

# MongoDB Clusters on Karbon Scenarios

---

The walkthrough scenarios in this document focuses on Deploying MongoDB Clusters of various Types within the Nutanix Kubernetes Engine (NKE aka Karbon) using a combination of Nutanix Cloud Management (NCM aka Calm) and the MongoDB Operator.

## To Run or Not Run a Database on Kubernetes Considerations

Below are snippets found in the following K8s best practice article - ["To run or not to run a database on Kubernetes: What to consider"](#) that I found relevant to topic overall.

### General Hosting Database Scenario Options:

- **Fully Managed Databases:** Consider leveraging DBaaS Solutions (i.e. Nutanix ERA) as a preferred low-ops choice that handles the maintenance tasks, like backup, patching and scaling.
- **Do-it-yourself on a VM:** You take full responsibility for building your database, scaling it, managing reliability, setting up backups and more...
- **Run it on Kubernetes:** Running a database on Kubernetes is closer to the full-ops option, that said, it is important to remember that pods (the database application containers) are transient, so the likelihood of database application restarts or failovers is higher. Also, some of the more database-specific administrative tasks—backups, scaling, tuning, etc.—are different due to the added abstractions that come with containerization.

### Considerations when running databases on Kubernetes:

Since pods are mortal, the likelihood of failover events is higher than a traditionally hosted or fully managed database. It will be easier to run a database on Kubernetes if it includes concepts like **sharding**, **failover elections** and **replication** built into its DNA (for example, **ElasticSearch**, **Cassandra**, or **MongoDB**)...

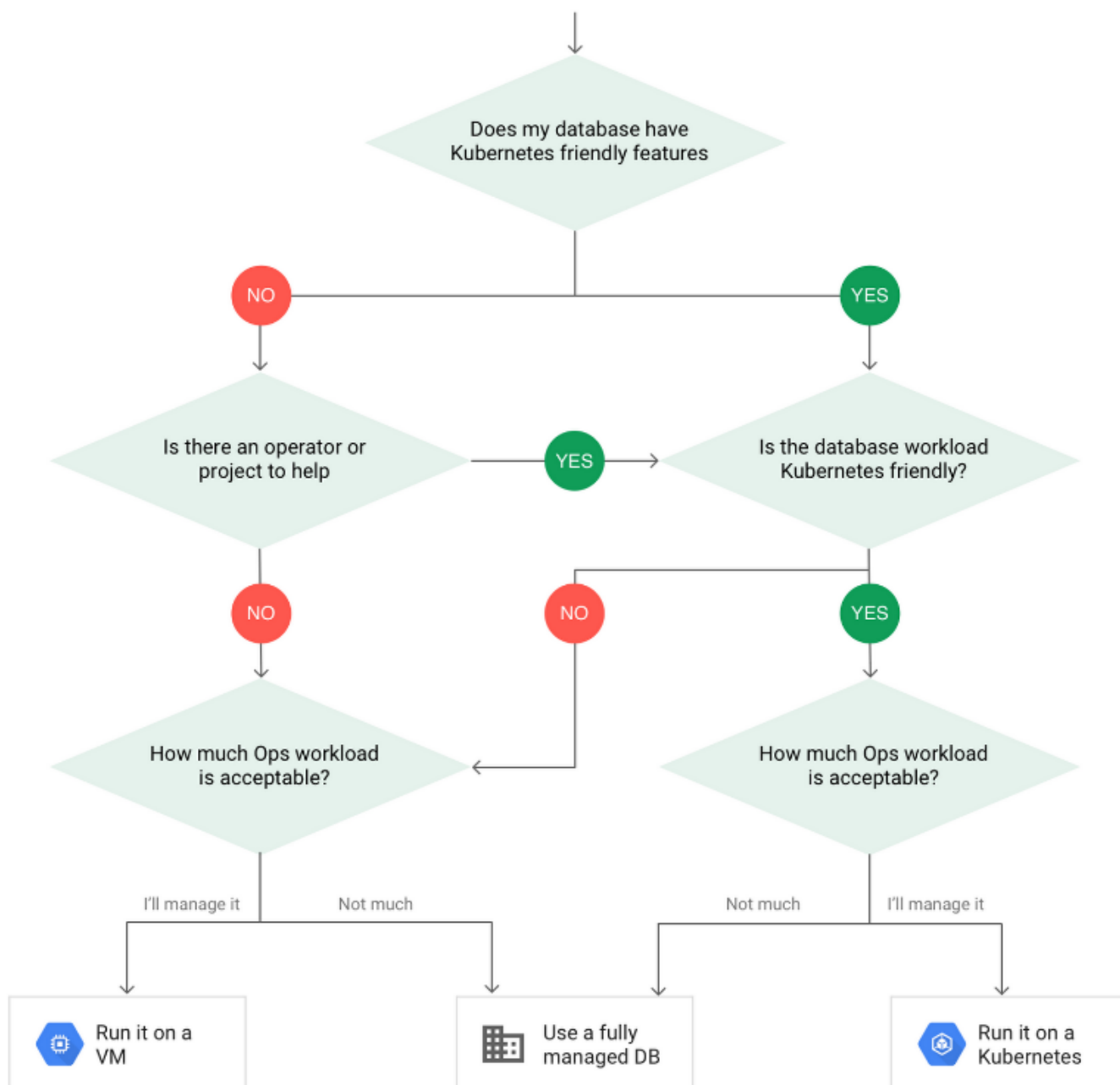
**MongoDB** is an open source document-oriented NoSQL database that stores data in flexible, JSON-like documents. **MongoDB** is an ideal candidate as it provides **high availability** and **redundancy** through **Replica sets** and horizontal **scalability** through **sharding**.

Some open source projects provide custom resources and operators to help with managing the database...

For example, if **MySQL** is needed to run on Kubernetes, there is the **Oracle MySQL Operator** and alternatively the **Crunchy Data** or **Zalando Operator** could be leveraged for **PostgreSQL**.

Finally, be sure you understand the replication modes available in the database. Asynchronous modes of replication leave room for data loss, because transactions might be committed to the primary database but not to the secondary database(s). So, be sure to understand whether you might incur data loss, and how much of that is acceptable in the context of your application...

After evaluating all of those considerations, you'll end up with a decision tree looking something like this:



## Leverage Nutanix DBaaS (NDB aka ERA) UI to Deploy MongoDB Standalone or ReplicaSets on VMs

**Nutanix Era** enables you to easily register, provision, clone, and administer all of your MongoDB databases on one or more Nutanix clusters with a single click.

Era supports single node and multiple node configurations. A single node configuration in MongoDB consists of a single mongod daemon running on a single database server VM.

**MongoDB Instance Summary**

Name: **mongodb-standalone**  
 Description:  
 Database Name on VM: **mongodb-standalone**    Size (GiB): **0**    Provisioned by Era: **2022-07-22 15:12:22**  
 Status: ●  
 Database Version: **4.4.2**

**Profiles**

Software: **MONGODB\_4.4\_OOB**  
 Network: **DEFAULT\_OOB\_MONGODB\_NETWORK**  
 Compute: **DEFAULT\_OOB\_COMPUTE**  
 Database Parameter: **DEFAULT\_MONGODB\_PARAMS**

The profiles were applied when the database was created but the settings may have been changed since then.

**Databases**

Name	System Database	Connect
admin	✓	<a href="#">See Description</a>
collection01		<a href="#">See Description</a>
config	✓	<a href="#">See Description</a>
local	✓	<a href="#">See Description</a>

**Time Machine**

Name: **mongodb-standalone\_TM**  
 Description: **Time Machine for instance 'mongodb-standalone'**  
 Age: **19 days 18 hours**    Clones: **0**  
 Last Update: **56 minutes ago**    SLA: **DEFAULT\_OOB\_BRASS\_SLA**  
 Size (GiB): **0.3**

**Database Server VM**

Name: **mongodb-standalone**  
 Description: **DBServer for MongoDB database mongodb-standalone**  
 Database Server VM's Time Zone: **UTC**

**Tags**

No tags found.

## Pros:

- **One-Click Provisioning:** Era enables you to easily provision database environments (either production or otherwise) on your Nutanix clusters.
- **Copy Data Management:** Era enables you to clone your databases and refresh the database clones by using snapshots or transaction logs.
- **Database Protection:** Era protects your database with full database consistent backups within a matter of minutes.
- **One-Click Patching:** Ensure data security with one-click patching to efficiently validate critical database updates. Era provides out-of-band patching of databases to eliminate database configuration sprawl.

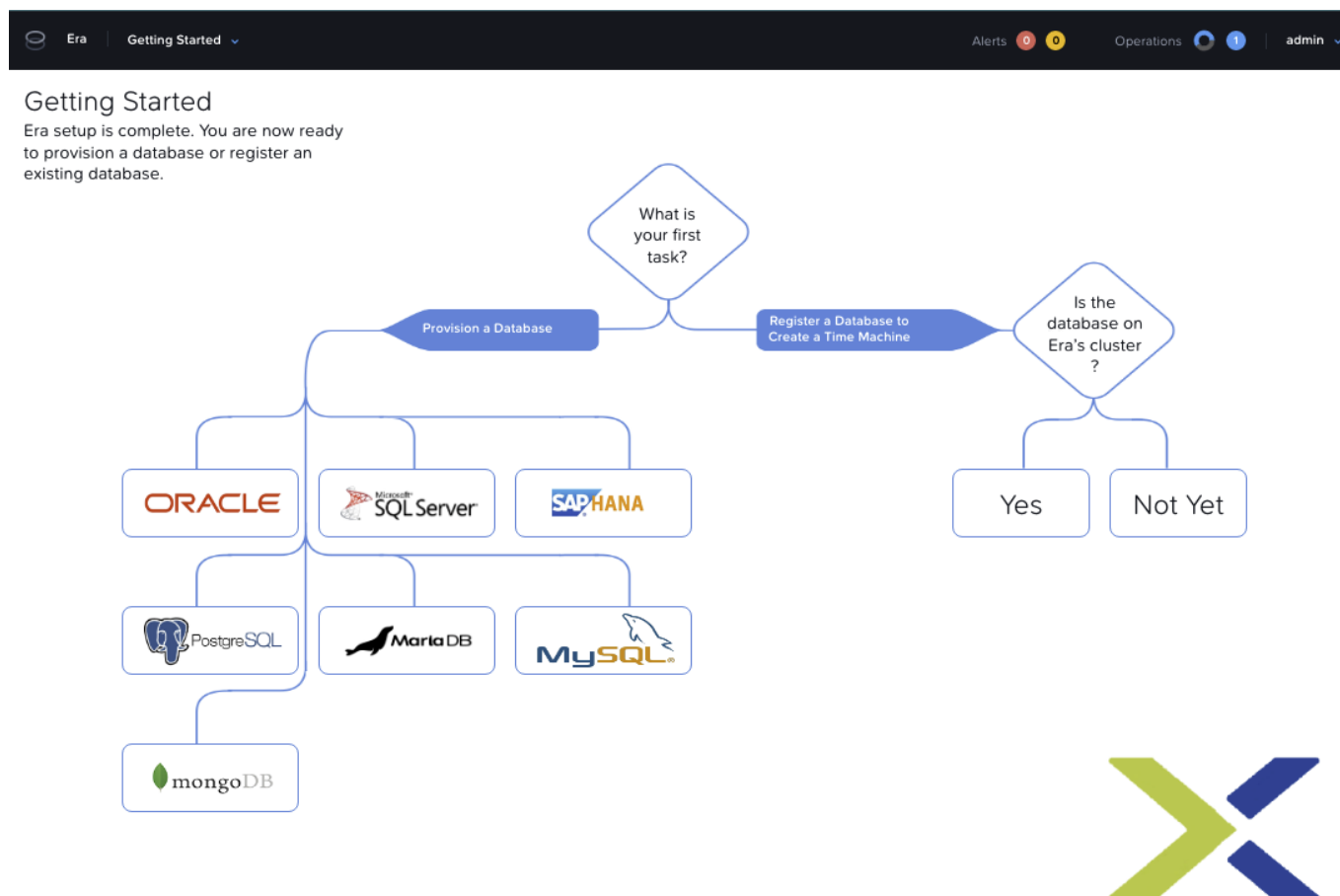
## Concerns/Limitations (As of ERA 2.4):

[https://portal.nutanix.com/page/documents/details?targetId=Nutanix-Era-User-Guide-v2\\_4:top-era-limitations-mongodb-c.html](https://portal.nutanix.com/page/documents/details?targetId=Nutanix-Era-User-Guide-v2_4:top-era-limitations-mongodb-c.html)

- Era supports MongoDB version 4.0.x.
- Era does not support MongoDB sharded systems.
- Era does not support MongoDB in-memory engine.
- Era does not support database restore for MongoDB replica set.
- Ubuntu and SUSE Linux operating systems are not supported.
- Era does not support provisioning and registration of multiple MongoDB user databases in the same database server VM or replica set.
- Era does not support multiple installations of MongoDB on the same database server VM.

