability for applications, you can also easily scale up your databases by adding more replicas to your Kubernetes deployment. As soon as a new pod is provisioned as part of the deployment object, Portworx will create a new persistent volume and provision replicas on the storage layer.

In addition to providing a layer of resiliency for applications with built-in replication, Portworx can also help provide high availability for applications that don't have redundancy built-in at the application layer.

Automated Capacity Management

Capacity management is a critical component to ensure application uptime. If your persistent volumes don't have any available capacity, your application will go offline. In traditional VMware environments, virtualization administrators use tools like vCenter or vRealize Operations Manager to monitor the available storage capacity for the VMs or the VMFS datastore. They expanded the underlying VMDK or LUN to ensure that the application has enough storage available. Portworx PX-Autopilot can automatically manage the underlying storage capacity for VMware Tanzu Kubernetes Grid clusters. PX-Autopilot provides a rule-based engine that responds to changes from a monitoring source. With PX-Autopilot, your VMware Tanzu Kubernetes Grid cluster can react dynamically, without admin intervention, to events such as these:

- Resizing PVCs when they are running out of capacity
- Scaling Portworx storage pools to accommodate increased usage
- Rebalancing volumes across Portworx storage pools when they come unbalanced

Automatically Grow PVCs

PX-Autopilot allows administrators to create Autopilot rules that can execute certain actions if conditions are met for resources with specific labels. Below is a sample Autopilot rule that monitors resources with the label *app: postgres* in the namespace with the label *type:db* and automatically doubles the size of the persistent volume if the persistent volume is more than 70% full. PX-Autopilot will keep expanding the persistent volume until it hits a maximum size of 400GB.

```
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: volume-resize
spec:
  ##### selector filters the objects affected by this rule given labels
  selector:
    matchLabels:
      app: postgres
  ##### namespaceSelector selects the namespaces of the objects affected by this rule
  namespaceSelector:
    matchLabels:
      type: db
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    expressions:
    - key: "100 * (px_volume_usage_bytes / px_volume_capacity_bytes)"
      operator: Gt
      values:
        - "70"
  ##### action to perform when condition is true
  actions:
  - name: openstorage.io.action.volume/resize
    params:
      # resize volume by scalepercentage of current size
      scalepercentage: "100"
      # volume capacity should not exceed 400GiB
      maxsize: "400Gi"
```

Automatically Expand Portworx Storage Pools

With PX-Autopilot, we can also create Autopilot rules that allow administrators to automatically expand their Portworx storage pools by provisioning additional VMDKs and attaching them to the TKC worker nodes. The YAML file lists an Autopilot rule that resizes your storage pool by adding new disks when the available capacity on your storage pool is less than 30%, for a maximum size of 900GB.

```
apiVersion: autopilot.libopenstorage.org/v1alpha1
kind: AutopilotRule
metadata:
  name: pool-expand
spec:
  enforcement: required
  ##### conditions are the symptoms to evaluate. All conditions are AND'ed
  conditions:
    expressions:
    # pool available capacity less than 30%
    - key: "100 * ( px_pool_stats_available_bytes/ px_pool_stats_total_bytes)"
      operator: Lt
      values:
        - "30"
    # volume total capacity should not exceed 400GiB
    - key: "px_pool_stats_total_bytes/(1024*1024*1024)"
      operator: Lt
      values:
       - "900"
  ##### action to perform when condition is true
  actions:
    - name: "openstorage.io.action.storagepool/expand"
      params:
        # resize pool by scalepercentage of current size
        scalepercentage: "50"
        # when scaling, add disks to the pool
        scaletype: "add-disk"
```

PX-Autopilot can completely automate storage capacity management using the YAML files listed above, or it can also help you put action approvals in place. In this case, PX-Autopilot will trigger approval requests for the administrators to approve using kubectl or by using the GitOps workflows. If the administrator approves these requests, it will carry out the expansion operations and help administrators manage their storage capacity.

Data Protection for VMware Tanzu

This use case focuses on data protection for containerized applications running on VMware Tanzu Kubernetes Grid clusters. Portworx PX-Backup provides a modern Kubernetes-native backup and restore solution for VMware Tanzu Kubernetes Grid clusters. When it comes to modern applications, traditional backup solutions won't work for the following reasons:

- **Traditional backup is machine-focused:** Traditional backup solutions talk to the underlying machines (bare metal hosts or virtual machines) and protect them as the primary unit. But they don't consider the applications running on top. Containerized applications are distributed in nature; each machine might have containers that might belong to different applications running on top, and each application might have containers that are spread across multiple machines. If you are just protecting underlying machines without understanding how modern applications are deployed and run in production, you might not be able to restore your applications as expected when needed.
- **Traditional backup doesn't speak Kubernetes**: Traditional backup solutions are more focused on connecting directly with the physical servers or connecting to virtualization managers like vCenter server and inventorying all the different virtual machines running on top of it. A production Kubernetes cluster consists of multiple control plane nodes and multiple worker nodes that are responsible for running your application using constructs like Kubernetes pods, deployments, services, or configmaps. If your backup solution is not able to understand and identify these constructs, you might not be able to restore your applications.
- **Traditional backup is centrally managed:** Traditional backup solutions don't have self-service or role-based access control built in. They are more focused on enabling the backup administrator or infrastructure administrator to create backup schedules and jobs and to ensure that all the jobs are completed successfully. With modern applications, you need

a more distributed approach, where the backup administrator will add the backup locations and create backup schedules. But individual application owners or developers might best know how to protect a particular application and would prefer self-service access and control over how their application is protected.

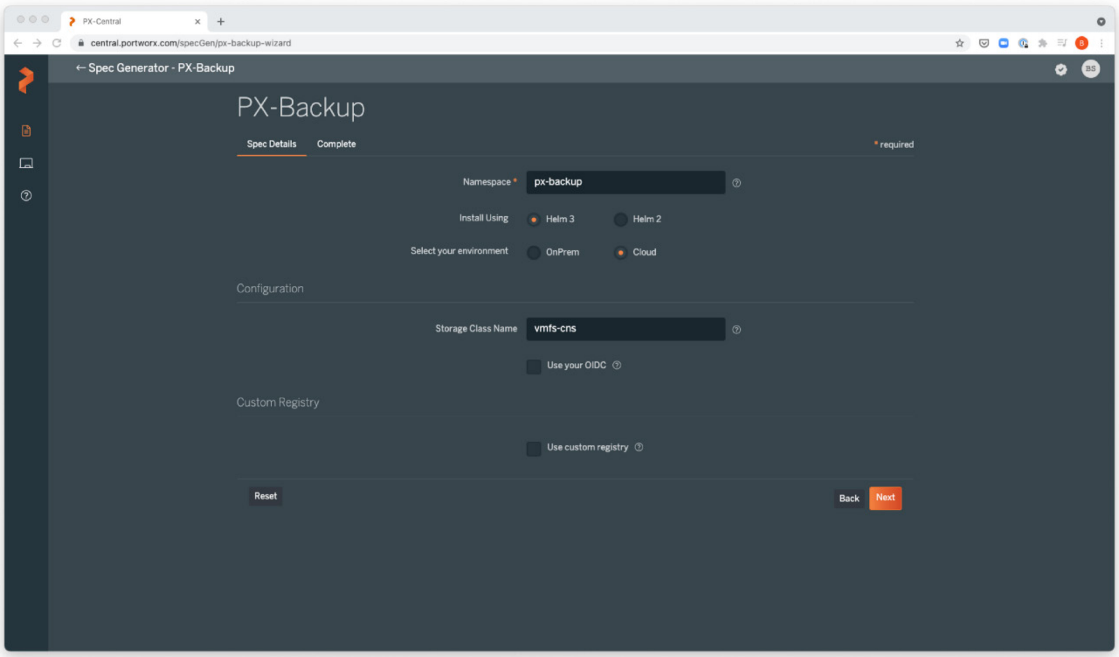
PX-Backup provides a modern data protection solution for VMware Tanzu that meets these specifications:

- **Container-granular:** PX-Backup runs on top of your VMware Tanzu Kubernetes Grid cluster. It can run on the same TKG cluster as your applications, or it can run on a dedicated TKG cluster. It helps you protect all the containers that are part of your application, not just the hosts that are running those containers.
- **Kubernetes namespace-aware:** PX-Backup talks to the Kubernetes API server, and it can identify all the namespaces configured inside the VMware Tanzu Kubernetes Grid cluster. It also identifies all the different Kubernetes objects from pods, deployments, services, config maps, secrets, etc., and it helps you backup everything that constitutes your containerized application.
- **Application-consistent:** Containerized stateful applications are distributed in nature, so it is essential to have a backup solution that can help take an application-consistent snapshot and not just crash-consistent snapshots. PX-Backup allows administrators to create pre- and post-backup rules that can be associated with backup jobs for your distributed applications.
- **Capable of backing up data and app configurations:** PX-Backup allows you to back up your entire application end to end. This includes all the Kubernetes objects, application configurations, and persistent volumes that store your application data.
- **Optimized for the multi-cloud world:** PX-Backup works with all Kubernetes distributions, so you can run your applications in VMware Tanzu Kubernetes Grid clusters on-prem and restore them to an Amazon EKS or a Google Kubernetes Engine cluster or another VMware Tanzu Kubernetes Grid cluster running in AWS or Azure.

Deployment and Validation

To install PX-Backup on your VMware Tanzu Kubernetes Grid cluster, use the following steps:

1. Deploy a TKG cluster on your vSphere cluster with at least three (3) worker nodes. If you are running PX-Backup on a dedicated TKG cluster, you don't need to install Portworx on your TKG cluster. If you are deploying PX-Backup on a shared cluster with applications that need to be protected, install Portworx on your TKG cluster before proceeding with PX-Backup installation.
2. Navigate to PX-Central and create a new spec. Select PX-Backup, and then click Next.
3. Enter a namespace where you want to install all the PX-Backup components. Select Helm3 and Cloud. Enter the name of the storage class that you want to use to install PX-Backup. Click Next.


PX-Central
central.portworx.com/specGen/px-backup-wizard
← Spec Generator - PX-Backup
Reset
Back
Next

PX-Backup

Spec Details | Complete

* required

Namespace *: px-backup

Install Using: [✓ Helm 3] Helm 2

Select your environment: OnPrem [✓ Cloud]

Configuration

Storage Class Name: vmfs-cns

☐ Use your OIDC

Custom Registry

☐ Use custom registry

4. Read through the License Agreement and click **Agree**.
5. Follow this two-step process to install PX-Backup on your VMware Tanzu Kubernetes Grid cluster.

```
helm repo add portworx http://charts.portworx.io/ && helm repo update
helm install px-central portworx/px-central --namespace px-backup --create-namespace --version 2.0.1
--set persistentStorage.enabled=true ,persistentStorage.storageClassName="vmfs-
cns",pxbackup.enabled=true
```

6. You can monitor the PX-Backup deployment using the following commands:

```
kubectl get pods -n px-backup -w
kubectl get po --namespace px-backup -ljob-name=pxcentral-post-install-hook  -o wide | awk '{print
$1, $3}' | grep -iv error
kubectl get svc -n px-backup
```

   Use the LoadBalancer IP for the px-backup-ui service to access the PX-Backup interface.

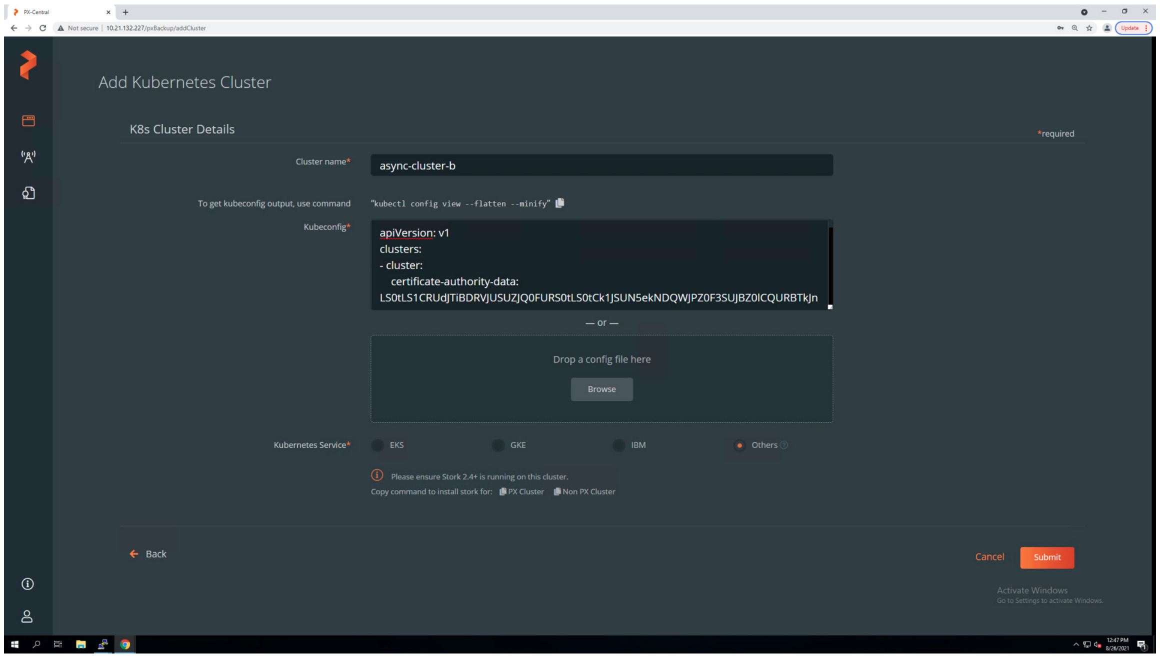
7. Log into the PX-Backup interface using the default credentials (admin/admin). You will be prompted to set a new password on your first login.
8. Once you log in, you can configure cloud accounts, backup locations, schedule policies, and backup rules.
   - **Cloud accounts:** These credentials allow PX-Backup to authenticate with clusters to create backups and restore to them. They also add and manage backup locations where backup objects are stored.
   - **Backup locations:** PX-Backup supports AWS S3, Azure Blob Storage, Google Cloud Object Storage, and any S3-compliant object store as the backup repository to store your backup objects.
   - **Schedule policies:** PX-Backup allows administrators to create periodic, hourly, daily, weekly, and monthly schedule policies that application owners can leverage to create their backup jobs.

