SOLUTION DOCUMENT

MayaData

MinIO object storage using OpenEBS LocalPV

MINIO

CONTENTS

BEFORE STARTING

MinIO is a high-performance object storage server designed for large-scale private cloud infrastructure. It provides a similar experience to cloud provided object stores like AWS S3, Azure Blob Storage, and GCP Cloud Storage. MinIO is designed in a cloud-native manner to scale sustainably in multi-tenant environments. Orchestration platforms like Kubernetes provide a perfect cloud-native environment to deploy and scale MinIO. MinIO can be provisioned with OpenEBS volumes using various OpenEBS storage engines such as Local PV, cStor, or Jiva based on the application requirement. In this document, we are installing a MinIO operator using Kubera on OpenEBS Local PV.

The MinIO operator offers a seamless way to create and update highly available distributed MinIO clusters. MinIO Operator brings native support for MinIO, Graphical Console for Admin and Users, and encryption to Kubernetes. It also offers MinIO tenant creation, management, upgrade, zone addition, and more.

INTRODUCTION

Kubera from MayaData is a SaaS and on-premise solution for the use of Kubernetes as a data layer. Capabilities include alerting, visualization, reporting, chaos engineering, per workload back-ups, rolling upgrades, compliance checks, troubleshooting, provisioning and management of underlying storage media and cloud volumes, and more. Kubera Propel is a module of Kubera focused on the use of Kubernetes for high-performance workloads.

The OpenEBS project (which Kubera Propel uses as a foundation) and MayaData popularized the Container Attached Storage pattern. Much more about this pattern is shared in various blogs on the Cloud Native Computing Foundation site, such as Container Attached Storage is Cloud Native Storage (CAS)

Today, OpenEBS and Kubera are used by well-known enterprises as well as thousands of community users world-wide. Popular use cases include:

- Containerized Data Management
- CI/CD
- ML & Data Ops
- Database as Service
- Content Management

OpenEBS is amongst the most broadly used cloud native - or Container Attached Storage - projects per various CNCF analysis and community surveys. OpenEBS is the most widely deployed open source example of a category of storage solutions sometimes called Container Attached Storage. OpenEBS was created by MayaData as an open source project and was donated to CNCF in early 2019. The open source OpenEBS Adopters list includes Arista, CNCF, Comcast, KPN, Orange, and others.

In this solution guide, we present a total solution based upon Kubera as well as Kubernetes v1.17.0.

System configuration

We will use EKS, where we will install MinIO object storage using OpenEBS. This guide will help you to install MinIO object storage in distributed mode using the MinIO Operator. In this case, a Local PV volume will be provisioned on one of the matching unclaimed block devices based on the storage request. This will use the entire blockdevice for storing the data for the particular application. The blockdevice can be mounted or be a raw device on the node where the application is scheduled. Another application cannot use this block device. Suppose users have limited block devices attached to some nodes. In that case, they can use nodeSelector in the application YAML to provision applications on particular nodes where the available blockdevice is present. Since our example setup is in distributed mode, it requires a minimum of four nodes, and at least a single unclaimed external disk should be attached to each of these nodes. Let's review our setup used for the configuration.

Our setup:

- 4 nodes in EKS
- 4 vCPUs / node
- 16 GB memory / node
- 1 SSD (100Gi) / node
- AWS instance type: t3.x.large
- Kubernetes version: v1.17

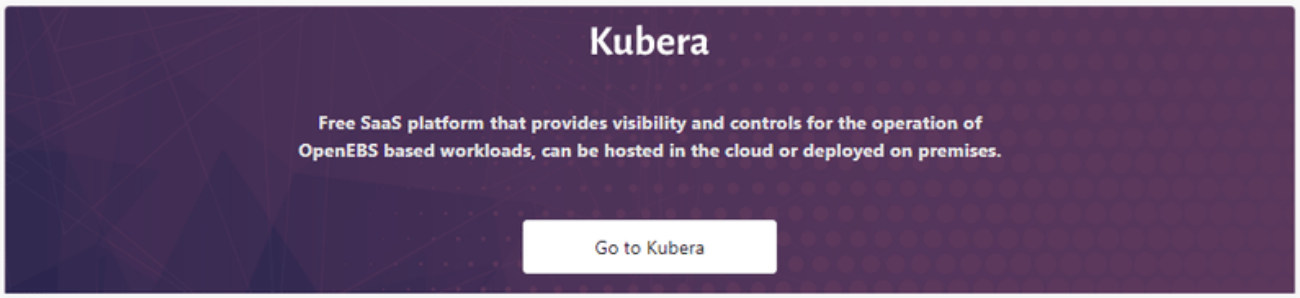
We will create a 100Gi volume for each node in each region where nodes are created.

GETTING STARTED

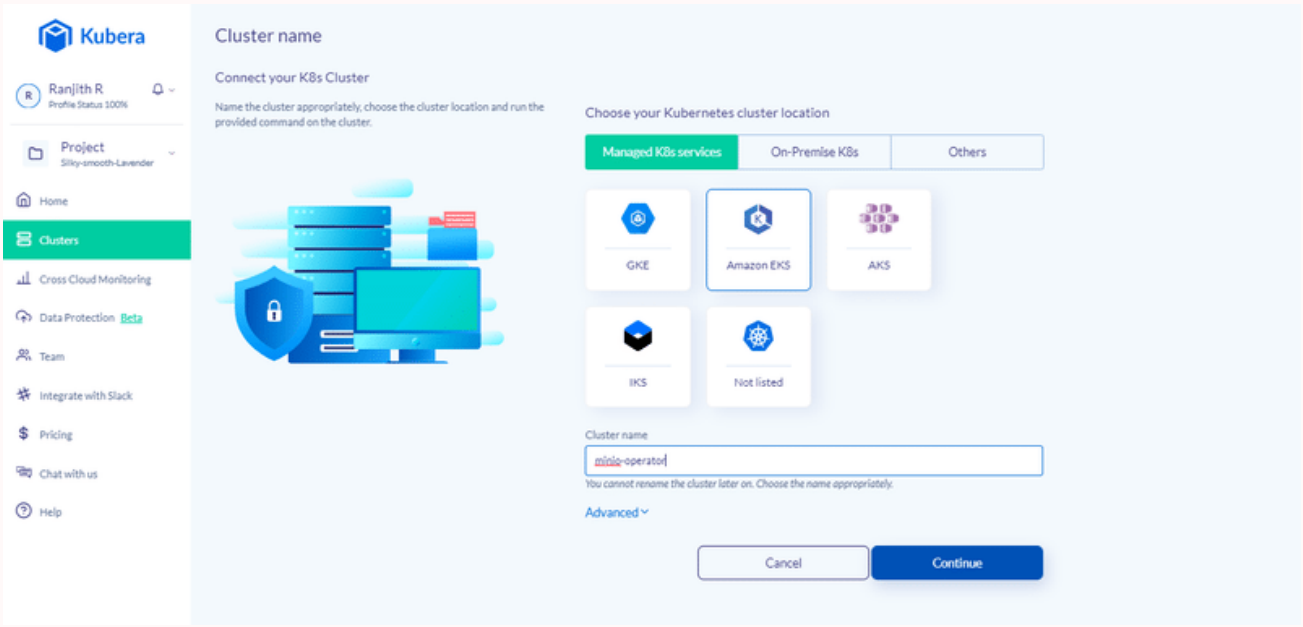
Let's start the installation of OpenEBS using the Kubera platform.

Installing OpenEBS using Kubera:

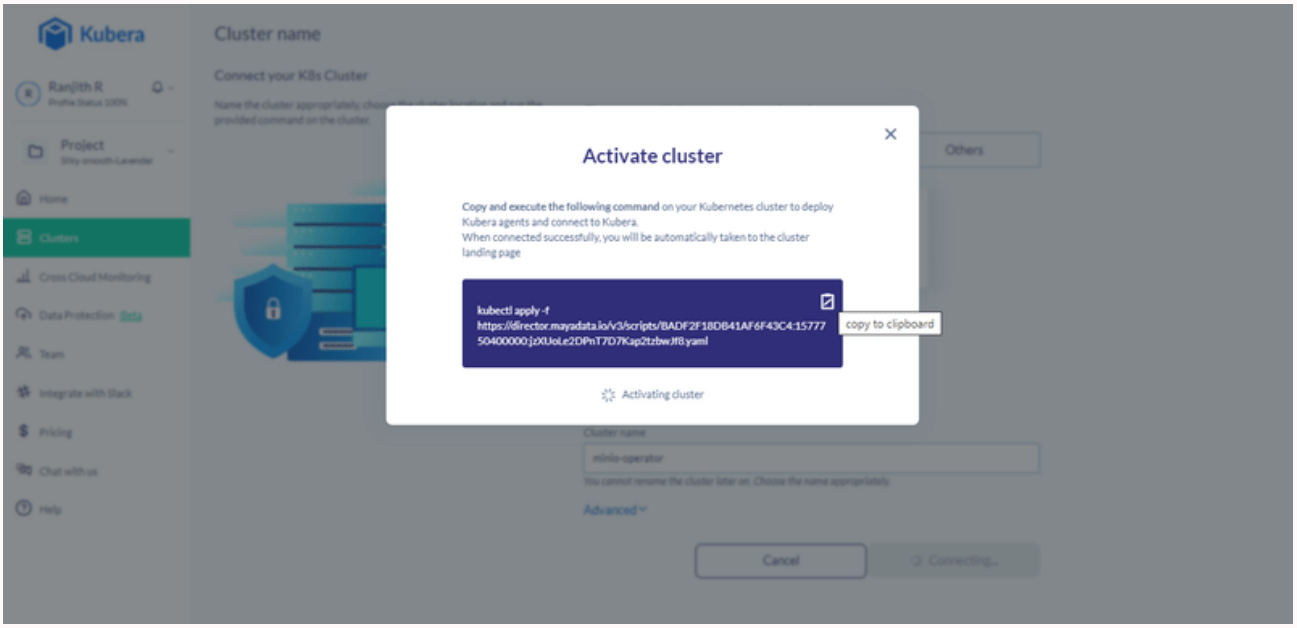
Login to your free Kubera account and click on Go to Kubera



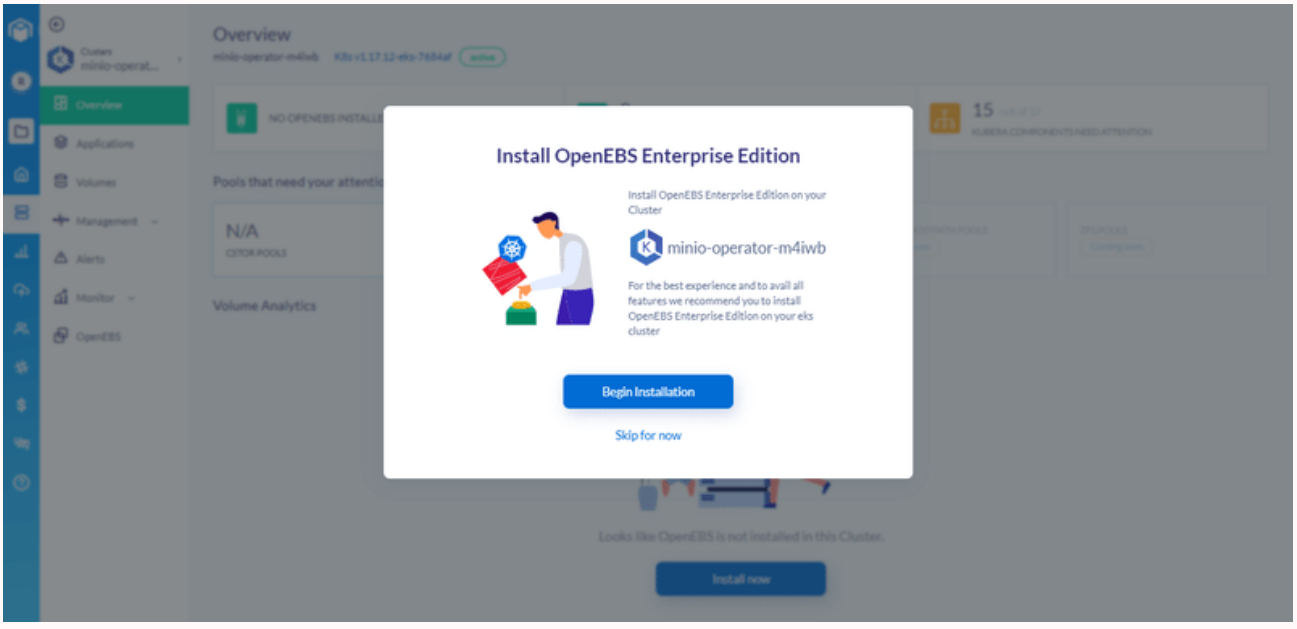
Follow the instructions to connect your cluster to your Kubera account.



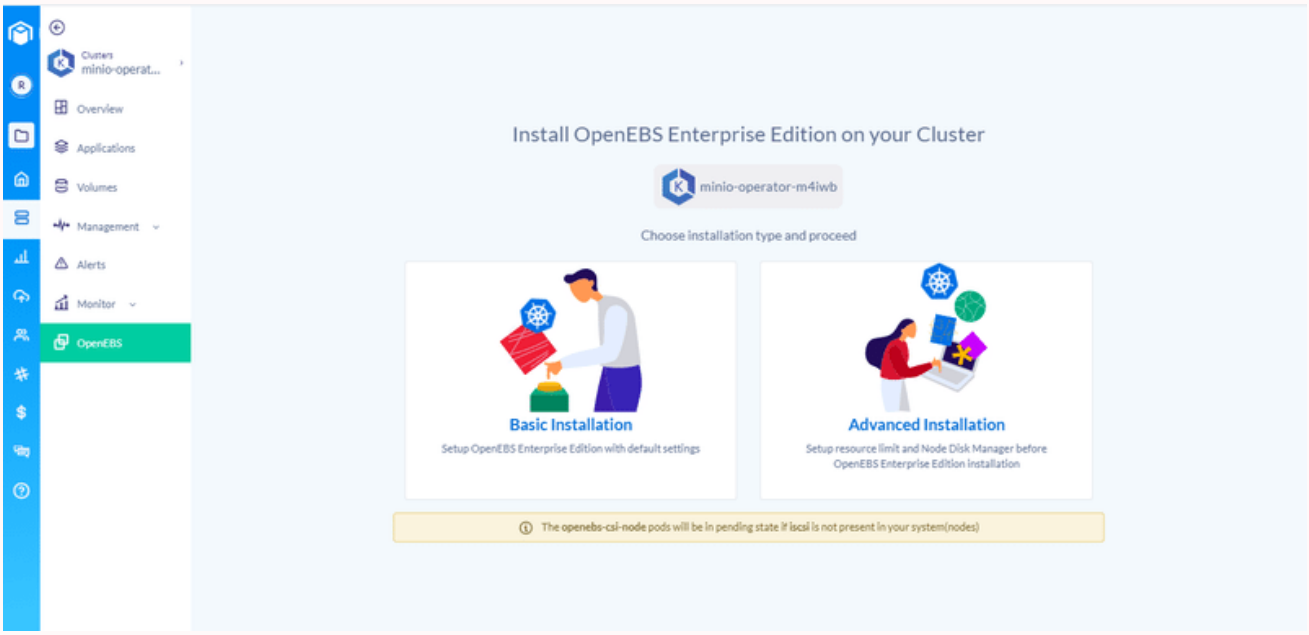
It will pop up a window with a command to connect your K8s cluster with the Kubera SaaS version. Copy and execute the command on your Kubernetes cluster.



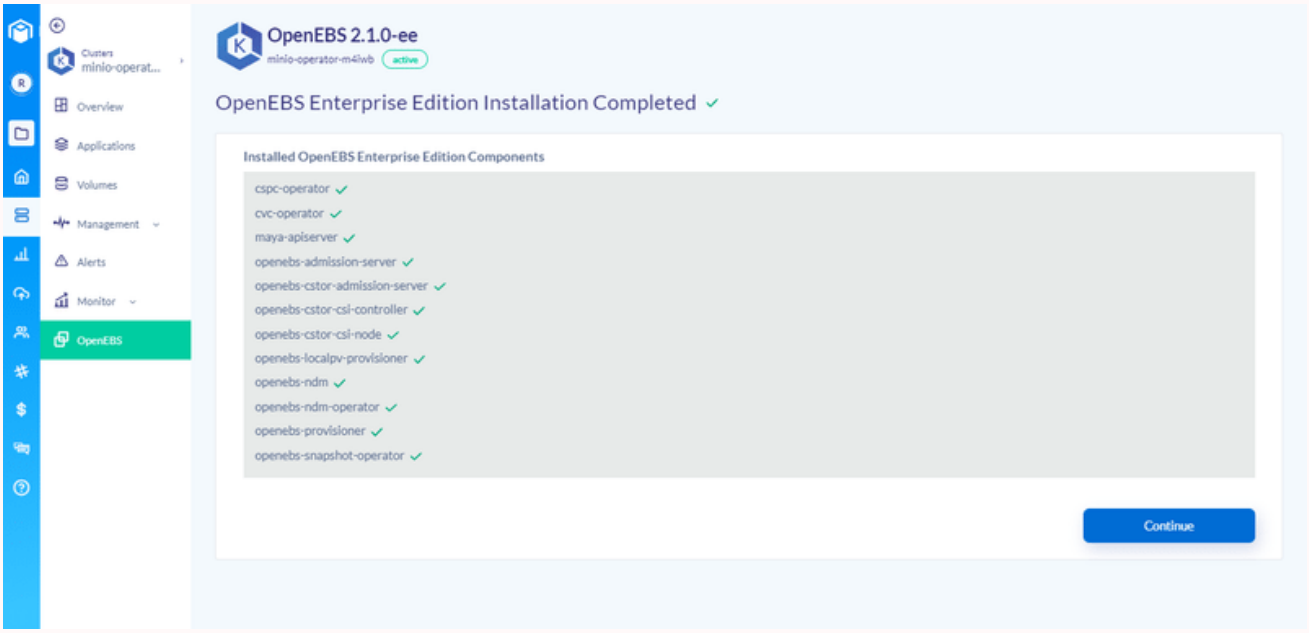
If OpenEBS was already installed using Kubera in your cluster, skip this process. If OpenEBS was not installed using Kubera, then click on **Begin Installation**, which will lead to a page where you can choose a way to install OpenEBS.



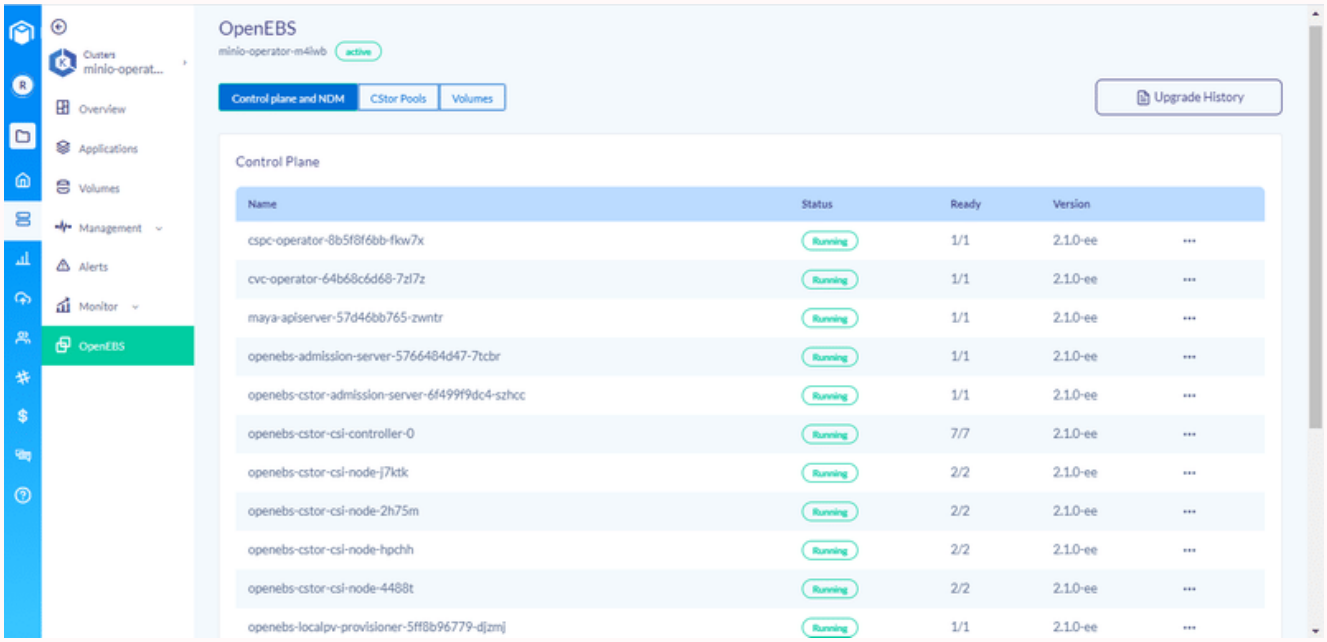
Follow the on-screen instructions titled **Basic Installation** for the default installation of OpenEBS Enterprise Edition on your K8s cluster.



Click on **Deploy OpenEBS** on the next screen and verify the installation status from the next screen. Please continue if everything is installed properly. If you run into errors or have questions, [community support](#) for OpenEBS is available on Slack.



OpenEBS installation will complete shortly, and you will see OpenEBS control-plane enabled on your cluster.



Attaching disks to nodes

Now, we will add an additional device to each node. Disks will be later consumed by MinIO instances using the openebs-device StorageClass, and these will be used as persistent storage for MinIO. This step can be done through your cloud vendor's web user interface, or if you are running in a VM, you can use your hypervisor to add an additional virtual device to each node. In this example, we have used AWS and added the disks using the AWS CLI tool.

Create a 100Gi volume for each Node.

```
$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-
south-1 --availability-zone ap-south-1a

$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-
south-1 --availability-zone ap-south-1b

$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-
south-1 --availability-zone ap-south-1c

$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-
south-1 --availability-zone ap-south-1c
```

Note: Provide the required size initially as Local PV storage will not allow you to expand the capacity later.

Get list of InstanceIds per each Zone:

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].
{Instances:InstanceId,AZ:Placement.AvailabilityZone}' --output table

-----------------------------------------
|           DescribeInstances           |
+--------------+------------------------+
|      AZ      |       Instances        |
+--------------+------------------------+
|  ap-south-1c |  i-0ea6279495a4d9e99   |
|  ap-south-1c |  i-00dfec00a401934b3   |
|  ap-south-1a |  i-027bf7d996376da58   |
|  ap-south-1b |  i-0afb2ff87ff1a46b4   |
+--------------+------------------------+
```

Get the list of VolumeIds per each Zone:

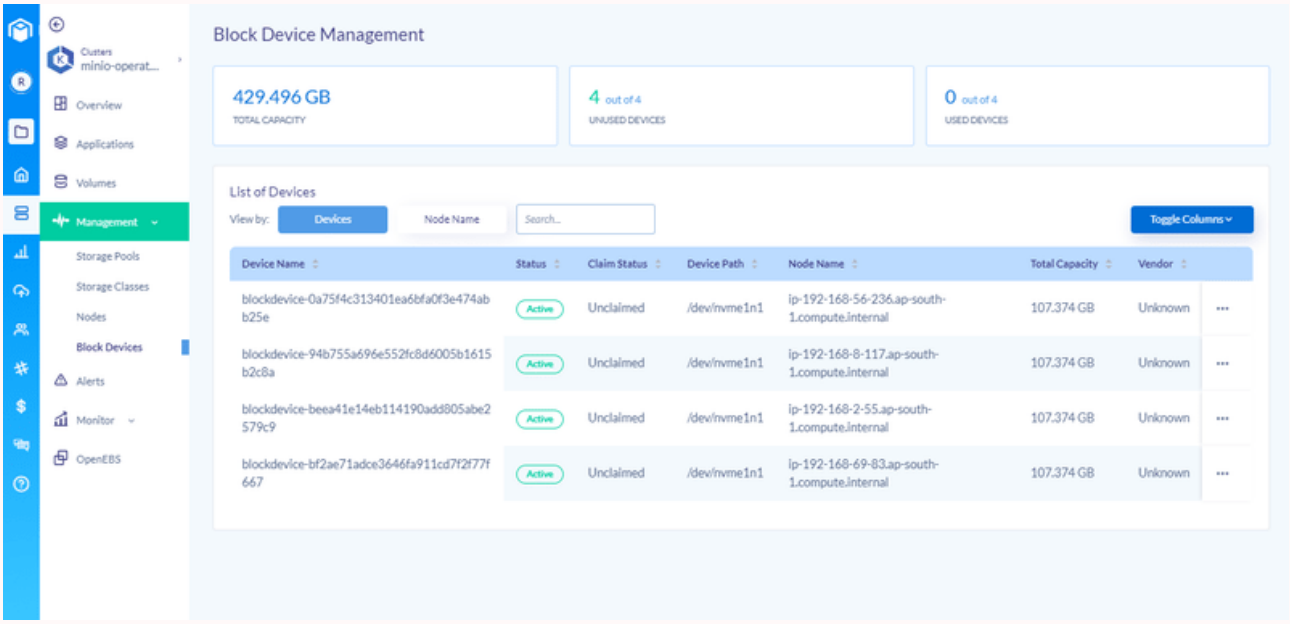
```
$ aws ec2 describe-volumes --filters Name=status,Values=available --
query "Volumes[*].{VolId:VolumeId,AZ:AvailabilityZone,Size:Size}" --
output table

-------------------------------------------------
|                DescribeVolumes                |
+--------------+-------+-----------------------+
|      AZ      | Size  |          VolId        |
+--------------+-------+-----------------------+
|  ap-south-1b |  100  |  vol-0edaef30570190551 |
|  ap-south-1c |  100  |  vol-0d6784f0fe10f03b7 |
|  ap-south-1c |  100  |  vol-04bdf2f839eaf1651 |
|  ap-south-1a |  100  |  vol-0be1f94ae9dd5d0ac |
+--------------+-------+-----------------------+
```

Repeat the following command for each device created and attach a device to each node.

```
# Disk 1 to worker node 1
$ aws ec2 attach-volume --volume-id vol-0be1f94ae9dd5d0ac --instance-id
i-027bf7d996376da58 --device /dev/sdf

# Disk 2 to worker node 2
$ aws ec2 attach-volume --volume-id vol-0edaef30570190551 --instance-id
i-0afb2ff87ff1a46b4 --device /dev/sdf

# Disk 3 to worker node 3
$ aws ec2 attach-volume --volume-id vol-0d6784f0fe10f03b7 --instance-id
i-0ea6279495a4d9e99 --device /dev/sdf

# Disk 4 to worker node 3
$ aws ec2 attach-volume --volume-id vol-04bdf2f839eaf1651 --instance-id
i-00dfec00a401934b3 --device /dev/sdg
```

Verify the BlockDevice information

You can verify the attached Block Device information from the Kubera portal under **Management > Block Devices** from the corresponding cluster page.



Verify default Storage Class

```
$ kubectl get sc
NAME                        PROVISIONER
RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gp2 (default)               kubernetes.io/aws-ebs
Delete          WaitForFirstConsumer   false                  41m
openebs-device              openebs.io/local
Delete          WaitForFirstConsumer   false                  18m
openebs-hostpath            openebs.io/local
Delete          WaitForFirstConsumer   false                  18m
openebs-jiva-default        openebs.io/provisioner-iscsi
Delete          Immediate              false                  18m
openebs-snapshot-promoter   volumesnapshot.external-
storage.k8s.io/snapshot-promoter   Delete          Immediate
false                  18m
```

From the default StorageClass, we use openebs-device for using Persistent storage for running MinIO.

INSTALL MINIO

In this section, we will install the MinIO in distributed mode using MinIO Operator on OpenEBS Local PV. The MinIO instance will be deployed in the **default** namespace.

Prerequisites

- Kubernetes >= v1.17.0.
- Install kubectl minio plugin using krew. Installation of krew can be done from here.
- Ensure that git is installed.
- Helm v3 (As we are using Helm V3)

Workflow

1. Install the MinIO plugin
2. Install the MinIO operator deployment
3. Install the MinIO cluster.
4. Access MinIO console

Install the MinIO operator plugin

The MinIO operator offers MinIO Tenant (MinIO cluster) creation, management of cluster, upgrade, zone addition, and more. Install the MinIO operator plugin using the following command.

```
$ kubectl krew install minio
```

Install the MinIO operator deployment

Let's get started by initializing the MinIO Operator deployment. This is a one time process.

```
$  kubectl minio init
CustomResourceDefinition tenants.minio.min.io: created
ClusterRole minio-operator-role: created
ServiceAccount minio-operator: created
ClusterRoleBinding minio-operator-binding: created
MinIO Operator Deployment minio-operator: created
```

Verify the MinIO operator is successfully installed.

```
$ kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
minio-operator-59b8965ff5-tzx8n   1/1     Running   0          18s
```

Install the MinIO Tenant/Cluster

A tenant is a MinIO cluster created and managed by the operator. Before creating a tenant, please ensure you have requisite nodes and drives in place.

In this guide, we are using 4 Nodes with one 100Gi block device attached per each node. Using the MinIO operator, the following command will generate a YAML file as per the given requirement, and the file can be modified as per your specific requirements.

```
$ kubectl minio tenant create --name tenant1 --servers 4 --volumes 4 --
capacity 400Gi -o > tenant.yaml
```

The above will create a YAML spec with 4 MinIO nodes with 100Gi volume. In this YAML file, we need to add the openebs-device storage class to create the 100Gi persistent volume using the device attached to each node.

Note: Ensure that the image version used for the MinIO console is 0.4.6 or higher. Otherwise, pods will be in crashloopbackoff state.

Add the following two changes to the tenant file created using the above command.

- Add the following to **spec.zones.volumeClaimTemplate.spec** under **Tenant** kind.

  storageClassName: openebs-device
  An example snippet of the modified tenant YAML file.

```
serviceName: tenant1-internal-service
  zones:
  - resources: {}
    servers: 4
    volumeClaimTemplate:
      apiVersion: v1
      kind: persistentvolumeclaims
      metadata:
        creationTimestamp: null
      spec:
        accessModes:
        - ReadWriteOnce
        storageClassName: openebs-device
        resources:
          requests:
            storage: 100Gi
      status: {}
    volumesPerServer: 1
```

- Also, set requestAutoCert: false so that MinIO will run in http mode. In this document, we have used http communication for accessing MinIO. The following is a sample snippet of the modified section.

```
mountPath: /export
 requestAutoCert: false
 serviceName: tenant1-internal-service
```

Apply the modified tenant YAML spec. The following command will install MinIO tenants under the default namespace.

```
$ kubectl apply -f tenant.yaml
tenant.minio.min.io/tenant1 created
secret/tenant1-creds-secret created
secret/tenant1-console-secret created
```

Verify the MinIO cluster creation is successfully running under the default namespace

```
$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
minio-operator-59b8965ff5-tzx8n     1/1     Running   0          6m46s
tenant1-console-6589f7574d-6kgnp    1/1     Running   0          19s
tenant1-console-6589f7574d-wt47v    1/1     Running   0          19s
tenant1-zone-0-0                    1/1     Running   0          51s
tenant1-zone-0-1                    1/1     Running   0          51s
tenant1-zone-0-2                    1/1     Running   0          51s
tenant1-zone-0-3                    1/1     Running   0          50s
```

Verify the MinIO persistent volume details.

```
$ kubectl get pvc
NAME                 STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS     AGE
0-tenant1-zone-0-0   Bound    pvc-eff2ebdc-1658-4525-b7e2-5d57b39f144b
100Gi      RWO            openebs-device   53s
0-tenant1-zone-0-1   Bound    pvc-1a5881ae-c65a-4ebe-9233-615c6fb7f364
100Gi      RWO            openebs-device   53s
0-tenant1-zone-0-2   Bound    pvc-bd8d3521-fea9-4a48-8f66-26c6d2808997
100Gi      RWO            openebs-device   53s
0-tenant1-zone-0-3   Bound    pvc-55d6aa94-37ed-4f14-bafb-dcee1d7af9f5
100Gi      RWO            openebs-device   52s

$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
 RECLAIM POLICY   STATUS   CLAIM                          STORAGECLASS
REASON   AGE
pvc-1a5881ae-c65a-4ebe-9233-615c6fb7f364   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-1   openebs-device
49s
pvc-55d6aa94-37ed-4f14-bafb-dcee1d7af9f5   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-3   openebs-device
49s
pvc-bd8d3521-fea9-4a48-8f66-26c6d2808997   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-2   openebs-device
53s
pvc-eff2ebdc-1658-4525-b7e2-5d57b39f144b   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-0   openebs-device
54s
```

Verify the MinIO cluster creation is successfully running under the default namespace.

```
$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
minio-operator-59b8965ff5-tzx8n     1/1     Running   0          6m46s
tenant1-console-6589f7574d-6kgnp    1/1     Running   0          19s
tenant1-console-6589f7574d-wt47v    1/1     Running   0          19s
tenant1-zone-0-0                    1/1     Running   0          51s
tenant1-zone-0-1                    1/1     Running   0          51s
tenant1-zone-0-2                    1/1     Running   0          51s
tenant1-zone-0-3                    1/1     Running   0          50s
```

Verify the MinIO persistent volume details.

```
$ kubectl get pvc
NAME                 STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS     AGE
0-tenant1-zone-0-0   Bound    pvc-eff2ebdc-1658-4525-b7e2-5d57b39f144b
100Gi      RWO            openebs-device   53s
0-tenant1-zone-0-1   Bound    pvc-1a5881ae-c65a-4ebe-9233-615c6fb7f364
100Gi      RWO            openebs-device   53s
0-tenant1-zone-0-2   Bound    pvc-bd8d3521-fea9-4a48-8f66-26c6d2808997
100Gi      RWO            openebs-device   53s
0-tenant1-zone-0-3   Bound    pvc-55d6aa94-37ed-4f14-bafb-dcee1d7af9f5
100Gi      RWO            openebs-device   52s

$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
 RECLAIM POLICY   STATUS   CLAIM                          STORAGECLASS
REASON   AGE
pvc-1a5881ae-c65a-4ebe-9233-615c6fb7f364   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-1   openebs-device
49s
pvc-55d6aa94-37ed-4f14-bafb-dcee1d7af9f5   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-3   openebs-device
49s
pvc-bd8d3521-fea9-4a48-8f66-26c6d2808997   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-2   openebs-device
53s
pvc-eff2ebdc-1658-4525-b7e2-5d57b39f144b   100Gi      RWO
 Delete           Bound    default/0-tenant1-zone-0-0   openebs-device
54s
```

Verify MinIO service status.

```
$ kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
kubernetes        ClusterIP   10.100.0.1      <none>        443/TCP
62m
minio             ClusterIP   10.100.59.34    <none>        80/TCP
57s
tenant1-console   ClusterIP   10.100.50.135   <none>
 9090/TCP,9443/TCP   25s
tenant1-hl        ClusterIP   None            <none>        9000/TCP
57s
```

Note: If the user needs to access MinIO outside the network, the service type can be changed or a new service should be added to use LoadBalancer or create Ingress resources for production deployment.
For ease of simplicity in testing the deployment, we are going to use NodePort. Please be advised to consider using LoadBalancer or Ingress, instead of NodePort, for production deployment.

Minio service will allow the user to access the console, and **tenant1-console** will allow access to the Admin console. In this guide, we have changed the service type of the services mentioned above, and the following is the output after the modification.

Now, MinIO has been installed successfully on your cluster.

```
$ kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
kubernetes        ClusterIP   10.100.0.1      <none>        443/TCP
64m
minio             NodePort    10.100.59.34    <none>        80:32095/TCP
3m10s
tenant1-console   NodePort    10.100.50.135   <none>
 9090:30383/TCP,9443:30194/TCP   2m38s
tenant1-hl        ClusterIP   None            <none>        9000/TCP
3m10s
```

Access MinIO Admin console

An Admin can access MinIO and do the configuration changes such as creating an account, group, bucket, and its configuration, the setting of user-level permission, file-level permission, etc.

For Admin access, use <Node_External_Ip>:<NodePort_of_tenant1-console_service> in your web browser.

Get the details of Node.

```
$ kubectl get node -o wide

NAME                                              STATUS   ROLES    AGE
 VERSION    INTERNAL-IP      EXTERNAL-IP      OS-IMAGE
 KERNEL-VERSION   CONTAINER-RUNTIME
ip-192-168-2-55.ap-south-1.compute.internal     Ready    <none>   51m
 v1.17.11   192.168.2.55     65.0.121.83      Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
ip-192-168-56-236.ap-south-1.compute.internal   Ready    <none>   57m
 v1.17.11   192.168.56.236   15.206.189.106   Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
ip-192-168-69-83.ap-south-1.compute.internal    Ready    <none>   57m
 v1.17.11   192.168.69.83    3.6.91.169       Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
ip-192-168-8-117.ap-south-1.compute.internal    Ready    <none>   57m
 v1.17.11   192.168.8.117    13.235.210.41    Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
```

Now, access the MinIO service over the browser using the following way.

```
http://3.6.91.169:30383
```

Note: Ensure Inbound Rules under VPC-> Security Groups are correctly configured to allow the traffic.

You should enter the Access key and Secret key to login into the admin console. These credentials can be obtained from the secret.

```
$ kubectl get secret tenant1-console-secret -oyaml
```

The following is a sample snippet of the output of the above command. It will show the Access key and Secret key in encoded form. The decoded value should be given in the web browser to login to the user console.

```
apiVersion: v1
data:
  CONSOLE_ACCESS_KEY: MmRkYzA2NGItYTMwZS00ZDg5LTgwODItNWMwYzFkYTRlOGNh
  CONSOLE_HMAC_JWT_SECRET:
ODkwYWFlYmEtMTAxYy00YTJmLTg0NDMtYmI1ZjAyMjcyNWFk
  CONSOLE_PBKDF_PASSPHRASE:
MDZhN2UzMmUtOWIxZi00MjI2LTk2MmItOTk4OTRmMGYwYjk2
  CONSOLE_PBKDF_SALT: OTg0OTM1YjAtNzgyMS00NWI3LWFmM2ItYzczNDZlNmUzYWNm
  CONSOLE_SECRET_KEY: MGQyY2NlZjktOWM0NC00N2JjLWFkMTYtM2RlNGExMjcwMzY1
kind: Secret
metadata:
  annotations:
```

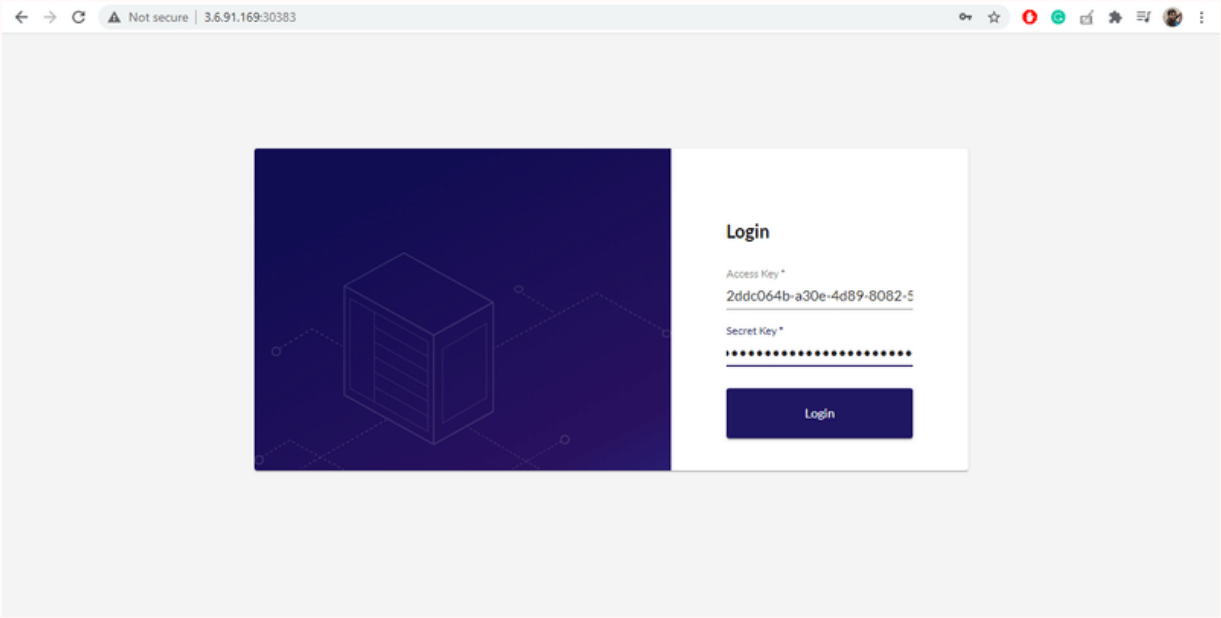
Decoding of the above credentials can be retrieved by following way.

Access key

```
$ echo 'MmRkYzA2NGItYTMwZS00ZDg5LTgwODItNWMwYzFkYTRlOGNh' | base64 -d
2ddc064b-a30e-4d89-8082-5c0c1da4e8ca
```

Secret key

```
$ echo 'MGQyY2NlZjktOWM0NC00N2JjLWFkMTYtM2RlNGExMjcwMzY1' | base64 -
d0d2ccef9-9c44-47bc-ad16-3de4a1270365
```



Access MinIO User console

The MinIO StatefulSet application is created using NodePort as the service type. To access MinIO over a web browser, use <Node_External_Ip>:<NodePort_of_minio_service> this way.

Get the details of Node

```
$ kubectl get node -o wide

NAME                                              STATUS   ROLES    AGE
VERSION    INTERNAL-IP      EXTERNAL-IP      OS-IMAGE
 KERNEL-VERSION   CONTAINER-RUNTIME
ip-192-168-2-55.ap-south-1.compute.internal     Ready    <none>   51m
v1.17.11   192.168.2.55     65.0.121.83      Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
ip-192-168-56-236.ap-south-1.compute.internal   Ready    <none>   57m
v1.17.11   192.168.56.236   15.206.189.106   Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
ip-192-168-69-83.ap-south-1.compute.internal    Ready    <none>   57m
v1.17.11   192.168.69.83    3.6.91.169       Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
ip-192-168-8-117.ap-south-1.compute.internal    Ready    <none>   57m
v1.17.11   192.168.8.117    13.235.210.41    Ubuntu 18.04.5 LTS
 5.4.0-1028-aws   docker://17.3.2
```

Now, access the MinIO service over the browser using the following way.

```
http://3.6.91.169:32095
```

Note: Ensure Inbound Rules under VPC-> Security Groups are correctly configured to allow the traffic.

You should enter the Access key and Secret key to login into the user console. These credentials can be obtained from the secret.

```
$ kubectl get secret tenant1-creds-secret -oyaml
```

The following is a sample snippet of the output of the above command. It will show the Access key and Secret key in encoded form. The decoded value should be given in the web browser to login to the user console.

```
$ kubectl get secret tenant1-creds-secret -oyaml
apiVersion: v1
data:
  accesskey: N2MyMTI0MWItZDczOS00NDEwLWE0OWQtOTkyODkwNDNiNDQ1
  secretkey: M2ZiNGFlZGQtYTU1Yy00YjM4LWJkNTQtODEyNmViOTg5ZmZk
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":
{"accesskey":"N2MyMTI0MWItZDczOS00NDEwLWE0OWQtOTkyODkwNDNiNDQ1","secre
tkey":"M2ZiNGFlZGQtYTU1Yy00YjM4LWJkNTQtODEyNmViOTg5ZmZk"},"kind":"Secr
et","metadata":{"annotations":
{},"creationTimestamp":null,"name":"tenant1-creds-
secret","namespace":"default"}}
```

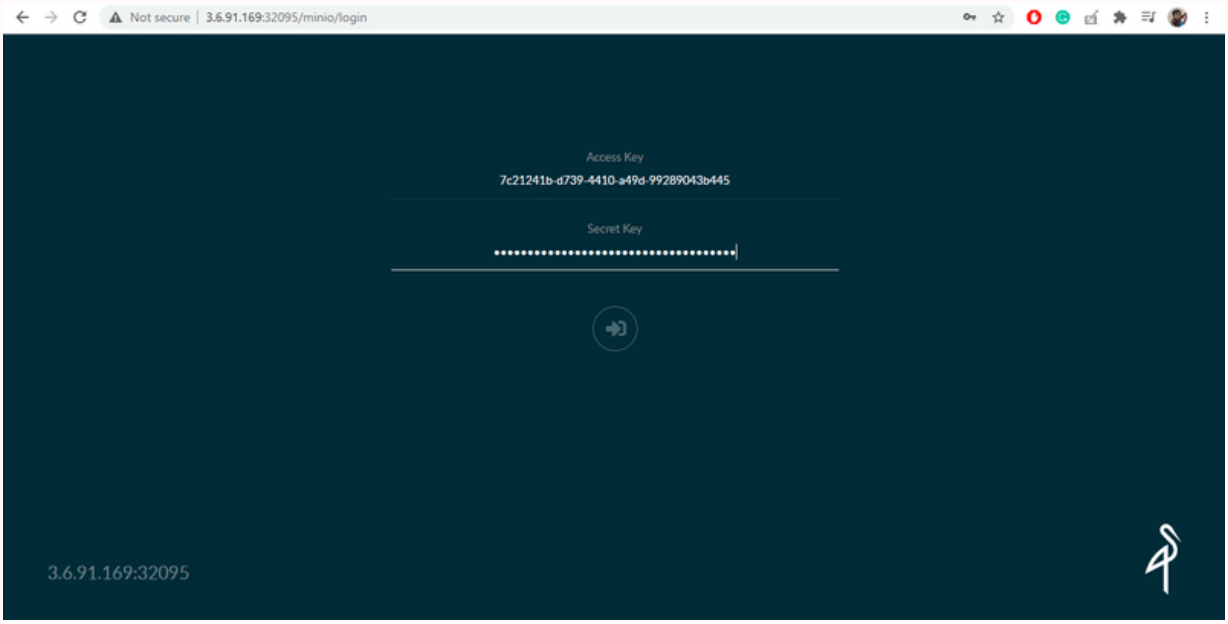
Decoding of the above credentials can be retrieved by following way.

Access key

```
$ echo 'N2MyMTI0MWItZDczOS00NDEwLWE0OWQtOTkyODkwNDNiNDQ1' | base64 -d
7c21241b-d739-4410-a49d-99289043b445
```

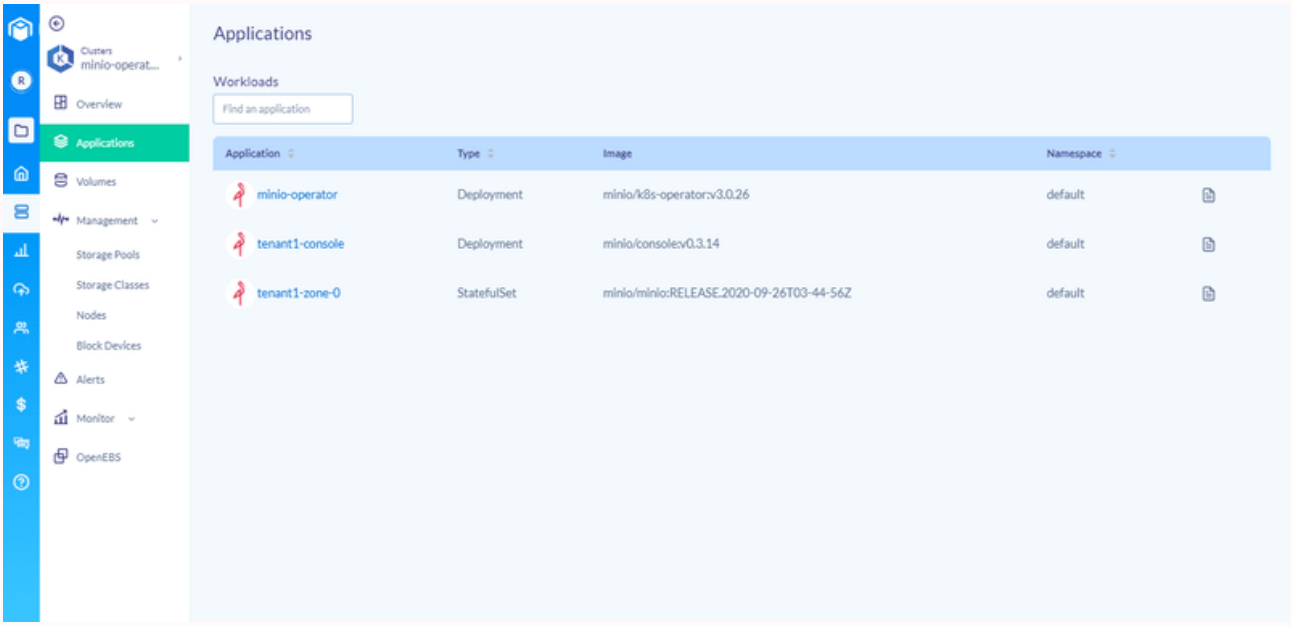
Secret key

```
$ echo 'M2ZiNGFlZGQtYTU1Yy00YjM4LWJkNTQtODEyNmViOTg5ZmZk' | base64 -
d3fb4aedd-a55c-4b38-bd54-8126eb989ffd
```

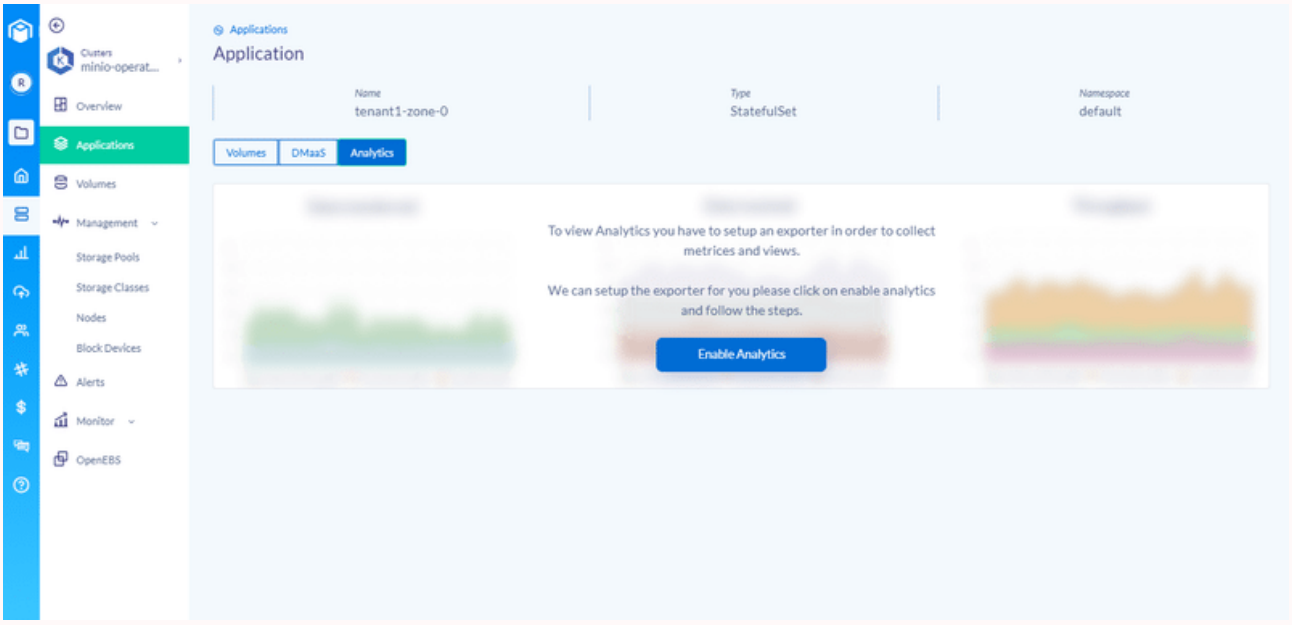


Monitoring MinIO

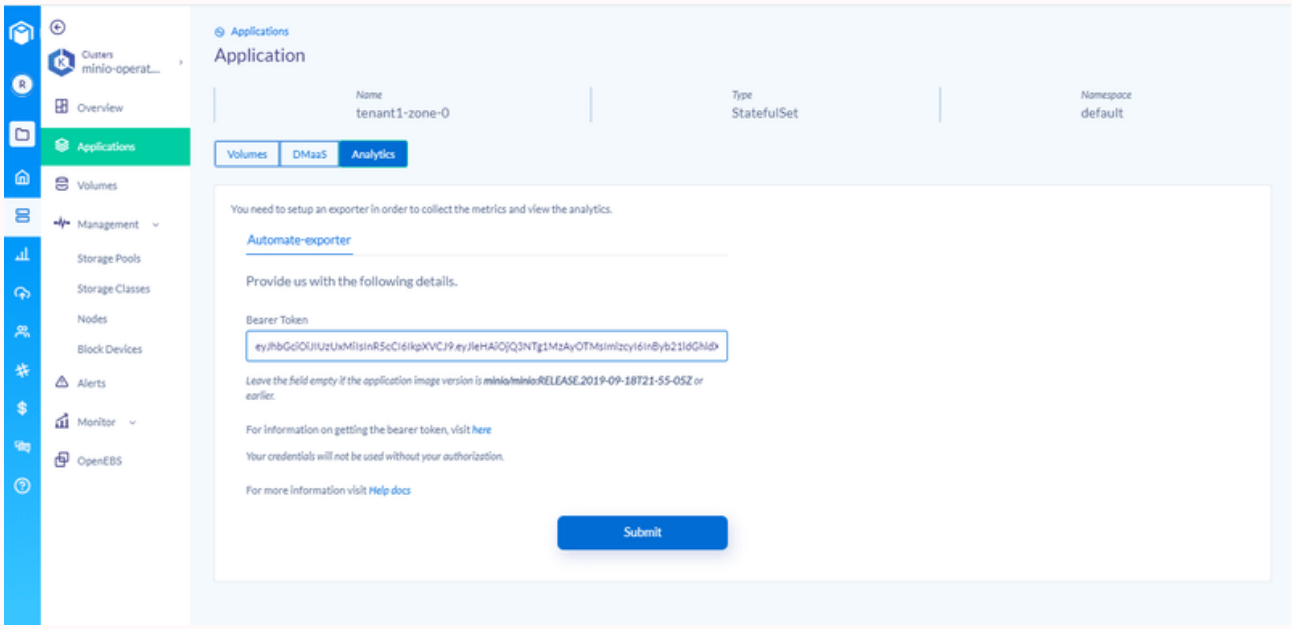
MinIO can be monitored from Kubera itself. For seeing the MinIO metrics, go to the application section and click on the required MinIO application.



Click on the Analytics section from the corresponding application page. Now, click on **Enable Analytics** on an application called **tenant1-zone-0**, a StatefulSet type, to see the metrics.



Next page, you have to provide the bearer token. A bearer token can be created using the instructions provided here. Click on the Submit button once the bearer token is provided.

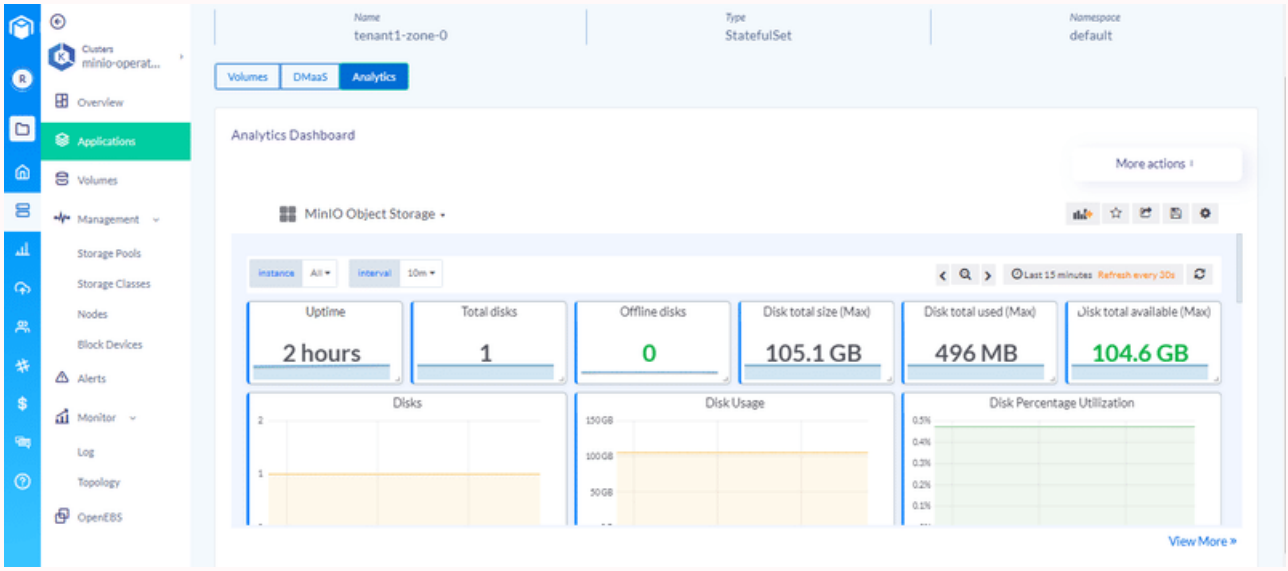


After providing the bearer token, click on **Submit**, proceed to the next section, and view the metrics displayed on the screen. It may take some time to display the metrics on the screen based on the configured interval.

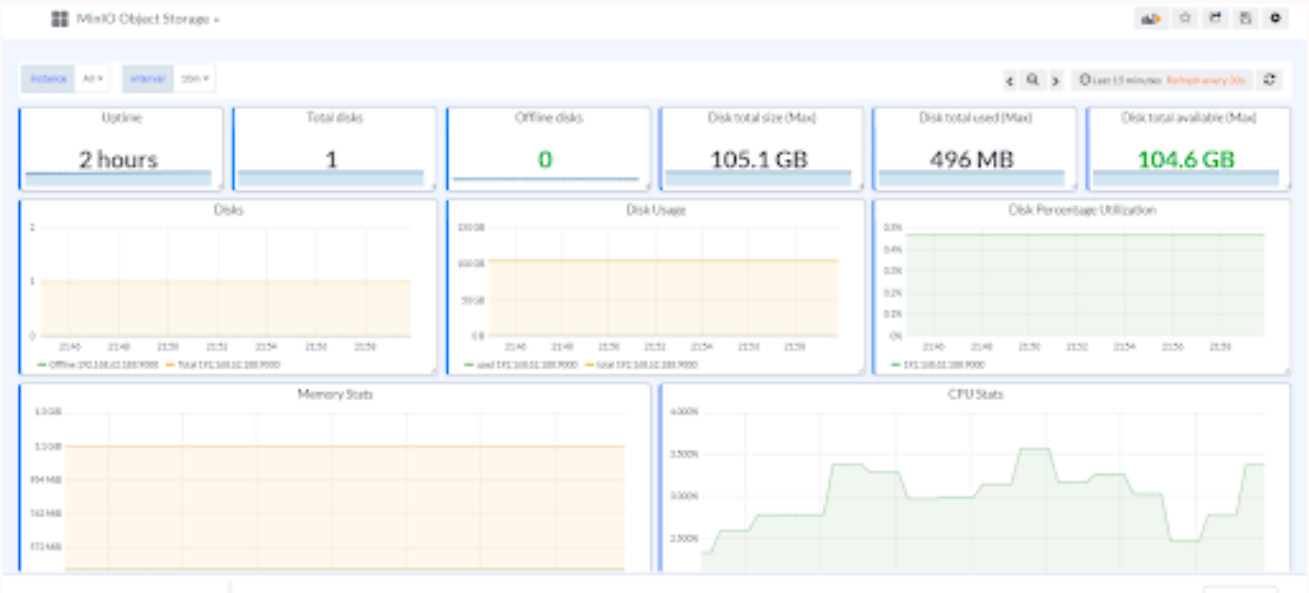
You can also import other Grafana dashboards as per your required metrics. In this guide, We have imported another dashboard by going to this path **[Dashboard_name] -> Import dashboard** and provided the following Grafana URL of the required MinIO dashboard.

https://grafana.com/grafana/dashboards/12563

The following are the sample snippets of the new MinIO dashboard.



Click on **View More** for getting a wider view of available metrics.



CONCLUSION

MinIO is a high-performance Object Storage server and we have deployed MinIO Operator which helps to manage and update highly available distributed MinIO clusters on a four node Kubernetes cluster using the well-known Container Attached Storage solution OpenEBS where OpenEBS Local PV as the underlying storage engine. We have used Kubera to orchestrate the storage requirement for MinIO. In addition to this, we showed how to check metrics, monitoring, and a number of other features that can be automatically configured for MinIO using Kubera.

MayaData

linkedin.com/company/mayadata/

twitter.com/MayaData

blog.mayadata.io/