- Era does not support MongoDB installations by any OS user other than 'mongod'.
- Era supports both XFS and ext4 file systems for registered databases but only supports XFS file system for provisioned databases.

**IMPORTANT:** Although all features required may not be immediately available - The KEY advantage to leveraging ERA over all solutions listed below is that it provides the above capabilities for MULTIPLE DB Platforms - e.g., MS SQL Server, Oracle (RAC), PostgreSQL, MySQL, MariaDB, SAP HANA AND MongoDB!!!




## Leverage Nutanix Self-Service (Calm) UI to Deploy MongoDB (All Scenarios) on VMs

Nutanix Calm could be leveraged to deploy MongoDB via the Self-Service Portal to provision VMs and leverage Day 2 Actions to Scale, Upgrade and/or Backup/Restore underlying clusters using any of the following scenarios:

- **Deploy MongoDB Standalone and/or ReplicaSets** by integrating directly with **Nutanix Era API**
- **Deploy MongoDB Standalone, ReplicaSets and/or ShardedClusters** by integrating with preferred IaaS endpoint (e.g., Nutanix AHV, vCenter, AWS, Google, Azure VMs, Terraform, etc.) to provision VM(s) and subsequently configure MongoDB using preferred **package management** (e.g. apt, yum, etc.), **configuration management** tools (e.g., ansible, chef, puppet, salt, etc.) and/or combination of linux / windows scripting technologies.
  - As an alternative, **available or custom MongoDB Docker container images** can be leveraged to deploy and isolate specific versions of mongod directly on VMs. Probably would

not recommend for multitude of reasons, but it's been done before.



## MongoDB Sharding Blueprint

by Nutanix

Version

4.2.0

▼

Launch

Clone

Overview

Actions Included

Change Log

---

MongoDB is a multi-node distributed database. MongoDB uses sharding to support deployments

**License:**

- Apache License 2.0
- GNU AGPL v3.0

**Hardware Requirement:**

- By default
  - Router Nodes - 3 VMs each 2 vCPU, 2GB RAM
  - Config Set - 3 VMs each 2 vCPU, 2GB RAM
  - Data Set - 3 VMs each 2 vCPU, 4GB RAM

**Resources Installed:**

- Config set with Mongo 3.4.2
- Data Set with Mongo 3.4.2
- Router VM with Mongo 3.4.2

**Operating System:**

- CentOS Linux release CentOS-7-x86\_64-2003
- Select the [AMI](#) according to the region on AWS.
- Select the [Image](#) according to the zone on GCP.
- Select the [Image](#) according to the location on Azure.

**Platform:**

- AHV
- AWS
- GCP
- Azure
- VMware

**Lifecycle:**

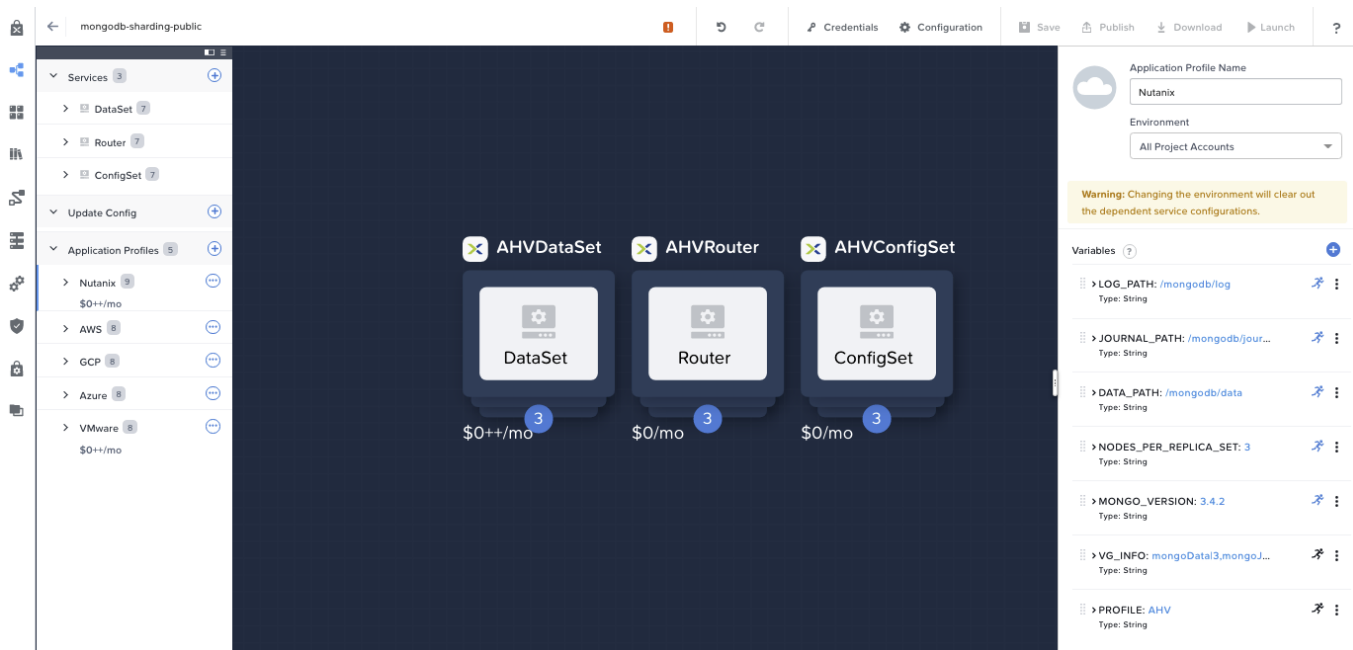
- ScaleOut
- ScaleIn

**Limitations:**

- For evaluation purposes only. Not recommended for production use.

**Other Instructions:**

- Open ports `39017` , `27017`



### Pros:

By Leveraging **NCM/Cal**m, you'll have the ability to provide end users the **Self-Service** ability to easily:

- Provision MongoDB Cluster in highly customized scenario to ensure production readiness and full compliance with customer standards (i.e., security policies, etc.).
- Provision MongoDB Clusters of any type to meet minimum requirements around compute and security, while providing day 2 actions to include advanced lifecycle scenarios that are very specific to MongoDB Operator (i.e., upgrade, scaling, etc.).
- Incorporate all internal runbook procedures required to properly manage the Full Lifecycle of Provisioning, Managing, Operating and Decommission any environment (e.g., DNS, IPAM, LoadBalancers, etc.)
- Integration with Service Management Portals such as **ServiceNow** for improved asset / incident management workflows.
- Team Level visibility around showback and quota utilization to determine whether there are opportunities to free up resources or need to expand resources on-demand.
- Enhanced RBAC to control access to what, who and where folks can provision resources across internal and public cloud environments.

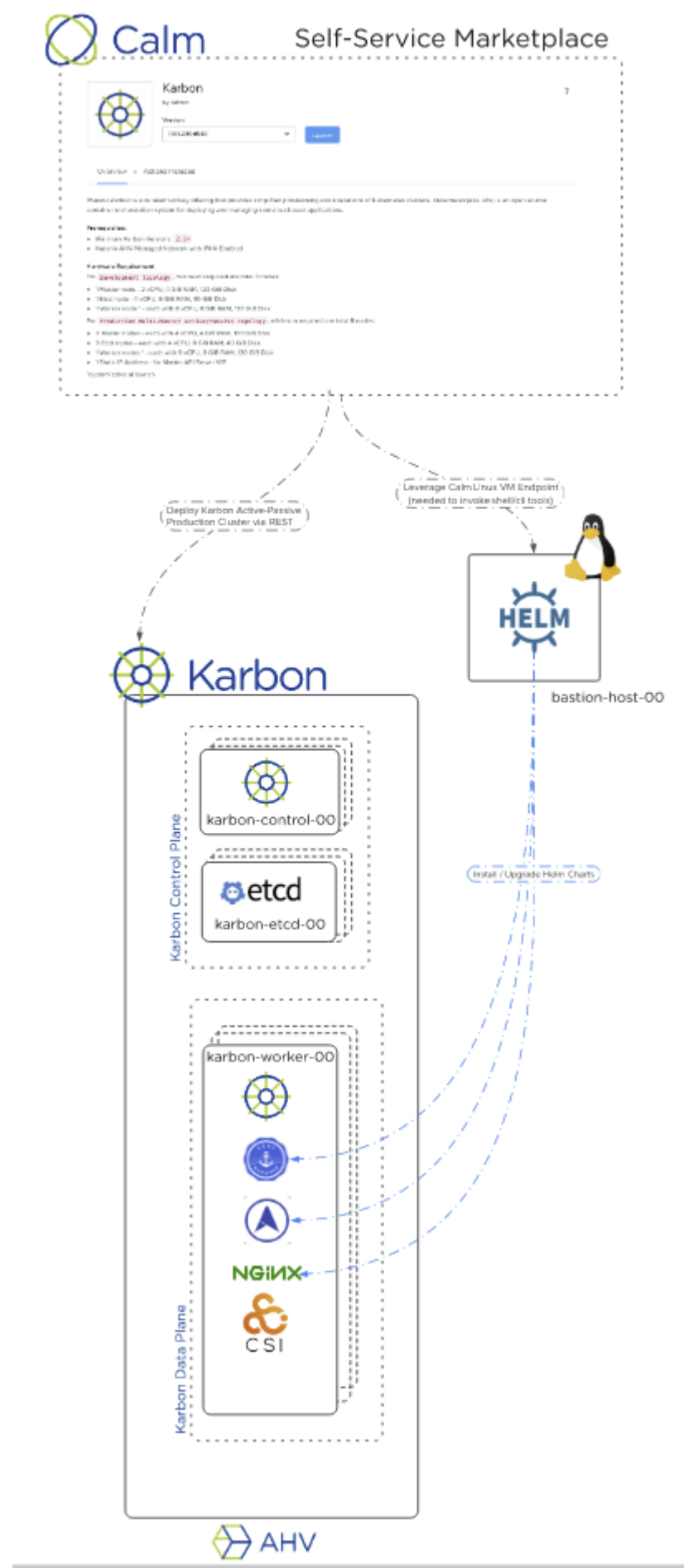
### Concerns/Limitations:

- While many of the ERA limitations (documented above) can be mitigated via highly customized automation & orchestration available directly via Calm blueprints - the effort to handle all the full lifecycle of all use cases "end to end" - would be a relatively significant effort in comparison to leveraging either ERA or the MongoDB Enterprise Operator on Kubernetes.
  - i.e., Automation that is leveraged to incorporate needs for Persistent Storage, Data Protection, Provisioning, Upgrading, Scaling, Quiescing, Self-Healing, Registration/De-Registration with Opsmanager, Snapshot/Restore, User/Secrets Management, would need to be continuously managed/tested/validated for a myriad of use cases and backward compatability - effectively slowing down adoption of newer mongodb releases that provide feature enhancements that could improve overall customer satisfaction.

