PURESTORAGE®
Uncomplicate Data Storage, Forever

TECHNICAL WHITE PAPER

MongoDB on Portworx

Deployment Guide

Contents

Introduction 3

How to Use This Guide 3

Considerations 3

Prerequisites 3

- Deploy Kubernetes Cluster 3
- Install and configure Portworx Enterprise 3
- StorageClass 5
- Install and configure Portworx Backup 5
- Portworx Data Services (PDS) 6

MongoDB Deployment—MongoDB Enterprise Kubernetes Operator 6

- Benefits 6
- Requirements 6
- Replica Set Deployment 8
- Monitor MongoDB Deployments 9

Data Protection using Portworx Backup 10

- Prerequisites 10
- Backup and Restore Kubernetes Cluster Objects 10

High Availability with Portworx 14

- Using MongoDB Built-in Features 14
- Using Portworx Enterprise 14

Volume Snapshots using Portworx 19

- Steps to perform VolumeSnapshot 19
- Steps to perform GroupVolumeSnapshot 22

Scalability with Portworx 23

- Steps to scale/expand PVC 24

MongoDB Deployments - Portworx Data Services (PDS) 26

- Benefits 26
- Requirements 26
- Deploy MongoDB as a Data Service 26
- Monitor MongoDB Deployment 30
- Protecting MongoDB Deployments using PDS 32
- Scale MongoDB Deployments using PDS 33

Additional References 36

Introduction

Containerizing MongoDB workloads offers numerous advantages, including enhanced scalability, faster deployments, improved resource management and developer-friendly environments that enhance user experiences.

Portworx® is a container-native storage solution explicitly designed for orchestrators such as Kubernetes. Its primary purpose is to oversee and furnish persistent storage solutions for containerized applications. Deploying MongoDB on Portworx is an effective way to combine the power of a versatile NoSQL database with a robust container-native storage solution to create a reliable and scalable database infrastructure in containerized environments, making it well-suited for modern, cloud-native applications.

How to Use This Guide

This deployment guide is designed for individuals with a foundational understanding of Kubernetes and database concepts. It provides a detailed, step-by-step walkthrough for deploying MongoDB alongside Portworx Enterprise and Portworx Data Services, tailored to address specific business requirements. The intended readership for this document encompasses platform engineers, system architects, database administrators, storage administrators and IT professionals.

Considerations

- This deployment guide can be used to deploy MongoDB and Portworx on several kubernetes platforms such as Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), Red Hat Openshift or Kubernetes.
- All deployment topologies showcased here are for "Non-Encrypted Connections".

Prerequisites

The following prerequisites are required for a successful MongoDB deployment withPortworx:

Deploy Kubernetes Cluster

Fully functional kubernetes cluster deployed with a supported Portworx environment.

Install and configure Portworx Enterprise

Portworx on the Kubernetes can be installed using an operator based deployment.

The Portworx operator is purpose-built for the management and automation of Portworx storage clusters within Kubernetes environments. It requires specific permissions to orchestrate and supervise the storage cluster within the Kubernetes cluster. Various Kubernetes platforms such as on-premises and cloud environments like AWS, Google Cloud, Azure and Oracle provide support for such operators.

The deployment manifest can be customized and downloaded from PX-Central, a web-based interface. Users can select Portworx's product catalog and tailor platform-specific parameters to their needs.

Portworx provides a range of editions for its container-native storage solution, each tailored to specific use cases and requirements.

Portworx Essentials: This is a no-cost edition, ideal for development environments. While it offers fundamental container storage features, it comes with certain limitations.

Portworx Enterprise: This is a licensed core edition designed for production-grade environments. It comes with an array of advanced functionalities, including data encryption, protection, automated failover and snapshot capabilities.

When considering Portworx, it's typically recommended to go for Portworx Enterprise, as it provides access to the full suite of features and allows for customization. This includes options such as selecting between internal and external ETCD, disk provisioning, enabling Stork, CSI integration, monitoring and more. This flexibility ensures that your storage solution aligns perfectly with your specific needs and use cases.

Requirements for installing Portworx Enterprise.

Requirements

- A valid account to login and access Portworx Central (Product catalog) to generate manifest/spec.
- Create a namespace in the Kubernetes cluster as "portworx".
- Create a secret in the target cluster with credentials (encoded to base64 format) to access the infrastructure resources to deploy the storage cluster.
- Apply the Portworx manifest/spec in the kubernetes cluster.

The prerequisites for Portworx Enterprise can be found at the following location :
https://docs.portworx.com/portworx-enterprise/install-portworx/prerequisites

Ensure Portworx Enterprise is installed and configured in the Kubernetes cluster. Portworx provides storage services and should be operational before deploying MongoDB.

A comprehensive installation procedure for Portworx Enterprise with requirements can be found on the following location:

https://docs.portworx.com/portworx-enterprise/install-portworx/kubernetes

StorageClass

A StorageClass is a provisioning system within Kubernetes that enables the dynamic allocation of persistent volumes (PVs) in a Kubernetes cluster. Kubernetes administrators define different storage classes and subsequently, pods have the capability to request the precise storage type required on-the-fly.

Define a Kubernetes StorageClass that utilizes Portworx as the storage provider. This StorageClass will be used to provision persistent volumes (PVs) for MongoDB pods.

It is advisable to configure specific parameters within the storage class when deploying database workloads with Portworx.

- *allowVolumeExpansion:* set to *true*, it allows resizing the volumes supported by the storageclass.
- *provisioner: portworx provisioner* - Volume plugin used to provision persistent volumes.
- *repl:* Indicates how many replicas of portworx volumes are to be maintained for High Availability. The minimum value to be set is 2.
- *io_profile:* The type of IO profile to be configured based on performance/workload requirements. For databases/statefulset workloads, the recommended setting is *"db_remote"*.
- *io_priority:* Indicates the priority of IOs. For databases/statefulset workloads, the recommended setting is *"high"*.
- *fs:* Indicates the file system type to be provisioned. For MongoDB, the recommended setting is *"xfs"* file system.

```
kind: StorageClass
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
metadata:
 name: px-ha-sc
provisioner: kubernetes.io/portworx-volume
parameters:
 repl: ""“2"""
 io_profile: """“db_remote""
 io_priority: """“high""
 fs: """“xfs""
```

Install and configure Portworx Backup

Portworx offers features for backup and recovery that are typically integrated with backup solutions. Portworx backup effectively, ensuring that your data is protected, recoverable and compliant with your organization's data management policies.

The prerequisites and setup procedure for Portworx Backupcan be found here.

Portworx Data Services (PDS)

Built on Portworx Enterprise, this Database Platform-as-a-Service (DBPaaS) for Kubernetes simplifies the deployment and management of data services like MongoDB within Kubernetes clusters. It is an unified, self-managed platform that automates the entire lifecycle of data services, improving availability, performance and cost efficiency.

The following links serve as convenient references for obtaining additional information about Portworx Data Services:

- To access login information, please visit: https://pds.docs.portworx.com/get-started
- For administration and operations details, please refer to: https://pds.docs.portworx.com/admin-guide

MongoDB Deployment—MongoDB Enterprise Kubernetes Operator

In this section, you will find comprehensive information covering various aspects of deploying and managing MongoDB workloads within a Kubernetes environment using the MongoDB Enterprise Kubernetes Operator. This includes details on prerequisites, deployment and monitoring techniques.

The MongoDB Enterprise Kubernetes Operator transforms the expertise required to create a MongoDB instance into a scalable, repeatable and standardized process. Kubernetes, while powerful, requires assistance in orchestrating stateful applications like databases. It must handle tasks like network configuration, persistent storage provisioning and resource allocation without the need for extensive manual intervention on each container.

The MongoDB Enterprise Kubernetes Operator efficiently manages the typical lifecycle events for a MongoDB cluster, including tasks such as storage and compute provisioning, network configuration, user setup and dynamic adjustments to these configurations. It accomplishes these tasks through the Kubernetes API and associated tools. The MongoDB Enterprise Kubernetes Operator collaborates with MongoDB Cloud Manager or Ops Manager, which further fine-tunes the MongoDB clusters.

Benefits

- The MongoDB Replica Set is automatically configured by MongoDB Ops Manager during deployment.
- MongoDB Ops Manager provides alerts for any potential risks that may arise within the deployments.
- MongoDB Ops Manager significantly simplifies the management of deployments.

Requirements

Deploying MongoDB clusters on Kubernetes using the MongoDB Enterprise Kubernetes Operator involves specific requirements to ensure a smooth and successful deployment.

1. A fully functional kubernetes cluster.
2. Deploy and configure MongoDB Enterprise Kubernetes Operator. A comprehensive setup process with prerequisites can be found on the following link https://www.mongodb.com/docs/kubernetes-operator/stable. MongoDB Enterprise Kubernetes Operator creates the following resources during deployment:

  - StatefulSets - To deploy MongoDB Ops Manager database
  - Services - Endpoints to access MongoDB Ops Manager and resources
  - Replica Sets - For Kubernetes Operator
  - Pods/containers - Runtime environment for applications/daemons

- The MongoDB Enterprise Kubernetes Operator usually initiates the deployment of MongoDB Ops Manager and multiple Custom Resource Definitions (CRDs):
- **MongoDB Ops Manager:** This is a custom resource definition responsible for deploying the MongoDB Ops Manager application, database and backup daemon. MongoDB Ops Manager monitors and manages MongoDB deployments.
- **CRDs:** The operator deploys or creates various processes, including but not limited to configuration maps, service accounts, access controls and user definitions.