- **Backup rules:** To ensure application consistency, PX-Backup allows administrators to create pre- and post-backup rules for their applications. Stateful and distributed applications like Cassandra, Elasticsearch, MongoDB, MySQL, PostgreSQL, and others need these backup rules to take application-consistent snapshots.

9. You can add your VMware Tanzu Kubernetes Grid clusters using the PX-Backup UI. Click on **Add Cluster** on the top right. Enter the name of the Tanzu cluster. Copy the kubectl command and run it against your TKG cluster. Select **Others,** and then click **Submit**.

If you are deploying PX-Backup on a cluster not running Portworx, you will have to install STORK using the command below:

```
curl -fsL -o stork-spec.yaml "https://install.portworx.com/2.6?comp=stork&storkNonPx=true"
kubectl apply -f stork-spec.yaml
```



Add Kubernetes Cluster

K8s Cluster Details

*required

Cluster name* async-cluster-b

To get kubeconfig output, use command "kubectl config view --flatten --minify"

Kubeconfig*

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5ekNDQWJPZ0F3SUJBZ0lCQURBTkJn
```

— or —

Drop a config file here

Browse

Kubernetes Service* EKS GKE IBM [✓ Others]

Please ensure Stork 2.4+ is running on this cluster.
Copy command to install stork for: PX Cluster Non PX Cluster

10. Once your VMware Kubernetes cluster is added, you can start backing up your applications using the PX-Backup UI.

Kubernetes Backup and Restore

As part of the research effort for this document, we validated a couple of use cases for VMware Tanzu Kubernetes Grid clusters.

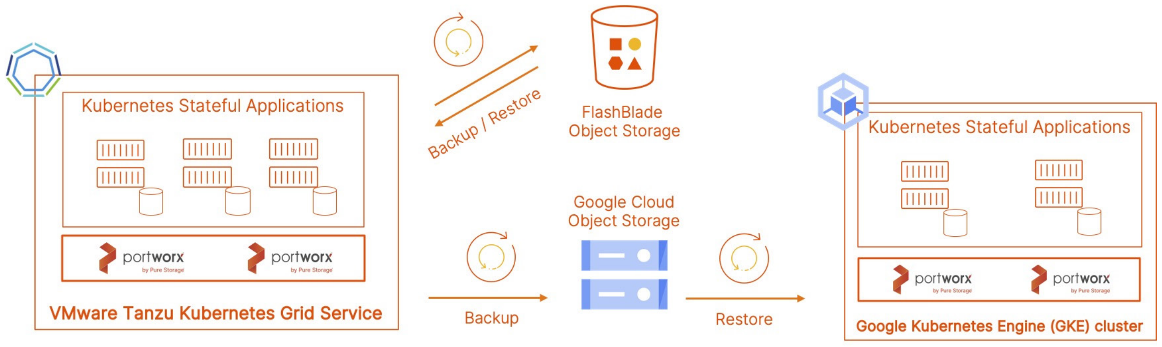
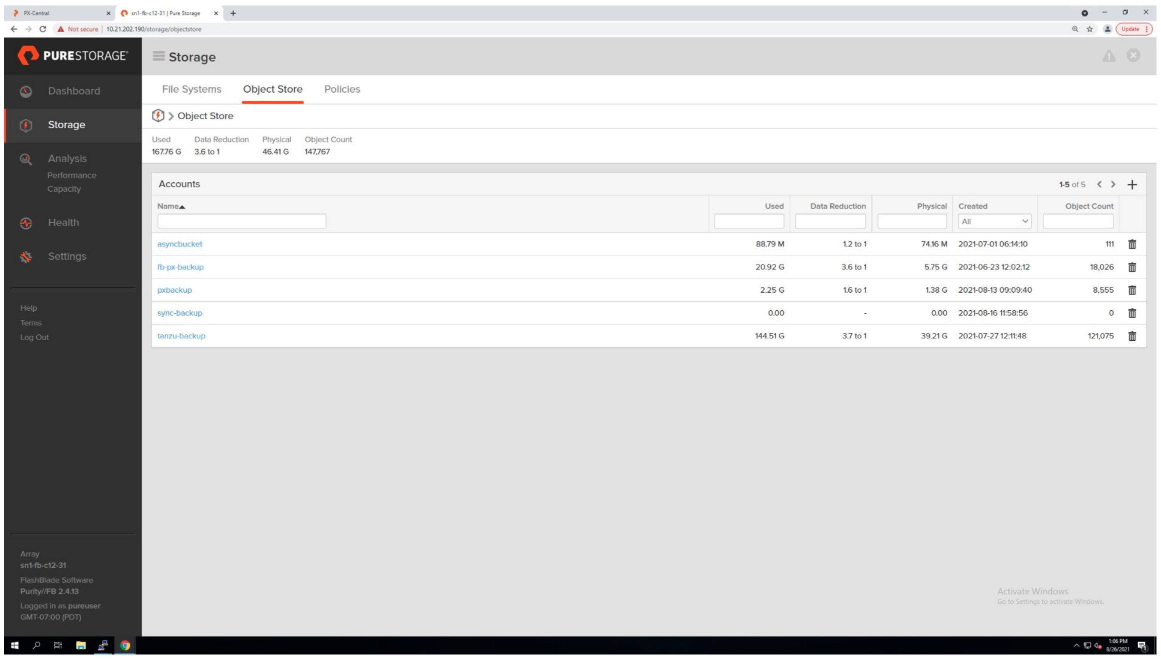

Kubernetes Stateful Applications
portworx by Pure Storage
portworx by Pure Storage
VMware Tanzu Kubernetes Grid Service
Backup / Restore
FlashBlade Object Storage
Google Cloud Object Storage
Backup
Restore
Kubernetes Stateful Applications
portworx by Pure Storage
portworx by Pure Storage
Google Kubernetes Engine (GKE) cluster

Figure 3. VMware Tanzu Kubernetes Grid clusters

Local Backup and Restore

This was completely an on-prem deployment scenario. Applications, VMware Tanzu Kubernetes Grid clusters, and the backup repository are configured on-prem. This is important when we have accidental data corruption or deletion and we want to restore from the last backup snapshot that is stored in our on-prem S3 compatible object repository. To set this up, follow these steps:

1. PX-Backup is deployed on a dedicated VMware Tanzu Kubernetes Grid cluster, and we have added a second VMware Tanzu Kubernetes Grid cluster that is running our applications.
2. For the backup repository, let's add an S3 bucket from a Pure FlashBlade system. Log into the FlashBlade UI and navigate to the **Storage** tab on the left. Next, select **Object Store** and click the **+** sign on the right to add a new account.

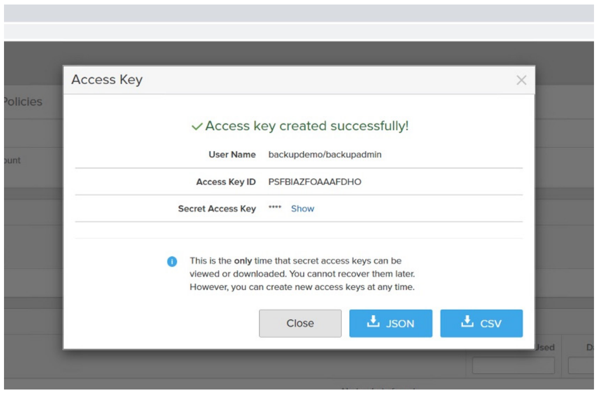


Used	Data Reduction	Physical	Object Count
167.76 G	3.6 to 1	46.41 G	147,767

Accounts 1-5 of 5

Name	Used	Data Reduction	Physical	Created	Object Count
asyncbucket	88.79 M	1.2 to 1	74.16 M	2021-07-01 06:14:10	111
fb-px-backup	20.92 G	3.6 to 1	5.75 G	2021-06-23 12:02:12	18,026
pxbackup	2.25 G	1.6 to 1	1.38 G	2021-08-13 09:09:40	8,555
sync-backup	0.00	-	0.00	2021-08-16 11:58:56	0
tanzu-backup	144.51 G	3.7 to 1	39.21 G	2021-07-27 12:11:48	121,075

3. Give the account a name, and then click **Create**. Click on the account that you created and create a new user and its access keys. Next, create a new bucket. Make a note of the bucket name and the Access Key ID and Secret Access Key.



✓ Access key created successfully!

User Name	backupdemo/backupadmin
Access Key ID	PSFBIAZFOAAAFDHO
Secret Access Key	****

This is the **only** time that secret access keys can be viewed or downloaded. You cannot recover them later. However, you can create new access keys at any time.

4. Navigate back to the PX-Backup UI and add a new AWS S3 compliant cloud account and backup location that maps back to the FlashBlade bucket.
5. Next, let's create a new backup. Click the VMware Tanzu Kubernetes Grid cluster that is running the application. Select the namespace that is running your application. PX-Backup allows users to customize backup jobs using namespaces, labels, and resource type.

   You can create different backup jobs. For example, if you have one application per namespace, you can create a backup job per namespace. If you have your Kubernetes objects labeled using a specific key-value pair, you can use that to create a backup job. Or, if you only want to back up specific Kubernetes object types, you can do that too. PX-Backup offers users complete flexibility by allowing them to choose how they want to protect their applications.

6. Once you have selected the objects that you want to back up, click the **Backup** button. Here you can enter and select the following details:

   a. Name for the backup job

   b. Backup location

   c. On-demand backup jobs or a scheduled backup job

   d. Pre- and post-backup rules for application-consistent backups

   e. Any labels you want to specify for the backup job

7. Next, click **Create**. You can monitor the backup jobs from the Backups tab.
8. The Backup Timeline gives administrators a single pane of glass view to monitor all backup jobs for a cluster over the past 24 hours or the past 30 days.
9. Once your backup job is successful, you can use that backup job to restore your application.
10. To restore your application, select the backup job that you want to restore from and click the **:** icon on the right. Next, click **Restore**. You can restore an application to the same or a different VMware Tanzu Kubernetes Grid cluster. You can also restore it to the same namespace and replace existing resources, or you can restore your application to a different namespace. PX-Backup also allows you to just restore specific Kubernetes objects, so if you only want to restore the config maps and service objects but not persistent volumes, PX-Backup allows you to do that as well.
11. Once your application is restored, you can monitor the restore and all previous restore operations from the Restore tab.

Remote Backup and Restore

In this scenario, we are still running our applications on-prem on VMware Tanzu Kubernetes Grid clusters, but instead of restoring them to the same or another TKG cluster, we will restore the application to a public cloud-managed cluster.