## Leverage Nutanix Self-Service (Calm) UI to Deploy MongoDB on NKE

---

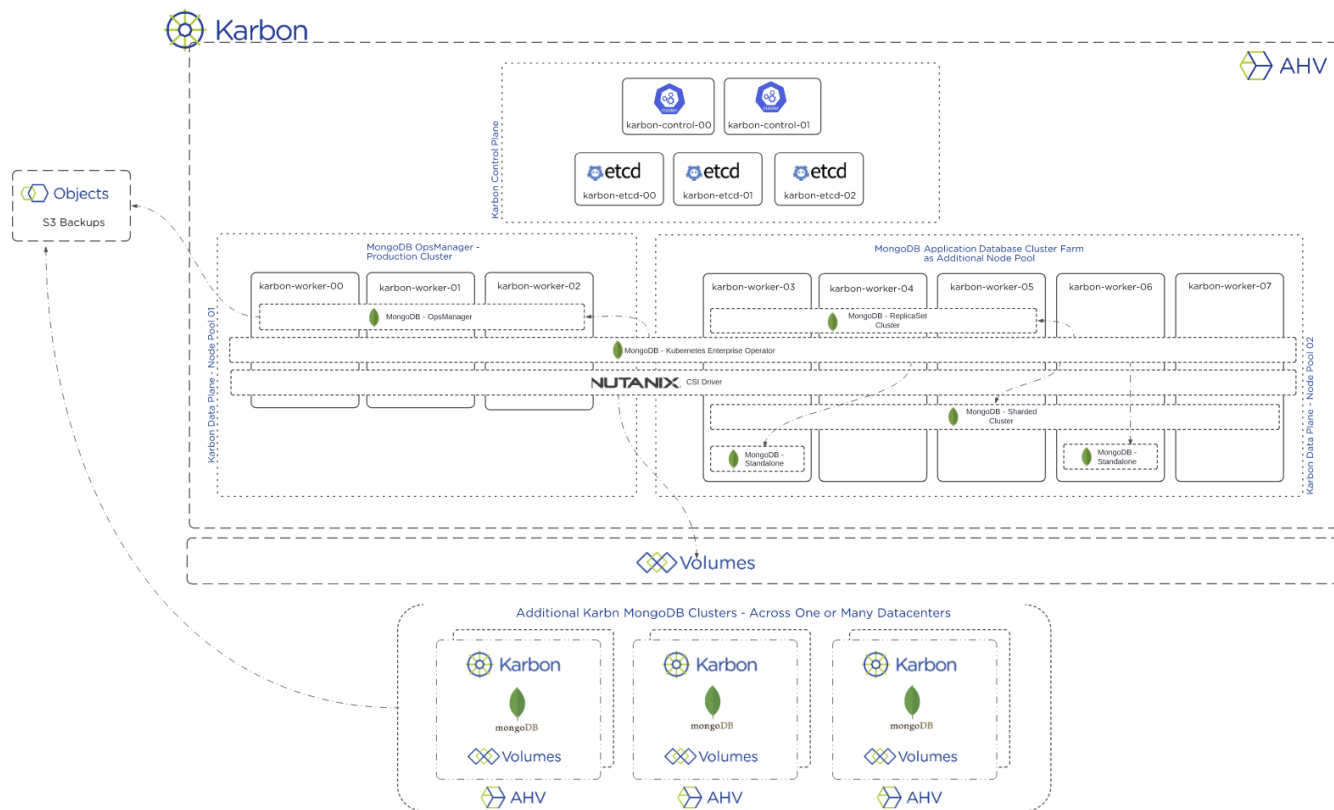
Nutanix Calm would be leveraged to deploy a dedicated Nutanix Karbon Production Cluster with the Nutanix CSI Driver, and subsequently deploy the MongoDB Enterprise Operator as a means to configure MongoDB custom resources - such as MongoDB, OpsManager and Users overall.



The MongoDB Enterprise Operator enables easy deploy of the following applications into Kubernetes clusters:



**MongoDB** - Replica Sets, Sharded Clusters and Standalones - with authentication, TLS and many more options. **Ops Manager** - our enterprise management, monitoring and backup platform for MongoDB. The Operator can install and manage Ops Manager in Kubernetes for you. Ops Manager can manage MongoDB instances both inside and outside Kubernetes.



By Leveraging **NCM/Calm**, you'll have the ability to provide end users the **Self-Service** ability to easily:

- Provision Karbon Cluster in highly customized scenario to ensure production readiness and full compliance with customer standards (i.e., security policies, ingress, image registries, limits/quotas, etc.).
- Curate Kubernetes Applications (along with MongoDB Operator) to fully include all customer specific requirements (i.e., naming standards, persistent storage layout, etc.).
- Provision MongoDB Clusters of any type to meet minimum requirements around compute and security, while providing day 2 actions to include advanced lifecycle scenarios that are very specific to MongoDB Operator (i.e., upgrade, scaling, etc.).
- Incorporate all internal runbook procedures required to properly manage the Full Lifecycle of Provisioning, Managing, Operating and Decommission any environment. (e.g., DNS, IPAM, LoadBalancers, etc.)
- Plugin Integration with Service Management Portals such as **ServiceNow** for improved asset / incident management workflows.

By Leveraging **NKE/Karbon**, you'll have the ability to easily:

- Provision Highly Available Production Clusters with Nutanix CSI Driver Auto-Provisioned
- Upgrade Kubernetes Clusters and underlying Node OS
- Scale Existing Worker Node Pools to add more Compute Resources
- Add Worker Node Pools for Specialized Workload Requirements (e.g., CPU/GPU/Memory Optimized, etc.)

- Easily connect to Kubernetes API Server via Kubectl via Karbon API or Plugins (i.e., `krew install karbon`)

By Leveraging the **Nutanix CSI Driver**, you'll have the ability to easily:

- Dynamically Provision Nutanix Volumes (RWO/BLOCK) or Nutanix Files (RWX/NFS)
- Leverage metrics to determine overall disk utilization from K8s or Nutanix Prism
- Expand, Clone and/or Snapshot Volumes
- Create Additional Storage Classes to handle advance use cases, such as:
  - Configuring Additional LVM Virtual Disks to Distribute IO
  - Workloads that require High Throughput/IO capabilities via ALL Flash Enabled Storage Pools.

By Leveraging the **MongoDB Enterprise Operator**, you'll have the ability to:

- Auto-Register and De-Register Clusters from OpsManager
- Configure S3 Backup within OpsManager and Continuously Backup all Registered Databases
- Create Multiple MongoDB Standalone, Replica sets and Sharded Clusters
- Upgrade and downgrade MongoDB server version
- Scale Replicas of All types up and down
- Use any of the Custom or Available Docker MongoDB images
- Connect to the replica set from inside the Kubernetes cluster without exposing Externally
- Secure client-to-server and server-to-server connections with mTLS/TLS
- Create users with SCRAM authentication
- Create custom roles
- Enable metrics targets that can be used with Prometheus and Grafana Dashboards for Enhanced Observability

### Pros:

- MongoDB Enterprise Operator is managed/supported by MongoDB
- NKE/Karbon is fully managed kubernetes distribution supported by Nutanix
- Nutanix CSI Driver could be leveraged on just about any Kubernetes Distribution (e.g., Red Hat Openshift, Rancher RKE/RKE2/K3s, Vanilla K8s, etc.) and Supported OS (e.g., CentOS,RHEL,Ubuntu, etc.) if other options are preferred.

### Concerns/Limitations:

- Karbon can only be deployed on Nutanix AHV
- Karbon manages entire stack - including Node OS - which is currently CENTOS
- Karbon doesn't include integrated dashboard to easily manage K8s objects from Prism Central
- Karbon is ultra-slim version of Kubernetes, so highly dependent on third-party solutions to manage Ingress, External Service LoadBalancing, Multi-Cluster Governance (i.e., Global Security Policies and Multi-Team)
- Team Level visibility and governance capabilities are limiting.

## Example Requirement Scenarios

Requirement: DBA Only Accessible Feature - Deploy New Dedicated MongoDB VM

Leverage Calm and Karbon to Deploy MongoDB OpsManager Cluster

- **Pre-Reqs:**

- In Calm UI, configure new project (i.e, development-team-a), account & environment and subsequently configure LDAP users/groups with appropriate role

development-team-a

API Equivalent

Overview

Users, Groups and Roles

Accounts

Environments

Policies

Project Description

No description added [Add Description](#)

Project Setup

Users, Groups & Roles

Add users and groups to have access control to the project

5 users have access >

Add Users

Accounts

Add resources you want this project to consume

2 accounts added >

Add Accounts

Environments

Define deployment infrastructure and VM/Pod defaults for quick app deployments

2 environments added >

Create Environment

Users, Groups and Roles


+ Add User

Name	Role	
admin	Admin	
ssp admins	Project Admin	<div>⌵ Delete</div>
ssp consumers	Consumer	<div>⌵ Delete</div>
ssp developers	Developer	<div>⌵ Delete</div>
ssp operators	Operator	<div>⌵ Delete</div>

☒ Allow Collaboration

Collaboration enables users of a project to manage each other's VMs and Applications. This flag cannot be changed once a user is added to the project and the project is saved. To change its value, all the users must be first removed from this project.

- Deploy and Configure Linux VM / Bastion Host Endpoint with necessary utilities and DNS A Host Record as needed



bastion-host-svm

Blueprint

by admin

Version

1.0.5-7f62e34

▼

Launch

Clone

Overview

Actions Included

The purpose of this bastion host is to provide a linux endpoint that has connectivity and the necessary utilities needed to manage applications.

**Utilities**

The following command line utilities have been installed and configured on the underlying bastion host based:

**Common Utilities**

- git
- jq
- yq

**Nutanix Utilities**

- calm dsl
- karbonctl

Applications > bastion-host-svm-kalm-main-default\_params-29954 RUNNING

Overview • Manage • Metrics • Recovery Points • Audit • ?

KC

Bastion\_HostProfileService - Install Kube CTL

Finished - 07/20/2022

⌵

KA

Bastion\_HostProfileService - Configure Kubectl Aliases

Finished - 07/20/2022

⌵

AK

Bastion\_HostProfileService - Install Kubectl and Kubens

Finished - 07/20/2022

⌵

KP

Bastion\_HostProfileService - Install Kube PS1

Finished - 07/20/2022

⌵

PM

Bastion\_HostProfileService - Install Kubectl Krew Package Manager

Finished - 07/20/2022

⌵

IS

Bastion\_HostProfileService - Install Stern

Finished - 07/20/2022

⌵

IJ

Bastion\_HostProfileService - Install JQ

Finished - 07/20/2022

⌵

IP

Bastion\_HostProfileService - Install Packages

Finished - 07/20/2022

⌵

IH

Bastion\_HostProfileService - Install Helm

Finished - 07/20/2022

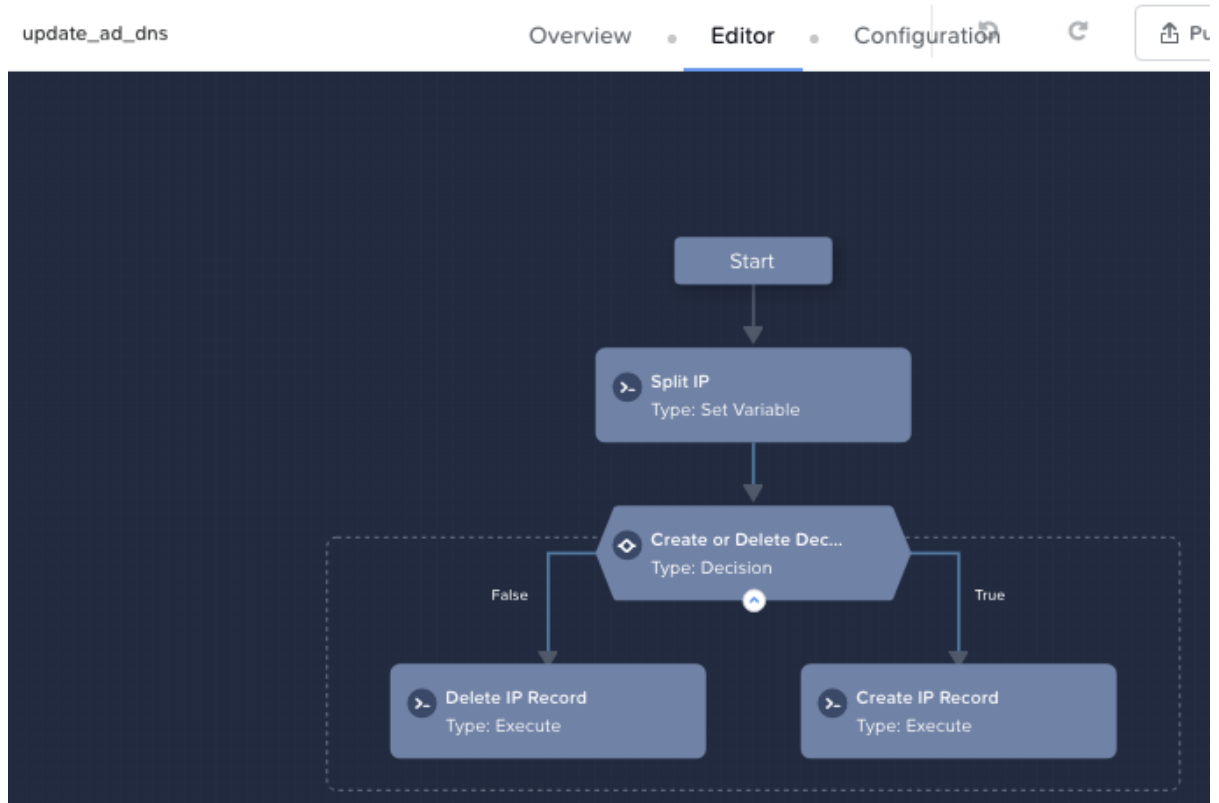
⌵

IV

Bastion\_HostProfileService - Install Vim

Finished - 07/20/2022

⌵



- As DBA, Deploy Karbon Production Cluster via Self-Service Marketplace with a minimum of 7 worker nodes



## karbon Blueprint

by admin

Version

1.0.5-7f62e34

Launch

Clone

Overview

Actions Included

*Nutanix Karbon* is a curated turnkey offering that provides simplified provisioning and operations of Kut applications.