3. Configure API Access—MongoDB Ops Manager

- To establish communication between the MongoDB Ops Manager and the Organization/project using APIs exclusively, programmatic access needs to be granted. This can be accomplished by creating an API key that consists of two components: a public key and a private key. It is important to configure the "Global owner" role for this key to ensure API endpoint access is provided to the MongoDB Ops Manager.
- Refer to the Managing Programmatic Access section for additional information.
- After creating the API key, save the key as a Kubernetes secret in the namespace where MongoDB deployments are targeted.

```
kubectl -n <namespace name> create secret generic <secret_name> \ --from-
literal="publicKey=<publicKey>" --from-literal="privateKey=<privateKey>
```

Replica Set Deployment

The MongoDB Replica Set instance is a multi-node deployment topology that includes a built-in high availability (HA) feature. It enables real-time data replication between the nodes. This topology is considered the optimal choice for production environments due to its robustness and reliability.

See the following sample file to create a YAML file as a replicaset instance definition and to apply it in the Kubernetes environment. (A namespace called “mongo” is created to deploy MongoDB StatefulSets)

```
#mongodb-rs.yaml
apiVersion: mongodb.com/v1
kind: MongoDB
metadata:
 name: mongodb-replicaset
spec:
 members: 3 #desired nodes
 version: "5.0.14-ent" # MongoDB version
 opsManager:
 configMapRef:
 name: my-project # Name of configmap
 credentials: organization-secret # Name of secret
 type: ReplicaSet # deployment type
 persistent: true
 podSpec:
 persistence:
 multiple:
 data:
 storage: 50Gi
 storageClass: px-ha-sc
 exposedExternally: true
```

Command to apply spec.

```
kubectl apply -f mongodb-rs.yaml -n mongo
```

This creates various resources such as StatefulSets, pods, services and persistentvolumeclaim (PVC) for the MongoDB Replica Set instances. These resources are tagged with the same project name as specified in the MongoDB Ops Manager under the designated organization.

Monitor MongoDB Deployments

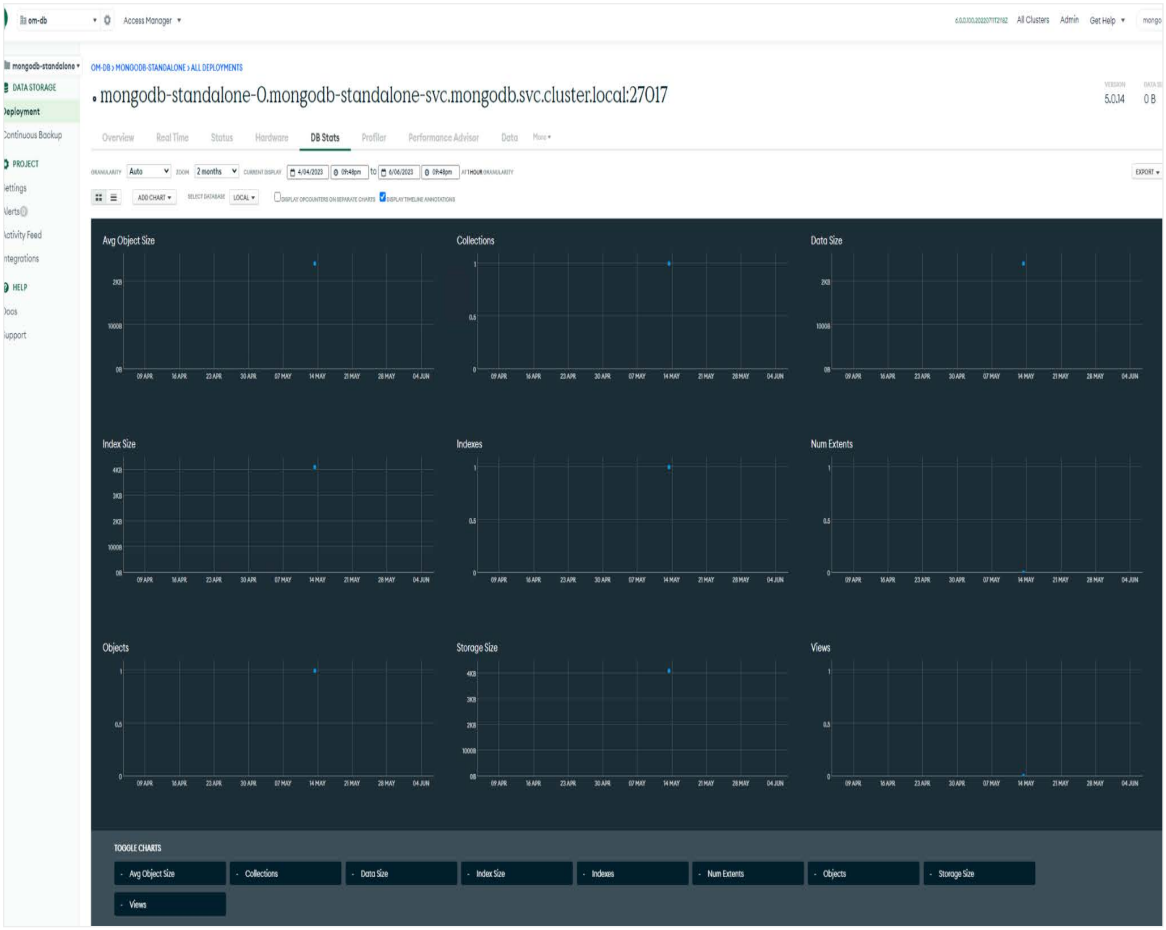
MongoDB Ops Manager simplifies the management and administration of MongoDB deployments, making it an essential tool for organizations running MongoDB in production environments. It enhances database security, availability and performance while providing valuable insights into the health and operation of MongoDB clusters.

MongoDB deployments can be managed and monitored using the MongoDB Ops Manager UI.

MongoDB Ops Manager UI displays the following metrics:

- Read/Write operations, Disk used, connections, IOPS.
- Real-time performance
- DB stats
- Hardware status
- Status

For instance, the below image is a reference for sample database statistics on MongoDB Ops Manager UI.



Data Protection using Portworx Backup

PX-Backup, or Portworx Backup, is a data protection and backup solution offered by Portworx. PX-Backup is designed to provide backup and disaster recovery capabilities for containerized applications, including those running on Kubernetes clusters. It is specifically built to work seamlessly with Portworx storage solutions, but it can also be used in conjunction with other storage systems.

It has the ability to backup and restore kubernetes native resources like deployments, secrets, configurations, service accounts etc. cluster level backup and application-consistent backups. PX-Backup allows backup and restore in multiple clusters. Portworx Backup is compatible with any Kubernetes cluster, including managed and cloud deployments.

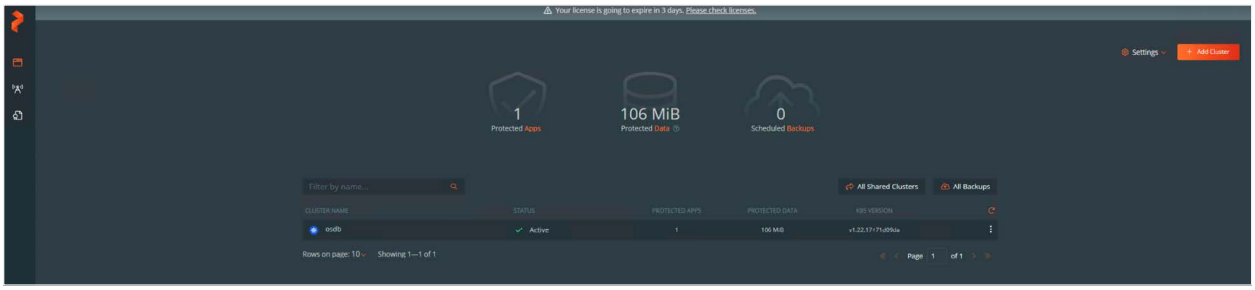
PX-Backup allows you to take on-demand and scheduled backups.

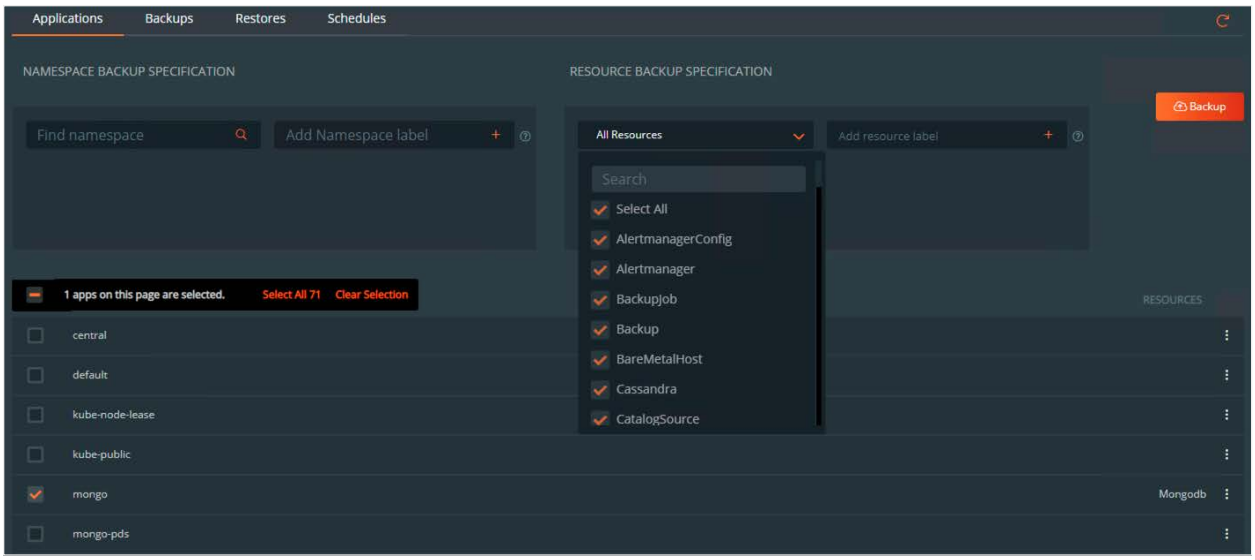
Prerequisites

- Add/configure backup target in PX-Backup UI.
- Add Pre & post-exec rules to maintain consistency.