1. Deploy PX-Backup on a dedicated VMware Tanzu Kubernetes Grid cluster and connect another VMware Tanzu Kubernetes Grid cluster running your applications. Add a Google Kubernetes Engine (GKE) cluster as the secondary cluster that will be used to restore the applications.
2. In this scenario, we are adding another backup repository in Google Cloud to store our backup snapshots outside of our on-prem data center as well.
3. To add a Google Cloud Object Storage bucket as a backup location, create a new Google Cloud service account and use the PX-Backup UI to add a new cloud account by using the JSON key for that service account.
4. Once you have added the cloud account, you can add a backup location for a Google Cloud object storage bucket by using the name of the bucket and the cloud account name that you just created in PX-Backup.
5. Next, you can start creating new backup jobs and use the remote backup repository to store your backup objects.
6. In case you need to restore from these remote backups, you can restore them to the original VMware Tanzu Kubernetes Grid clusters, or you can restore them to a Google Kubernetes Engine cluster.

This scenario is useful when your on-prem VMware Tanzu Kubernetes Grid cluster is unavailable or your on-prem backup repository is offline.

Disaster Recovery for VMware Tanzu

Portworx PX-DR allows organizations to build robust disaster recovery plans for applications running on VMware Tanzu Kubernetes Grid clusters. Any event that causes your applications to go offline counts as a disaster event. This can include things like natural disasters (floods and earthquakes), technical failures (network outage or power failures), or human actions (accidental misconfiguration, unauthorized access, and modification). For a business to recover from any of these events and continue serving traffic to customers, it requires proper planning and infrastructure. When planning for disasters, two main objectives will influence decisions:

- **Recovery Point Objective (RPO):** This is the maximum acceptable amount of time since the last data recovery point. In other words, it is the amount of data loss that your business can afford.
- **Recovery Time Objective (RTO):** This is the maximum acceptable delay between the interruption of service and restoration of service. In other words, it is the amount of time it takes to fully recover from a disaster.

Defining RPO and RTO requirements for each application is important because the greater the number of applications that require an RPO of zero (0) and the minimum amount of RTO, the higher the cost associated with deploying such an architecture. Organizations must find the right balance between application availability needs and the cost associated with designing and implementing such a solution.

Before configuring PX-DR in a synchronous or an asynchronous DR configuration, review the following terms:

- **ClusterPair:** To failover an application running on one Kubernetes cluster to another Kubernetes cluster, you need to migrate the resources between them. A *ClusterPair* is a trust object that is required for communication between the source and target Kubernetes clusters. This creates a pairing between the two clusters so that all Kubernetes resources can be migrated between them.

- **SchedulePolicy:** Schedule policies are used to specify when Portworx should trigger a specific operation. Schedule policies do not contain any actions themselves. A *SchedulePolicy* object has the following sections:
  - Interval: For interval operations, Portworx will trigger the operation at the specified frequency.
  - Daily: For daily operations, Portworx will trigger the operation at the specified time every day.
  - Weekly: For weekly operations, Portworx will trigger the operation at the specified day and time every week.
  - Monthly: For monthly operations, Portworx will trigger the operation at the specified day and time every month.
- **MigrationSchedule:** A *MigrationSchedule* object defines the actions that need to be taken when migrating resources from the source to the target VMware Tanzu Kubernetes Grid cluster. A MigrationSchedule object has the following sections:
  - Namespace: A MigrationSchedule object is created per namespace. You need to create a MigrationSchedule object for each application running in a different namespace.
  - ClusterPair: Each MigrationSchedule object uses a ClusterPair object to migrate resources between the source and the target Kubernetes clusters.
  - includeResources: Setting this flag to "true" implies that we want each migration to include all the different Kubernetes objects inside a namespace.
  - startApplications: Setting this flag to "false" tells Kubernetes to not deploy any pods on the target cluster until we need to failover an application from the source to the target site.
  - includeVolumes: Setting this flag to "true" implies that we need the migration object to copy all the data from the source to the target Kubernetes cluster.
  - schedulePolicyName: Each MigrationSchedule needs to be associated with a SchedulePolicy object for Portworx to trigger the migration operation at the intervals defined in the SchedulePolicy object.

Synchronous DR

For modern applications that need an RPO of zero (0), PX-DR can enable administrators to deploy a synchronous DR solution in which a single Portworx cluster is stretched across two different VMware Tanzu Kubernetes Grid clusters. These VMware Tanzu Kubernetes Grid clusters can reside on the same or different vSphere clusters in the same or different on-prem data centers if they can meet the 10ms round trip latency requirement for sync DR.

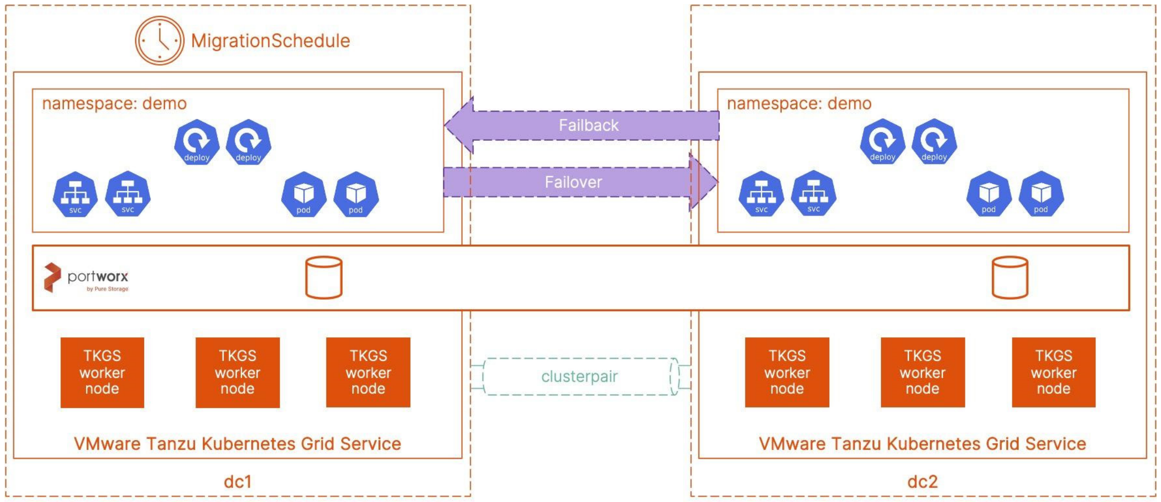


Figure 4. A Portworx cluster can be stretched across two different VMware Tanzu Kubernetes Grid clusters

Sync DR: Deployment and Validation

To deploy a sync DR topology using two VMware Tanzu Kubernetes Grid clusters, use the following steps:

> NOTE: PX-Enterprise release 2.8.2+ must be installed to use SyncDR.

1. Deploy two TKG clusters using individual YAML files. These clusters can be as small as one (1) control plane and three (3) worker nodes.
2. Once the TKG clusters are online, log into both of the clusters and label the worker nodes as site-a and site-b, respectively, using the commands below:

```
kubectl label nodes <<site-a-worker-node-1>> <<site-a-worker-node-2>> <<site-a-worker-node-3>> px-
dr=site-a
kubectl label nodes <<site-b-worker-node-1>> <<site-b-worker-node-2>> <<site-b-worker-node-3>> px-
dr=site-b
```

3. For sync DR, Portworx will deploy one stretched storage cluster. So, you will need to use an external etcd instance as the key value database (KVDB). You can deploy a single node etcd instance on a RHEL or CentOS VM using the commands below:

```
yum install etcd
#Edit the /etc/etcd/etcd.conf file and set the ETCD_INITIAL_CLUSTER,
ETCD_INITIAL_ADVERTISE_PEER_URLS, ETCD_ADVERTISE_CLIENT_URLS, ETCD_LISTEN_CLIENT_URLS to the
management IP address of the controller node to enable access by other nodes via the management
network:
#[Member]
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://10.0.0.11:2380"
ETCD_LISTEN_CLIENT_URLS="http://10.0.0.11:2379"
ETCD_NAME="controller"
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://10.0.0.11:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://10.0.0.11:2379"
ETCD_INITIAL_CLUSTER="controller=http://10.0.0.11:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster-01"
ETCD_INITIAL_CLUSTER_STATE="new"

systemctl enable etcd
systemctl start etcd
```

Verify that your TKG clusters can talk to the etcd instance over port 2379. For production workloads, we recommend installing a 3-node etcd cluster.

4. Next, generate a PX-cluster configuration with an external etcd instance using PX-Central and edit this YAML file to create two configuration files—one for site-a and one for site-b.
5. Edit each of the YAML files and edit the following arguments for the Portworx container:
    a. -c: You must use the same -c parameter for both TKG clusters. This parameter represents the Portworx cluster ID.
    b. -cluster_domain: This parameter is used to define sites/cluster domains inside the stretched Portworx cluster.

c. --node_pool_label: Use the key that you used in step 2 to label the nodes on each cluster.

d. -k: Verify that this is the correct endpoint for your external etcd installation.