### Prerequisites:

- Minimum Karbon Versions: **2.1+**
- Nutanix AHV Managed Network with IPAM Enabled

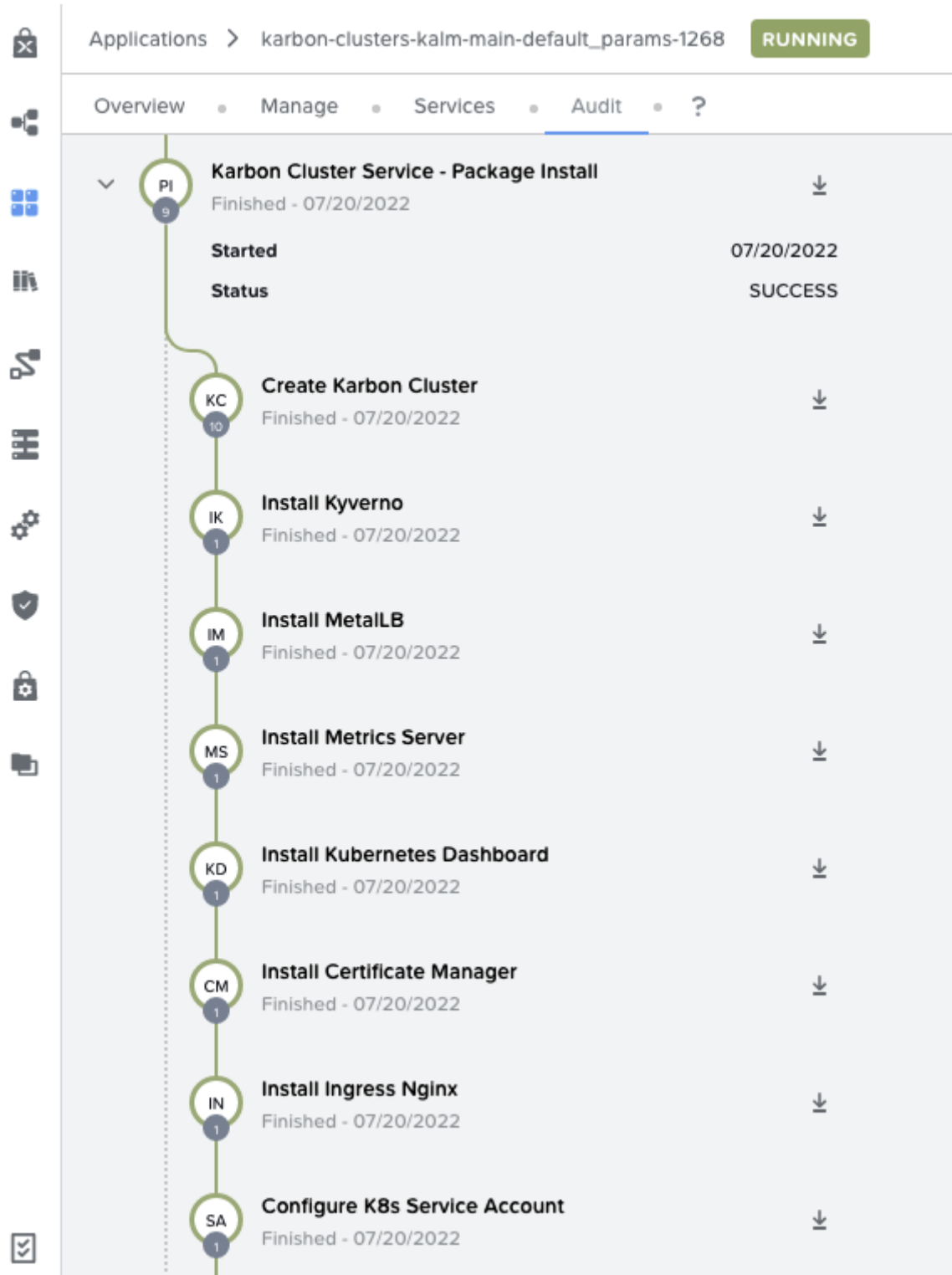
### Hardware Requirement:

For **Development topology**, minimum required are total 3 nodes:

- 1 Master node - 2 vCPU, 4 GiB RAM, 120 GiB Disk
- 1 Etcd node - 4 vCPU, 8 GiB RAM, 40 GiB Disk
- 1 Worker node \* - each with 8 vCPU, 8 GiB RAM, 120 GiB Disk

For **Production Multi-Master Active/Passive topology**, minimum required are total 6 nodes:

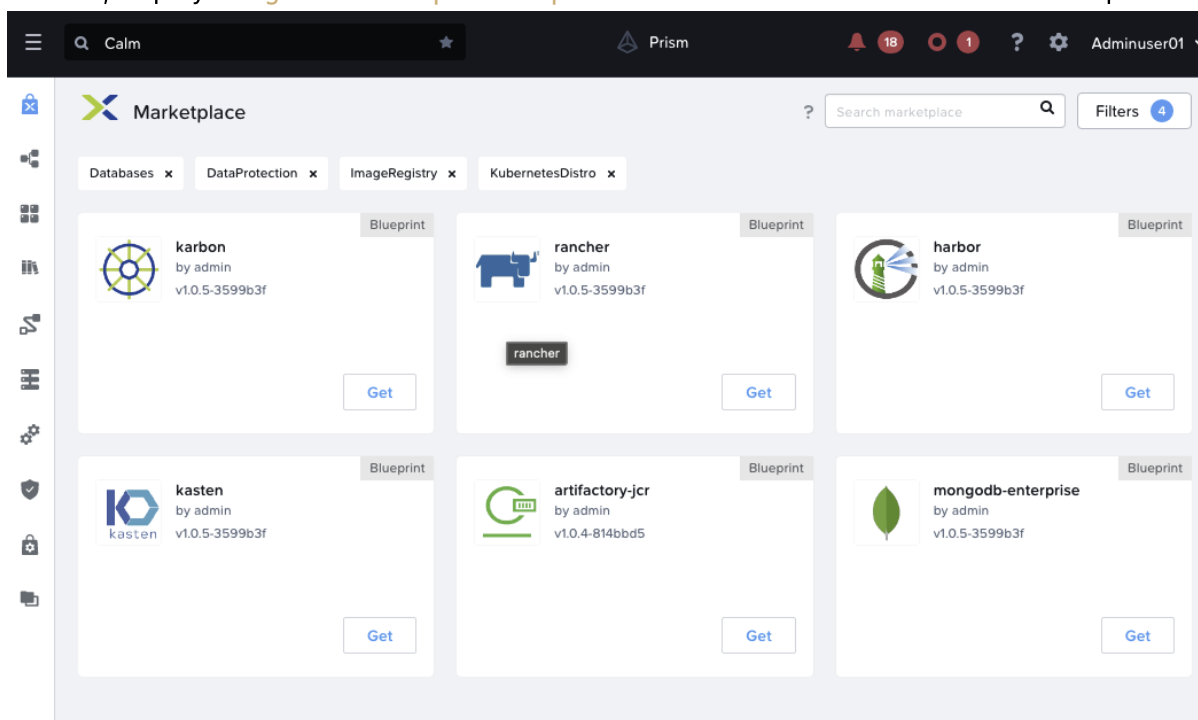
- 2 Master nodes - each with 4 vCPU, 4 GiB RAM, 120 GiB Disk
- 3 Etcd nodes - each with 4 vCPU, 8 GiB RAM, 40 GiB Disk
- 1 Worker nodes \* - each with 8 vCPU, 8 GiB RAM, 120 GiB Disk
- 1 Static IP Address - for Master API Server VIP



- Demo:



- As DBA, Deploy **MongoDB Enterprise Operator** to Karbon Production from Marketplace



## mongodb-enterprise Blueprint

by admin

Version

1.0.5-7f62e34

Launch

Clone

Overview

Actions Included

### MongoDB Enterprise Operator Helm Chart

The MongoDB Enterprise Kubernetes Operator provides a container image for the MongoDB Agent in Ops Manager.

This allows you to manage and deploy MongoDB database clusters with full monitoring, backups, and automation provided by The Operator enables easy deploy of the following applications into Kubernetes clusters:

**MongoDB** - Replica Sets, Sharded Clusters and Standalones - with authentication, TLS and many more options. **Ops Manager** Ops Manager in Kubernetes for you. Ops Manager can manage MongoDB instances both inside and outside Kubernetes.

#### Chart Details

This chart will do the following:

- Deploy MongoDB Enterprise Operator

#### Prerequisites

- Existing Karbon Cluster

The following services have been pre-configured:

- MetalLB** - [more info](#)
- Cert-Manager** - [more info](#)

#### Day 2 Actions

- Configure MongoDB OpsManager Cluster (+ Backend ApplicationDB ReplicaSet)
  - Expose MongoDB OpsManager via MetalLB Service LoadBalancer
- Configure MongoDB Standalone Instance
- Configure MongoDB ReplicaSet Cluster
- Configure MongoDB Sharded Cluster

◦ As DBA, Deploy MongoDB OpsManager Cluster as Day 2 Action

Applications > mongodb-operator-opsmgr-prod RUNNING

Overview • Manage • Services • Audit • ?

See helm-mongodb-enterprise-feat-mongodb-s-118ab56  
Commit History: [feat/mongodb-s](#)

Connectivity Details

OpsManager URL and API Keys for Deploying MongoDB AppDB:

```
OPSMANAGER_HOST=$(kubectl get svc mongodb-opsmgr-svc-ext -n mongodb-enterprise -o jsonpath="{.status.loadBalancer.ingress[0].ip}")
```

Variables (3)

**MongodbEnterprise Instance Name** ⓘ  
mongodb-enterprise

**Kubernetes Cluster Name** ⓘ  
kalm-main-12-1

**MongodbEnterprise Namespace** ⓘ  
mongodb-enterprise

### App Summary

Application UUID	b2575b58-172b-4147-8dc8-f61a66a2b601
Marketplace Item	<b>mongodb-enterprise (v1.0.5-3599b3f)</b>
Application Profile	Default
Provider	
Project	<a href="#">Bootcampinfra</a>
Environment	<a href="#">default</a>
Owner	adminuser01@ntnslab.local
Created On	22 days ago
Last Updated On	22 days ago

Applications > mongodb-operator-opsmgr-prod RUNNING

Overview • Manage • Services • Audit • ?

✓

>\_

Configure OpsManager Instance

Finished - 07/20/2022

Started07/20/2022

Run Byadminuser01@ntnslab.local

StatusSUCCESS

PE

Policy Execute - Approval

Finished - 07/20/2022

✓

OI

2

Configure OpsManager Instance

Finished - 07/20/2022

Started07/20/2022

StatusSUCCESS

MI

Helm\_MongodbEnterprise - Configuring MongoDB Instance

Finished - 07/20/2022

Started07/20/2022

StatusSUCCESS

Output • Script

View Macros Expanded

```
37 ---
38 apiVersion: mongodb.com/v1
39 kind: MongoDBOpsManager
40 metadata:
41   name: mongodb-opsmanager
42 spec:
43   replicas: $( echo $OPSMANAGER_REPLICA
44   version: $( echo $OPSMANAGER_VERSION
45   adminCredentials: om-admin-secret
46   externalConnectivity:
47     type: LoadBalancer
48   applicationDatabase:
49     members: $( echo $OPSMANAGER_APPDB_
50     version: $( echo $OPSMANAGER_APPDB_
51   configuration:
```

- [Manual] Login to MongoDB OpsManager UI and Show Initial OpsManager Cluster

Applications > mongodb-operator-opsmgr-prod RUNNING

Overview • Manage • Services • Audit • ?