Backup and Restore Kubernetes Cluster Objects

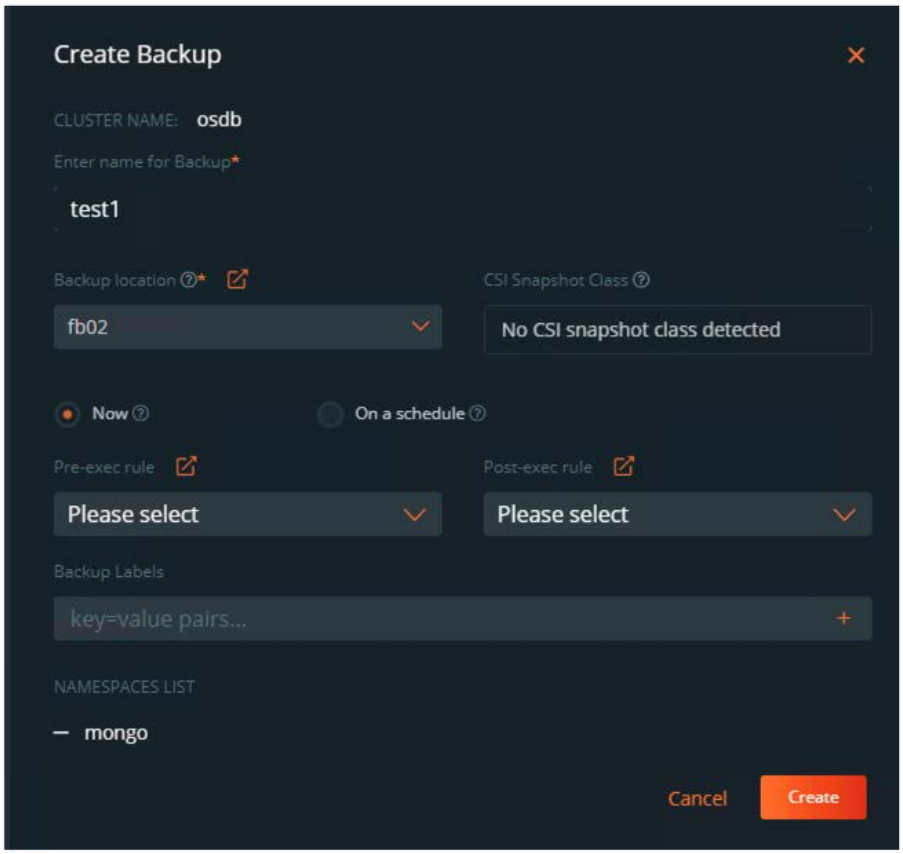
1. Login to PX-Backup UI with valid credentials.
2. Select cluster name to backup and select namespace(s) and resources. Then click **Backup**.



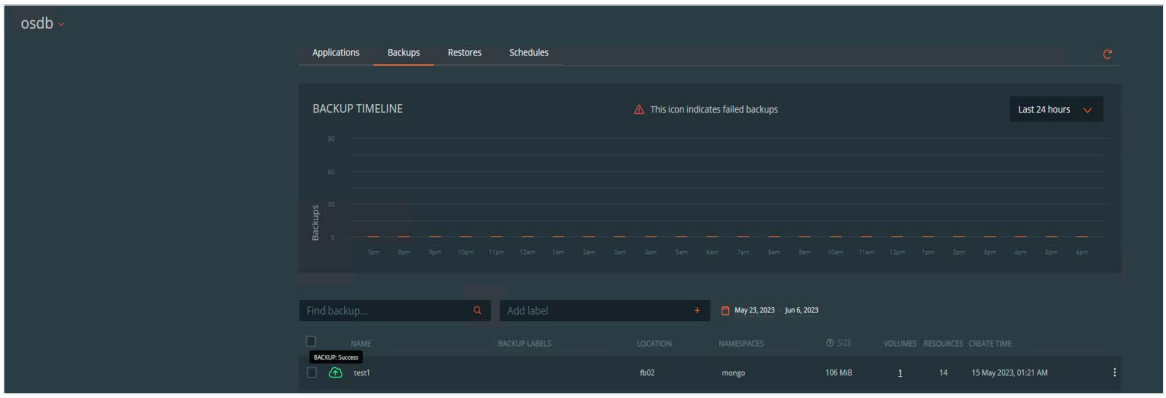


3. In the Create Backup dialog box, specify the name for backup, select the backup location and select pre-exec and post-exec rules from the dropdown list. Then click **Create**.

   Refer to the Portworx guide on configuring 3D snapshots for detailed instructions on creating pre and post snapshot rules for MongoDB.



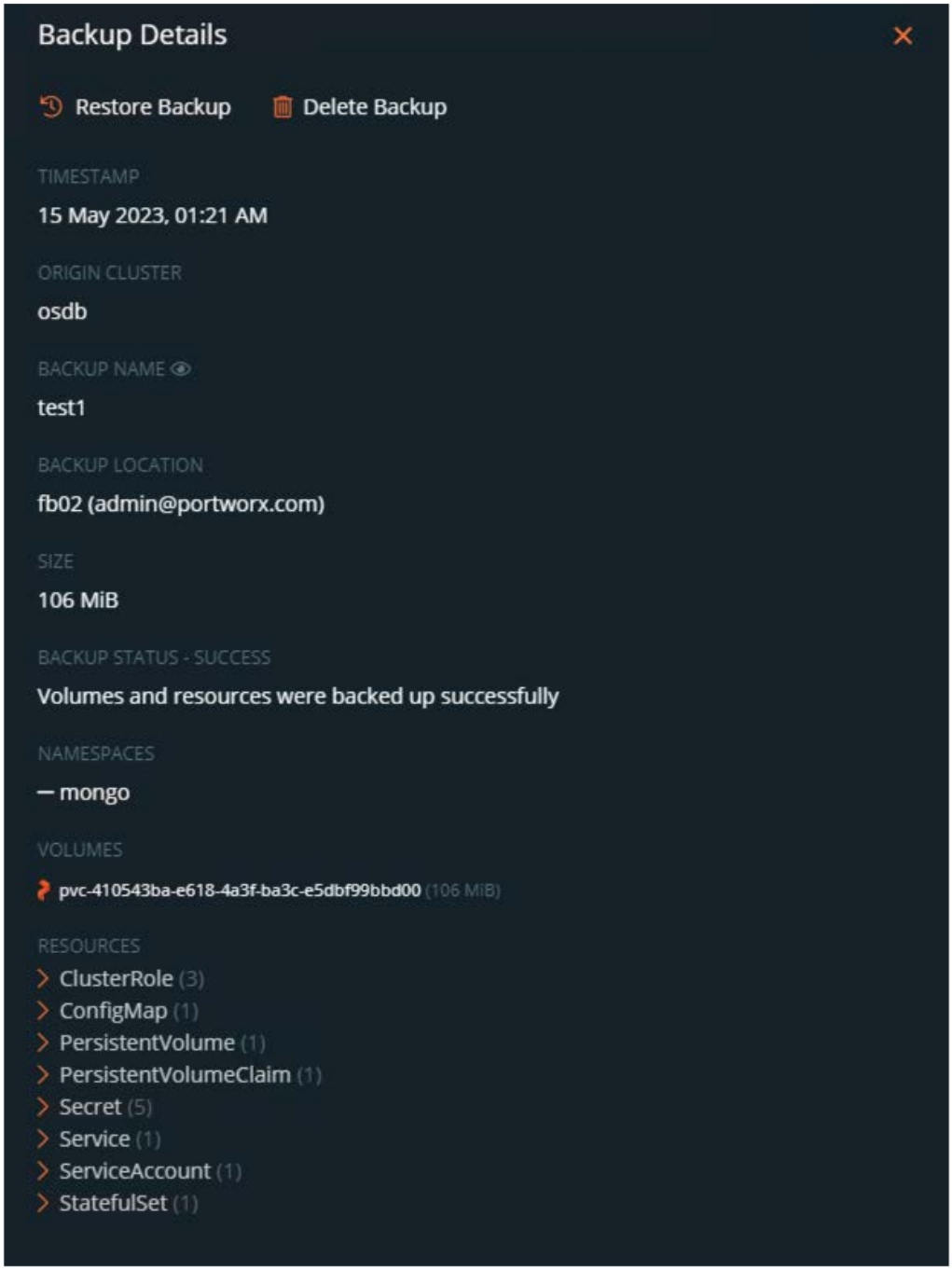
4. Initiates backup and lists the process in the "Backups" tab with status. If the backup icon turns green, backup is successful.