```
args:
  ["-k", "etcd:http://10.21.143.196:2379", "-c", "px-cluster-68d810ba-a73c-44c5-9395-b736e15ba68e",
"-cluster_domain", "dc1", "--node_pool_label=px-dr", "-s", "sc=vmfs-cns,size=150", "-secret_type",
"k8s", "-x", "kubernetes"]
```

6. Deploy these YAML files against your TKG clusters and wait for the Portworx cluster to come online. You can use the `storkctl get clusterdomainsstatus` command to verify that cluster domains are configured correctly.
7. Next, create an object store credential and a ClusterPair object. Object store credentials can be created by using the following command:

```
/opt/pwx/bin/pxctl credentials create \
--provider s3 \
--s3-access-key <aws_access_key> \
--s3-secret-key <aws_secret_key> \
--s3-region us-east-1  \
--s3-endpoint s3.amazonaws.com \
--s3-storage-class STANDARD \
clusterPair_<UUID_of_destination_cluster>
```

You can get the UUID of the site-b cluster using the following command:

```
kubectl exec -it <<portworx_pod>> -n kube-system -- nsenter --mount=/host_proc/1/ns/mnt bash
From inside the Portworx pod: /opt/pwx/bin/pxctl status
```

8. Apply the PX-DR license to the Portworx cluster from one of the Portworx pods using:

```
/opt/pwx/bin/pxctl license activate <<license_key>>
```

9. Next, create a ClusterPair spec on the site-b cluster. The ClusterPair specification is a namespaced object. So, ensure that you have the same namespace created on both sites.

```
storkctl generate clusterpair -n demo clusterpairname > clusterpairname.yaml
```

10. Edit the YAML file and remove the “<<insert_storage_options_here>>:” line. In sync DR configuration, the replication is handled at the storage layer.
11. Next, log into the site-a cluster and apply this configuration file using `kubectl apply -f clusterpairname.yaml`.
12. Verify that the ClusterPair object is online by using `storkctl get clusterpair -n demo.`
13. Next, go ahead and create a schedule policy that will be used by the MigrationSchedule object to copy Kubernetes objects and metadata between the two clusters.

```
apiVersion: stork.libopenstorage.org/v1alpha1
kind: SchedulePolicy
metadata:
  name: testpolicy
  namespace: demo
policy:
  interval:
    intervalMinutes: 1
  daily:
    time: "10:14PM"
  weekly:
    day: "Thursday"
    time: "10:13PM"
  monthly:
    date: 14
    time: "8:05PM"

kubectl apply -f testpolicy.yaml
```

14. Next, create a MigrationSchedule object that will replicate the application from site-a to site-b.

```
apiVersion: stork.libopenstorage.org/v1alpha1
kind: MigrationSchedule
metadata:
  name: demomigrationschedule
  namespace: demo
spec:
  template:
    spec:
      clusterPair: clusterpairname
      includeResources: true
      startApplications: false
      includeVolumes: false
      namespaces:
      - migrationnamespace
  schedulePolicyName: testpolicy

kubectl apply -f migrationschedule.yaml
```

15. Next, use the following commands to verify the status of the migration schedules and see if the different migration jobs are successful.

```
storkctl get migrationschedule -n demo
kubectl get migrationschedule -n demo
storkctl get migrations -n demo
kubectl get migrations -n demo
```

You have successfully deployed a sync DR topology for your TKG clusters. Next, use the following steps to failover and failback your application from site-a to site-b and vice-versa.

Sync DR Failover

To failover an application, you need to tell Stork and Portworx that one of your TKG clusters is down and inactive.

1. To initiate a failover, we first need to deactivate or mark our source cluster as inactive. After deactivating, the site-a cluster goes out of sync.

```
storkctl deactivate clusterdomain site-a
storkctl get clusterdomainsstatus
```

2. If you are just simulating a failover to test sync DR, you need to scale down your applications on the source side. In case of a disaster event, your source cluster will be offline, and the pods won't be in a running state.

```
kubectl scale --replicas 0 deployment/mysql -n demo
```

3. Now we can log into the site-b cluster and activate the migration to bring the application online on the secondary site.

```
storkctl activate migrations -n demo
```

4. Verify that your application is online by using the following command:

```
kubectl get all -n demo
```

You have successfully performed a failover operation in your sync DR topology.

Sync DR Failback

Once the primary cluster is back up and running, the Portworx nodes in that cluster will not immediately rejoin the stretched cluster. They will stay in an "Out of Quorum" state until you explicitly activate the cluster domain.

1. Activate site-a using storkctl:

```
storkctl activate clusterdomain site-a
storkctl get clusterdomainsstatus
```

2. Stop the application on the secondary site-b cluster using `kubectl scale --replicas 0 deployment/mysql -n demo`.
3. Start the application on the primary site-a cluster using `kubectl scale --replicas 1 deployment/mysql -n` demo.
4. Next, let's resume MigrationSchedule by using this command:

```
storkctl resume migrationschedule demo-migrationschedule -n demo
```

You have successfully performed a failback operation in your sync DR topology.

Asynchronous DR

In addition to synchronous DR, PX-DR can be deployed in an async DR configuration. This is useful in scenarios where your TKG clusters might be running in different data centers or different vSphere clusters or where the latency requirements of sync DR replication cannot be satisfied.

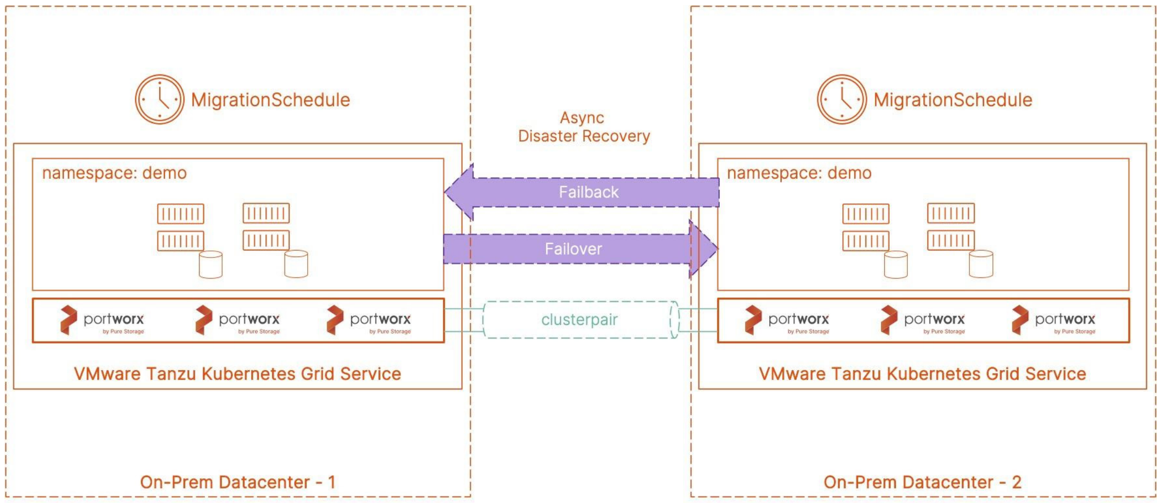


Figure 5. PX-DR can be deployed in an async DR configuration

Async DR: Deployment and Validation

Deploy a source and a destination TKG cluster with a minimum configuration of one (1) control plane and three (3) worker nodes.

1. Install Portworx version 2.8 using PX-Central. Unlike sync DR, you don't need to label the nodes or edit the Portworx spec file to add cluster domain and node label arguments. You can generate different Portworx specifications with a built-in etcd instance and apply it to your TKG cluster.
2. Once the Portworx pods are up and running in the kube-system namespace, exec into the Portworx pod on both clusters and apply the PX-DR license using the following command:

```
kubectl exec -it <<portworx_pod>> -n kube-system -- bash
From inside the Portworx pod: /opt/pwx/bin/pxctl license activate <<license_key>>
```

3. Edit the portworx-service in the kube-system namespace and change the service type to LoadBalancer. If you are using the Portworx operator-based deployment, you will have to add the following annotation to the Portworx storage cluster configuration:

```
metadata:
  annotations:
    portworx.io/service-type: "LoadBalancer"
```

4. Next, let's create an object store credential and a ClusterPair object. Object store credentials can be created by using the following command. You need to create the same credentials on both Portworx storage clusters.