Create

Start

Restart

Stop

Delete

Soft Delete

Configure OpsManager Instance

Configure MongoDB Standalone Instance

Run Action: Configure OpsManager Instance

OpsManager Backend AppDB Replicaset Count  
OpsManager Backend AppDB Replicaset Count  
3

OpsManager Replicaset Count  
OpsManager Replicaset Count  
3

OpsManager MongoDB AppDB Version  
OpsManager MongoDB AppDB Version  
4.4.4-ent

OpsManager Version  
OpsManager Version  
5.0.10

API Equivalent

Cancel Run

< BACK TO PROJECT

mongoDB Ops Manager

GLOBAL ADMIN

Admin

mongodb-opsmgr-prod

Overview

Ops Manager Config

Users

API Keys

Global Access List

Projects

Organizations

Logs

Version Manifest

Messages

Audits

MongoDB Usage

System Overview

Active Projects 5

Total Projects 5

Total Users 1

Enabled Hosts 21

Diagnostic Archive  
Download: .tar / .zip

Total Page Views

SERVICES

MONGODB PROCESSES

SYSTEM WARNINGS 20

mongodb-oplog-replicaset-0

State	Port	Version
mongodb-o...	27017	4.2.2

mongodb-oplog-replicaset-1

State	Port	Version
mongodb-o...	27017	4.2.2

mongodb-oplog-replicaset-2

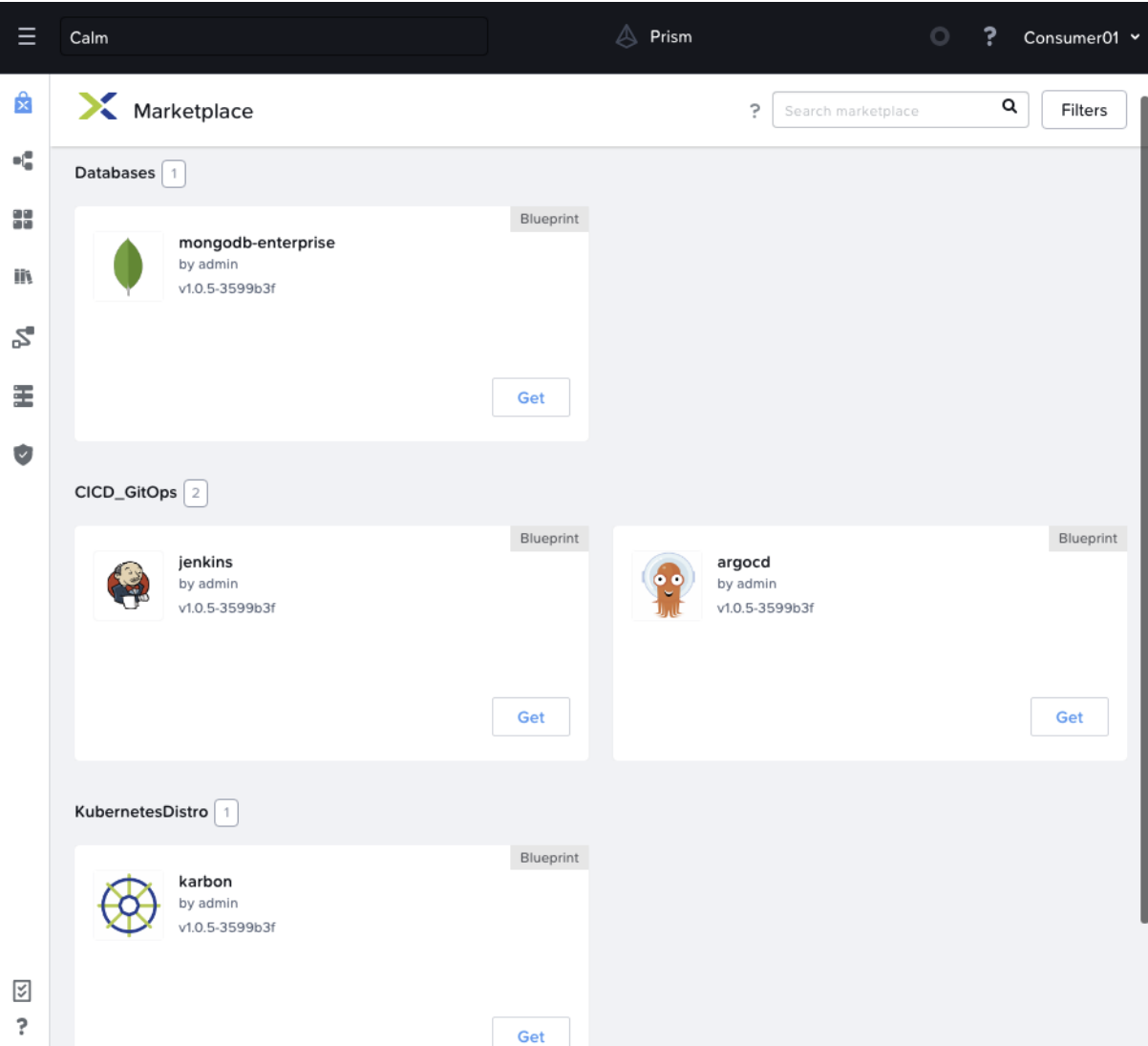
State	Port	Version
mongodb-o...	27017	4.2.2

mongodb-opmanager-0

State	Port	Version
Ops Manager	8080	5.0.10.10...

20 / 42

- As Developer/Consumer, Deploy Karbon Development Cluster from Marketplace



Applications > kalm-develop-5-2 **RUNNING** Delete ?

Overview • Manage • Services • **Audit** • ?

- Create**  
Finished - 07/20/2022  
Started: 07/20/2022  
Run By: consumer01@ntnslab.local  
Status: SUCCESS
- Policy Execute - Approval**  
Finished - 07/20/2022
- Karbon Cluster Service - Substrate Create**  
Finished - 07/20/2022
- Karbon Cluster Service - Package Install**  
Finished - 07/20/2022
- Karbon Cluster Service Create**  
Finished - 07/20/2022
- Karbon Cluster Service Start**  
Finished - 07/20/2022

- As Developer/Consumer, Deploy MongoDB Enterprise Operator to Karbon Development from Marketplace

Applications > mongodb-operator-development **RUNNING**

Overview • Manage • Services • **Audit** • ?

- Started**  
07/20/2022
- Status**  
SUCCESS
- Helm\_MongodbEnterprise - Install MongodbEnterprise Helm Chart**  
Output • Script  
View Macros Expanded ☒

```

1 WILDCARD_INGRESS_DNS_FQDN=kalm-main-12-1.ntnslab.local
2 NAMESPACE=mongodb-enterprise
3 INSTANCE_NAME=mongodb-enterprise
4 K8S_CLUSTER_NAME=kalm-develop-5-2
5
6 export KUBECONFIG=~/${K8S_CLUSTER_NAME}_${INSTANCE_NAME}.cfg
7
8 kubectl create ns ${NAMESPACE} --dry-run=client -o yaml | kubectl apply -f -
9
10 # this step will configure helm chart with ingress tls enabled and self-signed certs managed
11 helm repo add mongodb https://mongodb.github.io/helm-charts
12
13 helm repo update
14 helm search repo mongodb/enterprise-operator
15 helm upgrade --install ${INSTANCE_NAME} mongodb/enterprise-operator \
16   --set operator.watchNamespace='*' \
17   --namespace ${NAMESPACE} \
18   --wait-for-jobs \
19   --wait
20
21 kubectl wait --for=condition=Ready pod -l app.kubernetes.io/name=mongodb-enterprise-operator
22
23 helm status ${INSTANCE_NAME} -n ${NAMESPACE}

```

- Cheatsheet:

1. Login into Nutanix PC with `adminuser01@ntnslab.local` [Database-Admin]
2. Deploy MongoDB Enterprise Operator to Karbon Production from Marketplace
3. Deploy MongoDB OpsManager Cluster as Day 2 Action with Default Settings
4. Monitor OpsManager deployment progress from Calm Application Audit view and/or kubectl

```
## Set Default OPSMANAGER namespace
OPSMANAGER_NAMESPACE=mongodb-enterprise

## Setup Monitoring
watch kubectl get om,sts,pvc,po,svc -n ${OPSMANAGER_NAMESPACE}
watch "kubectl top nodes && echo '\n' && kubectl top pods -n
${OPSMANAGER_NAMESPACE}"

## Get OPSMANAGER URL
OPSMANAGER_HOST=$(kubectl get svc mongodb-opsmanager-svc-ext -n
${OPSMANAGER_NAMESPACE} -o jsonpath=
"{{.status.loadBalancer.ingress[0].ip}}")
OPSMANAGER_BASE_URL="http://opsmanager.${OPSMANAGER_HOST}.nip.io:8080"
echo "OPSMANAGER_BASE_URL=${OPSMANAGER_BASE_URL}"

## Troubleshooting
kubectl get om -o yaml -n ${OPSMANAGER_NAMESPACE}
```

5. Login to MongoDB OpsManager UI and show initial OpsManager Replicaset

```
## Set Default OPSMANAGER namespace
OPSMANAGER_NAMESPACE=mongodb-enterprise

## Login with admin user and password retrieved below
kubectl get secret om-admin-secret -o jsonpath='{.data.Password}' -n
${OPSMANAGER_NAMESPACE} | base64 -d && echo
```

6. Login into Nutanix PC with `consumer01@ntnslab.local` [Self-Service-User]
7. Deploy Karbon Development Cluster (i.e., `kalm-develop-5-2`) to Secondary AHV Cluster and define valid IP range to define for MetalLB (i.e., `10.38.5.90-10.38.5.91`)
8. Connect to Karbon Development Cluster (i.e., `kalm-develop-5-2`) cluster from kubectl
9. Deploy MongoDB Enterprise Operator to Karbon Development Cluster (i.e., `kalm-develop-5-2`) from Marketplace

Requirement: Deploy Container on existing VMS

Leverage MongoDB Enterprise Operator & Calm to Deploy MongoDB Instance and Auto-Register Into OpsManager

- Demo:

- Deploy **MongoDB Database Standalone Instance** as Day 2 Action on Production Cluster

The screenshot shows the Calm console interface. On the left, the 'Manage' tab is active, displaying a list of actions for the 'mongodb-operator-opsmgr-prod' application. The 'Run Action: Configure MongoDB Standalone Instance' dialog is open, showing the following configuration:

- MongoDB Standalone Instance Name: mongodb-demo-standalone
- Storage Class Name: default-storageclass
- MongoDB AppDB Journal Mount Size: 1Gi
- MongoDB AppDB Logs Mount Size: 500M
- MongoDB AppDB Data Mount Size: 10Gi
- MongoDB AppDB Memory Limits: 2G
- MongoDB AppDB CPU Limits: 2

On the right, the 'Configure MongoDB Standalone Instance' action history is visible, showing a successful execution on 07/20/2022 by adminuser01@ntnlab.local.

- Deploy **MongoDB Database Replica Set Cluster** as Day 2 Action on Production Cluster

The screenshot shows the Calm console interface. On the left, the 'Manage' tab is active, displaying a list of actions for the 'mongodb-operator-opsmgr-prod' application. The 'Run Action: Configure MongoDB ReplicaSet Cluster' dialog is open, showing the following configuration:

- MongoDB ReplicaSet Cluster instance Name, to be used for K8s Namespace and OpsManager Project: mongodb-demo-replicaset
- Storage Class Name: default-storageclass
- MongoDB AppDB Journal Mount Size: 1Gi
- MongoDB AppDB Logs Mount Size: 500M
- MongoDB AppDB Data Mount Size: 10Gi
- MongoDB AppDB Memory Limits: 2G
- MongoDB AppDB CPU Limits: 2
- MongoDB AppDB Container Image Name: mongodb-enterprise-database
- MongoDB AppDB Version: 4.4.4-ent
- MongoDB AppDB ReplicaSet Count: 3

On the right, the 'Configure MongoDB ReplicaSet Cluster' action history is visible, showing a successful execution on 07/20/2022 by adminuser01@ntnlab.local. Below the history, the 'View Macros Expanded' section shows the following YAML snippet:

```
107 ---
108 apiVersion: mongodb.com
109 kind: MongoDB
110 metadata:
111   name: $( echo $OM_PRO
112 spec:
113   members: $( echo $HOM
114   version: $( echo $HO
115   service: $( echo $OH
116   opsManager:
117     configMapRef:
118       name: $( echo $OM
119   credentials: organiza
120   persistent: true
121   storageVolumeName: $(
```



- Deploy **MongoDB Database Sharded Cluster** as Day 2 Action on Production Cluster

The screenshot displays the MongoDB OpsManager configuration interface. On the left, a sidebar lists actions like Create, Start, Restart, Stop, Delete, Soft Delete, and various configuration options for the MongoDB instance. The main panel shows configuration fields for the MongoDB AppDB, including:
 

- MongoDB AppDB Logs Mount Size: 500M
- MongoDB AppDB Data Mount Size: 10Gi
- MongoDB AppDB Memory Limits: 2G
- MongoDB AppDB CPU Limits: 2
- MongoDB AppDB Container Image Name: mongodb-enterprise-database
- MongoDB AppDB Version: 4.4.4-ent
- MongoDB AppDB ConfigServer Count: 3
- MongoDB AppDB Mongos Count: 2
- MongoDB AppDB Mongos per Shard Count: 3
- MongoDB AppDB Shard Count: 2

 On the right, a vertical timeline shows the execution of several tasks: 'Policy Execute - Approval' (Finished - 07/20/2022), 'Configure MongoDB Sharded Cluster' (Finished - 07/20/2022), 'Helm\_MongodbEnterprise - Get Kubeconfig' (Finished - 07/20/2022), and 'Helm\_MongodbEnterprise - Configure MongoDB Clusters' (Finished - 07/20/2022). The status for all these tasks is 'SUCCESS'.