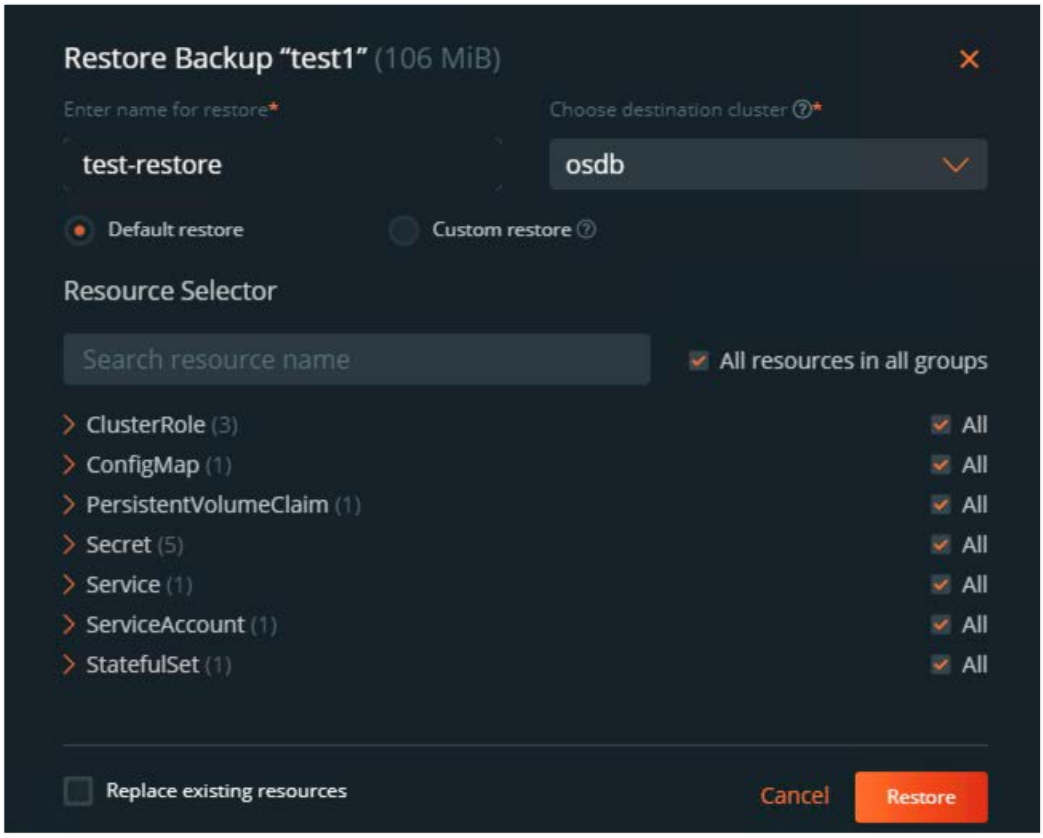


→ To restore backup, click on backup file → it opens a "Backup Details" window with 2 options listed (Restore Backup & Delete Backup)

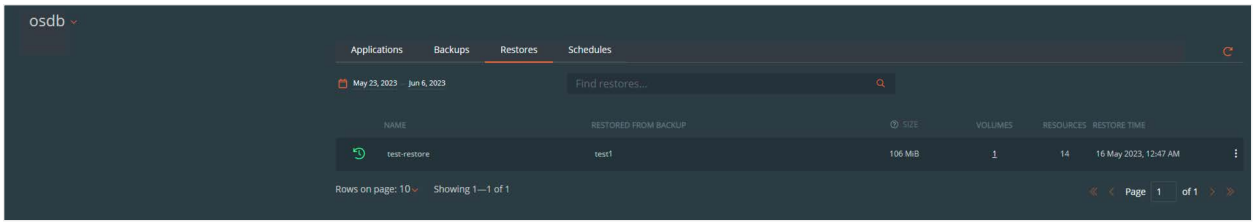
Click on "Restore Backup" → it opens a window to select the destination cluster for restore and there will be an option to select/unselect required resources to restore. Click on "Restore"

(There's an option to replace existing resources which need to be selected explicitly)





→ The restore status can be monitored in the "Restores" tab. If the restore icon turns green, the restore is successful.



High Availability with Portworx

High availability in the context of Portworx signifies its capacity, as a container-native storage solution, to maintain seamless and uninterrupted access to data and applications, even when confronted with hardware failures, network glitches, or unforeseen interruptions.

In this section, you will find comprehensive details on the methods to attain high availability for MongoDB workloads using Portworx.

Using MongoDB Built-in Features

Utilizing MongoDB's inherent capabilities, like the High Availability (HA) feature, Replica Set and sharded cluster deployments, enjoy the advantages of continuous data replication in real-time between primary and secondary nodes. This guarantees high availability at the database level. In case of a node or service failure, MongoDB seamlessly initiates a failover procedure by selecting one of the secondary nodes to assume control.

It's important to note that the HA feature operates exclusively at the database level.

Using Portworx Enterprise

Portworx Enterprise introduces high availability at the storage layer through its built-in persistent volume replication. The engine within Portworx distributes and synchronizes data across replica volumes based on the replication factor specified in the storage class definitions. This ensures the availability of data for MongoDB.

In the event of node failures, the integrated STORK scheduler identifies the presence of data copies and instructs the redeployment of MongoDB pods on those nodes. This feature provided by Portworx enhances the resilience of standalone MongoDB deployments. Rather than relying on a 2/3 node MongoDB cluster, leveraging Portworx replication offers faster failovers without the need for database-level replication.

Here are steps that illustrate how Portworx provides HA features for standalone MongoDB deployment.

1. Validate volume replication status.

Command to fetch pvc name for mongo-0 deployment/pod and assign it to variable "VOL"

```
VOL=`kubectl get pvc -n mongo| grep mongo-volume-mongo-0 | awk '{print $3}'`
```

Command to fetch portworx storage cluster name and assign it to variable "PX_POD"

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

Command to validate/verify volume replication status

```
kubectl exec -it $PX_POD -n portworx -- /opt/pwx/bin/pxctl volume inspect ${VOL}
```

Output:

```
Defaulted container "portworx" out of: portworx, csi-node-driver-registrar
 Volume         : 1047764872913900917
 Name   : pvc-de2c67dc-9d29-4574-9cea-3c76d0d9f075
 Size   : 200 GiB
 Format         : xfs
 HA     : 3
 IO Priority    : HIGH
 Creation time  : May 23 16:16:18 UTC 2023
 Shared         : no
 Status         : up
 State : Attached: b058be4a-efb4-4934-b7df-4cde7c67c1bb (10.21.227.157)
 Last Attached  : May 23 16:18:59 UTC 2023
 Device Path    : /dev/pxd/pxd1047764872913900917
 Labels: fs=xfs,io_priority=high,io_profile=db,namespace=mongo, pvc=mongo-volume-mongo-
0,repl=3,app=mongo
 Mount Options  : discard,nouuid
 Reads  : 30788
 Reads MS       : 48911
 Bytes Read     : 352538624
 Writes         : 3050973
 Writes MS      : 17647905
 Bytes Written  : 470908444672
 IOs in progress        : 0
 Bytes used     : 89 GiB
 Replica sets on nodes:         # data gets replicated to these 3 nodes
 Set 0
 Node : 10.21.227.158 (Pool be86c80a-0869-4636-89e5-6038241da5d6 )
 Node : 10.21.227.157 (Pool 239d74ac-c3d6-44c3-96c1-5093407f4cf2 )
 Node : 10.21.227.160 (Pool 9b4105aa-77dd-408e-878e-a288b5803195 )
 Replication Status     : Up
 Volume consumers       :
 - Name         : mongo-0 (29233bcc-43f7-4d40-8076-d55087ff0449) (Pod)
 Namespace      : mongo
 Running on     : osdb-zf5p2-worker-jcbvm
 Controlled by : mongo (StatefulSet)
```

2. **Pre-validation:** Verify database size in MongoDB pod.

Command to verify MongoDB pod status

```
kubectl get po -n mongo
```

Output:

```
NAME    READY STATUS     RESTARTS AGE
mongo-0         1/1      Running 0 14d
```

Command to verify the node name where MongoDB pod is running

```
kubectl get po mongo-0 -n mongo -o wide
```

Output:

```
NAME     READY STATUS    RESTARTS AGE IP         NODE
mongo-0 1/1      Running 0 14d 10.128.2.59 osdb-zf5p2-worker-jcbvm
```

Command to login MongoDB pod

```
kubectl exec -it mongo-0 -n mongo -- sh
```

Output:

```
$ mongosh
Current Mongosh Log ID: 647fc1aa433b7cf23ded05b6
```

Command to check the database list and size of each database

```
test> show dbs
```

Output:

```
admin 180.00 KiB
config 72.00 KiB
local   72.00 KiB
perf    53.42 GiB
```

3. Simulate node failure.

Command to cordon node / disable scheduling on node

```
kubectl adm cordon osdb-zf5p2-worker-jcbvm
```

Output:

```
node/osdb-zf5p2-worker-jcbvm cordoned
```

Command to check the status of the worker node

```
kubectl get nodes
```

Output:

```
NAME    STATUS  ROLES   AGE VERSION
osdb-zf5p2-master-0      Ready  master 28d v1.22.17+71d09da
osdb-zf5p2-master-1      Ready  master 28d v1.22.17+71d09da
osdb-zf5p2-master-2      Ready  master 28d v1.22.17+71d09da
osdb-zf5p2-worker-2k756          Ready  worker 28d v1.22.17+71d09da
osdb-zf5p2-worker-4tlf9          Ready  worker 28d v1.22.17+71d09da
osdb-zf5p2-worker-bggqt          Ready  worker 28d v1.22.17+71d09da
osdb-zf5p2-worker-jcbvm          Ready,SchedulingDisabled worker 28d v1.22.17+71d09da
osdb-zf5p2-worker-jjll2  Ready  worker 28d v1.22.17+71d09da
osdb-zf5p2-worker-thhjt          Ready  worker 28d v1.22.17+71d09da
```

4. Delete MongoDB pod.

Command to delete the pod

```
kubectl delete po mongo-0 -n mongo
```

Output:

```
pod "mongo-0" deleted
```

When the pod gets deleted, portworx's STORK scheduler colocates/redeploys the pod on any one of nodes where data copy exists.

Command to check pod status

```
kubectl get po mongo-0 -n mongo -o wide
```

Output:

```
NAME     READY STATUS   RESTARTS AGE IP         NODE
mongo-0 1/1 Running 0 24s 10.129.3.65 osdb-zf5p2-worker-thhjt
```

5. **Post validation:** Verify database size in MongoDB pod.

Command to login to MongoDB pod

```
kubectl exec -it mongo-0 -n mongo -- sh
```

Output:

```
$ mongosh
Current Mongosh Log ID: 647fc4cf9ad862ddd6cdea4f
```

Command to check the database list and size of each database.

```
test> show dbs
```

Output:

```
admin 180.00 KiB
config 72.00 KiB
local   72.00 KiB
perf    53.42 GiB
```

Volume Snapshots using Portworx

Volume snapshots allow capturing point-in-time copies of data volumes, providing data protection, backup and recovery capabilities in Kubernetes and containerized environments yet another way to protect MongoDB data. These snapshots can be used to restore/recover data in production environments and refresh lower environments by provisioning a new statefulset.Snapshots can be scheduled and taken on-demand.