```
/opt/pwx/bin/pxctl credentials create \
--provider s3 \
--s3-access-key <aws_access_key> \
--s3-secret-key <aws_secret_key> \
--s3-region us-east-1  \
--s3-endpoint s3.amazonaws.com \
--s3-storage-class STANDARD \
clusterPair_<UUID_of_destination_cluster>
```

5. You can get the UUID of the site-b cluster using the following command:

```
kubectl exec -it <<portworx_pod>> -n kube-system -- bash
From inside the Portworx pod: /opt/pwx/bin/pxctl status
```

6. Next, let's create a ClusterPair spec on the site-b cluster. A ClusterPair specification is created per namespace. So, ensure that you have the same namespace created on both sites.

```
storkctl generate clusterpair -n demo clusterpairname > clusterpairname.yaml
```

7. Once you have the YAML file generated, edit it to update the following options:

```
options:
    ip:     <ip_of_remote_px_node>
    port:   <port_of_remote_px_node_default_9001>
    token:  <destination_cluster_token>
    mode: DisasterRecovery
```

8. To fetch the destination Portworx cluster token, use the following command:

```
kubectl exec -it <<portworx_pod>> -n kube-system -- bash
/opt/pwx/bin/pxctl cluster token show
```

9. Use the updated YAML file and apply it against your source cluster:

```
kubectl apply -f clusterpair.yaml
```

10. Verify that the ClusterPair object is online by using this command:

```
storkctl get clusterpair -n demo
```

11. Create a SchedulePolicy object like the one you created in the sync DR configuration (Step 13 on your source cluster).

12. Next, create a MigrationSchedule object on the source cluster:

```
apiVersion: stork.libopenstorage.org/v1alpha1
kind: MigrationSchedule
metadata:
  name: demomigrationschedule
  namespace: demo
spec:
  template:
    spec:
      clusterPair: remotecluster
      includeResources: true
      startApplications: false
      namespaces:
      - demo
  schedulePolicyName: testpolicy

kubectl apply -f demomigrationschedule.yaml
```

In async DR scenarios, we will not use the "includeVolumes:false" parameters because we want PX-DR to migrate incremental copies of our persistent volumes from the source to the destination cluster as well.

13. You can monitor MigrationSchedule and the migration objects by using the following commands:

```
kubectl get migrationschedule -n demo
kubectl get migrations -n demo
kubectl describe migrationschedule -n demo
storkctl get migrations -n demo
storkctl get migrationschedule -n demo
```

You have successfully deployed an async DR configuration for your TKG cluster.

To expedite failback operations for AsyncDR, you should also create object store credentials and a ClusterPair object in the reverse direction as well (destination to source cluster).

Async DR Failover

To perform a failover operation from the source to the destination cluster, use the following steps:

1. If your source cluster is still online, use the following commands to suspend MigrationSchedule and scale down your application:

```
storkctl suspend migrationschedule demo-migrationschedule -n demo
kubectl scale --replicas 0 deployment/mysql -n demo
```

2. You can now log into your destination cluster and activate the migration using `storkctl activate migrations -n demo`.

You will see pods getting deployed for your application in the demo namespace. Use the `kubectl get all -n demo` command to verify that your application is up and running on the destination cluster:

You have successfully performed an async DR failover operation from the source to the destination site.

Async DR Failback

When your source cluster is back up and running, you can perform a failback operation to move the application back to the source cluster. Use the following steps to failback your application.

1. Create a new schedule policy and a migration schedule object on the destination cluster. All of this can be configured by using the commands in the async DR deployment and validation section.
2. Once the first migration is successful, you can suspend MigrationSchedule from the destination to the source site and scale down ther application

```
storkctl suspend migrationschedule demo-migrationschedule-failback -n demo
kubectl scale --replicas 0 deployment/mysql -n demo
```

3. Activate the migration on the source cluster to bring your application back online using `storkctl activate migrations -n demo.`
4. You will see pods getting deployed for your application in the demo namespace. Use the `kubectl get all -n demo` command to verify that your application is up and running on the destination cluster.

You have successfully performed an async DR failback operation from the source to the destination site.

VMware Tanzu Data Services and Portworx

VMware Tanzu Data Services offers an operator-based method to deploy distributed data services on VMware Tanzu Kubernetes Grid clusters. VMware Tanzu Data Services include the following:

- **VMware Tanzu RabbitMQ:** A fast and dependable message broker that is lightweight and easy to deploy on-premises and in the cloud
- **VMware Tanzu SQL:** Relational database-as-a-service for developers that is fully compatible with MySQL and PostgreSQL
- **VMware Tanzu Greenplum:** Massively parallel Postgres for analytics, machine learning, and AI
- **VMware Tanzu GemFire:** In-memory data grid powered by Apache Geode

VMware Tanzu Data Services provide an easy-to-install and operate experience for distributed, stateful applications. But to provide application resiliency at the storage layer and to provide all the advanced storage features needed, we recommend deploying VMware Tanzu Data Services on TKG clusters running Portworx. All the use cases that we discussed in the document—like high availability and replication, automated storage capacity management, data protection, and disaster recovery—work with all the VMware Tanzu Data Services as well.

Kubernetes Storage for Tanzu Data Services

VMware Tanzu Data Services like Tanzu RabbitMQ and Tanzu SQL can be deployed with optional high availability built into the application layer. But to provide additional resiliency and features, we tested VMware Tanzu Data Services with Portworx.

Replication and High Availability

Each of the VMware Tanzu Data Services has different requirements when it comes to replication and high availability. Portworx allows you to define application-specific storage classes for each of the VMware Tanzu Data Services and provide the best storage layer for that application. You can configure individual storage classes with parameters like the replication factor, volume groups, and io_priority. Using Portworx as the Kubernetes storage layer ensures that any data that is written by the application is stored reliably and can tolerate common failures—like pod restarts, node restarts, and node failures.—without any data loss.

- Storage Class for VMware Tanzu RabbitMQ:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: portworx-tanzu-rabbitmq
parameters:
  io_priority: high
  repl: "2"
  group: "rmq_vg"
provisioner: kubernetes.io/portworx-volume
allowVolumeExpansion: true
reclaimPolicy: Delete
```

- Storage Class for VMware Tanzu SQL – PostgreSQL:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: portworx-tanzu-postgres
provisioner: kubernetes.io/portworx-volume
allowVolumeExpansion: true
parameters:
  repl: "2"
  priority_io: “high”
  io_profile: “db_remote”
```

By leveraging different storage classes, you can configure different levels of replication factors for your applications.

Application Performance

Portworx storage classes also allow you to configure different IO profiles for your application. Each application writes data to disk in a different way, and Portworx can interpret the IO profile and present the best performance for your application. You can start with one of the five IO profiles that Portworx supports, and if you are not sure which one to use, you can just use “auto,” and Portworx will analyze your I/O pattern and choose the best profile for you.

- db: This profile is for databases that result in many flush operations on the disk.
- db_remote: This profile implements a write-back flush coalescing algorithm to coalesce multiple syncs that occur within a 50ms window into a single sync.
- sequential: This profile optimizes the read-ahead algorithm for sequential workloads, such as backup operations.

- random: This profile records the I/O pattern of recent access and optimizes the read-ahead and data layout algorithms for workloads involving short-term random patterns.
- auto: If you don't know which IO profile to select, you can use the "auto" profile. Portworx continuously analyzes the I/O pattern of the traffic in the background and applies the most appropriate pattern it sees.

In addition to configuring IO profiles, Portworx allows you to ensure data locality for your application pods by leveraging STORK (Storage Orchestrator Runtime for Kubernetes).

Automated Storage Capacity Management Using PX-Autopilot

PX-Autopilot allows you to create rules to automatically expand persistent volumes and Portworx storage pools as well as rebalance Portworx storage pools that are running Tanzu Data Services.

Data Protection for VMware Tanzu Data Services

Each of the VMware Tanzu Data Services supports a Kubernetes backup and restore solution. For example, VMware Tanzu SQL relies on pgBackRest for Postgres and Percona XtraBackup for Tanzu SQL with MySQL. But when you are running distributed stateful applications in production, you need a data protection solution that is consistent across all applications. PX-Backup provides a single solution that is easy to deploy and operate, but also allows you to provide backup and restore capabilities for all VMware Tanzu Data Services. PX-Backup can protect Tanzu Data Services that are running on-prem on VMware Tanzu Kubernetes Grid clusters, or it can also help you protect these data services if they are running on any managed public cloud Kubernetes service like GKE, EKS, AKS, or VMware Tanzu Kubernetes Grid running on AWS or Azure as well.

Using a single Kubernetes backup and restore tool to protect containerized applications reduces the amount of overhead involved compared to learning, deploying, and maintaining different tools for different applications. PX-Backup version 2.0 also provides role-based access control (RBAC) capabilities, so individual developers or application owners can protect their applications on their own.

PX-Backup offers the following users and responsibilities out of the box. But you can customize things for your organization as well.

- **Administrator:** This persona can add users, create custom roles, and create and share cloud accounts and backup locations.
- **Application owner:** This persona can create backup schedules, pre- and post-backup rules, and use existing cloud accounts.
- **Developer:** This persona can backup and restore their own applications on a self-service basis by leveraging existing cloud accounts, backup locations, backup schedule policies, and rules created by the above personas. These backups are granular, and they can be created per namespace, Kubernetes resource type, or Kubernetes label.

To deploy and install PX-Backup for your VMware Tanzu Data Services, refer to the deployment steps for PX-Backup in this document.

Disaster Recovery for Tanzu Data Services

Regardless of whether your production applications are running as virtual machines or containers, you need a robust disaster recovery solution to ensure business continuity. The same applies to VMware Tanzu Data Services. To ensure that VMware

Tanzu Data Services can meet their predefined, agreed-upon SLAs, PX-DR allows you to create synchronous and asynchronous disaster recovery solutions for these data services.

In addition to providing a unified and consistent solution that you can use with your VMware Tanzu Data Services, PX-DR also helps developers and operators to architect a robust DR solution. Sync DR can avoid data loss completely with zero RPO and very low RTO, and async DR can help specify RPO requirements using Portworx schedule policies. PX-DR ensures that Tanzu Data Services can come back online at the secondary site in case of a disaster event.

To deploy and configure PX-DR for VMware Tanzu Data Services, you can use the deployment instructions that we covered earlier in this document when discussing PX-DR in general. Based on your application SLAs, you can choose to deploy either synchronous replication with PX-DR sync DR or asynchronous replication with PX-DR async DR.

Conclusion

Portworx provides the best-in-class enterprise-grade data services for any application running on VMware Tanzu Kubernetes Grid clusters at any scale. Solving for speed, density, and scale, Portworx not only enables efficient, automatic provisioning on top of your VMware Tanzu Kubernetes Grid clusters, but it also provides advanced features like high availability and replication, automated capacity management, and dynamic provisioning using application-specific storage classes (IO_profiles, IO_priority, etc.). Portworx also provides customers with a complete disaster recovery and business continuity solution with PX-DR. PX-DR allows customers to build synchronous and asynchronous DR solutions for their VMware Tanzu Kubernetes Grid clusters. In addition to DR, PX-Backup completes your data protection solution with a Kubernetes-native backup and restore solution that can be leveraged to build architectures for local or remote backup and restore activities. Portworx from Pure Storage is the gold standard when it comes to Kubernetes Data Services, and it brings all of its capabilities to VMware Tanzu Kubernetes Grid clusters.

Additional Resources

- Learn more with Portworx blogs.
- Try out Portworx demos.
- Get more information with the VMware Tanzu and FlashArray PVD.

About the Author

Bhavin Shah is a technical marketing manager at Pure Storage. He is responsible for designing and architecting solutions around Portworx Enterprise, Backup, and Disaster Recovery for Kubernetes. Bhavin has worked in the data management ecosystem for the past eight years, focusing on building solutions around converged infrastructure, hyperconverged infrastructure, cloud, and Kubernetes. Bhavin joined Pure Storage in March 2021 and works in the Cloud Native Business Unit at Pure Storage.

©2021 Pure Storage, the Pure P Logo, and the marks on the Pure Trademark List at https://www.purestorage.com/legal/productenduserinfo.html are trademarks of Pure Storage, Inc. Other names are trademarks of their respective owners. Use of Pure Storage Products and Programs are covered by End User Agreements, IP, and other terms, available at: https://www.purestorage.com/legal/productenduserinfo.html and https://www.purestorage.com/patents.

The Pure Storage products and programs described in this documentation are distributed under a license agreement restricting the use, copying, distribution, and decompilation/reverse engineering of the products. No part of this documentation may be reproduced in any form by any means without prior written authorization from Pure Storage, Inc. and its licensors, if any. Pure Storage may make improvements and/or changes in the Pure Storage products and/or the programs described in this documentation at any time without notice.

THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. PURE STORAGE SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

Pure Storage, Inc.
650 Castro Street, #400
Mountain View, CA 94041