- [Manual] Login to **MongoDB OpsManager UI** and Show New Clusters Already Registered

The screenshot shows the 'All Clusters' page in the MongoDB OpsManager UI. At the top, there are filters for Availability, Type, Version, and Configuration. Below the filters, it indicates 'Showing 5 of 5 clusters.' and a checkbox for 'Show Inactive Clusters'. The clusters are listed in a table-like format with the following details:

Cluster Name	Version	Data Size	Nodes	Backup	SSL	Auth	Alerts
<b>mongodb-opsmanager-db/mongodb-demo-replicaset-project</b> Ops Manager	5.0.1	1.94 KB	3	OFF	OFF	OFF	
<b>mongodb-opsmanager-db/mongodb-demo-shardedcluster-project</b> Ops Manager	4.4.4	N/A	11	OFF	OFF	OFF	
<b>mongodb-opsmanager-db/mongodb-demo-standalone-project</b> Ops Manager	4.4.4	N/A	1	OFF	OFF	OFF	
<b>mongodb-opsmanager-db/mongodb-oplog-replicaset-project</b> Ops Manager							

- [Manual] Get **Organization ID, API Keys** via OpsManager App UI or Audit

```

at + kalm-main-12-1-context
OPS_MANAGER_NAMESPACE=mongodb-enterprise

OPS_MANAGER_NAMESPACE=mongodb-enterprise
OPS_MANAGER_HOST=$(kubectl get svc mongodb-opsmanager-svc-ext -n $(OPS_MANAGER_NAMESPACE) -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
OPS_MANAGER_BASE_URL="https://opsmanager.$(OPS_MANAGER_HOST).nip.io:8080"
OPS_MANAGER_API_KEY=$(kubectl get secrets mongodb-enterprise-mongodb-opsmanager-admin-key -n $(OPS_MANAGER_NAMESPACE) -o jsonpath='{.data.privateKey}' | base64 -d)
OPS_MANAGER_API_USER=$(kubectl get secrets mongodb-enterprise-mongodb-opsmanager-admin-key -n $(OPS_MANAGER_NAMESPACE) -o jsonpath='{.data.privateKey}' | base64 -d)
OPS_MANAGER_ORG_ID=$(curl -u $(OPS_MANAGER_API_USER):$(OPS_MANAGER_API_KEY) --digest -s --request GET "$OPS_MANAGER_HOST:8080/api/public/v1.0/orgs?pretty=true" | jq -r '.results[0].id')

echo "OPS_MANAGER_API_KEY=$(OPS_MANAGER_API_KEY)"
echo "OPS_MANAGER_API_USER=$(OPS_MANAGER_API_USER)"
echo "OPS_MANAGER_ORG_ID=$(OPS_MANAGER_ORG_ID)"
echo "OPS_MANAGER_BASE_URL=$(OPS_MANAGER_BASE_URL)"
    
```

Applications > mongodb-operator-opsmgr-prod **RUNNING**

Overview • Manage • Services • Audit •

### Connectivity Details

OpsManager URL and API Keys for Deploying MongoDB AppDB:

```
OPSMANAGER_HOST=$(kubectl get svc mongodb-opsmgr-svc-ext -n mongodb-enterprise -o jsonpath="{.status.loadBalancer.ingress[0].ip}")
OPSMANAGER_BASE_URL="http://opsmgr.${OPSMANAGER_HOST}.nip.io:8080"

OPSMANAGER_API_USER=$(kubectl get secrets mongodb-enterprise-mongodb-opsmgr-admin-key -n mongodb-enterprise -o jsonpath='{.data.publicKey}' | base64 -d)
OPSMANAGER_API_KEY=$(kubectl get secrets mongodb-enterprise-mongodb-opsmgr-admin-key -n mongodb-enterprise -o jsonpath='{.data.privateKey}' | base64 -d)
OPSMANAGER_ORG_ID=$(curl -u ${OPSMANAGER_API_USER}:${OPSMANAGER_API_KEY} --digest -s --request GET "${OPSMANAGER_HOST}:8080/api/public/v1.0/orgs?pretty=true" | jq -r '.results[0].id')
```

- Deploy MongoDB Database Replica Set Cluster as Day 2 Action on Development Cluster (with Creds)

OpsManager API Key [Optional]  
OpsManager API Key. i.e., 827c16bb-5f6e-4ed8-a234-95066d7a6684

OpsManager API Key User [Optional]  
OpsManager API Key User. i.e., jgejkwud

OpsManager Organization ID [Optional]  
OpsManager Organization ID. i.e., 62c7a4dbbdf127f78561be3

OpsManager External URL Endpoint [Optional]  
OpsManager External URL Endpoint - only if registering from external K8s clusters. i.e.,  
http://opsmanager-ext.ntnslab.local

API Equivalent Cancel Run

- Cheatsheet:

1. Login into Nutanix PC with `adminuser01@ntnslab.local` [Database-Admin]
2. Setup Monitoring via `kubectl` after each scenario below to monitor progress, updating `MONGODB_INSTANCE` var respectively:
  1. Deploy MongoDB Database Standalone Instance as Day 2 Action on Karbon Production Cluster (i.e., kalm-main-12-1) with Default Settings
  2. Deploy MongoDB Database Replica Set Cluster as Day 2 Action on Karbon Production Cluster (i.e., kalm-main-12-1) with Default Settings
  3. Deploy MongoDB Database Sharded Cluster as Day 2 Action on Karbon Production Cluster (i.e., kalm-main-12-1) with Default Settings
3. Login to MongoDB OpsManager UI and Show New Clusters Already Registered
4. Get Organization ID, API Keys via OpsManager App UI or Audit

```
## Set Default OPSMANAGER namespace
OPSMANAGER_NAMESPACE=mongodb-enterprise

# GET OPSMANAGER CRED FROM OPSMANAGER NAMESPACE
OPSMANAGER_NAMESPACE=mongodb-enterprise
OPSMANAGER_HOST=$(kubectl get svc mongodb-opsmanager-svc-ext -n
${OPSMANAGER_NAMESPACE} -o jsonpath="{.status.loadBalancer.ingress[0].ip}")
OPSMANAGER_BASE_URL="http://opsmanager.${OPSMANAGER_HOST}.nip.io:8080"
OPSMANAGER_API_USER=$(kubectl get secrets mongodb-enterprise-mongodb-opsmanager-admin-key -n ${OPSMANAGER_NAMESPACE} -o
```

```

jsonpath='{.data.publicKey}' | base64 -d)
OPSMANAGER_API_KEY=$(kubectl get secrets mongodb-enterprise-mongodb-opsmanager-admin-key -n ${OPSMANAGER_NAMESPACE} -o
jsonpath='{.data.privateKey}' | base64 -d)
OPSMANAGER_ORG_ID=$(curl -u
${OPSMANAGER_API_USER}:${OPSMANAGER_API_KEY} --digest -s --request GET
"${OPSMANAGER_HOST}:8080/api/public/v1.0/orgs?pretty=true" | jq -r
'.results[].id')

echo "OPSMANAGER_API_KEY=${OPSMANAGER_API_KEY}"
echo "OPSMANAGER_API_USER=${OPSMANAGER_API_USER}"
echo "OPSMANAGER_ORG_ID=${OPSMANAGER_ORG_ID}"
echo "OPSMANAGER_BASE_URL=${OPSMANAGER_BASE_URL}"

```

- Using Output from previous step, deploy **MongoDB Database Replica Set Cluster** as Day 2 Action on **Development Cluster (with Creds)**

Requirement: Ability to Deploy Different Mongo images/versions

Leverage MongoDB Enterprise Operator & Calm to upgrade existing MongoDB Environment.

- Demo:**
  - Leverage Operator to upgrade existing MongoDB instance as Day 2 Action
  - [Manual] Monitor MongoDB Upgrade occurring via kubectl
  - [Manual] Monitor OpsManager Output
- Cheatsheet:**

You can upgrade the major, minor, and/or feature compatibility versions of your MongoDB resource. These settings are configured in your resource's config map

- Find Available **MongoDB Enterprise Container Image Versions** (e.g., or use one of these **4.4.4-ent, 4.4.11-ent, 5.0.1-ent, 5.0.5-ent**, etc.)
- Setup Monitoring via kubectl** and initiate upgrade of MongoDB Cluster

```

## patch mongodb app enterprise version
MONGO_INSTANCE=mongodb-demo-replicaset

## for additional details - watch yaml file updates
kubectl get mongodb $MONGO_INSTANCE -o yaml -w

```

TODO: **Optionally Upgrade MongoDB Operator as Day 2 Action**

TODO: **Upgrade MongoDB Production Cluster as Day 2 Action**

Requirement: Grant permissions to requested user/svc accounts to enable access to container

Leverage Operator to Create custom roles and users with SCRAM authentication

- **Demo:**
  - Configure Custom Developer / Operations Roles as Day 2 Action
  - [Manual] Login to OpsManager and Show Access Manager in UI
- **Cheatsheet:**

Requirement: Ability to prevent creation should specific server metrics drop below critical thresholds (i.e., drive space, container # limits)

Leverage MongoDB Operator and K8s Constructs to Set/Enforce Resource Quotas / Limits / Affinity and Storage Persistence Configurations

- **Demo:**
  - [Manual] Show Resource Constraints for CPU and Memory via PodSpec YAML
  - [Manual] Show Scaling of StatefulSet Replicas via kubectl
  - [Manual] Show Scaling of Worker Nodes via Calm Day 2 Action
  - [Manual] Show Expanding of Volumes (PVC) via kubectl
  - [Manual] Show New ReplicaSet with High Compute Request Workflow
  - [Manual] Show Adding of Worker Node Pool to Karbon and Set Node Labels
  - [Manual] Re-Configure Node Affinity to Pins Pods to New Worker Node Pool
  - [Manual] Show Pod Location per Node

- **Cheatsheet:**

1. Show Resource Constraints for CPU and Memory via PodSpec YAML

```
## query yaml via kubectl
MONGO_INSTANCE=mongodb-demo-replicaset
kubectl get mongodb $MONGO_INSTANCE -n $MONGO_INSTANCE -o yaml
```

The Default PodSpec will Create a MongoDB Replicaset with following Defaults: - StatefulSet with 3 Replicas - CPU and Memory Limits of 2 CPU and 2GB of RAM - Multiple Mount Point Volumes (data:10Gi,journal:1Gi,log:500M), each with own PVC

2. **Setup Monitoring via kubectl** and Scale ReplicaSet Members from 3 to 5

```
## scale replicas by patching mongo instance
MONGO_INSTANCE=mongodb-demo-replicaset
kubectl patch mongodb $MONGO_INSTANCE --type merge -p '{"spec": {"members":5}}'
```

Add worker nodes to pool via Karbon as needed.