Here are comprehensive, step-by-step instructions for executing volume snapshots using Portworx.

Steps to perform VolumeSnapshot

1. Capture the PVC name for MongoDB deployment. (use below command)

```
kubectl get pvc -n <namespace>
```

2. Create a YAML file (refer below file and modify appropriate parameter values) by pre, post execution rules and volume snapshot definitions. (Rules are meant to ensure application consistent volume snapshots) and apply it.

```
kubectl apply -f mongo-vol-snap.yaml
```

It creates a volume snapshot.

```
 # mongo-vol-snap.yaml
apiVersion: stork.libopenstorage.org/v1
kind: Rule
metadata:
 name: mongo-presnap-rule
spec:
 - podSelector:
 app: mongo-0
 actions:
 - type: command
 value: mongosh --eval "printjson(db.fsyncLock())"
---
apiVersion: stork.libopenstorage.org/v1
kind: Rule
metadata:
 name: mongo-postsnap-rule
spec:
 - podSelector:
 app: mongo-0
 actions:
 - type: command
 value: mongosh --eval "printjson(db.fsyncUnlock())"
---
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
 name: mongo-snap
 namespace: default
 annotations:
 stork.rule/pre-snapshot: mongo-presnap-rule
 stork.rule/post-snapshot: mongo-postsnap-rule
spec:
 persistentVolumeClaimName: mongo-volume-mongo-0
```

3. Create another YAML file/spec for restoring snapshot to new PVC.

NOTE: *"storageClassName" value should be "stork-snapshot-sc" Stork StorageClass as mentioned in the below example.*

```
 apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mongo-snap-clone
 annotations:
 snapshot.alpha.kubernetes.io/snapshot: mongo-snap
spec:
 accessModes:
 - ReadWriteOnce
 storageClassName: stork-snapshot-sc
 resources:
 requests:
 storage: 200Gi
```

Once spec is applied, a new PVC will be created by stork.

4. Create a new MongoDB StatefulSet and attach a new PVC to ensure volume recovery from the snapshot.

5. Create a spec file and apply it. (Refer below sample file)

```
 apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: mongo2
spec:
 selector:
 matchLabels:
 app: mongo2
 serviceName: mongo2
 replicas: 1
 template:
 metadata:
 labels:
 app: mongo2
 spec:
 terminationGracePeriodSeconds: 10
 containers:
 - name: mongo2
 image: mongo
 resources:
 requests:
 cpu: ""“1"""”
 memory: 400Mi
 limits:
 cpu: ""“2"""”
 memory: 500Mi
 ports:
 - containerPort: 27017
 volumeMounts:
 - name: mongo2-volume
 mountPath: /data/db
 volumes:
 - name: mongo2-volume
 persistentVolumeClaim:
 claimName: mongo-volsnap-clone # attach new pvc from snapshot
```

It deploys a new MongoDB StatefulSet attached with new PVC cloned from the volume snapshot.

6. Validate data in the new MongoDB StatefulSet. (Size of volume including usage and free disk space in new MongoDB pod should match with original volume)

```
 # size of volume (/data/db) in MongoDB pod before snapshot
[root@linux-ctrlr]# kubectl exec -it mongo-0 -n mongo -- sh
$ df -h
Filesystem      Size Used Avail Use% Mounted on
overlay         120G 36G 85G 30% /
tmpfs   64M     0 64M 0% /dev
tmpfs   32G     0 32G 0% /sys/fs/cgroup
shm     64M     0 64M 0% /dev/shm
tmpfs   32G 58M 32G 1% /etc/passwd
/dev/pxd/pxd1047764872913900917 200G 90G 111G 45% /data/db
# size of volume (/data/db) in new MongoDB pod(mongo2) after snapshot
[root@linux-ctrlr]# kubectl exec -it mongo2-0 -n mongo -- sh
$ df -h
Filesystem      Size Used Avail Use% Mounted on
overlay         120G 32G 89G 27% /
tmpfs   64M     0 64M 0% /dev
tmpfs   32G     0 32G 0% /sys/fs/cgroup
shm     64M     0 64M 0% /dev/shm
tmpfs   32G 53M 32G 1% /etc/passwd
/dev/pxd/pxd162712120612106876 200G 90G 111G 45% /data/db
```

Steps to perform GroupVolumeSnapshot

1. To create a snapshot for a group of PVC's, Create a spec (refer below YAML file and modify appropriate parameter values such as namespace, app label) and apply the spec.

```
kubectl apply -f mongo-group-vol-snap.yaml
```

Portworx will quiesce IO operations on every volume within the group and subsequently initiate the snapshot.

```
 # mongo-group-vol-snap.yaml
apiVersion: stork.libopenstorage.org/v1
kind: GroupVolumeSnapshot
metadata:
 name: mongo-group-snap
 namespace: default
spec:
 pvcSelector:
 matchLabels:
 app: mongo
```

2. Create another YAML file/spec for restoring group snapshots by creating a VolumeSnapshotRestore spec.

```
 apiVersion: stork.libopenstorage.org/v1
kind: VolumeSnapshotRestore
metadata:
 name: mongo-group-snap-restore
 namespace: default
spec:
 groupSnapshot: true
 sourceName: mongo-group-snap
 sourceNamespace: default
```

It restores all volumes(PVC's) from the GroupSnapshot.

Scalability with Portworx

Scalability with Portworx is a critical aspect of its design, enabling organizations to efficiently grow their storage infrastructure as their needs expand. Portworx allows to expand the size of existing volumes without disrupting running applications. This means that as data grows, it's simple to increase the capacity of storage volumes.

In addition, the Portworx Autopilot feature offers automated detection and provisioning of storage resources when available capacity becomes constrained. It facilitates the scaling of storage for individual container volumes (PVCs) or the entire storage pool, all with customizable rules set by the user.

For cloud-based deployments, Portworx Autopilot serves as a cost-effective solution by intelligently addressing the common issue of paying for cloud storage at the provisioning stage rather than when it's actually consumed.

Refer to the documentation on How to use Portworx Autopilot for detailed instructions on utilizing the Autopilot feature in Portworx.

The following is an in-depth process that provides guidance on expanding storage capacity.

Steps to scale/expand PVC

1. Edit PVC spec of MongoDB deployment, change/modify the size of PVC and apply the spec. (Portworx does volume expansion seamlessly)

```
 # mongo2-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mongo-volsnap-clone
 namespace: mongo
 annotations:
 snapshot.alpha.kubernetes.io/snapshot: mongo-snapshot
spec:
 accessModes:
 - ReadWriteOnce
 storageClassName: stork-snapshot-sc
 resources:
 requests:
 #storage: 200Gi # Actual volume size
 storage: 300Gi # Desired volume size
```

Save and apply the spec → spec changes will be configured

2. Validate PVC capacity
   # Command to find PVC description

```
kubectl describe pvc mongo-volsnap-clone -n mongo
```

output:

```
# Size of PVC before expansion
Name:   mongo-volsnap-clone
Namespace:      mongo
StorageClass: stork-snapshot-sc
Status:         Bound
Volume:         pvc-9acf2d4b-ce78-431a-9009-1e5b8a5914e0
Labels:         <none>
Annotations: pv.kubernetes.io/bind-completed: yes
 pv.kubernetes.io/bound-by-controller: yes
 snapshot.alpha.kubernetes.io/snapshot: mongo-snapshot
 volume.beta.kubernetes.io/storage-provisioner: stork-snapshot
Finalizers:     [kubernetes.io/pvc-protection]
Capacity:       200Gi
Access Modes: RWO
VolumeMode:     Filesystem
Used By:        mongo2-0
Events: <none>
# Size of PVC after expansion
Name:   mongo-volsnap-clone
Namespace:      mongo
StorageClass: stork-snapshot-sc
Status:         Bound
Volume:         pvc-9acf2d4b-ce78-431a-9009-1e5b8a5914e0
Labels:         <none>
Annotations: pv.kubernetes.io/bind-completed: yes
 pv.kubernetes.io/bound-by-controller: yes
 snapshot.alpha.kubernetes.io/snapshot: mongo-snapshot
 volume.beta.kubernetes.io/storage-provisioner: stork-snapshot
 volume.kubernetes.io/storage-resizer: kubernetes.io/portworx-volume
Finalizers:     [kubernetes.io/pvc-protection]
Capacity:       300Gi
Access Modes: RWO
VolumeMode:     Filesystem
Used By:        mongo2-0
Events:
 Type    Reason  Age From        Message
 ----    ------  ---- ----       -------
 Normal VolumeResizeSuccessful 6s       volume_expand ExpandVolume succeeded for volume
mongo/mongo-volsnap-clone
```

Overall, Portworx's scalability features make it a powerful choice for organizations seeking to grow and manage their storage infrastructure in containerized environments. It allows the storage resources in a flexible and cost-effective manner, ensuring that applications have the storage capacity and performance required to meet evolving business demands.

MongoDB Deployments - Portworx Data Services (PDS)

Portworx Data Services stands as a valuable resource for the development community, providing the full range of features found in its database counterpart, all while alleviating the common operational complexities and demands typically encountered when developing new applications. This allows you to focus on core competencies. PDS features a streamlined interface that simplifies the configuration and management of MongoDB workloads, ensuring an exceptional developer experience.

Benefits

- Provides better user experience and ability to manage the entire life cycle process of data services (deployment, configuration, scaling, HA, backup, restore and maintenance/patching/upgrade) that are deployed.
- PDS offers a Database Platform As a Service where enterprises can manage their own containerized database workloads.
- PDS has inbuilt HA, scaling, backup and metrics features for MongoDB deployments.
- PDS deploys Enterprise MongoDB shard cluster by default that provides better availability, performance and scalability features.
- PDS introduces an AI-Powered Copilot tool designed to facilitate a more user-friendly and intuitive interaction with databases, bridging the gap between end users and database administrators.

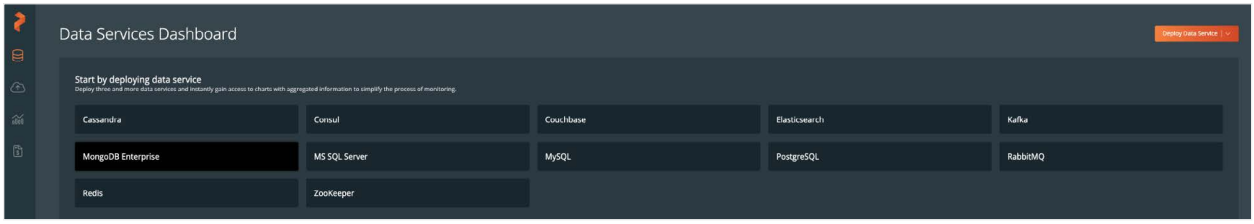
Requirements

- Any Kubernetes cluster in running state.
- Valid Portworx account to deploy and manage workloads through PDS portal.

Here's the process for deploying MongoDB on Kubernetes through the PDS Data Services catalog accessible within the PDS Portal.

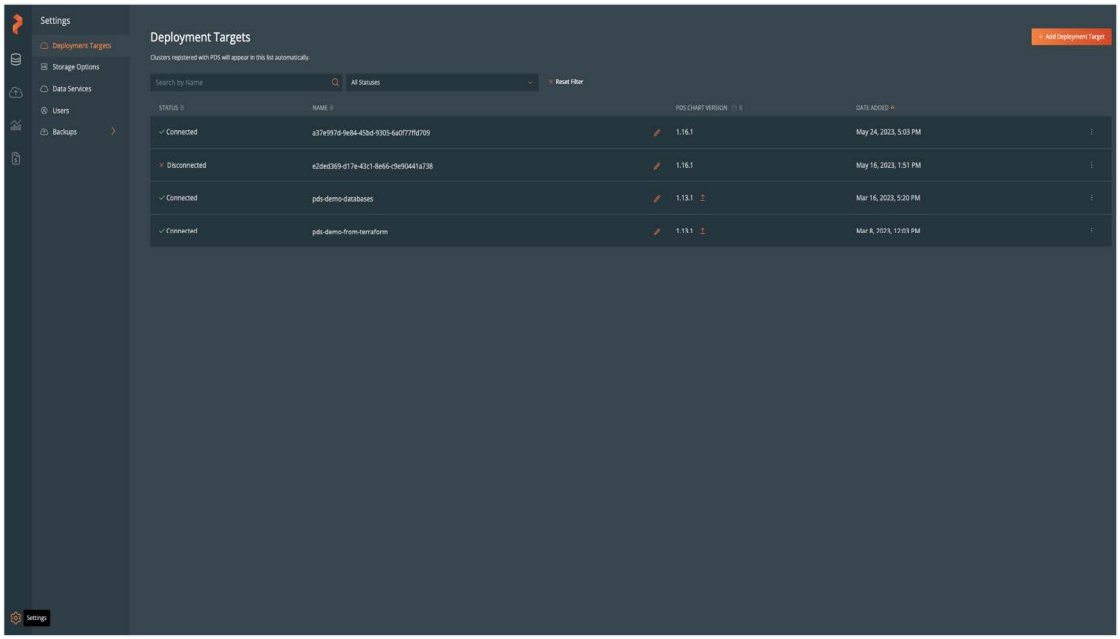
Deploy MongoDB as a Data Service

1. Sign in to the PDS portal using valid Portworx account credentials. This will direct to the Data Services dashboard, where a variety of services supported by the PDS platform are listed.

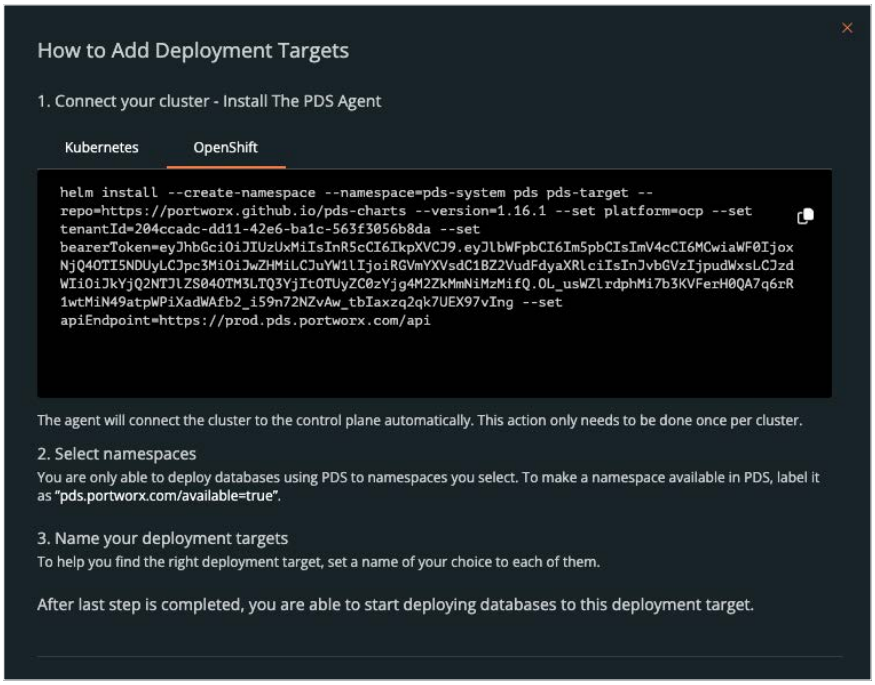


2. Before proceeding to deploy any of the data services, a deployment target (any operational Kubernetes cluster) should be added to the PDS platform.
   Here is the process to add deployment targets to the PDS portal.

- In the PDS portal, access the Settings → Deployment Targets section, with a list of existing or configured deployment targets.



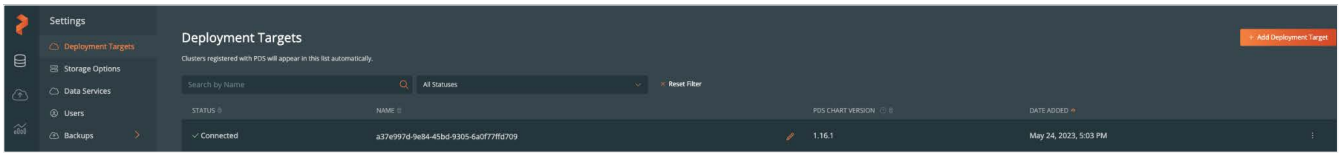
- Click **Add Deployment target** on the top-right corner of the PDS portal.
  A dialog box appears with a set of instructions to be executed on the Kubernetes cluster to configure it as a deployment target.



In the dialog box, perform the following actions:

- Install agent that connects cluster to control plane.
- Enable cluster's namespace details on PDS portal. (can be selected as target namespace to deploy any of data services)
- Edit/Name the deployment target with standard naming convention.

After completing the above 3 steps, the Kubernetes cluster is added to the PDS portal as one of the deployment targets.



3. To proceed with MongoDB deployment, in the Data Services Dashboard, select **MongoDB Enterprise data service**.

In the **Deploy MongoDB Enterprise** window, configure the following options:

- Version: Select the latest MongoDB version to deploy.
- Name**:** Specify valid name for planned deployment.
- Target: Select a Kubernetes cluster (added in previous steps) from the dropdown list where MongoDB workload is to be deployed.
- Namespace: Select namespace from dropdown list that are exposed to the PDS platform.
- Provision Load Balancer: Enable this option to deploy best practices.
- Application Configuration: Select appropriate configuration (Pre-loaded) needed to deploy MongoDB.
- Size: Select the appropriate resource settings (compute, memory and disk) to deploy MongoDB.
- No: of Nodes: Specify the deployment size (the number of MongoDB nodes).
- Storage Options: Select appropriate storage options such as file system type, replication factor (portworx's HA feature) and volume group specs.
- Backup: Select this option to enable or disable the backup process for planned MongoDB deployment.
- (Optional) Enable advanced settings: Select the appropriate MongoDB build version.

Deploy MongoDB Enterprise

Version*

6.0.3

Deployments

Updating this information after the service was deployed is currently not supported.

Name*

Test

Target*

a37e997d-9e84-45bd-9305-6a0f77ffd709

Namespace*

mongo-pds

Provision load balancer

See Connecting to your deployment.

Application Configuration

QaDefault

Show Details

Size

Resource Settings

Large
4-32 CPU, 6G-64G MEM, 500G STORAGE

No. of nodes*

4

Reducing number of nodes after deployment is currently not supported.
See recommendations

Storage Options

Updating this information after the service was deployed is currently not supported.

Solution Perf Test
XFS, Auto-detect, replication factor: 2

Backup

No Scheduled Backups

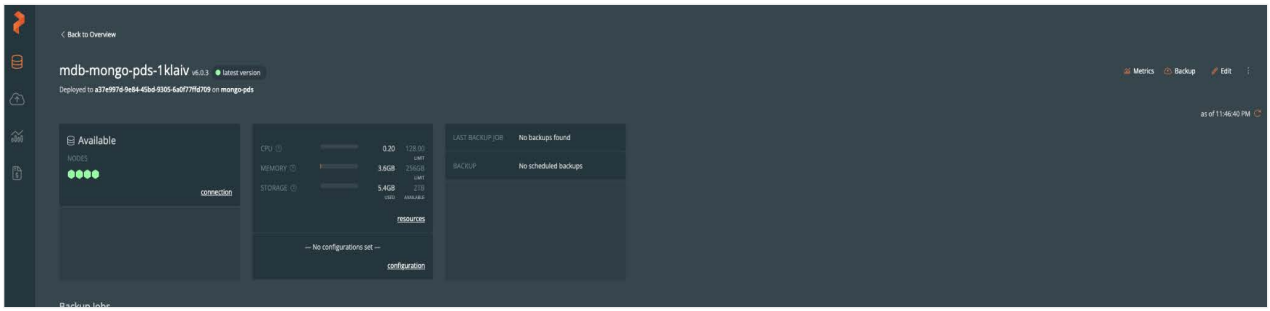
Enable advanced settings

Cancel

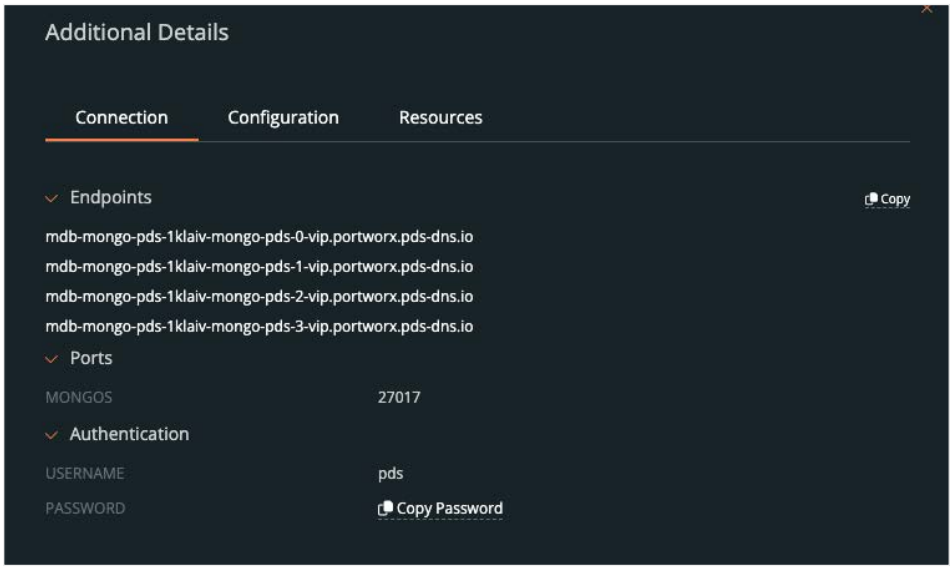
Deploy

4. Click **Deploy** to initiate MongoDB deployment on the targeted Kubernetes cluster using the PDS platform.

After MongoDB is successfully deployed, PDS lists the newly deployed MongoDB in the Data Services Dashboard. Click deployment to view the details.



To view connection details, click **MongoDB deployment** → **Connection**.



Monitor MongoDB Deployment

Portworx offers monitoring and reporting capabilities that help you track resource usage and identify performance bottlenecks. This information can be used to optimize resource allocation and scalability.

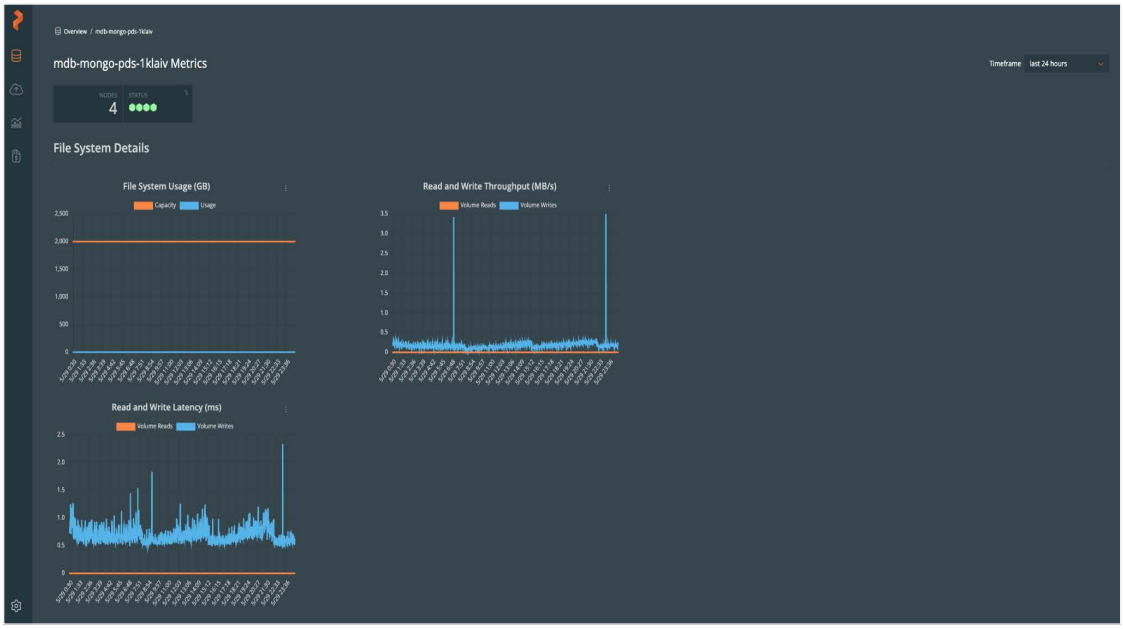
With PDS, Deployment and database metrics are thoughtfully organized, allowing for the monitoring of both infrastructure and database-specific parameters through the use of Prometheus and Grafana dashboards.

To monitor PDS deployments of MongoDB, perform the following tasks:

1. Click on the MongoDB deployment and then click **Metrics** on the top-right corner of the dashboard.

Metrics displays File system metrics and Application details.

2. File system metrics displays file system usage, read and write throughputs, read and write latency details of deployment.



3. Application Details metrics displays MongoDB-specific metrics such as total and active connections, documents count, replication lag, read and write ops/sec.

Protecting MongoDB Deployments using PDS

PDS places a strong emphasis on data protection, offering a fully managed backup service that includes continuous and consistent backups, point-in-time recovery and the flexibility to configure custom retention policies. Activating these features is a straightforward process, requiring just a few clicks.

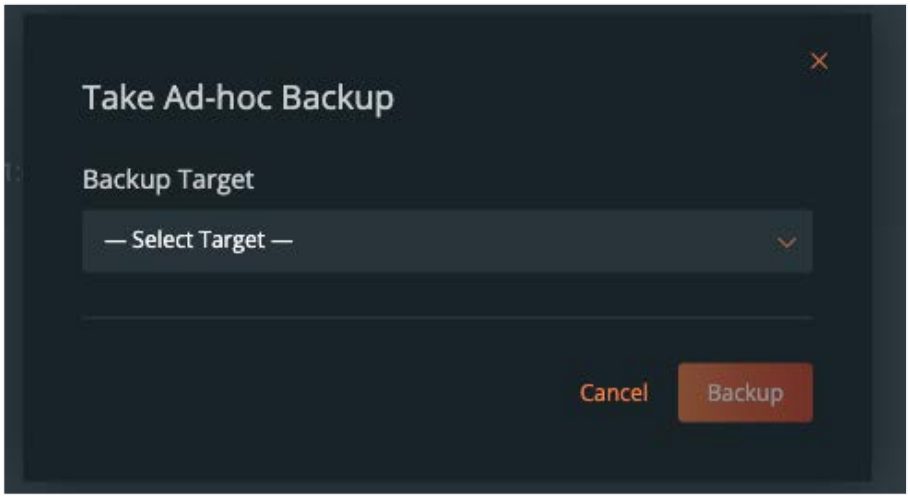
PDS hosted data services such as MongoDB can be protected using PDS inbuilt backup feature that protects deployment components. Backup features can be enabled and scheduled during deployment configuration or after deployment completion.

Requirements

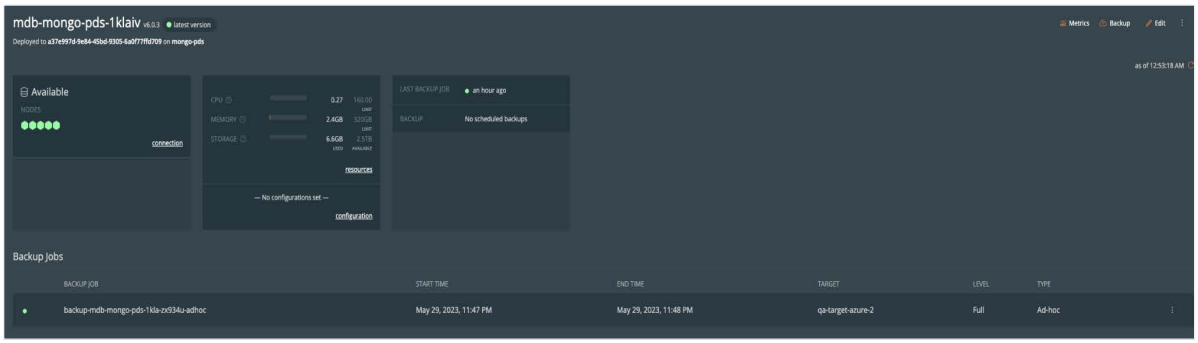
Configure backup target or access to pre-configured backup targets, where PDS stores backup files of deployment when backup triggers.

Performing Ad Hoc Backup using MongoDB Deployment on PDS

1. Click MongoDB deployment and then click on Backup option, which is on the top-right corner of the dashboard.



2. From the dropdown list, select the pre-configured backup target and Click **Backup**.
3. After successful completion of the backup, the backup job details such as type (full/differential), backup target, backup start and end times are tagged on the MongoDB deployment dashboard.



Comprehensive information on the restore process can be found here.

Scale MongoDB Deployments using PDS

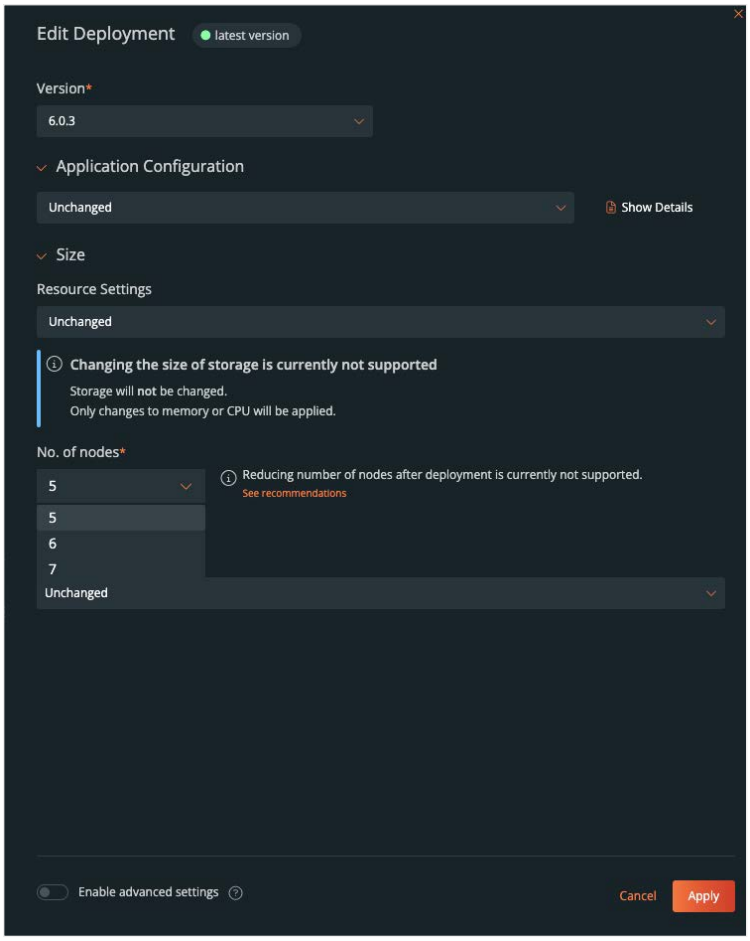
In light of the common challenges faced by organizations, such as the need to manage capacity and accommodate data growth, PDS offers continuous and online scaling capabilities, both horizontally and vertically:

- Horizontal scaling: PDS allows to scale MongoDB nodes online whenever required.
- Vertical scaling: PDS allows to scale compute resources (CPU and memory) online whenever required.

Here are comprehensive steps that provide guidance on achieving scalability using PDS:

Horizontal Scaling - MongoDB deployments

1. Click the MongoDB deployment and then click Edit, which is on the top-right corner of the dashboard. The Edit Deployment window appears, where you can perform horizontal and vertical scaling.



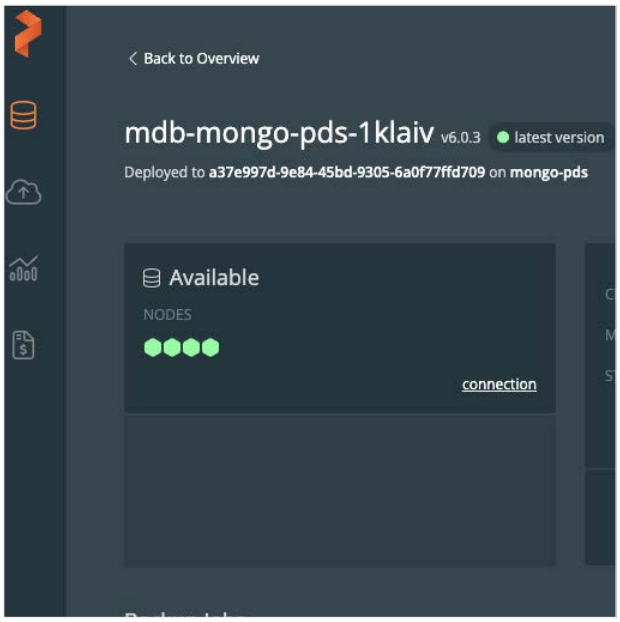
2. For horizontal scaling of MongoDB nodes, select the nodes from the **No: of nodes** dropdown list.
3. For vertical scaling of resources on each MongoDB node, select the appropriate setting from the **Resource Settings**.

4. Click **Apply**.

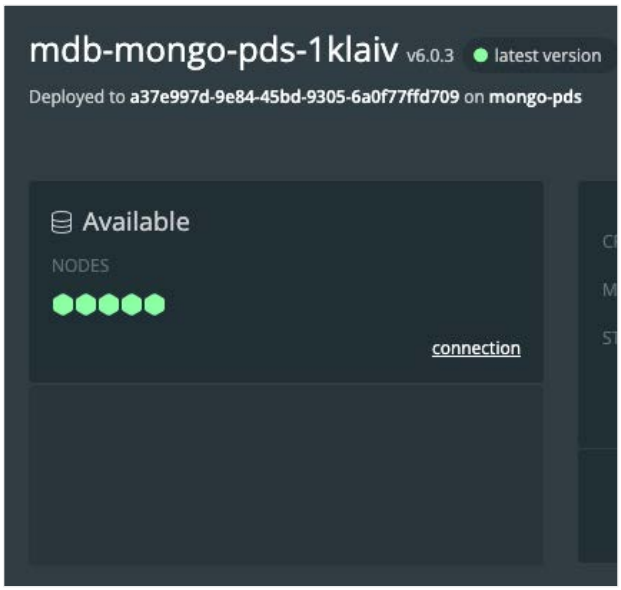
   The deployment properties are updated and the resources are scaled-up.

   The following image illustrates scaling MongoDB nodes from 4 to 5.

   Before scaling nodes:



After scaling nodes:



Vertical Scaling - MongoDB Deployments

PDS allows other customizations such as Storage Options and Resource Settings template. Users can create their own templates for resources settings (CPU, memory and disk) and storage options (file system type, volume group and replication factor).

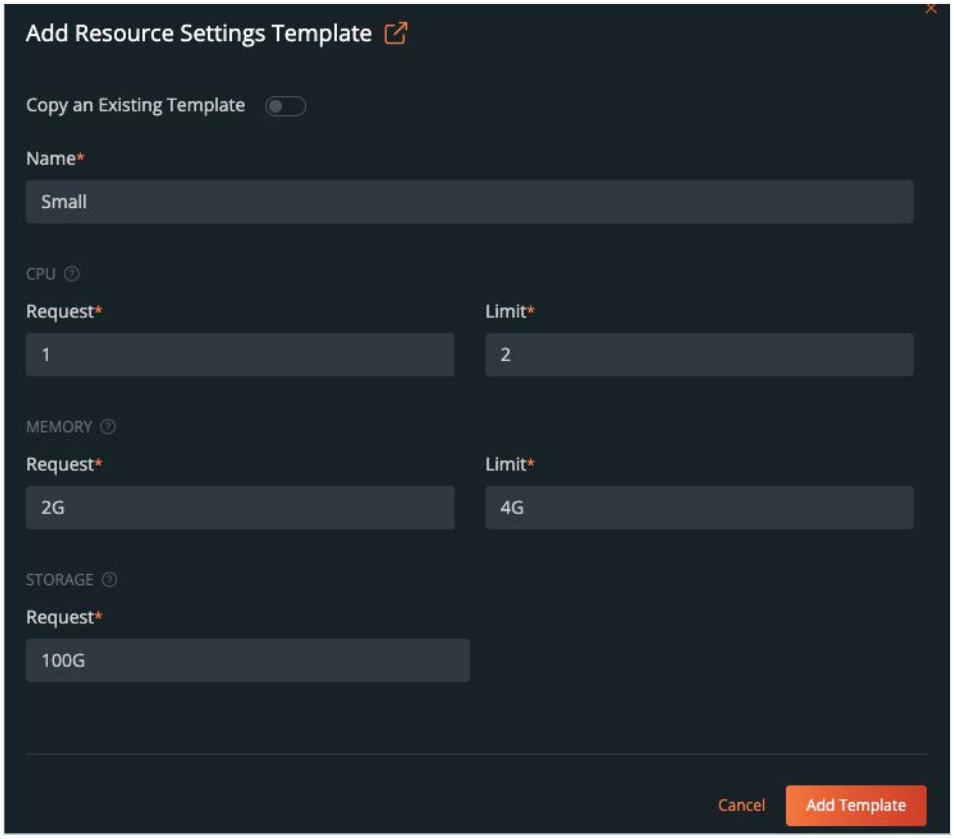
Here is a detailed, step-by-step process for customizing resource settings:

Customize Resource Settings

1. Click **Settings** → **Data services** → **MongoDB Enterprise** → **Resource Settings**.

2. In the Resource Settings tab, click **Add Template**.

The **Add Resource Settings Template** screen appears.



4. In the Add Resource Settings Template screen**,** specify the template name, select the resource request options and limits with respect to CPU, memory and disk.
3. Click **Add Template**. (The new template is displayed on the **Data services** → **Resource Settings** tab.

Additional References

- The MongoDB on Portworx Design guide provides detailed information regarding design aspects, available options, associated benefits, and recommended best practices for MongoDB on Portworx.
- For Portworx Solutions, architecture and features, see the Portworx Documentation.
- Refer to the Portworx Data Services documentation for more detailed information.
- The MongoDB Official Documentation provides additional details on MongoDB on Kubernetes using the MongoDB Enterprise Operator.

© 2024 Pure Storage, Inc. All rights reserved.
Pure Storage, the Pure Storage P Logo, Portworx, and the marks in the Pure Storage Trademark List are trademarks or registered trademarks of Pure Storage Inc. in the U.S. and/or other countries. The Trademark List can be found at purestorage.com/trademarks. Other names may be trademarks of their respective owners.
The Pure Storage product described in this documentation is distributed under a license agreement and may be used only in accordance with the terms of the agreement. The license agreement restricts its use, copying, distribution, decompilation and reverse engineering. 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.
THE 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.