1. **Setup Monitoring via kubectl** and **Deploy 2nd ReplicaSet** with more resources than what's available

- Deploy via Calm Day 2 Action a ReplicaSet 3 Member ReplicaSet with 4 vCPU and 8 GB of RAM
  - Show Pending Status on Calm, and Kubectl
  - Add Node Pool of 3 Worker Nodes with 8 vCPU/ 16 GB of RAM via Karbon UI and monitor via Kubectl (i.e., mongodb-node-pool)
    - [Creating a Karbon Node Pool](#)
  - Observe Completion in OpsManager UI, Kubectl, Calm UI

## 2. [Setup Monitoring via kubectl](#) and [Update Existing Worker Pool](#) with Node Labels and Configure, Taints, Tolerations and Node Affinity

- [Option 1](#): via Karbon UI, update node pool with label metadata ([karbon-node-pool=mongodb](#)):
- [Option 2](#): via Kubectl, label nodes with metadata ([karbon-node-pool=mongodb](#))

```
## Label New Node Pool
kubectl get nodes -o name | grep mongodb-pool | xargs -I {node} kubectl
label {node} karbon-node-pool=mongodb --overwrite

## Taint nodes of newly created pool
kubectl taint nodes -l karbon-node-pool=mongodb karbon-node-
pool=mongodb:NoSchedule

## validate that taints have been applied
kubectl describe nodes | grep -i taint
kubectl describe nodes -l karbon-node-pool=mongodb | grep -i taint
```

## 1. [Setup Monitoring via kubectl](#) and [Resize PV Storage](#) for Mount Points

```
## setup monitoring
MONGO_INSTANCE=mongodb-demo-replicaset
watch -n 1 "kubectl get po,pvc -l app=${MONGO_INSTANCE}-service -o
wide && echo && kubectl get mongodb && echo && kubectl top nodes"

## expand data,journal and log pvc storage size
MONGO_INSTANCE=mongodb-demo-replicaset

## data from 10Gi to 1000Gi
kubectl get pvc -l app=${MONGO_INSTANCE}-service -o name | grep data |
xargs -I {} kubectl patch {} -p='{"spec": {"resources": {"requests":
{"storage": "1000Gi"}}}}'

## journal from 1Gi to 100Gi
kubectl get pvc -l app=${MONGO_INSTANCE}-service -o name | grep
journal | xargs -I {} kubectl patch {} -p='{"spec": {"resources":
{"requests": {"storage": "100Gi"}}}}'

## logs from 500M to 100Gi
kubectl get pvc -l app=${MONGO_INSTANCE}-service -o name | grep log |
xargs -I {} kubectl patch {} -p='{"spec": {"resources": {"requests":
```

```

{"storage": "50Gi"}}}}'

## rolling restart of mongodb replicaset
kubectl rollout restart sts ${MONGO_INSTANCE}

#kubectl delete sts --cascade=orphan ${MONGO_INSTANCE}

```

## 2. Setup Monitoring via kubectl and configure LVM volume storage class

```

## Get Secret
NTNX_DYNAMIC_SECRET=$(kubectl get secrets -n kube-system -o name |
grep ntnx-secret | cut -d/ -f2)

## Configure LVM Storage Class
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
  name: lvm-enabled-storageclass
parameters:
  csi.storage.k8s.io/controller-expand-secret-name: $( echo
$NTNX_DYNAMIC_SECRET )
  csi.storage.k8s.io/controller-expand-secret-namespace: kube-system
  csi.storage.k8s.io/fstype: ext4
  csi.storage.k8s.io/node-publish-secret-name: $( echo
$NTNX_DYNAMIC_SECRET )
  csi.storage.k8s.io/node-publish-secret-namespace: kube-system
  csi.storage.k8s.io/provisioner-secret-name: $( echo
$NTNX_DYNAMIC_SECRET )
  csi.storage.k8s.io/provisioner-secret-namespace: kube-system
  flashMode: DISABLED
  storageContainer: Default
  chapAuth: ENABLED
  storageType: NutanixVolumes
  isLVMVolume: "true"
  numLVMDisks: "8"
provisioner: csi.nutanix.com
reclaimPolicy: Delete
allowVolumeExpansion: true
EOF

## get lvm sc yaml
kubectl get sc lvm-enabled-storageclass -o yaml

## get all sc
kubectl get sc

```

- Deploy New MongoDB ReplicaSet Cluster with Storage Class of type `lvm-enabled-storageclass`

### 3. Setup Monitoring via `kubectl` and configure Nutanix Files dynamic volumes storage class and Expand

```
## Get Secret
NTNX_DYNAMIC_SECRET=$(kubectl get secrets -n kube-system -o name |
grep ntnx-secret | cut -d/ -f2)

## set nfs server name - this is case sensitive
NFS_SERVER_NAME="BootcampFS"

## create dynamic storage class provisioner -

cat <<EOF | kubectl apply -f -
allowVolumeExpansion: true
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: dynamic-nfs-sc
provisioner: csi.nutanix.com
parameters:
  csi.storage.k8s.io/node-publish-secret-name: $( echo
$NTNX_DYNAMIC_SECRET )
  csi.storage.k8s.io/node-publish-secret-namespace: kube-system
  csi.storage.k8s.io/controller-expand-secret-name: $( echo
$NTNX_DYNAMIC_SECRET )
  csi.storage.k8s.io/controller-expand-secret-namespace: kube-system
  csi.storage.k8s.io/provisioner-secret-name: $( echo
$NTNX_DYNAMIC_SECRET )
  csi.storage.k8s.io/provisioner-secret-namespace: kube-system
  dynamicProv: ENABLED
  nfsServerName: $( echo $NFS_SERVER_NAME )
  storageType: NutanixFiles
EOF

## get dynamic nfs sc yaml
kubectl get sc dynamic-nfs-sc -o yaml

## get all sc
kubectl get sc
```

### 4. Setup Monitoring via `kubectl` and simulate node failure

- Cordon Node where MongoDB is running

```
## Setup Monitoring
MONGO_INSTANCE=mongodb-demo-replicaset
```



```

watch -n 1 "kubectl get po -l app=${MONGO_INSTANCE}-service -o wide &&
echo && kubectl get mongodb ${MONGO_INSTANCE} && kubectl get nodes"

## Find Node with Replicaset Member and CORDON.
MONGO_INSTANCE=mongodb-demo-replicaset
NODE=`kubectl get pods -l app=${MONGO_INSTANCE}-service -o wide | grep
-v NAME | awk '{print $7}' | head -n 1`
echo $NODE
kubectl cordon ${NODE}
kubectl get nodes

## Delete POD that lives on Node that has been CORDONED.
MONGO_INSTANCE=mongodb-demo-replicaset
POD=`kubectl get pods -l app=${MONGO_INSTANCE}-service -o wide | grep
-v NAME | awk '{print $1}' | head -n 1`
echo $POD
kubectl delete pod ${POD}

## UNCORDON NODE
MONGO_INSTANCE=mongodb-demo-replicaset
NODE=`kubectl get pods -l app=${MONGO_INSTANCE}-service -o wide | grep
-v NAME | awk '{print $7}' | head -n 1`
echo $NODE
kubectl uncordon ${NODE}

```

## Requirement: DR Option

Leverage MongoDB Operator and Objects to Configure OpsManager Backup via S3 Leverage Kasten and Objects to Configure OpsManager & MongoDB Backup Policy based on Label to Objects S3 Leverage Calm to Deploy Karbon and MongoDB Cluster to Secondary AHV Cluster [OPT] Leverage Calm to Deploy Karbon and MongoDB Cluster to Secondary Prism Central / AHV Cluster [OPT]

- **Demo:**
  - [Manual] Show Configuration of Objects S3 Backup via Operator and/or Opsmanager UI
  - [Manual] Show Kasten UI initiate Backups to S3 for both Production and Development
  - [Manual] Show Nutanix Objects UI Explorer for MongoDB Bucket and Kasten Bucket
  - [Manual] Show Karbon Pre-Deploy to Alternative Clusters [OPT]
- **Cheatsheet:**

```

## Get OpsManager vars
OPSMANAGER_NAMESPACE=mongodb-enterprise
OPSMANAGER_HOST=$(kubectl get svc mongodb-opsmanager-svc-ext -n
${OPSMANAGER_NAMESPACE} -o jsonpath="{.status.loadBalancer.ingress[].ip}")
OPSMANAGER_BASE_URL="http://opsmanager.${OPSMANAGER_HOST}.nip.io:8080"
OPSMANAGER_API_USER=$(kubectl get secrets mongodb-enterprise-mongodb-
opsmanager-admin-key -n ${OPSMANAGER_NAMESPACE} -o
jsonpath='{.data.publicKey}' | base64 -d)
OPSMANAGER_API_KEY=$(kubectl get secrets mongodb-enterprise-mongodb-
opsmanager-admin-key -n ${OPSMANAGER_NAMESPACE} -o

```

```

jsonpath='{.data.privateKey}' | base64 -d)
OPSMANAGER_ORG_ID=$(curl -u ${OPSMANAGER_API_USER}:${OPSMANAGER_API_KEY} -
--digest -s --request GET "${OPSMANAGER_HOST}:8080/api/public/v1.0/orgs?
pretty=true" | jq -r '.results[].id')

echo "OPSMANAGER_API_KEY=${OPSMANAGER_API_KEY}"
echo "OPSMANAGER_API_USER=${OPSMANAGER_API_USER}"
echo "OPSMANAGER_ORG_ID=${OPSMANAGER_ORG_ID}"
echo "OPSMANAGER_BASE_URL=${OPSMANAGER_BASE_URL}"

OM_PROJECT_NAME="mongodb-oplog-replicaset"

## Create the Oplog Store ReplicaSet
kubectl -n mongodb-enterprise create secret generic organization-secret \
--from-literal="user=${OPSMANAGER_API_USER}" \
--from-literal="publicApiKey=${OPSMANAGER_API_KEY}" \
--dry-run=client -o yaml | kubectl apply -n mongodb-enterprise -f -

## Create the s3 creds
kubectl create secret generic s3-credentials \
--from-literal=accessKey="yeuc0wTRkc6vA6nch_ESwaEUdiBwmI3n" \
--from-literal=secretKey="_rJ8M_Uc92Aj9GIm_E21gJBS4cGceZFv" \
-n mongodb-enterprise

## Create OplogStore Project ConfigMap and Replicaset and wait until
complete
cat <<EOF | kubectl apply -n mongodb-enterprise -f -
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: $( echo $OM_PROJECT_NAME )-config
data:
  baseUrl: $( echo $OPSMANAGER_BASE_URL )
  projectName: $( echo $OM_PROJECT_NAME )-project
  orgId: $( echo $OPSMANAGER_ORG_ID )
---
apiVersion: mongodb.com/v1
kind: MongoDB
metadata:
  name: $( echo $OM_PROJECT_NAME )
spec:
  members: 3
  version: 4.2.2
  type: ReplicaSet
  opsManager:
    configMapRef:
      name: $( echo $OM_PROJECT_NAME )-config
  credentials: organization-secret
  persistent: true
  type: ReplicaSet
EOF

## Monitor Oplog Replicaset Creation until Running

```

```

kubectl get mongodb $OM_PROJECT_NAME -o yaml
watch -n 1 kubectl get pod -l pod-anti-affinity=$OM_PROJECT_NAME

## Reconfigure OpsManager with Oplog Store and S3 Backup Configs
OM_PROJECT_NAME="mongodb-oplog-replicaset"

cat <<EOF | kubectl apply -n mongodb-enterprise -f -
---
apiVersion: mongodb.com/v1
kind: MongoDBOpsManager
metadata:
  name: mongodb-opsmanager
  namespace: mongodb-enterprise
spec:
  adminCredentials: om-admin-secret
  applicationDatabase:
    members: 3
    version: 4.4.4-ent
  configuration:
    automation.versions.source: remote
    mms.adminEmailAddr: cloud-admin@no-reply.com
    mms.fromEmailAddr: cloud-support@no-reply.com
    mms.ignoreInitialUiSetup: "true"
    mms.mail.hostname: email-smtp.nutanix.demo
    mms.mail.port: "465"
    mms.mail.ssl: "false"
    mms.mail.transport: smtp
    mms.minimumTLSVersion: TLSv1.2
    mms.replyToEmailAddr: cloud-support@no-reply.com
  externalConnectivity:
    type: LoadBalancer
  replicas: 3
  version: 5.0.10
  backup:
    enabled: true
    opLogStores:
      - name: oplog1
        # the MongoDB resource that will act as an Oplog Store
        mongodbResourceRef:
          name: mongodb-oplog-replicaset
  s3Stores:
    - name: s3store1
      s3SecretRef:
        name: s3-credentials
      pathStyleAccessEnabled: true
      # change this to a s3 url you are using
      s3BucketEndpoint: ntnx-objects.ntnslab.local
      s3BucketName: mongodb
EOF

```

- **Demo:**
  - [Manual] Show MongoDB OpsManager UI to Connect to see Realtime Usage of Mongo Clusters
  - [Manual] Show Prism Central Dashboard Widgets UI
  - [Manual] Show Prism Central Analysis Chart UI
- CheatSheet:
- Analysis Dashboard: [https://portal.nutanix.com/page/documents/details?targetId=Prism-Central-Guide-Prism-v5\\_20:mul-performance-management-pc-c.html](https://portal.nutanix.com/page/documents/details?targetId=Prism-Central-Guide-Prism-v5_20:mul-performance-management-pc-c.html)

Deploy Resource to Use Prometheus - <https://www.mongodb.com/docs/kubernetes-operator/v1.16/tutorial/deploy-prometheus/#deploy-prometheus>

```
## Install the MongoDB ReplicaSet with Prometheus Service Monitor
```

```
cat <<EOF | kubectl apply -f
```

```
---
```

```
apiVersion: v1
kind: Secret
metadata:
  name: metrics-endpoint-creds
  namespace: mongodb
type: Opaque
stringData:
  password: 'Not-So-Secure!'
  username: prometheus-username
```

```
---
```

```
apiVersion: mongodb.com/v1
kind: MongoDB
metadata:
  name: my-replica-set
spec:
  members: 3
  version: 5.0.6-ent

  cloudManager:
    configMapRef:
      name: <project-configmap>

  credentials: <credentials-secret>
  type: ReplicaSet

  persistent: true

  prometheus:
    passwordSecretRef:
      # SecretRef to a Secret with a 'password' entry on it.
```

```

    name: metrics-endpoint-password

# change this value to your Prometheus username
username: prometheus-username

# Enables HTTPS on the prometheus scrapping endpoint
# This should be a reference to a Secret type kubernetes.io/tls
# tlsSecretKeyRef:
#   name: <prometheus-tls-cert-secret>

# Port for Prometheus, default is 9216
# port: 9216
#
# Metrics path for Prometheus, default is /metrics
# metricsPath: '/metrics'

---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  # This needs to match `spec.ServiceMonitorSelector.matchLabels` from
  your
  # `prometheuses.monitoring.coreos.com` resource.
  labels:
    release: prometheus

  name: mongodb-sm

  # Make sure this namespace is the same as in `spec.namespaceSelector`.
  namespace: mongodb
spec:
  endpoints:

    # Configuring a Prometheus Endpoint with basic Auth.
    # `prom-secret` is a Secret containing a `username` and `password`
    entries.
    - basicAuth:
        password:
          key: password
          name: metrics-endpoint-creds
        username:
          key: username
          name: metrics-endpoint-creds

    # This port matches what we created in our MongoDB Service.
    port: prometheus

    # If using HTTPS enabled endpoint, change scheme to https
    scheme: http

    # Configure different TLS related settings. For more information, see:
    # https://github.com/prometheus-operator/prometheus-
    operator/blob/main/pkg/apis/monitoring/v1/types.go#L909

```

```

# tlsConfig:
#   insecureSkipVerify: true

# What namespace to watch
namespaceSelector:
  matchNames:
    # Change this to the namespace the MongoDB resource was deployed.
    - mongodb

# Service labels to match
selector:
  matchLabels:
    app: my-replica-set-svc

---
apiVersion: v1
kind: Secret
metadata:
  name: metrics-endpoint-creds
  namespace: mongodb
type: Opaque
stringData:
  password: 'Not-So-Secure!'
  username: prometheus-username
EOF
...

```

## Requirement: Create Incidents

- **Demo:**
  - [Manual] Show Prism Central Alert Policies - High CPU/Memory Usage based on VM Categories (i.e., AppFamily:KubernetesDistro)
  - [Manual] Show Prism Central Playbooks - Alert Policy Event triggering E-mail (i.e., "Create Incident on CPU or Memory Constrained Karbon Worker Nodes -AppFamily:KubernetesDistro" Playbook)
- CheatSheet:

### Importing Prism Central Playbooks

- Configure PC Alert Policies: [https://portal.nutanix.com/page/documents/details?targetId=Prism-Central-Guide-Prism-v5\\_20:mul-alert-policies-configure-pc-t.html](https://portal.nutanix.com/page/documents/details?targetId=Prism-Central-Guide-Prism-v5_20:mul-alert-policies-configure-pc-t.html)
- Auto Categorize New VMs (Ideal for Setting Up Nutanix Policies (e.g., Data Protection, Network Security, Envents, Alerts, etc.) on demand): <https://www.nutanix.dev/playbooks/auto-categorize-new-vms/>
- Send Incident Alerts to Email, Slack or MSTEams: <https://www.nutanix.dev/playbooks/send-alert-details-to-slack-email-msteams/>
- Send Incident Alerts to PagerDuty: <https://www.nutanix.dev/playbooks/send-alerts-to-pagerduty/>

- ServiceNow Integration with Prism Central (X-Play Support):  
[https://portal.nutanix.com/page/documents/details?targetId=Prism-Central-Guide-Prism-v5\\_20:mul-service-now-integration-pc.html](https://portal.nutanix.com/page/documents/details?targetId=Prism-Central-Guide-Prism-v5_20:mul-service-now-integration-pc.html)

Requirement: Messaging to users to communicate submitted / completed requests

- Demo:
  - [Manual] Show Playbook with New VM Create Event Notification
- CheatSheet:

This playbook sends a Slack message when a VM is created, with the user and VM details. It uses the Lookup VM Details, REST API, and Slack actions. It could easily be modified to support Microsoft Teams and/or e-mails as well -

[https://github.com/nutanixdev/playbooks/tree/master/vm\\_created\\_alert](https://github.com/nutanixdev/playbooks/tree/master/vm_created_alert)

Requirement: Tracking against containers for users and teams

- Demo:
  - [Manual] Show Mongo Team/User usage for Mongo
  - [Manual] Show Scenarios with Rancher, Kubecost, Kubernetes Dashboard

## Troubleshooting

Monitor Deployment Progress via Kubectl

```
## setup monitoring for target instance, update MONGO_INSTANCE var
MONGO_INSTANCE=mongodb-demo-replicaset
MONGO_SVC_NAME=$(kubectl get svc -o name -n $MONGO_INSTANCE | grep -v
external | cut -d/ -f2 | egrep "service|svc")
watch -n 1 "kubectl get po,pvc -l app=${MONGO_SVC_NAME} -n $MONGO_INSTANCE
-o wide && echo && kubectl get mongodb ${MONGO_INSTANCE} -n
${MONGO_INSTANCE}"
```

Connecting to mongodb shell via kubectl

```
## find mongo instance user secret and get standard connection info,
update MONGO_INSTANCE var
MONGO_INSTANCE=mongodb-demo-replicaset
MONGO_SECRET_NAME=$(kubectl get secret -o name -n $MONGO_INSTANCE | grep -
i $MONGO_INSTANCE | cut -d/ -f2)
MONGO_USER_NAME=$(kubectl get secret $MONGO_SECRET_NAME -n $MONGO_INSTANCE
-o jsonpath='{.data.username}' | base64 -d)
MONGO_USER_PASS=$(kubectl get secret $MONGO_SECRET_NAME -n $MONGO_INSTANCE
-o jsonpath='{.data.password}' | base64 -d)
MONGO_CONNECTION_STD=$(kubectl get secrets $MONGO_SECRET_NAME -n
${MONGO_INSTANCE} -o jsonpath='{.data.connectionString.standard}' | base64
-d)
```

```
echo $MONGO_CONNECTION_STD
```

```
## enter mongo shell interactively using standard connection string
kubectl run -i -t --rm --image=mongo:5.0 mongosh-$RANDOM -- mongosh
"$MONGO_CONNECTION_STD"
```

## Executing some basic MongoDB queries

```
## insert quick snippets of data to validate
db.ships.insert({name: 'USS Enterprise-
D', operator: 'Starfleet', type: 'Explorer', class: 'Galaxy', crew: 750, codes:
[10, 11, 12]})
db.ships.insert({name: 'USS
Prometheus', operator: 'Starfleet', class: 'Prometheus', crew: 4, codes:
[1, 14, 17]})
db.ships.insert({name: 'USS
Defiant', operator: 'Starfleet', class: 'Defiant', crew: 50, codes: [10, 17, 19]})
db.ships.insert({name: 'IKS Buruk', operator: 'Klingon
Empire', class: 'Warship', crew: 40, codes: [100, 110, 120]})
db.ships.insert({name: 'IKS Somraw', operator: 'Klingon
Empire', class: 'Raptor', crew: 50, codes: [101, 111, 120]})
db.ships.insert({name: 'Scimitar', operator: 'Romulan Star
Empire', type: 'Warbird', class: 'Warbird', crew: 25, codes: [201, 211, 220]})
db.ships.insert({name: 'Narada', operator: 'Romulan Star
Empire', type: 'Warbird', class: 'Warbird', crew: 65, codes: [251, 251, 220]})

## query data
db.ships.findOne()
db.ships.find().pretty()
db.ships.find({}, {name: true, _id: false})
```

## Connecting to database instance options

If Database are deployed within K8s cluster (typically as Statefulset), but DON'T need external connectivity access - the typical scenarios would be:

- Internal Clients (e.g., frontend apps / database clients) would connect to Internal ClusterIP Service for scenarios where it's ok to be Load Balanced between different DB replica instances
  - Internal DNS Lookup for Service will return ClusterIP Address that frontends all the database pod IPs. The path taken would leverage kube-proxy/iptables pathway.
- Alternatively, Internal Clients could leverage **Headless** Services to connect to specific Stateful Pod / DB Replica Instance (i.e., secondary readonly instance) -

<https://kubernetes.io/docs/concepts/services-networking/service/#headless-services>

- With **Headless** Service, the ClusterIP attribute would be set to **None**, therefore no **ClusterIP** is allocated, **kube-proxy** won't be leveraged to handle services and there is no load balancing or proxying would be done by Kubernetes Platform. A DNS Service Lookup will simply return the Pod IP Address.



- In both cases, DNS is configured automatically based on whether or not the **Service** has **Selectors** defined
  - When **Selector** is defined, **Endpoints** are configured with Respective A Host Records in DNS (i.e., coredns) for each service / statefulset pod.

Alternatively, if database are external/outside the kubernetes clusters, then you'd want to leverage Service of type **ExternalName** and DNS CNAME lookups

- Without **Selector**, A **CNAME** records could be leveraged with **ExternalName** type as means of connecting to external service from within Kubernetes
  - <https://kubernetes.io/docs/concepts/services-networking/service/#externalname>
  - <https://www.googblogs.com/kubernetes-best-practices-mapping-external-services/>

Lastly, If Database are deployed within K8s cluster but need external connectivity access - the typical scenarios would be:

- Leverage Port Forwarding for temporary access
  - <https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/>
- OR Leverage External DNS Operator and Headless Service (with type=NodePort) to Create DNS Records for External IP of Host(s) (vs PodIPs)
  - <https://github.com/kubernetes-sigs/external-dns>
  - <https://github.com/kubernetes-sigs/external-dns/blob/master/docs/tutorials/hostport.md>

Additional References if needed:

- <https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider>
- <https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/>
- <https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/>

## Production Best Practice Notes

- Single Instance of Ops Manager for all MongoDBs
- One Operator PER Kubernetes Namespace
- One Kubernetes Namespace per OpsManager Organization
- One ConfigMap per MongoDB Instance
- Map Internal to External DNS names with TLS [OPT]
- Enable TLS with Cert-Manager [OPT]
- Enable Authentication using MongoDBUsers CRD and K8s Secrets (or Vault Alternative) [OPT]
- Enable LDAP AuthN/Z [OPT]
- Set Resource Constraints for all
- Configure NodeAffinity if there are specialized workload / placement constraints
- Configure Multiple Mount Points. Mount Point == PVC. Each PVC can be expanded on Demand
- Setup NodeAffinity and PodAffinity Accordingly based on Node Selector Labels
- Replicated block storage across multiple nodes and data centers to increase availability
- Secondary data backup storage (for example, NFS or S3)
- Cross-cluster disaster recovery volumes
- Recurring volume snapshots

- Recurring backups to secondary storage
- Non-disruptive upgrades

## References

- MongoDB on Nutanix - <https://portal.nutanix.com/page/documents/solutions/details?targetId=BP-2023-MongoDB-on-Nutanix:BP-2023-MongoDB-on-Nutanix>
- Nutanix Playbooks Library - <https://www.nutanix.dev/playbooks/>
- OpsManager Architecture in K8s - <https://www.mongodb.com/docs/kubernetes-operator/v1.16/tutorial/om-arch/>
- MongoDB Architecture in K8s - <https://www.mongodb.com/docs/kubernetes-operator/stable/tutorial/mdb-resources-arch/>
- Running MongoDB OpsManager in Kubernetes - <https://www.mongodb.com/blog/post/running-mongodb-ops-manager-in-kubernetes>
- Deploy a MongoDB ReplicaSet - <https://www.mongodb.com/docs/kubernetes-operator/v1.16/tutorial/deploy-replica-set/>
- Manage MongoDB Database Users - <https://www.mongodb.com/docs/kubernetes-operator/v1.16/tutorial/manage-database-users-scam/>
- Deploy a MongoDB Sharded Cluster - <https://www.mongodb.com/docs/kubernetes-operator/v1.16/tutorial/deploy-sharded-cluster/>
- Observability of the MongoDB Kubernetes Operator in Production <https://www.youtube.com/watch?v=JqpQPrJSgS8>
- Deploy a Resource to Use with Prometheus - <https://www.mongodb.com/docs/kubernetes-operator/v1.16/tutorial/deploy-prometheus/#deploy-prometheus>
- Mapping to External Services - <https://cloud.google.com/blog/products/gcp/kubernetes-best-practices-mapping-external-services>
- To Run or Not Run a Database on Kubernetes: What to Consider - <https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider>