HCLSoftware

HCL CDP v12.1.8 Platform OpenShift Installation Guide

Contents

Chapter 1. Introduction

This section details the installation and configuration of the services and components required for deploying HCL CDP (Customer Data Platform) on the Red Hat OpenShift platform.

Chapter 2. Prerequisites

There are two parts of the deployment:

- Micro services based components, to be deployed on containerization platform. Here, Red Hat OpenShift is the choice of the Container orchestration platform.
- Virtual Machine (VM) based deployment.

Before proceeding with the installation of the required components, you have to ensure that Red Hat OpenShift is installed and properly configured. Refer to the Red Hat OpenShift documentation for installation guidelines.

Also, below tools are expected to present in the deployment to facilitate the installation:

1. OpenShift CLI ("oc" command)
2. Kubernetes command line tool ("kubectl" command)
3. Helm CLI tool ("helm" command)

Hardware Prerequisites

Given below is the minimum hardware requirement for the HCL CDPdeployment.

Standalone VMs

Server/Node	No. of Server/Node	CPU	Memory	Storage	Network
Bastion host	1	2 vCPUs for a 7h 12m burst	8.0 GiB	EBS only	Low to Moderate
Aerospike DB	2	4 vCPUs	16.0 GiB	150 GB NVMe SSD	Up to 10 Gigabit
DMP server	1	2 vCPUs	8.0 GiB	EBS only	Up to 12.5 Gigabit
Druid server	2	16 vCPUs	128.0 GiB	EBS only	Up to 10 Gigabit
Mongo DB Server	4	4 vCPUs	16.0 GiB	150 GB NVMe SSD	Up to 10 Gigabit
Mongo DB Server	1	2 vCPUs for a 4h 48m burst	2.0 GiB	EBS only	Up to 5 Gigabit
Nifi DB	3	2 vCPUs	16.0 GiB	75 GB NVMe SSD	Up to 10 Gigabit
SFTP Server	1	2 vCPUs	8.0 GiB	EBS only	Up to 12.5 Gigabit
postgres-1	2	2 vCPUs	8.0 GiB	EBS only	Up to 12.5 Gigabit
RabbitMq	1	2 vCPUs	8.0 GiB	EBS only	Up to 10 Gigabit
TC Redis	1	2 vCPUs	16.0 GiB	EBS only	Up to 10 Gigabit

Server/Node	No. of Server/Node	CPU	Memory	Storage	Network
Redis & RMQ for Celery	1	2 vCPUs	16.0 GiB	EBS only	Up to 10 Gigabit
Scheduler	1	2 vCPUs	16.0 GiB	EBS only	Up to 10 Gigabit

EKS Cluster

Server/Node	CPU	Memory	Storage	Network
3 Nodes	32 vCPUs	64.0 GiB	EBS only	12.5 Gigabit
18 Nodes	4 vCPUs	16.0 GiB	EBS only	Up to 12.5 Gigabit

EMR Cluster

Server/Nodes	CPU	Memory	Storage	Network
Primary Node	4 vCPUs	16.0 GiB	EBS only	Up to 10 Gigabit
Core Node	4 vCPUs	16.0 GiB	EBS only	Up to 10 Gigabit

Software requirements

Given below is a list of components to be deployed and configured before the deployment for CDP Core modules. CDP utilizes the functionalities from these modules to perform the end-to-end operations.

Application/Service	Deployment Type	Version
HashiCorp Vault	Helm chart	Vault v1.17.2
Red Hat Quay Operator		
RHBK Operator, OpenShift RBAC	NA	
NFS	Storage Class on OpenShift	
AMQ streams	OpenShift Operator	
SMTP server		
Trino	Helm chart	v0.7.0
Apache Spark (Spark ETL jobs)	Helm chart	3.5.1
HAProxy(route/ingress)	Route on OpenShift	
MinIO	Helm chart	v5.0.15
Apache Airflow	Helm chart	2.9.2

Application/Service	Deployment Type	Version
Stackable Operator for Apache Spark (certified) / Spark Helm Operator(Community)	Helm chart	24.3.0
PostgreSQL	PgSQL on VM	
AMQ broker	OpenShift Operator	
3Scale	Api-Gateway	

Chapter 3. Supported Cloud Platforms

HCL CDP is compatible with a range of leading cloud platforms, ensuring flexibility and scalability for deployment. The supported platforms include:

- **Amazon Elastic Kubernetes Service (Amazon EKS):** Managed Kubernetes service provided by AWS for running containerized applications.
- **Google Cloud Platform (GCP):** Offers robust Kubernetes-based services for scalable deployments.
- **Microsoft® Azure:** Provides integrated cloud solutions for hosting containerized workloads using Azure Kubernetes Service (AKS).
- **Red Hat OpenShift®:** An enterprise-grade Kubernetes platform optimized for managing containerized applications at scale.

Chapter 4. Downloading HCL CDP Artifacts

This page explains on how to download the required images and charts from Flex Net Operations (FNO).

Start by downloading all the necessary Cloud Native CDP images from the FNO (Field Network Operations). These images are essential for the deployment process. The Cloud Native CDP images, by default, include charts that are pre-configured with application resources. These charts simplify the deployment process.

Before proceeding with the implementation of Cloud Native CDP, ensure that the Cloud Native environment is properly set up. This environment is crucial for running the platform smoothly. The downloaded charts use Helm as a package manager for Kubernetes. Helm is responsible for managing, updating, and deploying the charts on the Kubernetes cluster. The Helm charts deploy the CDP components onto the specified Kubernetes cluster, allowing the platform to function as expected.

After downloading, extract the ZIP file to a designated location on the cloud VM (Virtual Machine) where you plan to deploy the CDP. This location will serve as the base for managing and deploying the platform components.

By following this sequence, you'll be able to successfully set up and deploy the Cloud Native CDP platform on OpenShift.

1. Download the required HCL CDP artifacts from Flex Net Operations (FNO). To roll out the Helm charts, you must specify the offering-related details and requirements. Contact support team to get a Helm chart.
2. To import the downloaded Docker images for all the products, run the following command.

   ```
   docker load -i product_image_name.tar
   ```

3. To verify all products images are loaded and available for use, run the following command.

   ```
   docker images
   ```

4. To tag the images appropriately, run the following command.

   ```
   docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
   ```

5. To push the images to the docker registry, run the following command.

   ```
   docker push TARGET_IMAGE[:TAG]
   ```

6. To push the product helm charts to container registry, run the following command.

   ```
   helm push product_chart_name.tgz oci://container_registry_path
   ```

Chapter 5. Deploying HCL CDP

This section provides step-by-step instructions for deploying each service, component, and module required forHCL CDP. Follow the guidelines in each subsection to ensure the successful deployment and configuration of the respective elements.

Installing HashiCorp Vault

This section provides a step-by-step guide to installing HashiCorp Vault.

HashiCorp Vault, a powerful tool secures secrets management and key management services (KMS). HashiCorp Vault ensures robust encryption, secure access to sensitive data, and centralized management of secrets across your applications and infrastructure. The page covers the installation process, initial configuration, and integration with your deployment environment to enhance security and compliance.

Prerequisite:

Make sure to assign required privileges in the security context (scc) to the below service accounts on the namespace, where the vault will be deployed.

- vault-cluster-agent-injector
- vault-cluster

In case of **anyuid** scc, add the below sections under user:

```
oc edit scc anyuid
- system:serviceaccount:vault-cluster-agent-injector:vault-server
- system:serviceaccount:vault-cluster:vault-serve
```

Installing HashiCorp Vault

To install HashiCorp Vault, follow the steps below:

1. As a first step in installation, add the HashiCorp value repository using the below helm command.

   ```
   helm repo add HashiCorp https://helm.releases.hashicorp.com
   helm repo update
   ```

2. Create a vault-values-override.yaml file to deploy the vault in HA mode and vault data persistence mode. A sample yaml file along with configuration is shown below.

   ```
   # Vault Helm Chart Value Overrides
   global:
     enabled: true
     tlsDisable: true
     resources:
       requests:
         memory: 256Mi
         cpu: 250m
       limits:
         memory: 256Mi
         cpu: 250m
   ```

```
server:
  # Use the Enterprise Image
  # image:
    # repository: "hashicorp/vault-enterprise"
    # tag: "1.16.1-ent"

  # These Resource Limits are in line with node requirements in the
  # Vault Reference Architecture for a Small Cluster
  resources:
    requests:
      memory: 8Gi
      cpu: 2000m
    limits:
      memory: 16Gi
      cpu: 2000m

  # For HA configuration and because we need to manually init the vault,
  # we need to define custom readiness/liveness Probe settings
  readinessProbe:
    enabled: true
    path: "/v1/sys/health?standbyok=true&sealedcode=204&uninitcode=204"
  livenessProbe:
    enabled: true
    path: "/v1/sys/health?standbyok=true"
    initialDelaySeconds: 60

  # extraEnvironmentVars is a list of extra environment variables to set with the stateful set. These
 could be
  # used to include variables required for auto-unseal.
  extraEnvironmentVars:
    # VAULT_CACERT: /vault/userconfig/tls-ca/ca.crt

  # extraVolumes is a list of extra volumes to mount. These will be exposed
  # to Vault in the path `/vault/userconfig/<name>/`.
  extraVolumes:
    # - type: secret
      # name: tls-server
    # - type: secret
      # name: tls-ca
    # - type: secret
      # name: kms-creds

  # This configures the Vault Statefulset to create a PVC for audit logs.
  # See https://www.vaultproject.io/docs/audit/index.html to know more
  auditStorage:
    enabled: true

  standalone:
    enabled: false

  # Run Vault in "HA" mode.
  ha:
    enabled: true
    replicas: 2
    raft:
      enabled: true
      setNodeId: true
```

```
      config: |
        ui = true
        cluster_name = "vault-cluster-with-integrated-storage"
        listener "tcp" {
          tls_disable = 1
          address = "[::]:8200"
          cluster_address = "[::]:8201"
        }

        storage "raft" {
          path = "/vault/data"
          retry_join {
            leader_api_addr = "http://vault-cluster-0.vault-test-internal:8200"
          }
          retry_join {
            leader_api_addr = "http://vault-cluster-1.vault-test-internal:8200"
          }
          autopilot {
            server_stabilization_time = "100s"
            last_contact_threshold = "10s"
            min_quorum = 5
            cleanup_dead_servers = false
            dead_server_last_contact_threshold = "10m"
            max_trailing_logs = 1000
            disable_upgrade_migration = false
          }
        }

# Vault UI
ui:
  enabled: true
  serviceType: "LoadBalancer"
  serviceNodePort: null
  externalPort: 8200
```

In case, if there are more that 2 vaults replica, you can adjust and update the below properties, accordingly:

- retry_join
- replicas

3. After configuring the yaml file, you can install the Vault using Helm chart as shown below.

```
helm upgrade --install vault hashicorp/vault --namespace vault --set
"global.openshift=true" --set
"server.image.repository=docker.io/hashicorp/vault" --set
 "injector.image.repository=docker.io/hashicorp/vault-k8s"
 -f /<path-to-your-vault-override-values-file>/vault-override-values.yaml --debug
```

4. On successful installation of the vault, you can verify the installation using the below command.

```
oc get pods --namespace vault
```

Note: After installation, make sure to unseal the vault as shown below. Otherwise, the pods will keep restart.

```
NAME                                    READY   STATUS    RESTARTS        AGE
vault-0                                 1/1     Running   0               95m
vault-1                                 1/1     Running   2 (37m ago)     39m
vault-agent-injector-67c4c7c489-wx5p9   1/1     Running   6 (101m ago)    107m
```

Configuring HashiCorp Vault

This section provides a step-by-step guide to configure HashiCorp Vault.

After installing the HashiCorp Vault, configuring the Vault is a crucial step to ensure that it integrates effectively within your infrastructure and meets your security requirements. This process involves initializing and unsealing Vault, setting up Kubernetes authentication, creating routes for UI access, and defining user access policies.

Initialize Vault Pod

1. Initiate the vault-0 pod for execution in the OpenShift environment.

```
oc exec -it vault-0 -- /bin/sh -n vault
vault operator init
```

2. On successful initializing, the output will be as shown below:

```
$ vault operator init
Unseal Key 1: he8HRofmeEDrgLKOg9EDMqb1DGY/GMluGecHWEuiajs/
Unseal Key 2: fC/Ouew4GupSLZ1f0qDSJHU63bACFzEERW4LjDISwPSk
Unseal Key 3: GVY8sWWlafxz+7pbvFasSlLpYfYL+60gGmP6M22NEEyA
Unseal Key 4: Qq3n3SQ0tDJLkoeAwt4mG8McAWn79F/xm2TwLFPZzdS3
Unseal Key 5: N5ETxm3ZPczf6gN1/mKzR/u+DIk29ZxQDMPM/6CXup0C

Initial Root Token: hvs.A2kzLVXS8h51JBTSovnOH9XF

Vault initialized with 5 key shares and a key threshold of 3. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal it
before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to
reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
```

```
existing unseal keys shares. See "vault operator rekey" for more information.
```

Note: Make sure to store the Unseal keys and Root Token, safely. By default, the Vault is sealed, and to unseal them, three keys out of five are required. You can use the root token to login to Vault either via Vault UI or from inside the pod.

Unseal Vault

After initializing the vault and obtaining the keys and token, you can unseal the Vault from vault-0 pod using the following commands.

1. Run the vault operator unseal command thrice with 3 different Unseal keys retrieved in the previous step. After unsealing the Vault thrice, the vault status will be changed from Sealed=true to Sealed=False.

   ```
   oc exec -it vault-0 -- /bin/sh -n vault
   vault operator unseal
   ```

2. Now, join the Vault-1 pod with cluster, and unseal the vault-1 using the keys found during the vault-0 initialization.

   ```
   vault operator raft join http://vault-0.vault-internal:8200
   vault operator unseal
   ```

3. After joining the cluster and Vault-1 pod, login into the vault-0 pod. Login to the Vault using the root password retrieved during vault-0 initialization.

   ```
   vault login <root password>
   # output will be - Success! You are now authenticated. The token information displayed below
   ```

4. Now, enable kubernetes authentication.

   ```
   vault auth enable kubernetes
   ```

5. Perform authorization method to use location of kubernetes host.

   ```
   vault write auth/kubernetes/config \ kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"
   ```

6. Verify the cluster setup.

   ```
   vault operator raft list-peers
   ```

   On successful verification, the output will be as shown below.

   ```
   Node                                    Address                        State       Voter
   ----                                    -------                        -----       -----
   a1799962-8711-7f28-23f0-cea05c8a527d    vault-0.vault-internal:8201    leader      true
   e6876c97-aaaa-a92e-b99a-0aafab105745    vault-1.vault-internal:8201    follower    true
   ```

Create route to access Vault UI

After unsealing the vault, explore the vault service and create a route to access the Vault UI.

1. Check for available services in the Vault.

   ```
   oc get svc -n vault
   #output will be like as shown below
   vault-cluster-ui   LoadBalancer   172.30.229.200   <pending>  8200:30530/TCP
   ```

2. Expose the Vault service.

```
oc expose svc vault-cluster-ui -n vault
```

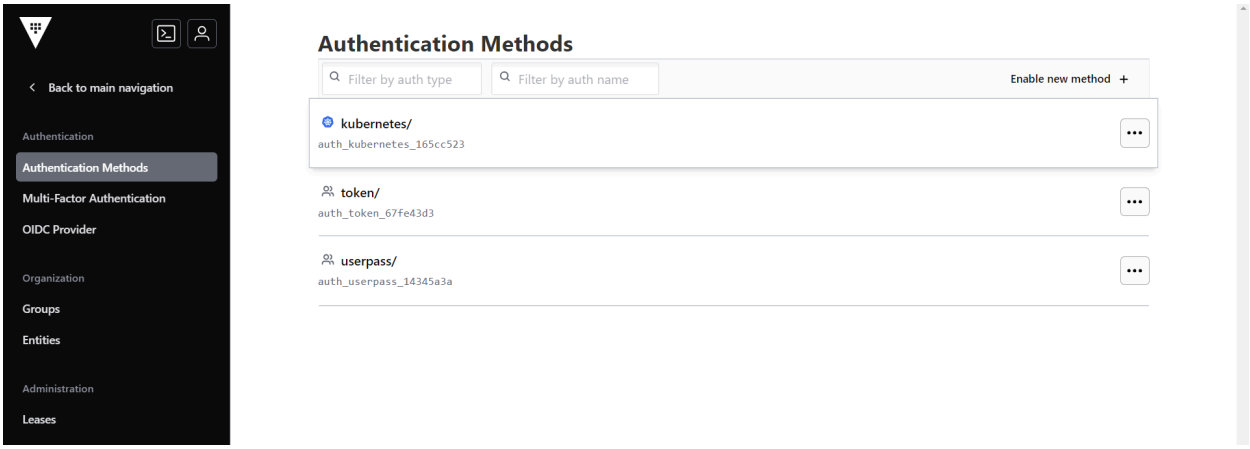
3. Now, create a route to access the Vault UI.

```
oc get routes -n vault
```

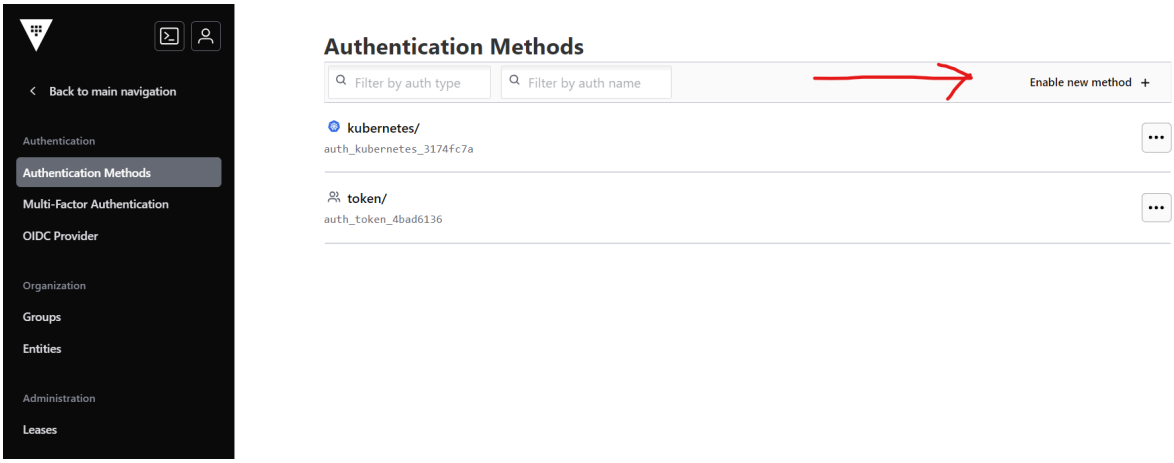
4. Copy the route HOST/PORT URL and access the Vault UI. To login, use **Token** as User name, and **<root token>**, which is received after vault initialization done earlier on vault-0 pod, as password.

Create user for Vault Access

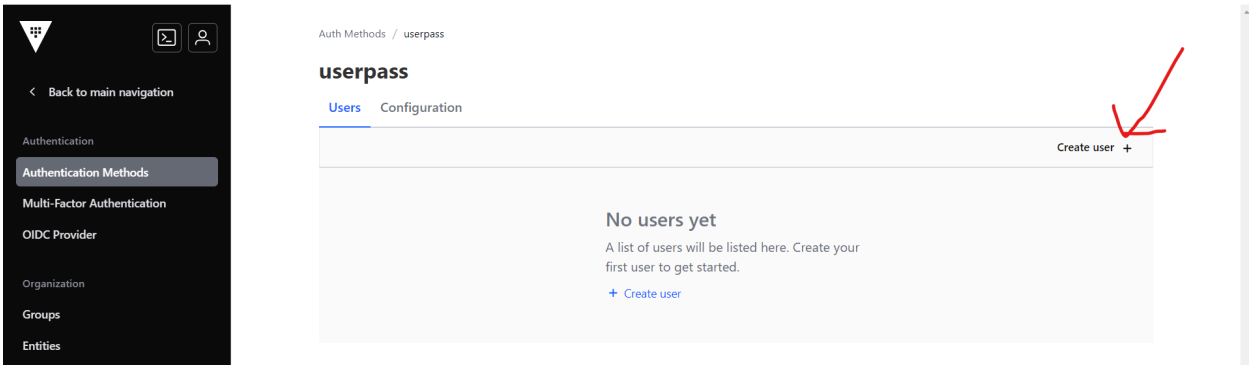
1. Using the port URL and credentials, login to the Vault UI. Navigate to Access > Authentication Method > Authentication Method.



2. In the Authentication Method section, click the Enable new method as shown below.

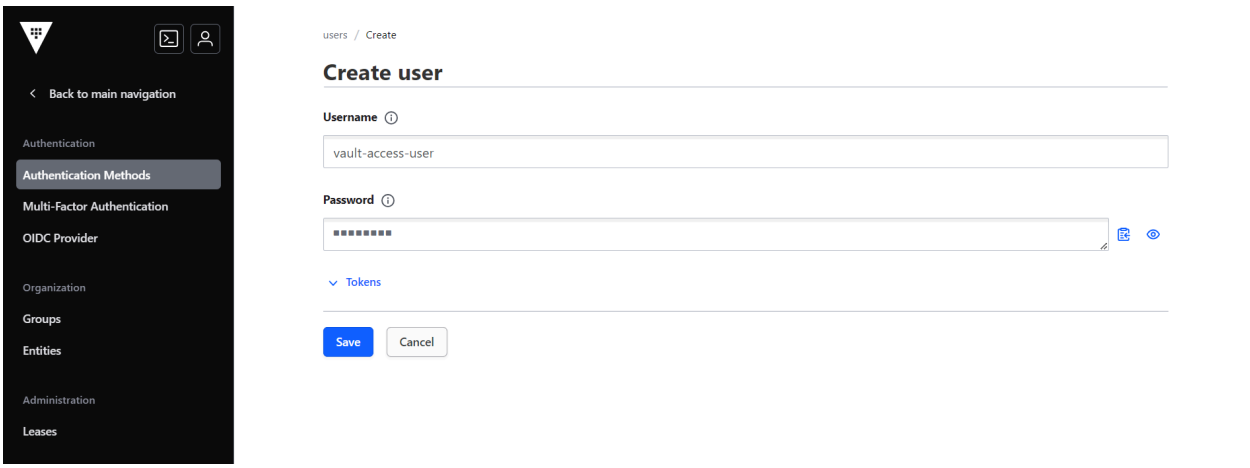


3. Click the **Username & Password** icon, and click **Create user**.

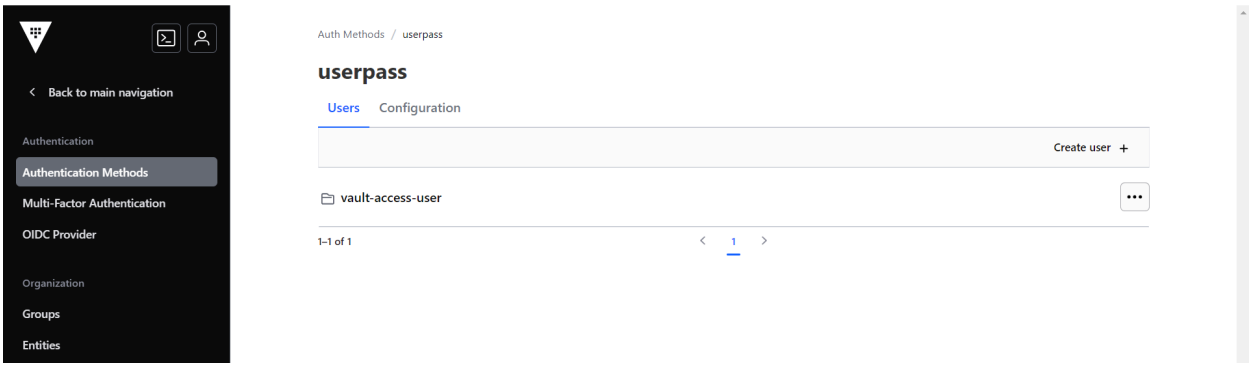


4. In the Create user section, enter the username and password.

Note: Make sure to use base 64 bit encoded password.



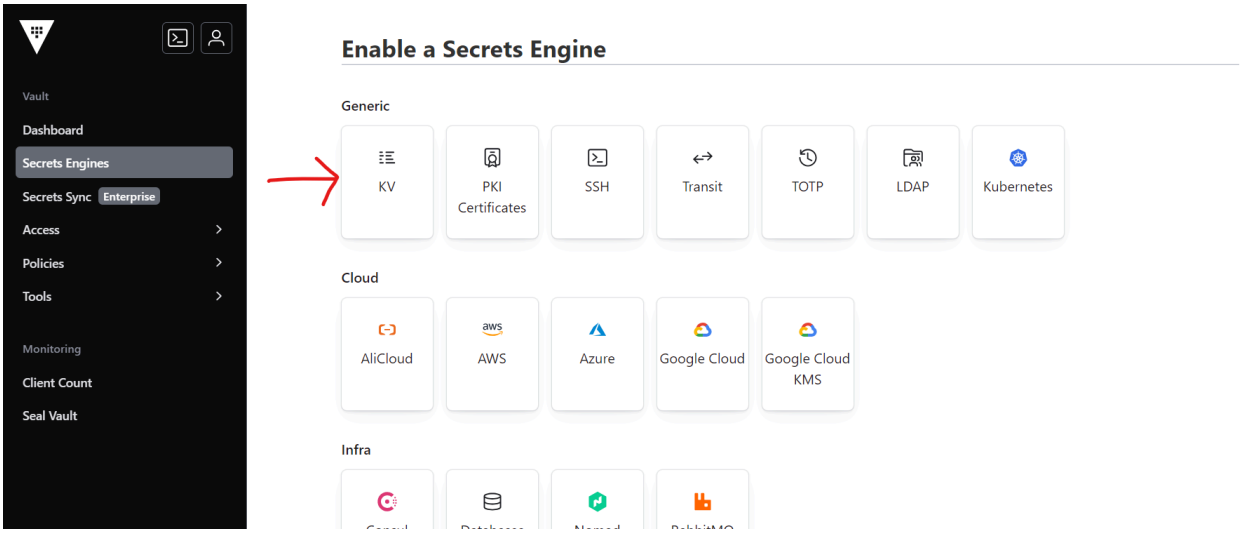
5. The new user will be created listed in the userpass section. The new user can access the vault to retrieve the secrets.



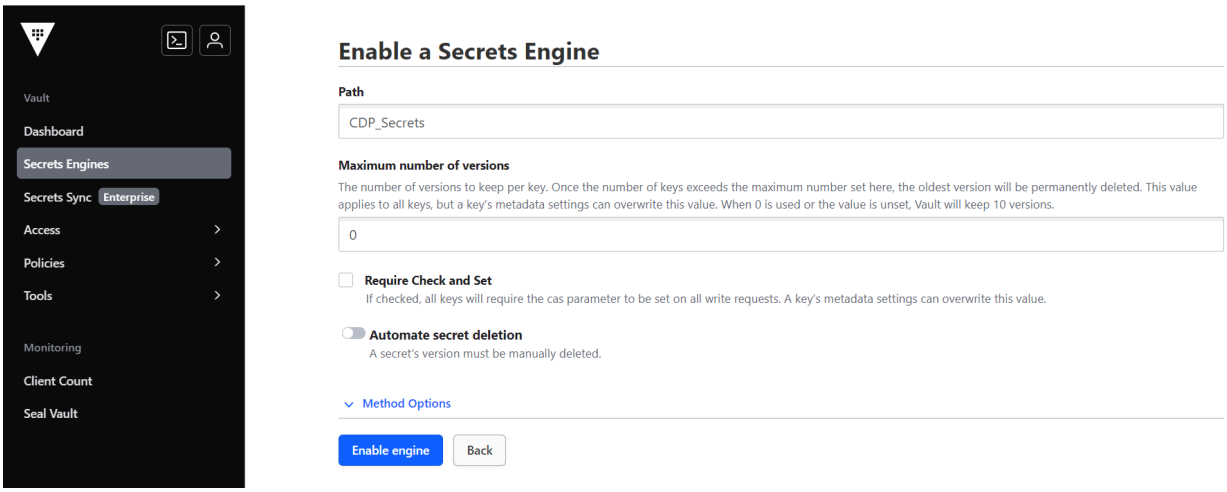
Create Secrets Engine to store secrets in Vault

To create secretes engine, follow the steps below:

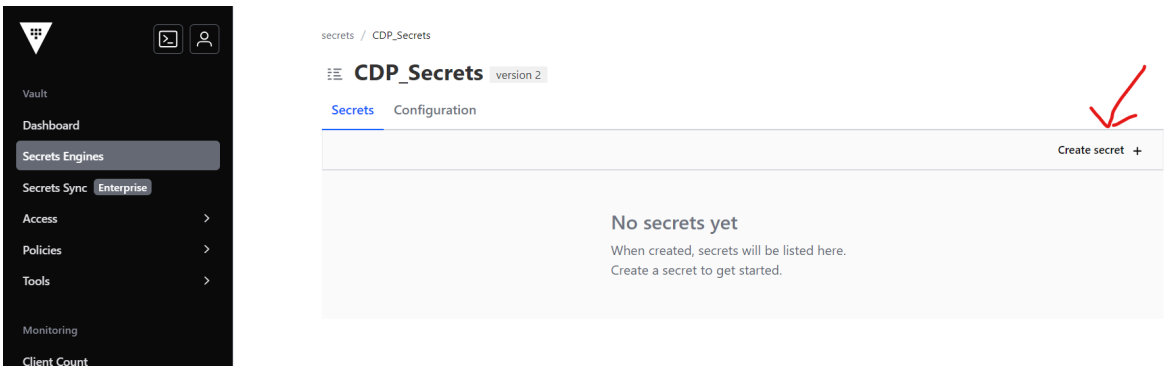
1. In the Vault UI, navigate to Secrets Engines > Enable new engine, and click KV icon.



2. Enter the name of the KV secret folder, inside which you can create the secrets, and click Enable engine to create the KV folder.



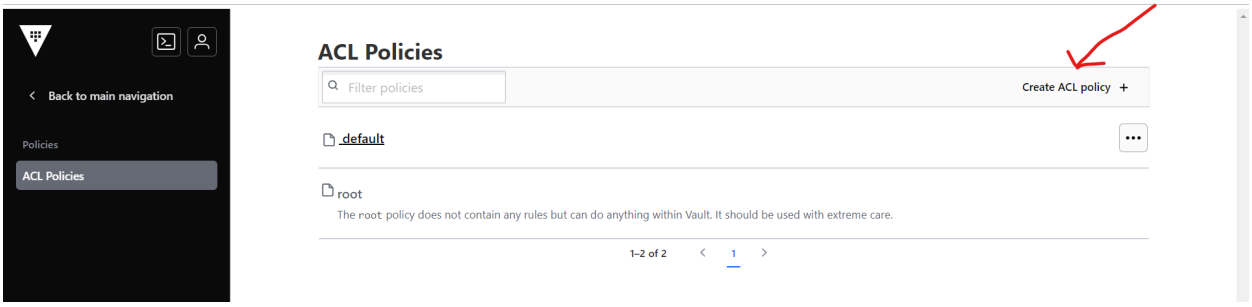
3. In the KV Folder section, click Create secret option to create new secrets.



Create User Policy

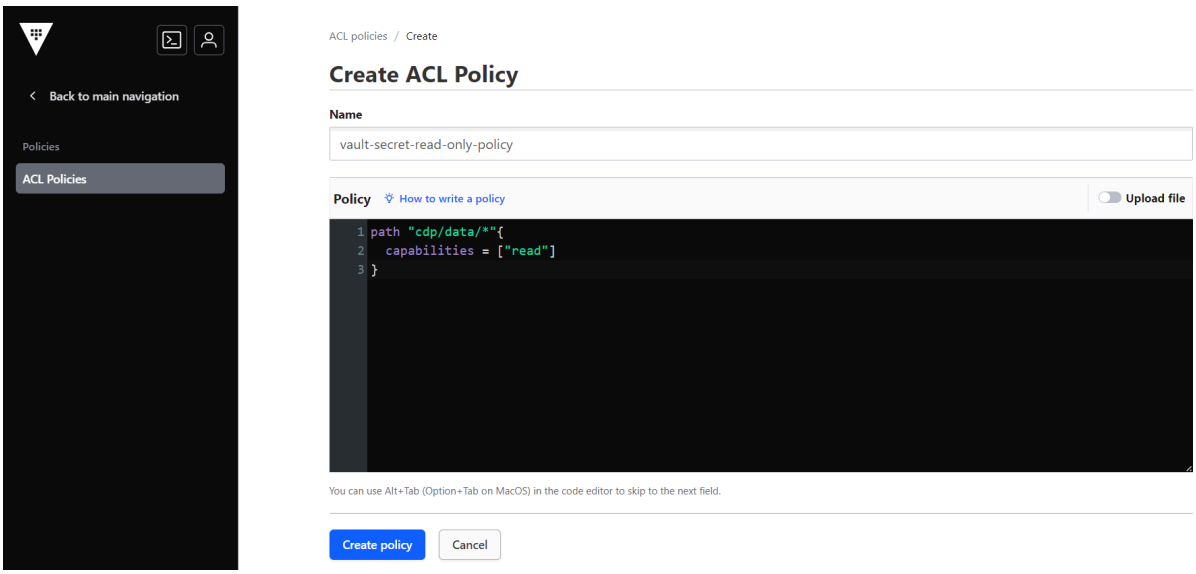
After creating secret folder, you can create policy to access the secrets from Vault.

1. In the Vault UI, click Policies.
2. In the ACL Policies section, click Create ACL policy.



3. As a result, the Create ACL Policy section will be displayed. In the Create ACL Policy section, enter policy name and policy as shown below.

```
path "CDP_Secrets/data/*"{
capabilities = ["read"]
}
```



4. Click Create policy to create the policy.
5. Now, you can add the policy to a vault user account. Run the below command with the user credentials to add the policy in vault-0.

```
vault write auth/userpass/users/<your-vault-access-user>
password=<your-vault-user-password> policies=<your-vault-access-policy>
```

Example:

```
vault write auth/userpass/users/vault-access-user password=mypassword
 policies=vault-secret-read-only-policy
```

Installing AMQ Streams Operator for Kafka

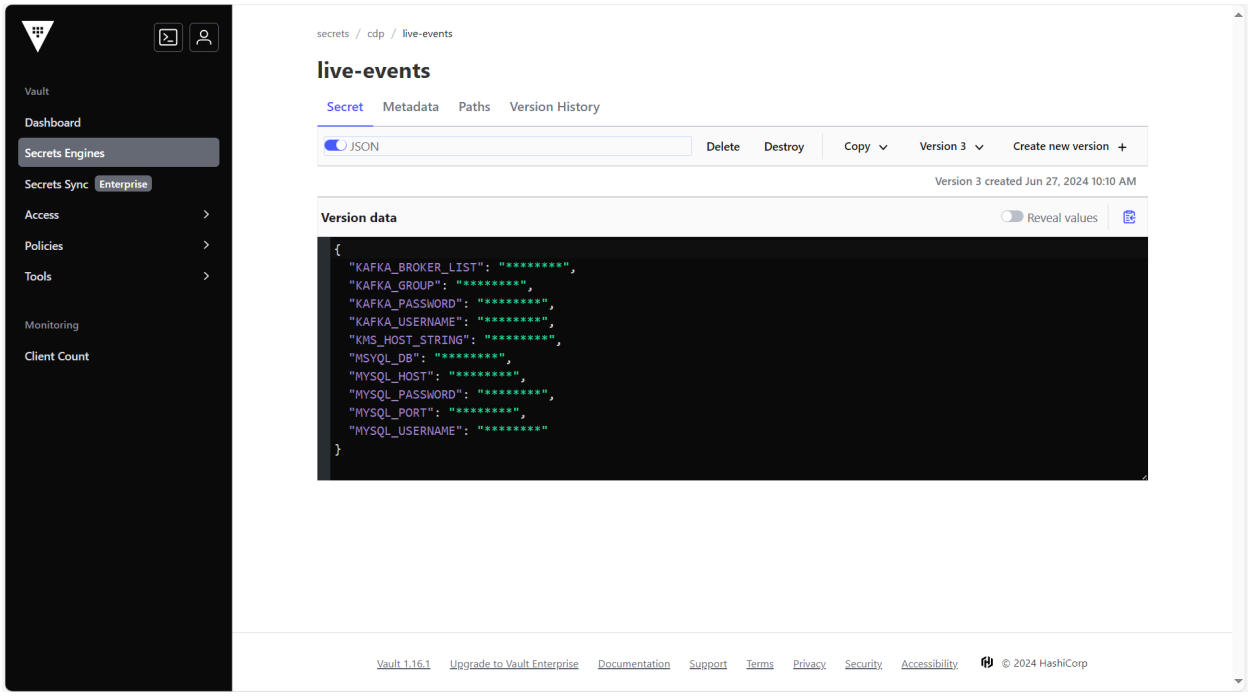
This section provides a detailed guide to installing and configuring the AMQ Streams Operator for Kafka on Red Hat OpenShift.

AMQ Streams simplifies the deployment and management of Apache Kafka clusters in a Kubernetes environment, enabling reliable real-time data streaming and messaging capabilities.

The guide includes steps for installing the AMQ Streams Operator from the OpenShift Operator Catalog, deploying a Kafka cluster, configuring essential resources like secrets and users, and setting up a sample Kafka cluster with Zookeeper.

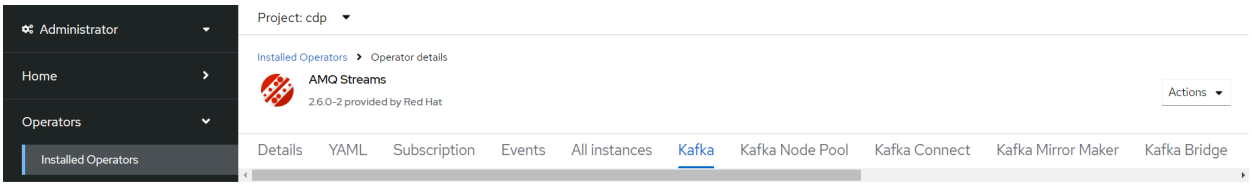
1. In the OpenShift Operator catalog, search and install the AMQ Streams operator.

2. In the **Operators** > **Installed Operations**, deploy a Kafka Cluster Instance.



Note: Make sure to create a secret with username and password for Kafka user and create a Kafka user. For more information on creating user and secrets, refer and .

3. After creating the user account and secrets, in the Operator details section, for the AMQ Streams, select Kafka and click **Create Kafka**.



4. In the **Create Kafka** section, click the YAML View option, and create a Kafka cluster instance with zookeeper. A sample config.yaml file is provided below.

```
apiVersion: kafka.strimzi.io/v1beta2x
kind: Kafka
metadata:
  labels:
    app: sbi-kafka
  name: hclsw-sbi-dev-kafka
  namespace: kafka-ns
spec:
  entityOperator:
    topicOperator: {}
    userOperator: {}
  kafka:
    authorization:
      superUsers:
        - hclsw-sbi-kafka-user
      type: simple
    config:
      default.replication.factor: 2
      inter.broker.protocol.version: '3.6'
      min.insync.replicas: 2
      offsets.topic.replication.factor: 2
      transaction.state.log.min.isr: 2
      transaction.state.log.replication.factor: 2
    listeners:
      - authentication:
          type: scram-sha-512
        configuration:
          preferredNodePortAddressType: Hostname
        name: external
        port: 9094
        tls: false
        type: nodeport
      - authentication:
          type: scram-sha-512
        name: internal
        port: 9092
        tls: false
        type: internal
    replicas: 2
    resources:
      limits:
        cpu: '4'
        ephemeral-storage: 20Gi
        memory: 8Gi
      requests:
        cpu: '2'
        ephemeral-storage: 20Gi
```

```
        memory: 2Gi
    storage:
      class: nfssc
      deleteClaim: false
      size: '20'
      type: persistent-claim
    version: 3.6.0
  zookeeper:
    replicas: 3
    resources:
      limits:
        cpu: '4'
        ephemeral-storage: 20Gi
        memory: 8Gi
      requests:
        cpu: '2'
        ephemeral-storage: 20Gi
        memory: 2Gi
    storage:
      class: nfssc
      deleteClaim: true
      size: 20Gi
      type: persistent-claim
```

5. On successful deployment of cluster, the two broker pods and three zookeeper pods will be created.

```
NAME                                                  READY   STATUS    RESTARTS   AGE
hclsw-sbi-dev-kafka-entity-operator-c89bd7567-spkvb   3/3     Running   3          27d
hclsw-sbi-dev-kafka-kafka-0                           1/1     Running   0          10d
hclsw-sbi-dev-kafka-kafka-1                           1/1     Running   0          10d
hclsw-sbi-dev-kafka-zookeeper-0                       1/1     Running   1          27d
hclsw-sbi-dev-kafka-zookeeper-1                       1/1     Running   0          27d
hclsw-sbi-dev-kafka-zookeeper-2                       1/1     Running   0          27d
```

6. To access the Kafka within the cluster, the Bootstrapper URL will be created as shown below. `<Your-cluster-name>-kafka-bootstrap.cdp.svc:9092`
7. Similarly, to access the Kafka from outside the cluster (DMP Producer application), the Bootstrapper URL will be created as shown below. `<hostname1>:31571,<hostname2>:31571`
8. Once a Kafka instance is deployed follow below steps to create a secret and user for accessing Kafka.

Creating secret for Kafka User

The `Secret` object type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, private source repository credentials, and so on. You can create a kafka-cdp-secret on the OpenShift cluster for Kafka User.

Note: Make sure that the username and password must be in base64 format.

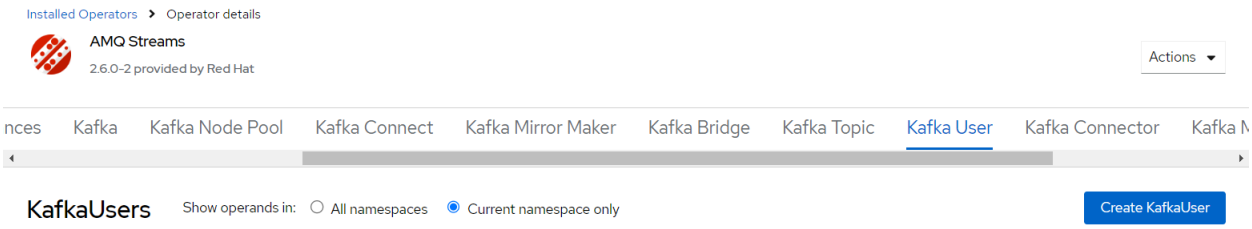
```
apiVersion: v1
data:
  password: WWQ4cXloWmNuREpXUUpMMzg2dGZ4UQ==
  username: aGNsc3ctc2JpLWthZmthLWNkcC11c2Vy
kind: Secret
metadata:
  name: kafka-cdp-secret
```

```
  namespace: cdp
type: Opaque
```

Creating Kafka User

To create a Kafka user, follow the steps below:

1. In the **AMQ Streams** section, select Kafka User and click **Create KafkaUser** for accessing Kafka and performing operations on topics and create consumer groups.



2. Create the user with label having Kafka cluster name e.g. hclsw-sbi-dev-kafka.

```
strimzi.io/cluster: hclsw-sbi-dev-kafka
```

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  labels:
    app: sbi-kafka
    strimzi.io/cluster: hclsw-sbi-dev-kafka
  name: hclsw-sbi-kafka-user
  namespace: kafka-ns
spec:
  authentication:
    password:
      valueFrom:
        secretKeyRef:
          key: password
          name: kafka-cdp-secret
    type: scram-sha-512
  authorization:
    acls:
      - host: '*'
        operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic
        type: allow
      - host: '*'
        operation: All
        resource:
          name: '*'
          patternType: literal
          type: group
        type: allow
    type: simple
```

3. In the YAML view, update the authorization block to allow permissions to user on topic and Group.

```
spec:
  authentication:
    password:
      valueFrom:
        secretKeyRef:
          key: password
          name: hclsw-sbi-kafka-secret
    type: scram-sha-512
  authorization:
    acls:
      - host: '*'
        operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic
        type: allow
      - host: '*'
        operation: All
        resource:
          name: '*'
          patternType: literal
          type: group
        type: allow
    type: simple
```

Deploying Kafka UI for AMQ Streams

This section provides a comprehensive guide for deploying Kafka UI

Kafka UI, a web-based interface for managing and monitoring Apache Kafka clusters deployed through AMQ Streams. Kafka UI simplifies interactions with Kafka by providing an intuitive dashboard for exploring topics, consumers, and producers, as well as monitoring real-time cluster performance.

The page covers creating secrets for secure authentication, configuring necessary services and deployments, and integrating the UI with the Kafka cluster. By following this guide, you will enable seamless access to Kafka's operational insights, facilitating efficient cluster management and troubleshooting.

Prerequisites

Make sure the following things are in place, before creating the kafka-ui-secrets.yaml file:

1. Convert bootstrap servers URL to base-64.

```
<Your-cluster-name>-kafka-bootstrap.cdp.svc:9092
```

2. Convert password **kafka-ui-passwd** to base-64.
3. Convert the **sasl-jass-config** value to base-64.

```
org.apache.kafka.common.security.scram.ScramLoginModule required
 username="kafka-user-used-in-kafka-cdp-secret" password="base64 password";
```

4. Convert sasl-mechanism **SCRAM-SHA-512** to base-64.
5. Convert security-protocol **SASL_PLAINTEXT** to base-64

Deploy Kafka UI

To deploy the Kafka UI, follow the steps below:

1. Update the converted values in the below yaml file.

```
apiVersion: v1
data:
  bootstrap-servers: aGNsc3ctc2JpLWRldi1rYWZrYS1rYWZrYS1ib290c3RyYXAuY2RwLnN2Yzo5MDky
  password: a2Fma2EtdWktcGFzc3dk
  sasl-jaas-config:
 b3JnLmFwYWNoZS5rYWZrYS5jb21tb24uc2VjdXJpdHkuc2NyYW0uU2NyYW1Mb2dpbk1vZHVsZSByZXF1aXJlZCB1c2VybmFtZT0iaGN
sc3ctc2JpLWthZmthLXVzZXIiIHBhc3N3b3JkPSJZZDhxeWhaY25ESldRSkwzODZ0ZnhRIjs=
  sasl-mechanism: U0NSQU0tU0hBLTUxMg==
  security-protocol: U0FTTF9TU0w=
kind: Secret
metadata:
  name: kafka-ui-secret
type: Opaque
```

2. Create the Kafka UI Secret resource in the specified namespace.

```
oc apply -f kafka-ui-secret.yaml -n <your namespace>
```

3. Create a service definition in `kafka-ui-service.yaml` to expose the Kafka UI within the cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: kafka-ui-service
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
  selector:
    app: kafka-ui
```

4. Deploy the Kakfa UI service.

```
oc apply -f kafka-ui-service.yaml -n <your namespace>
```

5. Define the Kafka UI deployment in `kafka-ui-deployment.yaml`, including environment variables and secrets.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kafka-ui
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kafka-ui
  template:
```

```
    metadata:
      labels:
        app: kafka-ui
    spec:
      containers:
        - env:
            - name: KAFKA_CLUSTERS_0_NAME
              value: kafka-cluster
            - name: KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS
              valueFrom:
                secretKeyRef:
                  key: bootstrap-servers
                  name: kafka-ui-secret
            - name: KAFKA_CLUSTERS_0_PROPERTIES_SECURITY_PROTOCOL
              valueFrom:
                secretKeyRef:
                  key: security-protocol
                  name: kafka-ui-secret
            - name: KAFKA_CLUSTERS_0_PROPERTIES_SASL_MECHANISM
              valueFrom:
                secretKeyRef:
                  key: sasl-mechanism
                  name: kafka-ui-secret
            - name: KAFKA_CLUSTERS_0_PROPERTIES_SASL_JAAS_CONFIG
              valueFrom:
                secretKeyRef:
                  key: sasl-jaas-config
                  name: kafka-ui-secret
            - name: KAFKA_CLUSTERS_0_METRICS_PORT
              value: "11001"
            - name: DYNAMIC_CONFIG_ENABLED
              value: "true"
            - name: AUTH_TYPE
              value: LOGIN_FORM
            - name: SPRING_SECURITY_USER_NAME
              value: admin
            - name: SPRING_SECURITY_USER_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: password
                  name: kafka-ui-secret
          image: provectuslabs/kafka-ui:latest
          name: kafka-ui
          ports:
            - containerPort: 8080
```

6. Deploy kafka UI in the namespace.

```
oc apply -f kafka-ui-deployment.yaml -n <your namespace>
```

7. On successful deployment, expose the Kafka UI service to create a route and access the Kafka UI.

```
oc expose svc kafka-ui-service -n <your namespace>
oc get routes -n <your namespace>
```

```
Login username : - admin
Login password : - kafka-ui-passwd
```

8. Use the exposed route URL to access Kafka UI in your browser.

   Login credentials:
   - **Username:** `admin`
   - **Password:** `kafka-ui-passwd`
9. Upon successful deployment, you can use Kafka UI to manage and monitor your Kafka clusters effectively.

Installing Trino

This section provides a step-by-step guide to installing Trino.

Trino is a SQL query engine and does not store data. Instead, Trino interacts with various databases or directly on object storage. Trino parses and analyzes the SQL query you pass in, creates and optimizes a query execution plan that includes the data sources, and then schedules worker nodes that are able to intelligently query the underlying databases they connect to.

Here, Trino is deployed in accordance with Minio for Dataset Store, Hive Metastore for Metadata, and Redis for table schema.

Prerequisites:

Before performing the installation, make sure to have the following things in place:

- **kubectl**: Install primary command-line tool for managing Kubernetes clusters, which allows to inspect, manipulate, and administer cluster resources.
- **helm**: A package manager for Kubernetes. Helm allows you to deploy, upgrade, and manage applications within your cluster using pre-defined charts.

To install Trino and its component, follow the steps below:

1. As a first step to access the resources needed for deploying Trino on Kubernetes, clone the GitHub repository.

   ```
   git clone https://github.com/minio/blog-assets.git
   cd blog-assets/trino-on-kubernetes
   ```
2. Create a new namespace for Trino in kubectl to provide isolated environments for applications.

   ```
   kubectl create namespace trino --dry-run=client -o yaml | kubectl apply -f -
   ```
3. Create a generic Kubernetes secret for sourcing data from a JSON file to secure the Redis table schema.

   ```
   kubectl create secret generic redis-table-definition --from-file=redis/test.json -n trino || true
   ```
4. Then, Add the Bitnami and Trino repositories to Helm configuration.

   ```
   helm repo add bitnami https://charts.bitnami.com/bitnami || true
   helm repo add trino https://trinodb.github.io/charts/ || true
   ```
5. After adding the repositories, install PostgreSQL.

   ```
   helm upgrade --install hive-metastore-postgresql bitnami/postgresql -n trino -f
    hive-metastore-postgresql/values.yaml
   ```

6. Update the /blog-assets/trino-on-kubernetes/hive-metastore/values.yaml file with Minio details and deploy the Hive Metastore within the Trino namespace.

```
helm upgrade --install my-hive-metastore -n trino -f hive-metastore/values.yaml ./charts/hive-metastore
```

7. Redis is a high-speed, in-memory data store used to hold Trino table schema for enhanced query performance. Deploy Redis in the Trino namespace using helm chart.

```
helm upgrade --install my-redis bitnami/redis -n trino -f redis/values.yaml
```

8. Update the security context to allow permissions to default service account on namespace where trino will be deployed.

```
oc edit scc anyuid
users:
- system:serviceaccount:trino:default
```

9. Update /blog-assets/trino-on-kubernetes/trino/values.yaml with Minio details, and deploy Trino as the distributed SQL query engine that will connect to MinIO and other data sources.

```
helm upgrade --install my-trino trino/trino --version 0.7.0 --namespace trino -f trino/values.yaml
```

10. On successful deployment, verify that all the components are running correctly.

```
kubectl get pods -n trino
```

11. As a result, the output will be as shown below.

```
$ oc get pods
NAME                                  READY   STATUS    RESTARTS   AGE
hive-metastore-postgresql-0           1/1     Running   0          6h59m
my-hive-metastore-0                   1/1     Running   0          6h57m
my-redis-master-0                     1/1     Running   0          6h56m
my-redis-replicas-0                   1/1     Running   0          6h56m
my-redis-replicas-1                   1/1     Running   0          6h56m
my-redis-replicas-2                   1/1     Running   0          6h55m
my-trino-coordinator-767ff55b85-vp7wm 1/1     Running   0          6h40m
my-trino-worker-5f6bcd5c46-7n9gf      1/1     Running   0          6h47m
my-trino-worker-5f6bcd5c46-gchqd      1/1     Running   0          6h46m
$
```

Installing Spark

This section provides a step-by-step guide to installing Spark

Apache Spark is an open-source unified analytic engine for large-scale data processing. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance.

To install spark, follow the steps below:

1. Add bitnami Repositories to helm configuration.

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

2. Deploy Spark in a namespace using Helm.

```
helm install my-spark bitnami/spark --namespace spark
```

3. On successful deployment, verify that all the components are running correctly.

```
oc get pods -n spark
```

4. As a result, the output will be as shown below.

```
my-spark-master-0 1/1 Running 0 2d4h
my-spark-worker-0 1/1 Running 0 2d4h
my-spark-worker-1 1/1 Running 0 2d4h
```

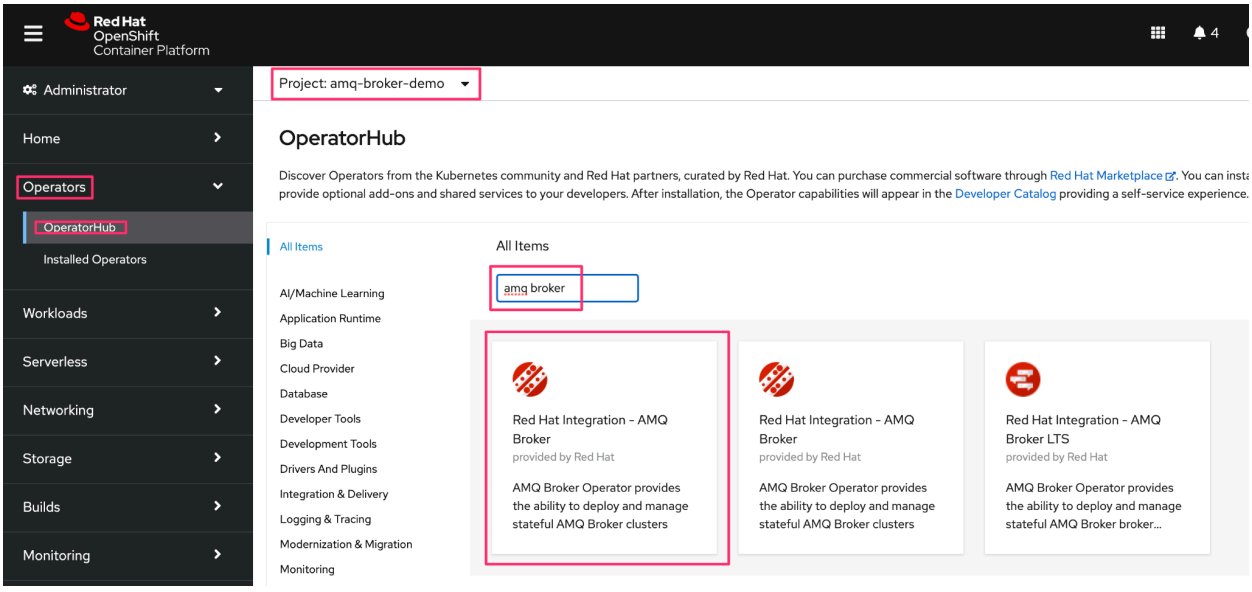
Installing AMQ Broker

This section provides a step-by-step guide to installing AMQ Broker.

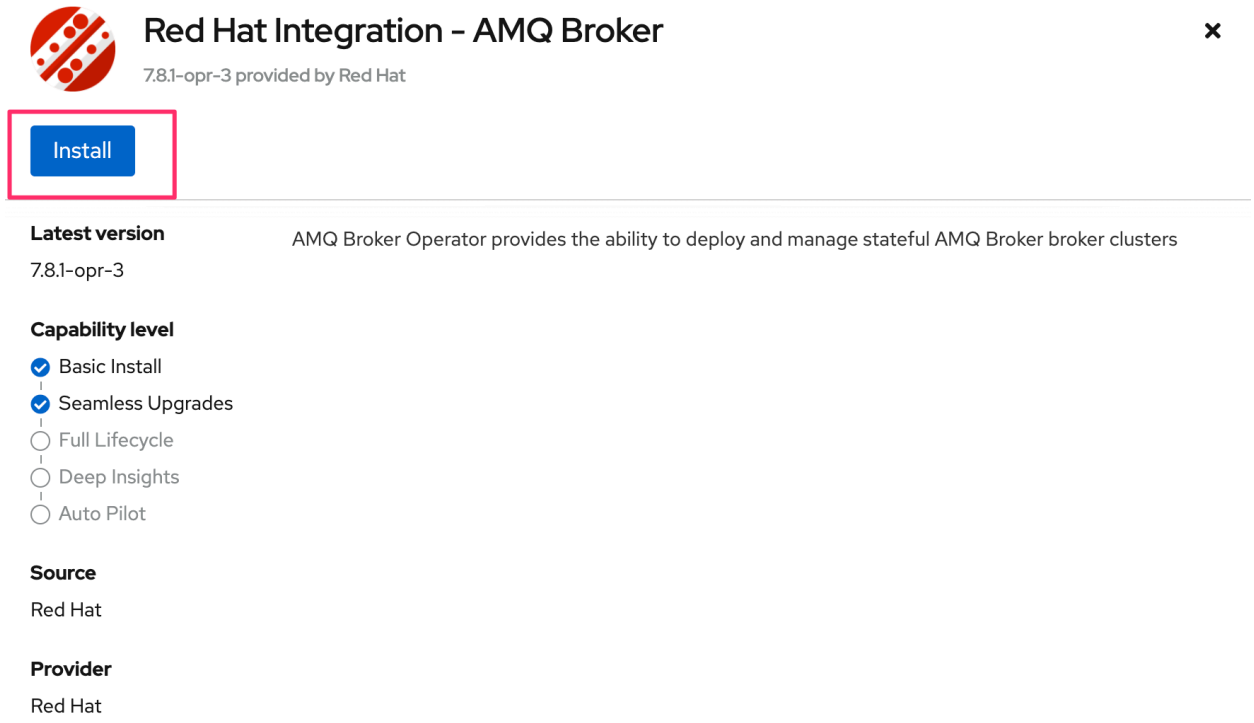
AMQ Broker is a high-performance messaging implementation based on ActiveMQ Artemis. It uses an asynchronous journal for fast message persistence, and supports multiple languages, protocols, and platforms.

To install AMQ broker, follow the steps below:

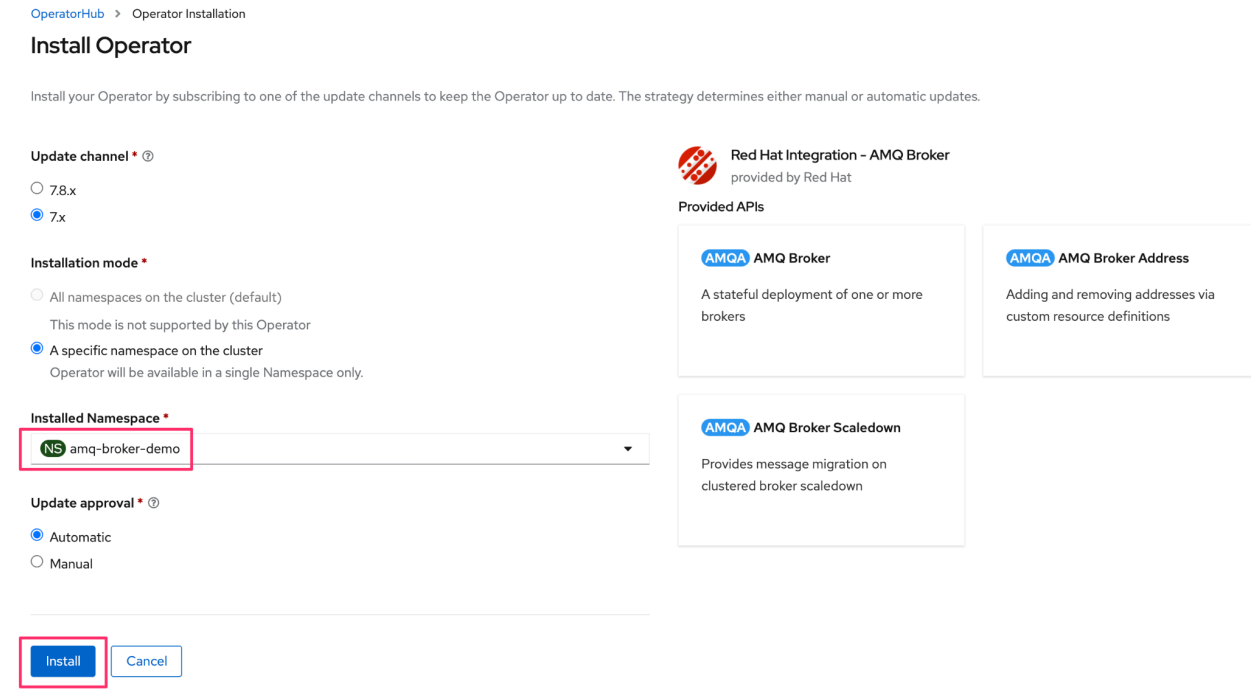
1. First, install the AMQ Operator from the Operator Hub. In the OpenShift Container platform, click Operators > OperatorHub, and select the project.



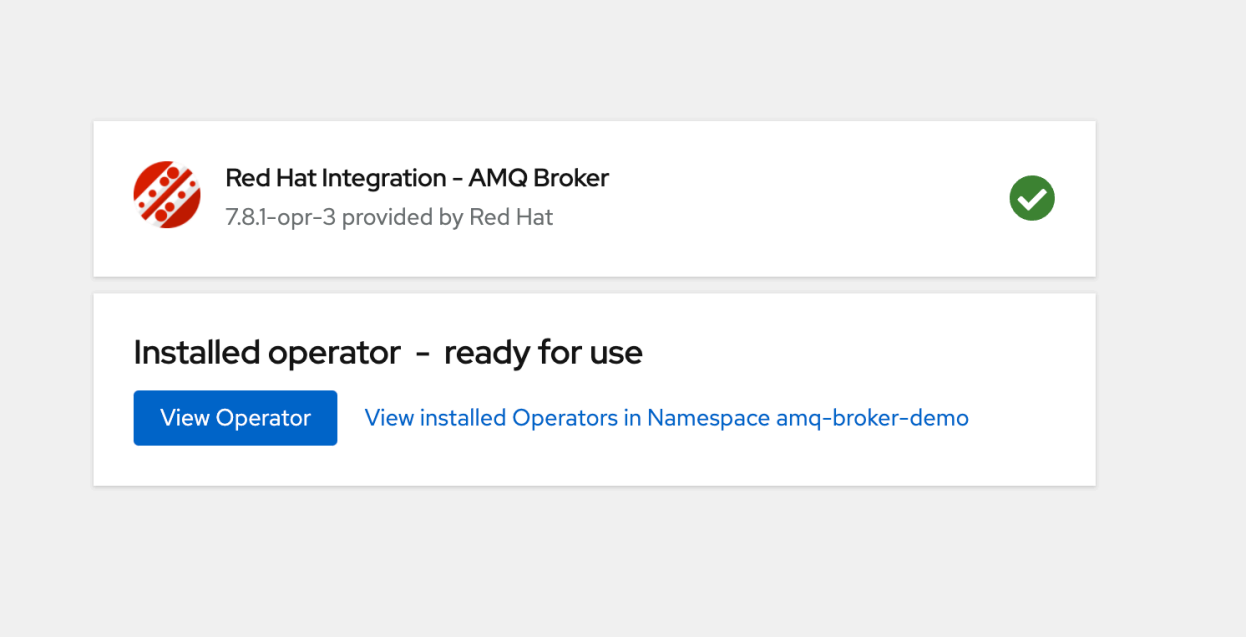
2. In the OperatorHub, search for AMQ Broker, and in the Red Hat Integration - AMQ Broker, click Install.



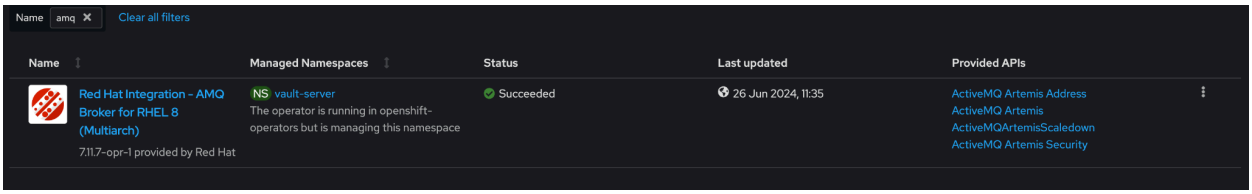
3. Select corresponding options like channel, installation mode and approvals, and click Install.



4. On successful installation, you can view the below page.



5. You can view the installed application as shown below.



Create ActiveMQ Artemis instance:

Using the OpenShift Container Platform web console:

1. Log in to the console as a user that has privileges to deploy CRs in the project in which you are creating the deployment.
2. Start a new CR instance based on the main broker CRD. In the left pane, click Administration > Custom Resource Definitions.
3. Click the ActiveMQArtemis CRD, and then click the **Instances** tab.
4. Click **Create ActiveMQArtemis**. Within the console, a YAML editor opens, enabling you to configure a CR instance.

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
 creationTimestamp: '2024-07-23T05:45:12Z'
 generation: 2
 managedFields:
   - apiVersion: broker.amq.io/v1beta1
     fieldsType: FieldsV1
     fieldsV1:
       'f:spec':
         .: {}
```

```
        'f:deploymentPlan':
          'f:jolokiaAgentEnabled': {}
          'f:image': {}
          .: {}
          'f:size': {}
          'f:persistenceEnabled': {}
          'f:managementRBACEnabled': {}
          'f:requireLogin': {}
          'f:messageMigration': {}
          'f:journalType': {}
    manager: Mozilla
    operation: Update
    time: '2024-07-23T06:42:10Z'
  - apiVersion: broker.amq.io/v1beta1
    fieldsType: FieldsV1
    fieldsV1:
      'f:status':
        .: {}
        'f:conditions': {}
        'f:deploymentPlanSize': {}
        'f:podStatus':
          .: {}
          'f:ready': {}
        'f:scaleLabelSelector': {}
        'f:upgrade':
          .: {}
          'f:majorUpdates': {}
          'f:minorUpdates': {}
          'f:patchUpdates': {}
          'f:securityUpdates': {}
        'f:version':
          .: {}
          'f:brokerVersion': {}
          'f:image': {}
          'f:initImage': {}
    manager: amq-broker-operator
    operation: Update
    subresource: status
    time: '2024-07-23T12:50:35Z'
 name: amq-broker
 namespace: amq-broker-ns
 resourceVersion: '2781361'
 uid: 977dc755-fa47-46b0-a222-046467f524eb
spec:
 deploymentPlan:
   image: placeholder
   jolokiaAgentEnabled: false
   journalType: nio
   managementRBACEnabled: true
   messageMigration: false
   persistenceEnabled: true
   requireLogin: false
   size: 1
status:
 conditions:
   - lastTransitionTime: '2024-07-23T05:45:12Z'
     message: ''
     observedGeneration: 2
```

```
      reason: ValidationSucceded
      status: 'True'
      type: Valid
    - lastTransitionTime: '2024-07-23T12:50:35Z'
      message: ''
      reason: Applied
      status: 'True'
      type: BrokerPropertiesApplied
    - lastTransitionTime: '2024-07-23T12:50:35Z'
      message: ''
      reason: AllPodsReady
      status: 'True'
      type: Deployed
    - lastTransitionTime: '2024-07-23T12:50:35Z'
      message: ''
      reason: ResourceReady
      status: 'True'
      type: Ready
    - lastTransitionTime: '2024-07-23T12:50:35Z'
      message: ''
      reason: VersionMatch
      status: 'True'
      type: BrokerVersionAligned
  deploymentPlanSize: 1
  podStatus:
    ready:
      - amq-broker-ss-0
  scaleLabelSelector: 'ActiveMQArtemis=amq-broker,application=amq-broker-app'
  upgrade:
    majorUpdates: true
    minorUpdates: true
    patchUpdates: true
    securityUpdates: true
  version:
    brokerVersion: 7.12.1
    image:
  'registry.redhat.io/amq7/amq-broker-rhel8@sha256:10855749104ac3ecefe62a15aa18e22d485b4e9c7cbf9fc1894366
ed02fdf28d'
    initImage:
  'registry.redhat.io/amq7/amq-broker-init-rhel8@sha256:3adf4e74ed54043c867f6f444e4dc97471194994b6bb5060f
4e9437c2db14029'
```

5. When you have finished configuring the CR, click Create.

Creating ActiveMQArtemisAddress (if required)

To create ActiveMQArtemisAddress yaml, follow the steps below:

1. Log in to the console as a user that has privileges to deploy CRs in the project for the broker deployment.
2. Start a new CR instance based on the address CRD. In the left pane, click Administration > Custom Resource Definitions.
3. Click the ActiveMQArtemisAddresss CRD, click the **Instances** tab.
4. Click **Create ActiveMQArtemisAddress**. Within the console, a YAML editor opens, enabling you to configure a CR instance.

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisAddress
metadata:
 creationTimestamp: '2024-07-02T11:41:02Z'
 generation: 1
 managedFields:
   - apiVersion: broker.amq.io/v1beta1
     fieldsType: FieldsV1
     fieldsV1:
       'f:spec':
         .: {}
         'f:addressName': {}
         'f:queueName': {}
         'f:routingType': {}
     manager: Mozilla
     operation: Update
     time: '2024-07-02T11:41:02Z'
 name: artemis-address-queue
 namespace: vault-server
 resourceVersion: '48399557'
 uid: bd5194d5-748e-441d-b054-1258e00a2154
spec:
 addressName: myAddress0
 queueName: myQueue0
 routingType: anycast
```

5. When you have finished configuring the CR, click Create.

Configuring ActiveMQ Service (Node Port)

Similarly, create another service for node port, and in the YAML editor, configure the service as shown below.

```
kind: Service
apiVersion: v1
metadata:
 name: amq-broker-svc
 namespace: vault-server
 uid: 6e4a975d-3cd0-41a0-9f82-28d5275e76d6
 resourceVersion: '48454106'
 creationTimestamp: '2024-07-02T12:47:04Z'
 labels:
   ActiveMQArtemis: amq-broker
   application: amq-broker-app
 managedFields:
   - manager: Mozilla
     operation: Update
     apiVersion: v1
     time: '2024-07-02T12:54:04Z'
     fieldsType: FieldsV1
     fieldsV1:
       'f:metadata':
         'f:labels':
           .: {}
           'f:ActiveMQArtemis': {}
           'f:application': {}
       'f:spec':
         'f:allocateLoadBalancerNodePorts': {}
         'f:externalTrafficPolicy': {}
```

```
         'f:internalTrafficPolicy': {}
         'f:ports':
           .: {}
           'k:{"port":61616,"protocol":"TCP"}':
             .: {}
             'f:port': {}
             'f:protocol': {}
             'f:targetPort': {}
         'f:selector': {}
         'f:sessionAffinity': {}
         'f:type': {}
spec:
 clusterIP: 172.30.41.165
 externalTrafficPolicy: Cluster
 ipFamilies:
   - IPv4
 ports:
   - protocol: TCP
     port: 61616
     targetPort: 61616
     nodePort: 31635
 internalTrafficPolicy: Cluster
 clusterIPs:
   - 172.30.41.165
 type: NodePort
 ipFamilyPolicy: SingleStack
 sessionAffinity: None
 selector:
   ActiveMQArtemis: amq-broker
   application: amq-broker-app
status:
 loadBalancer: {}
```

Now, using the assigned port number (31635 - in this case), you can access the AMQ Artemis Service from outside the cluster.

Example:

```
$./artemis queue stat --url tcp://10.14.108.233:31635
Connection brokerURL = tcp://10.14.108.233:31635
|NAME                     |ADDRESS                  |CONSUMER_COUNT |MESSAGE_COUNT |MESSAGES_ADDED
 |DELIVERING_COUNT |MESSAGES_ACKED |SCHEDULED_COUNT |ROUTING_TYPE |
|DLQ                      |DLQ                      |0              |0             |0              |0
      |0              |0              |ANYCAST      |
|ExpiryQueue              |ExpiryQueue              |0              |0             |0              |0
      |0              |0              |ANYCAST      |
|TEST                     |TEST                     |0              |51            |301            |0
      |250            |0              |ANYCAST      |
|
activemq.management.3dcbfac0-42f6-4772-bfe6-12550f24f042|activemq.management.3dcbfac0-42f6-4772-bfe6-12550f24f0
42|1              |0             |0              |0                |0              |0               |MULTICAST
    |
|myQueue0                 |myAddress0               |0              |0             |0              |0
      |0              |0              |ANYCAST      |
$
```

Installing HA Proxy

This section provides a step-by-step guide to installing HA Proxy.

HA Proxy, which stands for High Availability Proxy, is a software TCP/HTTP Load Balancer and proxying solution. Its most common use is to improve the performance and reliability of a server environment by distributing the workload across multiple servers (e.g. web, application, database).

To install HA proxy, follow the steps below:

1. Add HA proxy repositories to helm configuration.

   ```
   helm repo add haproxytech https://haproxytech.github.io/helm-charts
   helm repo update
   ```

2. Deploy HAProxy using Helm.

   ```
   helm install haproxy-ingress haproxytech/kubernetes-ingress --namespace haproxy --create-namespace
   ```

3. On successful deployment of HAProxy, you can verify the pods.

   ```
   kubectl get pods --namespace haproxy
   ```

Installing Apache Airflow

This section provides a step-by-step guide to installing Apache Airflow.

Apache Airflow (or simply Airflow) is a platform to programmatically author, schedule, and monitor workflows. In Airflow, workflows are defined as code, and they are more maintainable, versionable, testable, and collaborative.

You can use Airflow to author workflows as directed acyclic graphs (DAGs) of tasks. The Airflow scheduler executes tasks on an array of workers while following the specified dependencies. Availability of a rich set of command line utilities help performing complex surgeries on DAGs. The user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed.

To install the Apache Airflow, follow the steps below:

1. Add Apache Airflow repositories to helm configuration.

   ```
   helm repo add apache-airflow https://airflow.apache.org
   helm repo update
   ```

2. Download the Helm charts to local.

   ```
   mkdir -p /home/user/airflow-chart-download
   cd /home/user/airflow-chart-download
   # Download the chart locally,
   # it should download a tarball in the same directory
   helm pull apache-airflow/airflow
   # untar the downloaded tarball
   tar -zxvf airflow-1.14.0.tgz
   ```

3. Update the airflow image with the HCL provided airflow image details in the helm chart inside the values.yaml as below.

```
vi /home/user/airflow-chart-download/airflow/values.yaml
```

Edit below properties

```
defaultAirflowRepository: hclcr.io/unica/airflow-custom
defaultAirflowTag: "1"
airflowVersion: "2.8.3"
```

The properties will look like below:

```
67    # Default airflow repository -- overridden by all the specific images below
68    #defaultAirflowRepository: apache/airflow
69    defaultAirflowRepository: hclcr.io/unica/airflow-custom
70
71    # Default airflow tag to deploy
72    #defaultAirflowTag: "2.8.3"
73    defaultAirflowTag: "1"
74
75    # Default airflow digest. If specified, it takes precedence over tag
76    defaultAirflowDigest: ~
77
78    # Airflow version (Used to make some decisions based on Airflow Version being deployed)
79    airflowVersion: "2.8.3"
```

4. Update the security context in PostgreSQL values.yaml as below to disable seccompProfile.

```
vi /home/user/airflow-chart-download/airflow/charts/postgresql/values.yaml
```

5. Update the OpenShift SCC and grant necessary permissions to all the below listed service accounts to deploy the pods.

```
<your airflow deployment namespace>-create-user-job
if your namespace is airflow, the below all service accounts will be created when you deploy the helm
 chart.
so grant the necessary permission in the SCC for all the service accounts.
airflow-create-user-job
airflow-migrate-database-job
airflow-redis
airflow-scheduler
airflow-statsd
airflow-triggerer
airflow-webserver
airflow-worker
default
```

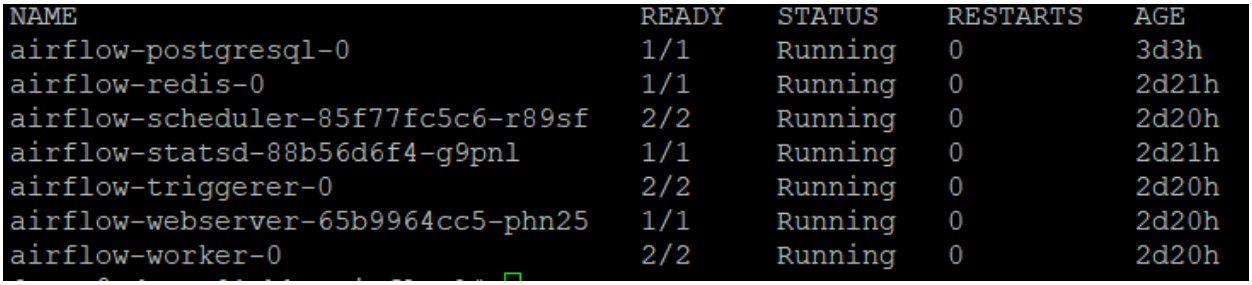
6. Deploy the Apache Airflow using Helm.

```
helm upgrade --install airflow /home/user/airflow-chart-download/airflow --namespace airflow
 --create-namespace --debug --timeout 15m
```

7. After deploying Apache Airflow, verify the deployed pods.

```
kubectl get pods --namespace airflow
```

As a result, the output will be as shown below:



```
NAME                                READY   STATUS    RESTARTS   AGE
airflow-postgresql-0                1/1     Running   0          3d3h
airflow-redis-0                     1/1     Running   0          2d21h
airflow-scheduler-85f77fc5c6-r89sf  2/2     Running   0          2d20h
airflow-statsd-88b56d6f4-g9pnl      1/1     Running   0          2d21h
airflow-triggerer-0                 2/2     Running   0          2d20h
airflow-webserver-65b9964cc5-phn25  1/1     Running   0          2d20h
airflow-worker-0                    2/2     Running   0          2d20h
```

Configuring Apache Airflow

Expose the Airflow web server service, which will create the route to access the Airflow UI.

```
oc expose svc airflow-webserver -n airflow
oc get routes -n airflow
```

Access the airflow UI application using the HOST/PORT URL.

Installing MinIO

This section provides a step-by-step guide to installing MinIO.

MinIO is an object storage solution that provides an Amazon Web Services S3-compatible API and supports all core S3 features. MinIO is built to deploy anywhere - public or private cloud, bare metal infrastructure, orchestrated environments, and edge infrastructure.

To install the MinIO, follow the steps below:

1. Add the MinIO Operator repositories to helm configuration.

   ```
   helm repo add minio-operator https://operator.min.io
   helm repo update
   ```

2. Validate the repositories contents using helm search.

   ```
   helm search repo minio-operator
   ```

   As a result, the output will be as shown below.

   ```
     NAME                            CHART VERSION   APP VERSION     DESCRIPTION
   minio-operator/minio-operator   4.3.7           v4.3.7          A Helm chart for MinIO Operator
   minio-operator/operator         5.0.10          v5.0.10         A Helm chart for MinIO Operator
   minio-operator/tenant           5.0.10          v5.0.10         A Helm chart for MinIO Operator
   ```

3. Download the MinIO Operator Helm chart from github to your local.

   ```
   curl -O https://raw.githubusercontent.com/minio/operator/master/helm-releases/operator-5.0.15.tgz
   ```

4. After downloading the tar file, extract the chart.

   ```
   tar -xvf operator-5.0.15.tgz
   ```

5. Create a minio operator namespace.

```
kubectl create namespace minio-operator --
```

6. Update the values.yaml file within the extracted chart directory, and comment out the seccompProfile values.

```
containerSecurityContext:
 runAsUser: 1000
 runAsGroup: 1000
 runAsNonRoot: true
 allowPrivilegeEscalation: false
 capabilities:
 drop:
 - ALL
 # seccompProfile: # type: RuntimeDefault
```

7. Similarly, update the Ingress details of config and host values.

```
###
# Configures `Ingress <https://kubernetes.io/docs/concepts/services-networking/ingress/>`__ for the
 Operator Console.
#
# Set the keys to conform to the Ingress controller and configuration of your choice.
# Set console.ingress.number to any port. For example:
# You may choose port number 9443 for HTTPS or 9090 for HTTP, as desired.
ingress:
enabled: true #false
ingressClassName: ""
labels: { }
annotations: { }
tls: [ ]
host: console.minio-operator.apps.ocp415.manishkr.nonprod.hclpnp.com #console.local
path: /
pathType: Prefix
number: 9090
###
```

8. Now, navigate to the template folder in the extracted file, and open minio.min.io_tenants.yaml for editing.
9. Remove all the seccompProfile sections, and Install the chart using below command.

```
helm install --namespace minio-operator <chart-name> <chart-path>
```

10. After installation, you can verify the operator.

```
kubectl get all -n minio-operator
```

As a result, the output will be as shown below.

```
NAME                                  READY   STATUS    RESTARTS   AGE
pod/console-68d955874d-vxlzm          1/1     Running   0          25h
pod/minio-operator-699f797b8b-th5bk   1/1     Running   0          25h
pod/minio-operator-699f797b8b-nkrn9   1/1     Running   0          25h

NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE
service/console    ClusterIP   10.43.195.224   <none>        9090/TCP,9443/TCP   25h
service/operator   ClusterIP   10.43.44.204    <none>        4221/TCP            25h
service/sts        ClusterIP   10.43.70.4      <none>        4223/TCP            25h

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/console           1/1     1            1           25h
deployment.apps/minio-operator    2/2     2            2           25h
```

```
NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/console-68d955874d      1         1         1       25h
replicaset.apps/minio-operator-699f797b8b   2         2         2       25h
```

11. Now, retrieve the console access token.

    ```
    kubectl get secret/console-sa-secret -n minio-operator -o json | jq -r ".data.token" | base64 -d
    ```

    As a result, the output will be shown as below:

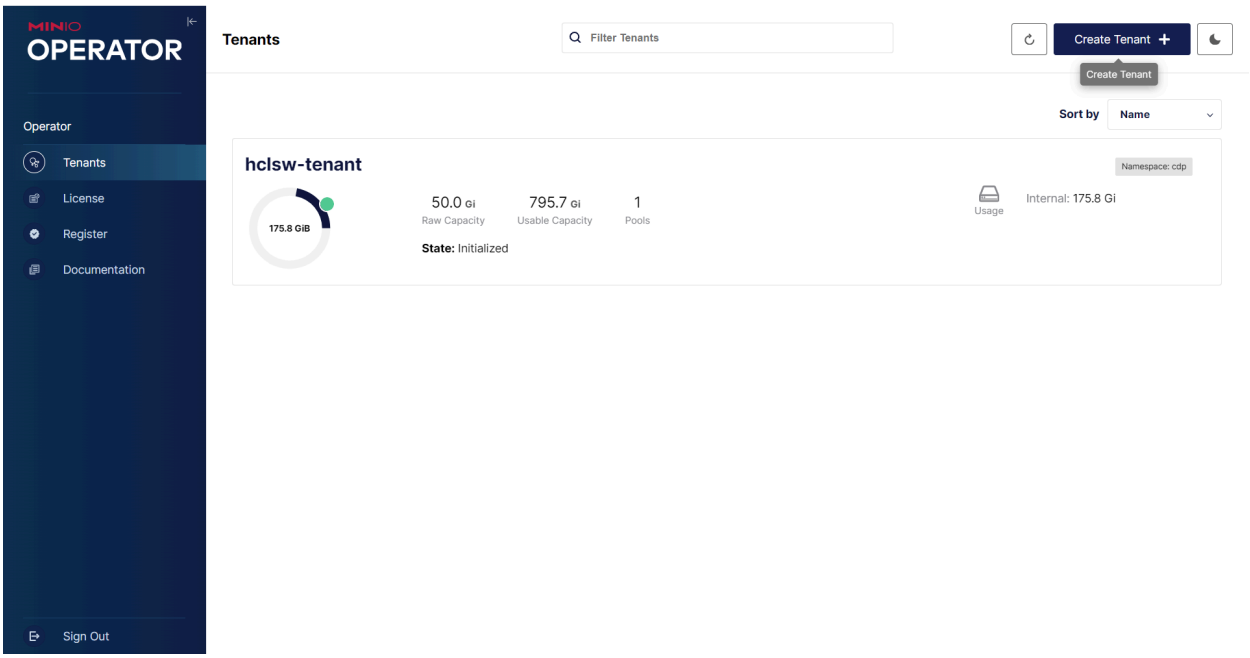
    ```
    eyJhbGciOiJSUzI1NiIsImtpZCI6IlRtV2x3Z1RILVREaThhQm9iemFfLW95NHFHT0ZZOHFBRjlZalBRcWZiSDgifQ.eyJpc3MiOiJrd
    WJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJtaW5pby1vcGVyY
    XRvciIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJjb25zb2xlLXNhLXNlY3JldCIsImt1YmVybmV0Z
    XMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJjb25zb2xlLXNhIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlY
    WNjb3VudC9zZXJ2aWNlLWFjY291bnQudWlkIjoiY2M1ZjEwYzktYzU1ZC00MjNiLTgxM2MtNmU5ZDY2ZGI5NDYyIiwic3ViIjoic3lzd
    GVtOnNlcnZpY2VhY2NvdW50Om1pbmlvLW9wZXJhdG9yOmNvbnNvbGUtc2EifQ.F-Pt5nU9xaugjRksWAOTShBW_eNTf8UwXvLfGxEK6l
    3_41NYsLgvTg5m0hYLUiYr6v2HwkEu0XzqTJbPoeSrFds8BOjeiCoP2Lmw4tRPo9tSXhAq-_elWt83YpJl-zjUpna5nVSWJWXKgj1Iga
    -9gw-Q63UygEcyTJ9_AwCNU9T0HdPzqccS9XrEUdsXFQxR9RwZY4TGC8K7cD9sc_OmfEiuyilRgyC_gFRvtCQfFv1DP0GKyjMGo2ffu-
    2Tq2U7zK5epWdqmNSvbIa0ZRoPlPedZ6nYY935lNgTIIW1oykRYrgwZZiv4CzfTH2gPswjtPc5ICtDDRUjYEhdTq3gtw
    ```

12. If you configured the svc/console service for access through ingress, a cluster load balancer, you can access the console using the configured hostname and port. Once you access the console, use the Console JWT to log in.

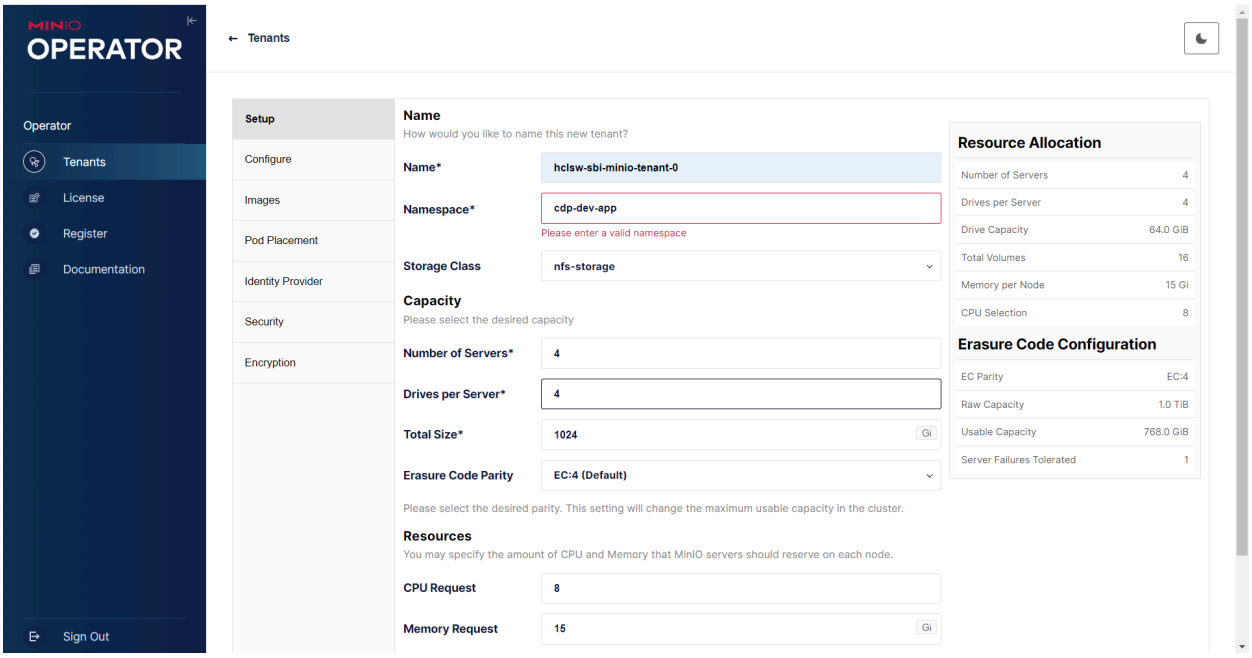
    ```
    oc expose svc/console -n minio-operator
    ```

Creating and configuring Tenant

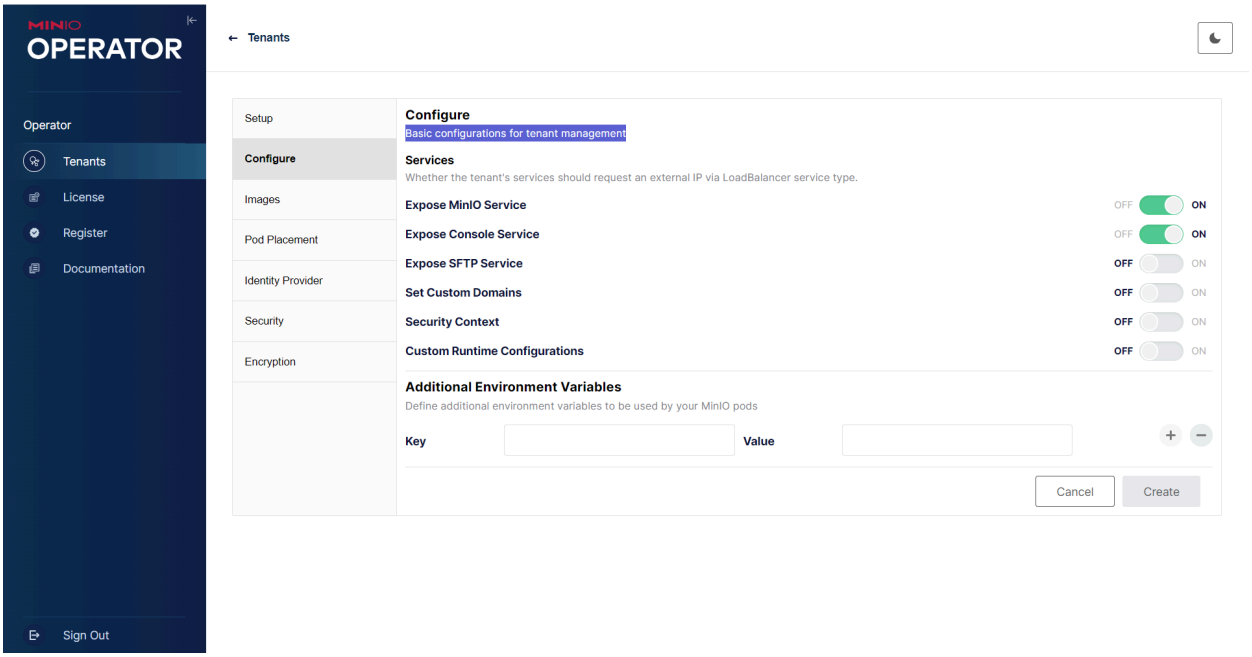
1. In the MinIO portal, select **Tenant**, and click **Create Tenant**.



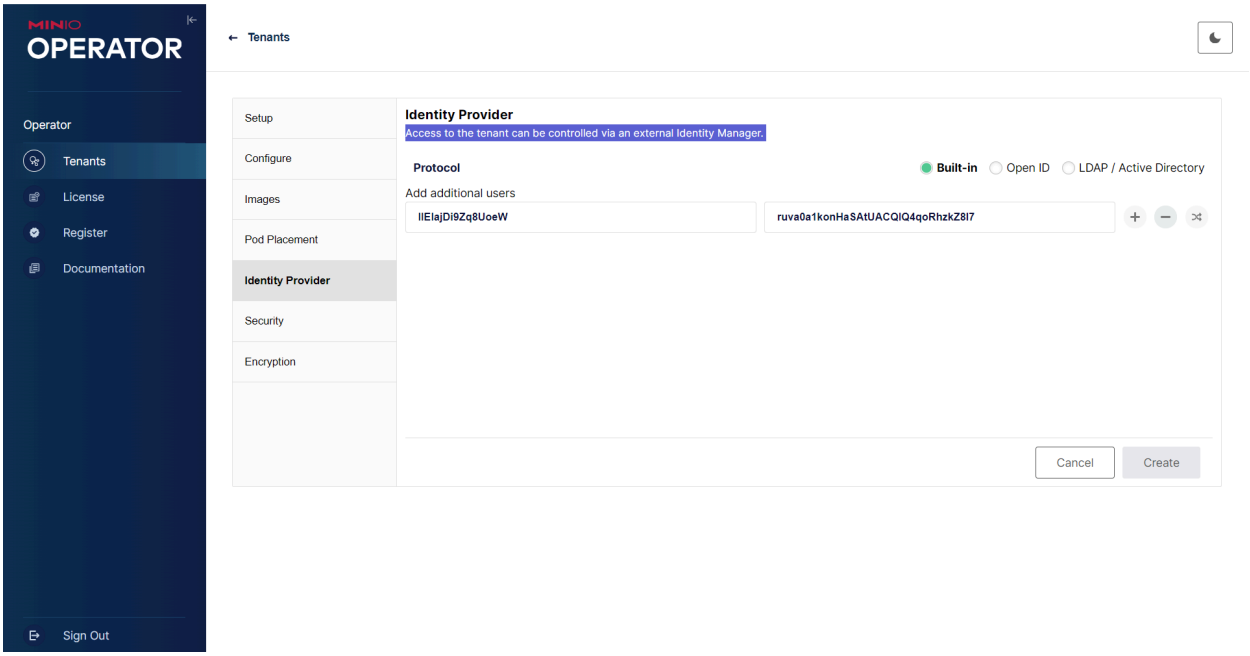
2. In the Tenants page, click **Setup**, and enter tenant name and other details.



3. Now, click **Configure**, and set the basic configurations.



4. Similarly, configure Identity Provider, Security and Encryption.



5. Store the MinIO credentials.json for Tenant Management, and verify the Tenant resources created under specified namespace - cdp.

```
[root@cdpsvc01 ~]# oc get svc -n cdp
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
                AGE
hclsw-tenant-console                LoadBalancer   172.30.90.139   <pending>     9090:32037/TCP
                17d
hclsw-tenant-hl                     ClusterIP      None            <none>        9000/TCP
                17d
minio                               LoadBalancer   172.30.43.239   <pending>     80:32045/TCP
                17d

[root@cdpsvc01 ~]# oc get pods -n cdp  | grep tenant
hclsw-tenant-pool-0-0                              2/2     Running                 0          17d
hclsw-tenant-pool-0-1                              2/2     Running                 0          6d2h

[root@cdpsvc01 ~]# oc get sts -n cdp  | grep tenant
hclsw-tenant-pool-0   2/2     17d
[root@cdpsvc01 ~]#
```

6. Expose MinIO service for bucket creation.

```
[root@cdpsvc01 ~]# oc get routes -n cdp | grep minio
minio minio-cdp.apps.ocp415.manishkr.nonprod.hclpnp.com minio http-minio None
```

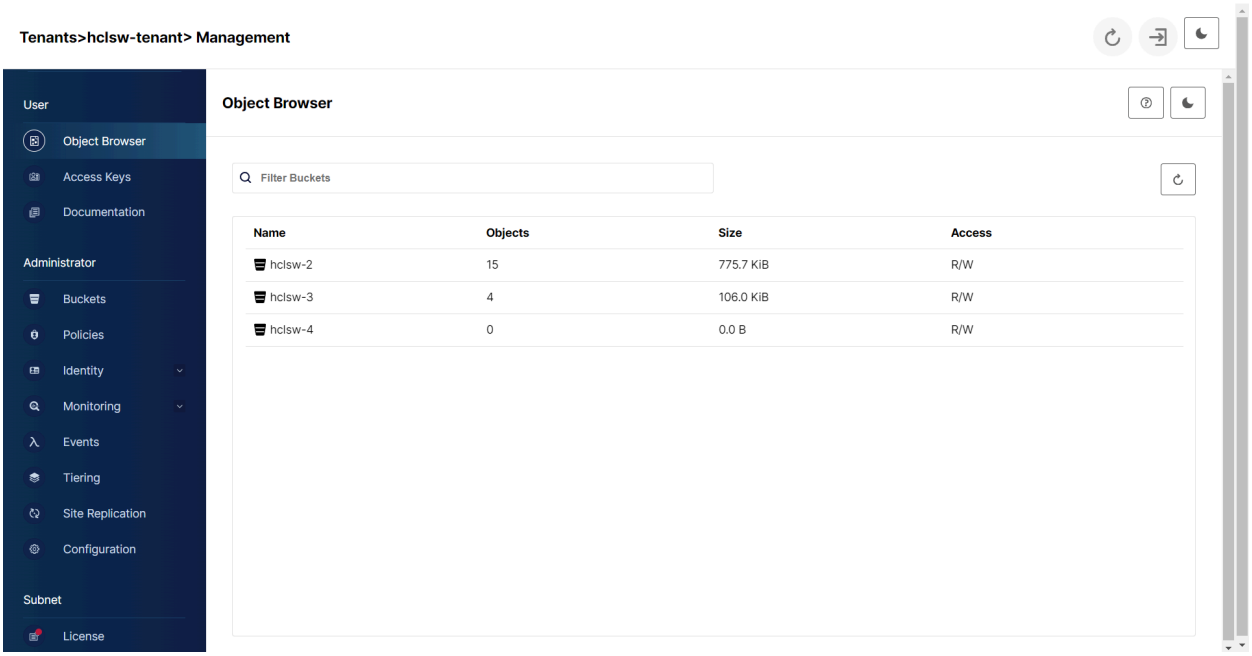
7. Verify MinIO Client commands.

```
sh-5.1$ mc alias set test http://minio-cdp.apps.ocp415.manishkr.nonprod.hclpnp.com  rAtUbaYVYdtkrRBo
 etFo134HCqcPs5yJzWn0WjA6E61vsEHG --api s3v4 --path auto
Added `test` successfully.
sh-5.1$ mc ls test
[2024-06-08 06:49:17 UTC]     0B hclsw-2/
[2024-06-08 07:05:52 UTC]     0B hclsw-3/
[2024-06-08 07:08:51 UTC]     0B hclsw-4/
```

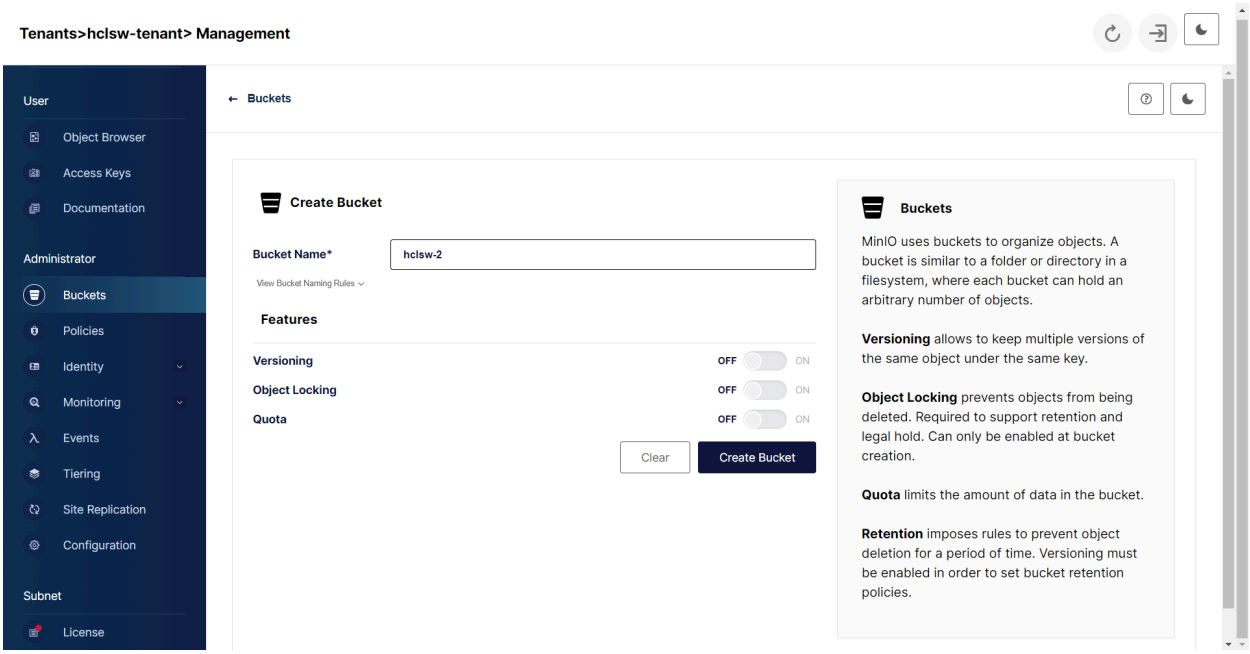
```
sh-5.1$ mc mb test/test-bkt
Bucket created successfully `test/test-bkt`.
sh-5.1$ mc ls test

[2024-06-08 06:49:17 UTC]     0B hclsw-2/
[2024-06-08 07:05:52 UTC]     0B hclsw-3/
[2024-06-08 07:08:51 UTC]     0B hclsw-4/
[2024-06-25 09:55:42 UTC]     0B test-bkt/
```

Creating Bucket in Management Console

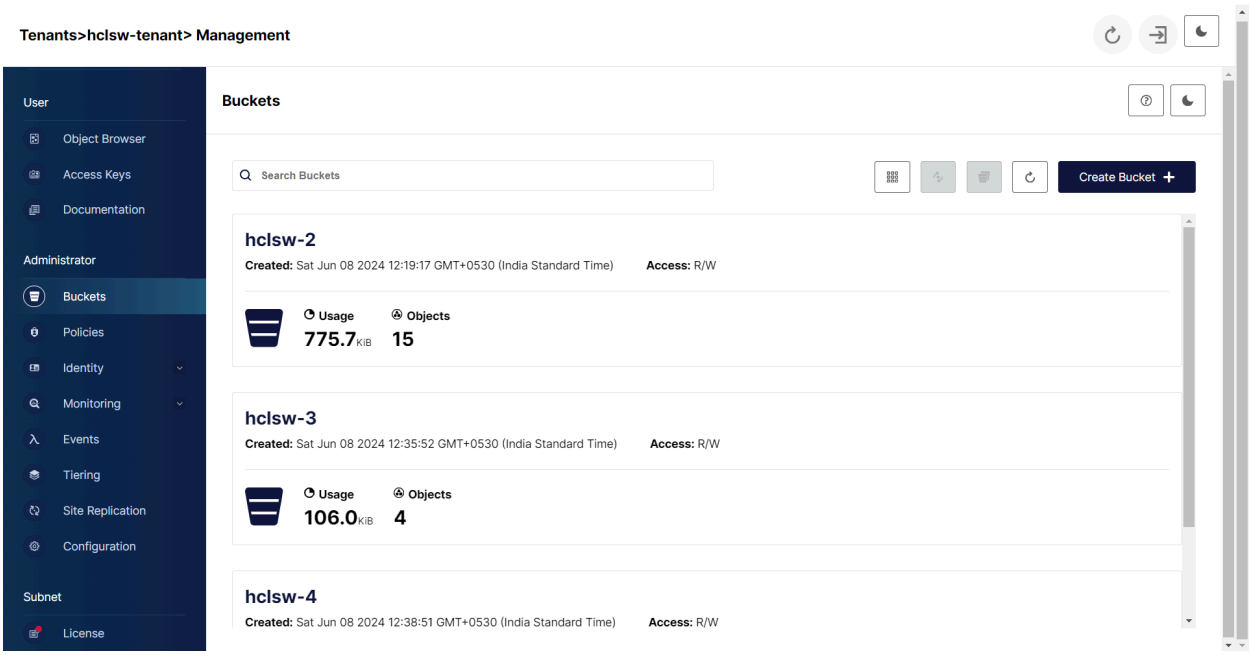
1. Open the Management console using console URL.



2. In the console, Under Administrator section, click Buckets and enter bucket details. Click Create Bucket to create the new bucket.



3. You search for the existing buckets in the search bucket option as shown below.



Chapter 6. List of Components for Native VM

This section lists all the required component of Native VM and their deployment types.

Application/Services	**Deployment Type**
Aerospike	Standard
MongoDB	Standard
Druid	Standard
Mysql	Standard
PgSql	Docker Compose
NIFI	Docker Compose
Redis	VM Based Deployment
RabbitMQ	VM Based Deployment

Installing Aerospike

This section provides a step-by-step guide to installing Aerospike.

To install the aerospike, follow the steps below:

1. Download the latest tarball of the server, and unpack it.

   ```
   wget -O aerospike.tgz
    https://enterprise.aerospike.com/enterprise/download/server/6.3.0/artifact/el8_amd64
   tar -xzf aerospike.tgz
   ```

2. Install the Aerospike package.

   ```
   cd aerospike-server-enterprise_6.3.0.17_tools-8.5.1_el8_x86_64/
   ./asinstall
   ```

3. Update the service in the user.conf file.

   ```
   vi /etc/systemd/system/aerospike.service.d/user.conf
   [Service]
   User=aerospike
   Group=aerospike
   ```

4. Update the instance store drive rules and cleanup the drive.

   ```
   lsblk # check the instance store disck
   file -s /dev/nvme3n1
   vi /etc/udev/rules.d/99-aerospike.rules
   KERNEL=="nvme3n1", OWNER="aerospike"
   udevadm control --reload-rules
   udevadm trigger
   blkdiscard /dev/nvme3n1
   blkdiscard -z --length 8MiB /dev/nvme3n1
   ```

5. Update cluster host and port details, storage in the conf file.

```
cd /etc/aerospike/
vi aerospike.conf # Update the Cluster host/port details and storage (Device path of your actual disk
 path ex: /dev/nvme3n1) as per below example.
```

Note: If nvme3n1 disk is available, add the path of actual mounted disk, as /data.

```
# Aerospike database configuration file for use with systemd.

service {
        cluster-name aerospike
        proto-fd-max 15000
        user aerospike
        group aerospike
        pidfile /var/run/aerospike/asd.pid
        feature-key-file /etc/aerospike/features.conf
        run-as-daemon
}

logging {
        file /var/log/aerospike/aerospike.log {
                context any info
        }
        file /var/log/aerospike/udf.log {
                context udf info
        }
}

network {
        service {
                address puapso1aaxcdaerdbs1001.axcd.aws.hclsw.internal
                port 13000
        }

        heartbeat {
                mode mesh
                port 13001
                address puapso1aaxcdaerdbs1001.axcd.aws.hclsw.internal
                mesh-seed-address-port puapso1baxcdaerdbs1002.axcd.aws.hclsw.internal 13001
                mesh-seed-address-port puapso1aaxcdaerdbs1001.axcd.aws.hclsw.internal 13001
                interval 150
                timeout 100
        }

        fabric {
                address puapso1aaxcdaerdbs1001.axcd.aws.hclsw.internal
                port 13002
        }

        info {
                address puapso1aaxcdaerdbs1001.axcd.aws.hclsw.internal
                port 13003
        }
}

namespace cdpstore {
```

```
        replication-factor 2
        memory-size 12G
        high-water-disk-pct 50
        high-water-memory-pct 75
        stop-writes-pct 90
        default-ttl 31536000
        nsup-period 300
        rack-id 1
        storage-engine device {
                device /dev/nvme3n1
                write-block-size 256K
        }
}
```

6. Update the license details in the features.conf file.

```
vim features.conf
```

Sample features.conf file is shown below.

```
# generated 2023-12-08 18:52:53
feature-key-version 2
serial-number <serial no>
account-name Aerospike
account-ID <account id>
valid-until-date 2024-04-26
asdb-change-notification true
asdb-cluster-nodes-limit 8
asdb-compression true
asdb-encryption-at-rest true
asdb-flash-index true
asdb-ldap true
asdb-pmem true
asdb-rack-aware true
asdb-secrets true
asdb-strong-consistency true
asdb-vault true
asdb-xdr true
database-recovery true
elasticsearch-connector true
graph-service true
mesg-jms-connector true
mesg-kafka-connector true
presto-connector true
pulsar-connector true
spark-connector true
----- SIGNATURE ------------------------------------------------
 MEYCIQDKnSmT9dIagWuUPrrsUH4m20NlQw3G7RoADPFPuhHjZwIhAJBlXCYHYNti
sUrMCvIQodb1H7jRui+tQ8IqVQ1cZGsZJA==
----- END OF SIGNATURE ------------------------------------------
```

7. Perform the following commands to update the configuration.

```
mkdir -p /var/run/aerospike/
mkdir -p /var/log/aerospike/
chown -R aerospike: /var/run/aerospike/
chown -R aerospike: /var/log/aerospike/
chown -R aerospike: /opt/aerospike/
semanage port -l | grep 13000
semanage port -a -t http_port_t -p tcp 13000-13003
```

```
systemctl status aerospike
systemctl start aerospike
systemctl restart aerospike
tail -F /var/log/aerospike/aerospike.log
asadm -p 13000
asadm -p 13000 -e 'info'
asadm -p 13000 -e 'asinfo -v services'
info
show config
asinfo -v services enable
asinfo -v features
nc -zv puapso1baxcdaerdbs1002.axcd.aws.hclsw.internal 13001
```

Installing Node Exporter

To install the Node exporter, follow the steps below:

1. Download the latest tarball of node exporter and unpack it.

```
wget
 https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.0.linux-amd64.ta
r.gz
tar xfz node_exporter-*.*-amd64.tar.gz
sudo mv node_exporter-*.*-amd64/node_exporter /usr/local/bin/
sudo useradd -rs /bin/false node_exporter
```

2. Update the node_exporter.services file as shown below.

```
vi /etc/systemd/system/node_exporter.service
[Unit]
Description=Node Exporter
After=network.target
[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter
[Install]
WantedBy=multi-user.target
```

3. Update the mode and service of the node exporter as shown below.

```
chown -R root: /usr/local/bin/node_exporter
chmod -R 755 /usr/local/bin/node_exporter
restorecon -Rv /usr/local/bin
semanage port -l | grep 9100
systemctl daemon-reload
systemctl start node_exporter
systemctl status node_exporter
systemctl restart aerospike
systemctl enable node_exporter
----------For Debugging only Start---------
/usr/local/bin/node_exporter --help
sudo -H -u node_exporter bash -c "/usr/local/bin/node_exporter"
su -s /bin/bash node_exporter
ps aux | grep node
----------For Debugging only End---------
curl -kv  http://localhost:9100/metrics
```

Installing Aerospike exporter

To install the aerospike exporter, follow the steps below:

1. Download the latest Aerospike exporter, and unpack it.

```
wget
 https://github.com/aerospike/aerospike-prometheus-exporter/releases/download/v1.16.0/aerospike-promethe
us-exporter-1.16.0-1.el8.x86_64.rpm
rpm -Uvh aerospike-prometheus-exporter-1.16.0-1.el8.x86_64.rpm
```

2. Update the aerospike-prometheus-exporter.service files. Use the following command to open the file in editor.

```
vim /usr/lib/systemd/system/aerospike-prometheus-exporter.service.
```

```
[Unit]
Description=Aerospike Prometheus Exporter Service
Documentation=https://github.com/aerospike/aerospike-prometheus-exporter
Wants=network.target
After=network-online.target

[Service]
ExecStart=/usr/bin/aerospike-prometheus-exporter --config /etc/aerospike-prometheus-exporter/ape.toml

[Install]
WantedBy=multi-user.target
```

```
vi /etc/systemd/system/multi-user.target.wants/aerospike-prometheus-exporter.service
```

```
[Unit]
Description=Aerospike Prometheus Exporter Service
Documentation=https://github.com/aerospike/aerospike-prometheus-exporter
Wants=network.target
After=network-online.target
[Service]
ExecStart=/usr/bin/aerospike-prometheus-exporter --config /etc/aerospike-prometheus-exporter/ape.toml
[Install]
WantedBy=multi-user.target
```

```
vim /etc/aerospike-prometheus-exporter/ape.toml
```

The default ape.toml is shown below.

```
[Agent]
      # Metrics Serving modes
      PROMETHEUS = true
      OPEN_TELEMETRY = false

      # labels to add to the prometheus metrics for e.g. labels={zone="asia-south1-a", platform="google
 compute engine"}
      labels = {latitude="0",longitude="0"}

      # mention cloud provider (supported: aws, gcp, azure ) so exporter collects few details like
 region, zone etc.,
      cloud_provider = ""

      # metrics server timeout in seconds
      timeout = 10
```

```
     # support system statistics also
     refresh_system_stats = false

     # Exporter logging configuration
     # Log file path (optional, logs to console by default)
     # Level can be info|warning,warn|error,err|debug|trace ('info' by default)
     log_file = ""
     log_level = ""

     # Exporter HTTPS (TLS) configuration
     # HTTPS between Prometheus and Exporter

     # TLS certificates.
     # Supports below formats,
     # 1. Certificate file path                                        - "file:<file-path>"
     # 2. Environment variable containing base64 encoded certificate -
"env-b64:<environment-variable-that-contains-base64-encoded-certificate>"
     # 3. Base64 encoded certificate                                   -
"b64:<base64-encoded-certificate>"
     # Applicable to 'root_ca', 'cert_file' and 'key_file' configurations.

     # Server certificate
     cert_file = ""

     # Private key associated with server certificate
     key_file = ""

     # Root CA to validate client certificates (for mutual TLS)
     root_ca = ""

     # Passphrase for encrypted key_file. Supports below formats,
     # 1. Passphrase directly                                                    - "<passphrase>"
     # 2. Passphrase via file                                                    -
"file:<file-that-contains-passphrase>"
     # 3. Passphrase via environment variable                                    -
"env:<environment-variable-that-holds-passphrase>"
     # 4. Passphrase via environment variable containing base64 encoded passphrase -
"env-b64:<environment-variable-that-contains-base64-encoded-passphrase>"
     # 5. Passphrase in base64 encoded form                                      -
"b64:<base64-encoded-passphrase>"
     key_file_passphrase = ""

     # prometheus binding port
     bind = ":9145"

     # Basic HTTP authentication for '/metrics'.
     # Supports below formats,
     # 1. Credential directly                                                    - "<credential>"
     # 2. Credential via file                                                    -
"file:<file-that-contains-credential>"
     # 3. Credential via environment variable                                    -
"env:<environment-variable-that-contains-credential>"
     # 4. Credential via environment variable containing base64 encoded credential -
"env-b64:<environment-variable-that-contains-base64-encoded-credential>"
     # 5. Credential in base64 encoded form                                      -
"b64:<base64-encoded-credential>"
     basic_auth_username = ""
```

```
      basic_auth_password = ""

[Agent.OpenTelemetry]
      # NOTE: currently supports only gRPC endpoints

      # OTel service-name
      service_name = "aerospike-server-metrics-service"

      # OTel Endpoint
      endpoint = ""

      # OTel SSL/TLS, for HTTPS endpoints
      endpoint_tls_enabled = true

      # OTel headers
      headers = {}

      # OTel server-stat fetch interval (default 15, not recommended to to reduce this)
      server_stat_fetch_interval = 15

      # OTel metric push interval (default 60, not recommended to to reduce this)
      push_interval = 60

[Aerospike]
      db_host = "localhost"
      db_port = 13000

      # TLS certificates.
      # Supports below formats,
      # 1. Certificate file path                                        - "file:<file-path>"
      # 2. Environment variable containing base64 encoded certificate -
 "env-b64:<environment-variable-that-contains-base64-encoded-certificate>"
      # 3. Base64 encoded certificate                                   -
 "b64:<base64-encoded-certificate>"
      # Applicable to 'root_ca', 'cert_file' and 'key_file' configurations.

      # root certificate file
      root_ca = ""

      # certificate file
      cert_file = ""

      # key file
      key_file = ""

      # Passphrase for encrypted key_file. Supports below formats,
      # 1. Passphrase directly                                                          - "<passphrase>"
      # 2. Passphrase via file                                                          -
 "file:<file-that-contains-passphrase>"
      # 3. Passphrase via environment variable                                          -
 "env:<environment-variable-that-holds-passphrase>"
      # 4. Passphrase via environment variable containing base64 encoded passphrase -
 "env-b64:<environment-variable-that-contains-base64-encoded-passphrase>"
      # 5. Passphrase in base64 encoded form                                            -
 "b64:<base64-encoded-passphrase>"
      key_file_passphrase = ""
```

```
     # node TLS name for authentication
     node_tls_name = ""

     # Aerospike cluster security credentials.
     # Supports below formats,
     # 1. Credential directly                                                    - "<credential>"
     # 2. Credential via file                                                    -
"file:<file-that-contains-credential>"
     # 3. Credential via environment variable                                    -
"env:<environment-variable-that-contains-credential>"
     # 4. Credential via environment variable containing base64 encoded credential -
"env-b64:<environment-variable-that-contains-base64-encoded-credential>"
     # 5. Credential in base64 encoded form                                      -
"b64:<base64-encoded-credential>"
     # Applicable to 'user' and 'password' configurations.

     # database user
     user = ""

     # database password
     password = ""

     # authentication mode: internal (server authentication) [default], external (e.g., LDAP), pki.
     auth_mode = ""

     # timeout for sending commands to the server node in seconds
     timeout = 5

     # Number of histogram buckets to export for latency metrics. Bucket thresholds range from 2^0 to
2^16 (17 buckets).
     # e.g. latency_buckets_count=5 will export first five buckets i.e. <=1ms, <=2ms, <=4ms, <=8ms and
<=16ms.
     # Default: 0 (export all threshold buckets).
     latency_buckets_count = 0

     # Order of context - namespace, set, latencies, node-stats, xdr, user, jobs, sindex

     # Metrics Allowlist - If specified, only these metrics will be scraped. An empty list will exclude
all metrics.
     # Commenting out the below allowlist configs will disable metrics filtering (i.e. all metrics will
be scraped).

     # Metrics Blocklist - If specified, these metrics will be NOT be scraped. An empty list will
include all metrics.
     # Commenting out the below blocklist configs will disable metrics filtering (i.e. no metrics will
be blocked/filtered).

     # globbing pattern (wildcard) are allowd for allowlist and blocklist
     # for example "batch_index_*_buffers"

     # Namespace metrics allowlist, to control which Namespace metrics should be collected.
     # namespace_metrics_allowlist = []
     # namespace_metrics_blocklist = []

     # set_metrics_allowlist = []
     # set_metrics_blocklist = []
```

```
      # Latencies histogram allowlist, to control which Histogram-name metrics should be collected.
      # Note: globbing patterns (wildcard) are not supported for this latency metric configuration.
      # latencies_metrics_allowlist = []
      # latencies_metrics_blocklist = []

      # node_metrics_allowlist = []
      # node_metrics_blocklist = []

      # Support only from Aerospike versions 5.0 and above
      # xdr_metrics_allowlist = []
      # xdr_metrics_blocklist = []

      # User statistics are available in Aerospike 5.6+
      # Note globbing patterns (wildcard) are not supported for this User configuration.
      # user_metrics_users_allowlist = []
      # user_metrics_users_blocklist = []

      # sindex_metrics_allowlist = []
      # sindex_metrics_blocklist = []

namespace_metrics_allowlist=[
"client_read_[a-z]*",
"stop_writes",
"storage-engine.file.defrag_q",
"client_write_success",
"memory_*_bytes",
"objects",
"*_available_pct"
]

# Set metrics allowlist
set_metrics_allowlist=[
"objects",
"tombstones"
]

# Node metrics allowlist
node_metrics_allowlist=[
"uptime",
"cluster_size",
"batch_index_*",
"xdr_ship_*"
]

# XDR metrics allowlist (only for Aerospike versions 5.0 and above)
xdr_metrics_allowlist=[
"success",
"latency_ms",
"throughput",
"lap_us"
]

# Job (scans/queries) metrics allowlist
job_metrics_allowlist = [
"rps",
"active-threads",
"job-progress",
```

```
"run-time",
"recs-throttled",
"recs-succeeded",
"recs-failed",
"net-io-bytes"
]

# Secondary index metrics allowlist
sindex_metrics_allowlist = [
"entries",
"ibtr_memory_used",
"nbtr_memory_used",
"query_basic_complete",
"query_basic_error",
"query_basic_abort",
"query_basic_avg_rec_count"
]

# Metrics Blocklist - If specified, these metrics will be NOT be scraped.

# Namespace metrics blocklist
namespace_metrics_blocklist=[
"memory_used_sindex_bytes",
"client_read_success"
]

# Set metrics blocklist
# set_metrics_blocklist=[]

# Node metrics blocklist
node_metrics_blocklist=[
"batch_index_*_buffers"
]
```

3. Start the aerospike-prometheus-exporter services.

```
systemctl daemon-reaload
systemctl start aerospike-prometheus-exporter
systemctl status aerospike-prometheus-exporter
ps aux | grep expo
journalctl -u aerospike-prometheus-exporter
/usr/bin/aerospike-prometheus-exporter --help
semanage port -l | grep 9145
systemctl enable aerospike-prometheus-exporter
systemctl status aerospike-prometheus-exporter
curl -kv  http://localhost:9145/metrics
```

Installing MongoDB

This section provides a step-by-step guide to installing MongoDB.

Prerequisites

Make sure the following things are in place, before you start installation:

- Create a mount point on the instance volume (nvme).
- Attach the additional SSD to the newly created mount point, so that the initial nvme volume size can be expanded.

To install the MongoDB, follow the steps below:

1. Edit the /etc/yum.repos.d/mongodb-org-6.0.repo with the following content.

```
vi /etc/yum.repos.d/mongodb-org-6.0.repo
[mongodb-org-6.0]
name=MongoDB Repository baseurl= https://repo.mongodb.org/yum/redhat/8/mongodb-org/6.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://pgp.mongodb.com/server-6.0.asc
```

2. Install the MongoDB.

```
yum install -y mongodb-org
```

3. Update the yum.conf with bind IP address, port number, storage db path.

```
echo
 "exclude=mongodb-org,mongodb-org-database,mongodb-org-server,mongodb-mongosh,mongodb-org-mongos,mongo
db-org-tools" >> /etc/yum.conf
sed -i "s|OPTIONS=-f /etc/mongod.conf|OPTIONS=--replSet cdprsg
 -f /etc/mongod.conf" /usr/lib/systemd/system/mongod.service
 vi /etc/mongod.conf
```

   Use the following the mongodb.conf file.

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /sdd
  journal:
    enabled: true
#  engine:
#  wiredTiger:

# network interfaces
net:
  port: 51090
  bindIp: puapso1ahxcdmgodbs1001.hxcd.aws.hclsw.internal

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

security:
```

```
  authorization: disabled

operationProfiling:
  mode: all
  slowOpThresholdMs: 200

#replication:
#  replSetName: "cdprsg"

#sharding:

## Enterprise-Only Options:

#auditLog:

#snmp:
```

4. Now, change the file ownership and reload deamon usng the following commands.

```
chown -R mongod: /sdd
chcon -R --reference=/var/lib/mongo /sdd
semanage port -a -t mongod_port_t -p tcp 51090
systemctl daemon-reload
systemctl restart mongod
systemctl status mongod
```

5. Validate the connectivity with other nodes.

```
nc -zv puapso1bhxcdmgodbs1002.hxcd.aws.hclsw.internal 5190
nc -zv puapso1ahxcdmgodbs1003.hxcd.aws.hclsw.internal 5190
nc -zv puapso1bhxcdmgodbs1004.hxcd.aws.hclsw.internal 5190
nc -zv puapso1ahxcdmgodbs1005 5190
```

6. Connect the MongoDB cluster with the specified hostname.

```
mongosh --host puapso1ahxcdmgodbs1001.axcd.aws.hclsw.internal --port 51090
```

7. Configure replica set and initiate it.

```
use admin;
conf={ _id: "cdprsg", members: [ { _id: 0, host: "puapso1ahxcdmgodbs1001.hxcd.aws.hclsw.internal:51090",
 tags: {usage: "schedule"} }, { _id: 1, host: "puapso1bhxcdmgodbs1002.hxcd.aws.hclsw.internal:51090",
 tags: {usage: "schedule" }}, { _id: 2, host: "puapso1ahxcdmgodbs1003.hxcd.aws.hclsw.internal:51090",
 tags: {usage: "segloader" }}, { _id: 3, host: "puapso1bhxcdmgodbs1004.hxcd.aws.hclsw.internal:51090",
 tags: {usage: "analytics" }}] }
rs.initiate(conf);
rs.addArb("puapso1ahxcdmgodbs1005:51090");
conf = rs.conf()
conf.members[3].hidden = true
rs.status()
```

8. Create three users in the admin database with specific roles and privileges.

```
use admin;
db.createUser( { user: "rtsms", pwd: "<password>", roles: [{ role: "readWrite", db: "segmentation" },
 { role: 'dbAdmin', db: 'segmentation' }] });
```

```
db.createUser( { user: "segdbconnector", pwd: "<password>", roles: [{ role: "readWrite", db:
 "segmentation" }] });
db.createUser( { user: "pmm", pwd: "Vism6lUIkw5O6BX7fxJB9w", roles: [ { role: 'backup', db: 'admin' },
 { role: 'clusterMonitor', db: 'admin' }, { role: 'read', db: 'local' }, { role: 'readWrite', db:
 'admin' }, { role: 'restore', db: 'admin' }] });
```

Installing Node Exporter

To install the node exporter, follow the steps below:

1. Download the latest node exporter package, and extract the package.

   ```
   wget
    https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.0.linux-amd64.ta
   r.gz
   tar xfz node_exporter-*.*-amd64.tar.gz
   ```

2. Move the extracted binary from the extracted directory to the local bin location.

   ```
   sudo mv node_exporter-*.*-amd64/node_exporter /usr/local/bin/
   ```

3. Optionally, create a user.

   ```
   sudo useradd -rs /bin/false node_exporter
   ```

4. Update the service file as shown below:

   ```
   vi /etc/systemd/system/node_exporter.service
   [Unit]
   Description=Node Exporter
   After=network.target
   [Service]
   User=node_exporter
   Group=node_exporter
   Type=simple
   ExecStart=/usr/local/bin/node_exporter
   [Install]
   WantedBy=multi-user.target
   ```

5. Reload Daemon to ensure the Node Exporter service is properly configured and running.

   ```
   restorecon -Rv /usr/local/bin
   systemctl daemon-reload
   systemctl start node_exporter
   systemctl status node_exporter
   systemctl enable node_exporter
   curl -kv  http://localhost:9100/metrics
   ```

Installing MongoDB Exporter

To install the MongoDB exporter, follow the steps below:

1. Download the MongoDB exporter, a tool for collecting metrics about your MongoDB cluster.

   ```
   wget
    https://github.com/percona/mongodb_exporter/releases/download/v0.40.0/mongodb_exporter-0.40.0.linux-
   64-bit.rpm
   ```

2. Install the MongoDB package.

```
rpm -Uhv mongodb_exporter-0.40.0.linux-64-bit.rpm
```

3. Update the MongoDB exporter service file.

```
vi /etc/systemd/system/mongodb_exporter.service
[Unit]
Description=Prometheus MongoDB Exporter
Documentation= https://github.com/percona/mongodb_exporter
After=network.target
[Service]
Type=simple
User=mongodb_exporter
Group=mongodb_exporter
 Environment="OPTIONS=--mongodb.uri=mongodb://
pmm:Wrtu9v0UGaqgohsYePtiMQ==@puapso1ahxcdmgodbs1001.hxcd.aws.hclsw.internal:51090 --compatible-mode"
EnvironmentFile=-/etc/default/mongodb_exporter
ExecStart=/usr/bin/mongodb_exporter $OPTIONS
SyslogIdentifier=mongodb_exporter
Restart=always
[Install]
WantedBy=multi-user.target
```

4. Reload Daemon to ensure the MongoDB exporter service is properly configured and running.

```
systemctl daemon-reload
systemctl start mongodb_exporter
systemctl status mongodb_exporter
systemctl enable mongodb_exporter
curl -kv  http://localhost:9216/metrics
```

Installing Druid

This section provides a step-by-step guide to installing Druid.

Pre-requisites

Before installing Druid, make sure the following things are in place:

1. Create directories and place the respective files provided as a release artifact, in the specified locations.
    a. create /home/druid/allmetrics directory, and place all_metrics.sh and allmetrics_airflow.json files inside the folder.
    b. create /home/druid/cdpdashboard/cdpleadconv directory, and place cdp_leadconv.sh and cdp_leadconv.json files inside the folder.
    c. create /home/druid/cdpdashboard/cdpuniqueusers directory, and place cdp_uniqueusers.sh and cdp_uniqueusers.json files inside the folder.
    d. create /home/druid/cdpdashboard/events directory, and place events_druid_with_python.sh and events_v1.json files inside the folder.
    e. create /home/druid/daily directory, and place uploadmysqlmappingfiles.sh files inside the folder.
2. Install mc client. Follow the steps below to install mc client.
    a. Download the mc client, and update the execute permissions using below command.

```
curl -fsSLhttps://dl.min.io/client/mc/release/linux-amd64/mc > /usr/bin/mc && chmod
 +x /usr/bin/mc
```

b. Navigate vi /etc/profile, and update the mc path in the system profile file using the below information.

```
export PATH="/usr/bin/mc:$PATH"
```

c. Verify the version of the mc client using below command.

```
mc --version
```

Setting up MySQL

To set up MySQL, follow the steps below:

1. Install the MySQL server.

```
yum install mysql-server
```

2. Start the MySQL server.

```
systemctl start mysqld
```

3. On successful starting up of the server, check for the status.

```
systemctl status mysqld
mysql -u root
```

4. Create a user for the MySQL server.

```
CREATE USER 'druid'@'localhost' IDENTIFIED BY 'Oc3f4EagD5cUnZnUy5uMQ==';
CREATE DATABASE druid DEFAULT CHARACTER SET utf8mb4;
```

5. Grant comprehensive privileges on the druid database to the user.

```
GRANT CREATE, ALTER, DROP, INSERT, UPDATE, INDEX, DELETE, SELECT, REFERENCES ON druid.* TO
 'druid'@'localhost' WITH GRANT OPTION;FLUSH PRIVILEGES;
```

Installing Druid and MySQL connector

To install Druid binary and MySQL connector, follow the steps below:

1. Download the latest package of Apache Druid binary.

```
wget https://archive.apache.org/dist/druid/25.0.0/apache-druid-25.0.0-bin.tar.gz
```

2. Extract the tar package, and move the directory to the local folder.

```
tar -xzf apache-druid-28.0.1-bin.tar.gz
mv apache-druid-25.0.0 /sdd/
```

3. Download the MySQL Connector/J library (version 5.1.49) to the current working directory.

```
wget https://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.49/mysql-connector-java-5.1.49.jar
```

4. Copy the jar file to the local directory as shown below.

```
cp mysql-connector-java-5.1.49.jar /sdd/apache-druid-25.0.0/extensions/mysql-metadata-storage/
```

Updating Druid Configuration

To update the druid configuration, follow the steps below:

1. Open common.runtime.properties files in text editor, and update mysql-metadata-storage and druid-s3-extensions in druid.extensions.loadList along with other extensions section.

```
vi /sdd/apache-druid-25.0.0/conf/druid/single-server/small/_common/common.runtime.properties
druid.extensions.loadList=["druid-hdfs-storage", "druid-kafka-indexing-service", "druid-datasketches",
 "druid-multi-stage-query","mysql-metadata-storage", "druid-s3-extensions"]
```

2. Comment the Derby configurations, and update the following properties in common.runtime.properties.

```
druid.metadata.storage.type=mysql
 druid.metadata.storage.connector.connectURI=jdbc:mysql://
<host>/druid?allowPublicKeyRetrieval=true&useSSL=false
druid.metadata.storage.connector.user=druid
druid.metadata.storage.connector.password=<password>
```

3. Update the following to store segments in S3.

```
druid.storage.type=s3
druid.storage.bucket=druid-bucket
druid.storage.baseKey=druid/segments
```

4. Add the druid.storage.disableAcl=true at the end of the file.

```
nohup ./start-single-server-small &
ps -eaf | grep supervise
```

Installing MySQL

This section provides a step-by-step guide to installing MySQL.

To install MySQL and create a database, follow the steps below:

1. Install the MySQL server.

```
yum install mysql-server
```

2. Enable the MySQL server.

```
systemctl enable mysqld
```

3. Connect a MySQL database server as the root user.

```
mysql -u root
```

4. Create a database named vrm in MySQL.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| vrm                |
+--------------------+
5 rows in set (0.00 sec)
```

5. Create three users as admin, cdpapp and cdpapp_ro users along with permissions as shown below.

```
CREATE USER 'admin'@'%' IDENTIFIED BY '<password>'
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%' WITH GRANT OPTION;
CREATE USER 'cdpapp_rw'@'%' IDENTIFIED BY '<password>'
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, RELOAD, REFERENCES, INDEX ON *.* TO `cdpapp_rw`@`%` WITH
 GRANT OPTION;
CREATE USER 'cdpapp_ro'@'%' IDENTIFIED BY '<password>'
GRANT SELECT ON *.* TO `cdpapp_ro`@`%` WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

6. Finally, copy the initial data dump in the database.

```
# mysql -u <user> -p -P <Port> -D vrm < <path_to_dumpfile> eg. vrm_schema_only_dump.sql
# mysql -u <user> -p -P <Port>-D vrm < <path_to_dumpfile> eg. db_dump.sql
```

Installing Nifi

This section provides a step-by-step guide to installing Nifi.

To install Nifi, follow the steps below:

1. Mount an external file system disk to a VM at /data.

```
mount external file system disk to the VM at /data
```

2. Perform docker installation with below commands.

```
yum update -y
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
 --allowerasing
wget https://github.com/opencontainers/runc/releases/download/v1.1.12/runc.amd64 -O /usr/local/sbin/runc
cp /usr/local/sbin/runc /usr/bin/runc
systemctl start docker
curl -L "https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-$(uname -s)-$(uname
 -m)" -o /usr/local/bin/docker-compose
mv /usr/local/bin/docker-compose /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose
systemctl status docker
ln -s /data /disk1
mkdir -p /disk1/nifi
mkdir -p /disk1/nifi/data
mkdir -p /disk1/nifi/conf
mkdir -p /disk1/nifi/logs
Donwload the postgresql-42.6.0.jar file and copy into /disk1/nifi
cd /disk1/nifi; wget  https://jdbc.postgresql.org/download/postgresql-42.6.0.jar
```

3. Create a docker compose file.

```
vi /home/user/postgres/docker-compose.yaml
```

```
version: "3"
services:
zookeeper:
hostname: zookeeper
container_name: zookeeper
image: bitnami/zookeeper:3.9.1
restart: always
environment:
```

```
- ALLOW_ANONYMOUS_LOGIN=yes
networks:
- nifinet
nifi00:
image: apache/nifi:1.24.0
container_name: nifi00
restart: always
ports:
- 8080:8080
volumes:
- /disk1/nifi/custom_processors:/opt/nifi/nifi-current/extensions
- /disk1/nifi/postgresql-42.6.0.jar:/opt/nifi/nifi-current/lib/postgresql-42.6.0.jar
- nifi-conf:/opt/nifi/nifi-current/conf
- nifi-logs:/opt/nifi/nifi-current/logs
networks:
- nifinet
environment:
- NIFI_WEB_HTTP_PORT=8080
- NIFI_CLUSTER_IS_NODE=true
- NIFI_CLUSTER_NODE_PROTOCOL_PORT=8082
- NIFI_ZK_CONNECT_STRING=zookeeper:2181
- NIFI_ELECTION_MAX_WAIT=1 min
- NIFI_SENSITIVE_PROPS_KEY=bf4xSLVSAmtex/qtcP5uMTbCrxaP+8q5WjELaYXTkkQ=
- JVM_ARGS=-XX:MaxDirectMemorySize=1GB -Xms512m -Xmx1g
networks:
nifinet:
driver: bridge
volumes:
nifi-data:
driver: local
driver_opts:
type: none
o: bind
device: /disk1/nifi/data
nifi-conf:
driver: local
driver_opts:
type: none
o: bind
device: /disk1/nifi/conf
nifi-logs:
driver: local
driver_opts:
type: none
o: bind
device: /disk1/nifi/logs
```

4. Deploy docker containers using a compose file and validate the containers are running.

```
cd /home/user/postgres
docker-compose up -d
docker ps
on inside postgres container run the below command
psql -U cdpadmin -W -d analytics -h localhost -p 5432
```

Installing PostgreSQL

This section provides a step-by-step guide to installing PostgreSQL.

Installing PostgreSQL on Master VM

To install PostgreSQL on master VM, follow the steps below:

1. Mount an external file system disk to a VM at /data.

```
mount external file system disk to the VM at /data
```

2. Perform docker installation with below commands.

```
yum update -y
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
 --allowerasing
wget https://github.com/opencontainers/runc/releases/download/v1.1.12/runc.amd64 -O /usr/local/sbin/runc
systemctl start docker
curl -L "https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-$(uname -s)-$(uname
 -m)" -o /usr/local/bin/docker-compose
mv /usr/local/bin/docker-compose /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose
systemctl status docker
ln -s /data /disk1
mkdir -p /disk1/postgress
mkdir -p /disk1/postgress/data
```

3. Create a docker compose file as shown below.

```
- vi /home/user/postgres/docker-compose.yaml
```

```
version: "3"
services:
postgres:
image: postgres:15.3
restart: always
environment:
POSTGRES_USER: cdpadmin
POSTGRES_PASSWORD: ugGShnko8j5rNbsoA245Iw
POSTGRES_DB: analytics
volumes:
- db-data:/var/lib/postgresql/data
ports:
- 5432:5432
shm_size: 1gb
postgres-exporter:
image: prometheuscommunity/postgres-exporter
ports:
- 9187:9187
environment:
DATA_SOURCE_NAME:
 "postgresql://cdpadmin:ugGShnko8j5rNbsoA245Iw@10.214.101.138:5432/analytics?sslmode=disable"
links:
- postgres
volumes:
db-data:
driver: local
driver_opts:
type: none
```

```
o: bind
device: /disk1/postgress/data
```

4. Deploy docker containers using a compose file and validate the containers are running.

```
cd /home/user/postgres
docker-compose up -d
docker ps
on inside postgres container run the below command
psql -U cdpadmin -W -d analytics -h localhost -p 5432
```

Installing PostgreSQL on Slave VM

To install PostgreSQL on slave VM, follow the steps below:

1. Mount an external file system disk to a VM at /data.

```
mount external file system disk to the VM at /data
```

2. Perform docker installation with below commands.

```
yum update -y
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
 --allowerasing
wget https://github.com/opencontainers/runc/releases/download/v1.1.12/runc.amd64 -O /usr/local/sbin/runc
systemctl start docker
curl -L "https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-$(uname -s)-$(uname
 -m)" -o /usr/local/bin/docker-compose
mv /usr/local/bin/docker-compose /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose
systemctl status docker
ln -s /data /disk1
mkdir -p /disk1/postgress
mkdir -p /disk1/postgress/data
```

3. Create a docker compose file.

```
- vi /home/user/postgress/docker-compose.yaml
```

```
version: "3"
services:
postgres:
image: postgres:15.3
restart: always
environment:
POSTGRES_USER: cdpadmin
POSTGRES_PASSWORD: wOH3S62YxZ7Y3E7PGdQwbA
POSTGRES_DB: analytics
volumes:
- db-data:/var/lib/postgresql/data
ports:
- 5432:5432
shm_size: 58gb
postgres-exporter:
image: prometheuscommunity/postgres-exporter
ports:
- 9187:9187
environment:
```

```
DATA_SOURCE_NAME:
 "postgresql://cdpadmin:wOH3S62YxZ7Y3E7PGdQwbA@10.214.103.150:5432/analytics?sslmode=disable"
links:
- postgres
volumes:
db-data:
driver: local
driver_opts:
type: none
o: bind
device: /disk1/postgress/data
```

4. Deploy docker containers using a compose file and validate the containers are running.

```
cd /home/user/postgress
docker-compose up -d
docker ps
on inside postgres container run the below command
psql -U cdpadmin -W -d analytics -h localhost -p 5432
```

Whitelisting Ports

White list ports to allow communication between Master and Slave nodes. Once the Postgress installation is completed on both Master and Slave, you need to white-list the 5432 and 9187 ports from Master and Slave hosts (Inbound and Outbound).

Source: Master and Slave Hosts

Destination : Kubernetes Pods.

Setting up Replication on Master

To perform replication, follow the steps below:

1. Connect to database and create a user with privilege.

```
psql -U cdpadmin -W -d analytics -h localhost -p 5432
CREATE USER replication REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED PASSWORD 'replicationpa55word';
\du;
ALTER ROLE replication CONNECTION LIMIT -1;
```

2. Edit the /disk1/postgress/data/postgresql.conf file, update the below lines and save the file.

```

max_connections = 1000
wal_level = replica
max_wal_size = 100GB
max_wal_senders = 10
wal_keep_size = 3000
```

3. Similarly, edit the /disk1/postgress/data/pg_hba.conf file, add the below line and save the file.

```
host replication replication <Slave Host IP>/0 md5
```

4. After updating the postgresql.conf and pg_hba.conf, stop and start the postgress container.

```

cd /home/kgunda/postgress/
systemctl stop docker
```

```
systemctl start docker
cd /home/user/postgress
docker-compose down
docker-compose up -d
docker ps
```

Setting up Replication On Slave

To set up replication on slave VM, follow the steps below:

1. Similarly, perform replication on Slave using the below commands.

```
psql -U cdpadmin -W -d analytics -h localhost -p 5432
CREATE USER replication REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED PASSWORD 'replicationpa55word';
\du;
ALTER ROLE replication CONNECTION LIMIT -1;
```

2. Edit the /disk1/postgress/data/postgresql.conf file, update the below lines and save the file.

```
max_connections = 1000
wal_level = replica
max_wal_size = 100GB
max_wal_senders = 10
wal_keep_size = 3000
```

3. Edit the /disk1/postgress/data/pg_hba.conf file, add the below line and save the file.

```

host replication replication <Master Host IP>/0 md5
cd /disk1/postgress/data
rm -rfv *
```

4. For pg_basebackup working, install the postgressql-client on Slave Host.

```

dnf update -y
dnf install -y
 https://download.postgresql.org/pub/repos/yum/reporpms/EL-9-x86_64/pgdg-redhat-repo-latest.noarch.rpm
dnf -qy module disable postgresql
dnf install -y postgresql15
```

5. Once postgressql-client installation completed, run the below command from Slave.

```

pg_basebackup -h <Master Host IP> -U replication -p 5432 -D /var/lib/postgresql/12/main/ -Fp -Xs -P -R
```

6. After pg_basebackup completed, restart the postgress DB on Slave.

```

cd /home/user/postgres
systemctl stop docker
systemctl start docker
docker-compose down
docker-compose up -d
```

Validating replication and Creating table

To validate replication on VMs and create table, index and segments, follow the steps below:

1. Valid the replication on Master VM.

```
psql -U cdpadmin -W -d analytics -h localhost -p 5432
SELECT * FROM pg_stat_replication;
```

2. Once replication is successfully validated, create the following tables.

```
psql -U cdpadmin -W -d analytics -h localhost -p 5432

CREATE TABLE public.sst_events (
    apid character varying(128),
    hostname character varying(128),
    event_type character varying(64) NOT NULL,
    input_event text,
    cid character varying(64) NOT NULL,
    ds character varying(64) NOT NULL,
    ts bigint NOT NULL,
    reason character varying(1024),
    he character varying(256),
    hm character varying(256),
    vizid character varying(256),
    wspn character varying(256),
    subid character varying(1024),
    id character varying(1024),
    crmid character varying(256),
    ts_div_1000 bigint GENERATED ALWAYS AS (ts / 1000) STORED);

CREATE TABLE public.sst_demo_events (
    apid        VARCHAR(128),
    hostname    VARCHAR(128),
    event_type  VARCHAR(64) NOT NULL,
    input_event TEXT,
    cid         VARCHAR(64) NOT NULL,
    ds          VARCHAR(64) NOT NULL,
    ts          BIGINT NOT NULL,
    reason      VARCHAR(1024),
    he          VARCHAR(256),
    hm          VARCHAR(256),
    vizid       VARCHAR(256),
    wspn        VARCHAR(256),
    subid       VARCHAR(1024),
    id          VARCHAR(1024),
    crmid       VARCHAR(256),
    ts_div_1000 BIGINT GENERATED ALWAYS AS (ts / 1000) STORED
);

CREATE TABLE public.source_outbound (
    input_event TEXT,
    key         VARCHAR,
    event       VARCHAR,
    cid         VARCHAR,
    rid         VARCHAR(255),
    id          INTEGER NOT NULL DEFAULT nextval('source_outbound_id_seq'::regclass),
    ts          BIGINT,
    srcid       INTEGER,
    PRIMARY KEY (id)
);

CREATE TABLE public.source_failure (
```

```
    input_event  TEXT,
    key          VARCHAR,
    event        VARCHAR,
    cid          VARCHAR,
    errorcode    VARCHAR,
    errordetails TEXT,
    rid          VARCHAR(255),
    id           INTEGER NOT NULL DEFAULT nextval('source_failure_id_seq'::regclass),
    ts           BIGINT,
    srcid        INTEGER,
    PRIMARY KEY (id)
);

CREATE TABLE public.destination_outbound (
    input_event           TEXT,
    key                   VARCHAR,
    event                 VARCHAR,
    cid                   VARCHAR,
    rid                   VARCHAR(255),
    id                    INTEGER NOT NULL DEFAULT nextval('destination_outbound_id_seq'::regclass),
    ts                    BIGINT,
    destinationinstanceid INTEGER,
    srcid                 INTEGER,
    PRIMARY KEY (id)
);
```

3. Create the following index.

```
CREATE INDEX sst_events_apid ON public.sst_events (apid);
CREATE INDEX sst_events_cid ON public.sst_events (cid);
CREATE INDEX sst_events_crmid ON public.sst_events (crmid);
CREATE INDEX sst_events_ds ON public.sst_events (ds);
CREATE INDEX sst_events_event_type ON public.sst_events (event_type);
CREATE INDEX sst_events_he ON public.sst_events (he);
CREATE INDEX sst_events_hm ON public.sst_events (hm);
CREATE INDEX sst_events_id ON public.sst_events (id);
CREATE INDEX sst_events_subid ON public.sst_events (subid);
CREATE INDEX sst_events_ts ON public.sst_events (ts);
CREATE INDEX sst_events_ts_div_1000_temp ON public.sst_events (ts_div_1000 DESC);
CREATE INDEX sst_events_vizid ON public.sst_events (vizid);
CREATE INDEX sst_events_wspn ON public.sst_events (wspn);

CREATE INDEX sst_demo_events_apid ON public.sst_demo_events (apid);
CREATE INDEX sst_demo_events_cid ON public.sst_demo_events (cid);
CREATE INDEX sst_demo_events_crmid ON public.sst_demo_events (crmid);
CREATE INDEX sst_demo_events_ds ON public.sst_demo_events (ds);
CREATE INDEX sst_demo_events_event_type ON public.sst_demo_events (event_type);
CREATE INDEX sst_demo_events_he ON public.sst_demo_events (he);
CREATE INDEX sst_demo_events_hm ON public.sst_demo_events (hm);
CREATE INDEX sst_demo_events_id ON public.sst_demo_events (id);
CREATE INDEX sst_demo_events_subid ON public.sst_demo_events (subid);
CREATE INDEX sst_demo_events_ts ON public.sst_demo_events (ts);
CREATE INDEX sst_demo_events_ts_div_1000_temp ON public.sst_demo_events (ts_div_1000) DESC;
CREATE INDEX sst_demo_events_vizid ON public.sst_demo_events (vizid);
CREATE INDEX sst_demo_events_wspn ON public.sst_demo_events (wspn);

CREATE INDEX idx_cid ON public.source_outbound (cid);
CREATE INDEX idx_event ON public.source_outbound (event);
CREATE INDEX idx_key ON public.source_outbound (key);
```

```
CREATE INDEX idx_rid ON public.source_outbound (rid);
CREATE INDEX idx_srcid ON public.source_outbound (srcid);
CREATE INDEX idx_ts ON public.source_outbound (ts);

CREATE INDEX idx_cid_source_failure ON public.source_failure (cid);
CREATE INDEX idx_errorcode_source_failure ON public.source_failure (errorcode);
CREATE INDEX idx_errordetails ON public.source_failure (errordetails);
CREATE INDEX idx_errordetails_source_failure ON public.source_failure (errordetails);
CREATE INDEX idx_event_source_failure ON public.source_failure (event);
CREATE INDEX idx_key_source_failure ON public.source_failure (key);
CREATE INDEX idx_rid_source_failure ON public.source_failure (rid);
CREATE INDEX idx_srcid_source_failure ON public.source_failure (srcid);
CREATE INDEX idx_ts_source_failure ON public.source_failure (ts);

CREATE INDEX idx_cid_destination_outbound ON public.destination_outbound (cid);
CREATE INDEX idx_destinationinstanceid_destination_outbound ON public.destination_outbound
 (destinationinstanceid);
CREATE INDEX idx_event_destination_outbound ON public.destination_outbound (event);
CREATE INDEX idx_key_destination_outbound ON public.destination_outbound (key);
CREATE INDEX idx_rid_destination_outbound ON public.destination_outbound (rid);
CREATE INDEX idx_srcid_destination_outbound ON public.destination_outbound (srcid);
CREATE INDEX idx_ts_destination_outbound ON public.destination_outbound (ts);
```

4. Similarly, create Sequence.

```
CREATE SEQUENCE source_outbound_id_seq;
CREATE SEQUENCE source_failure_id_seq;
CREATE SEQUENCE destination_outbound_id_seq;
```

Installing DMP Producer

This section provides a step-by-step guide to installing DMP Producer.

Prerequisites

Make sure the following things are in place, before installing:

- Root access to the VM
- Required software like Java, MySQL, Kafka, must be installed on the VM

To install DMP Producer, follow the steps below:

1. Copy necessary files to the mount points e.g., /data, /hdd, and create symbolic links.

```
ln -s /data/ /disk1/
ls -al
```

2. Link the DMP Producer configuration.

```
ln -s /data/config/dmp-producer /mnt/sst
```

3. Update the property files like dmp-producer.properties, msk.properties and piistorage.properties.

- Locate the files at/disk1/config/dmp-producer/dmp-producer.properties, and update the dmp-producer.properties.

```
#Kafka producer configurations
KafkaProducerConfig=
# Properties for DB e.g., MySQL
DBConfig=
#SQS config
SQSConfig=
#SES Properties
SESConfig=
#SMTP
SMTPConfig=
```

- Locate the /disk1/msk/msk.properties, and update msk.properties.

```
# Properties for DB e.g., MySQL
DBConfig=
```

- Locate /disk1/piistorage/config/piistorage.properties, and update piistorage.properties.

```
# Properties for DB
DBConfig=
# Properties for Aerospike
AerospikeConfig=
# Properties for Cron Expression
CronExpressionConfig=
```

4. Copy the DMP Producer JAR file to /disk1/config/dmp-producer/.

```
cp /path/to/dmp-producer.jar /disk1/config/dmp-producer/
```

5. Create a user for DMP system.

```
useradd -c "DMP-System User" cdpservice
Update the /etc/passwd file to:
cdpservice:x:1002:1003:DMP-System User:/home/cdpservice:/sbin/nologin
```

6. Create a service file at /usr/lib/systemd/system/dmp-producer.service.

```
[Unit]
Description=DMP Producer Service
After=network.target
[Service]
User=cdpservice
Group=cdpservice
ExecStart=/usr/bin/java -jar /disk1/config/dmp-producer/dmp-producer.jar
SuccessExitStatus=143
StandardOutput=/disk1/logs/dmp-producer/dmp-producer-out.log
StandardError=/disk1/logs/dmp-producer/dmp-producer-err.log
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

7. Set Ownership and Permissions for the user.

```
chown -R cdpservice:cdpservice /disk1 /mnt
touch /disk1/logs/dmp-producer/dmp-producer-out.log
touch /disk1/logs/dmp-producer/dmp-producer-err.log
touch /disk1/logs/dmp-producer-gc.log
chcon -t user_home_t /disk1/logs/dmp-producer/dmp-producer-out.log
```

```
chcon -t user_home_t /disk1/logs/dmp-producer/dmp-producer-err.log
chcon -t user_home_t /disk1/logs/dmp-producer-gc.log
```

8. Start the DMP Producer Service.

```
systemctl start dmp-producer
systemctl status dmp-producer
```

Troubleshooting

If you encounter issues, use the following commands for troubleshooting:

```
ls -Z /disk1/logs/dmp-producer/dmp-producer-gc.log
ausearch -m avc -ts recent | audit2allow -M dmp_producer
chcon -t init_var_run_t /disk1/logs/dmp-producer/*
chcon -t init_var_run_t /disk1/logs/dmp-producer-gc.log
ausearch -m avc -ts recent
semodule -i dmp_producer.pp
systemctl start dmp-producer
systemctl status dmp-producer
systemctl stop dmp-producer
```

Installing Redis

This section outlines the process for installing and configuring Redis.

Redis, an in-memory data structure store widely used as a database, cache, and message broker. Redis is essential for high-performance data management and real-time processing in modern applications. This page explains how to set up Redis, enable remote access, and customize its configuration.

Prerequisites

Make sure the following things are in place, before installing:

- Root access to the VM

To install and configure Redis, follow the steps below:

1. Run the following bash command to update all existing system packages:

```
dnf update
```

2. After updating the packages, install the EPEL repository to access additional packages.

```
dnf install epel-release
```

3. Install Redis using the following command.

```
dnf install redis
```

4. Check if Redis is installed and its service status.

```
systemctl status redis
```

5. Start the Redis service if it is not running.

```
systemctl start redis
```

6. To configure Redis for Remote Access, navigate to the Redis configuration directory.

```
cd /etc
```

7. Open the Redis configuration file in a text editor as shown below.

```
vi redis.conf
```

8. Find the line containing `bind 127.0.0.1` and comment it out by adding `#` at the beginning, and add the following line to allow connections from any IP address:.

```
#bind 127.0.0.1
bind 0.0.0.0
```

9. Stop and restart the Redis service to apply the new configuration.

```
systemctl stop redis
systemctl start redis
```

10. Now, Redis is configured and ready to use with remote access enabled.

Installing RabbitMq

This section provides a detailed guide on installing and configuring RabbitMQ.

RabbitMQ, a robust message broker facilitates communication between distributed applications. RabbitMQ is critical for managing asynchronous workflows and ensuring reliable message delivery in a microservices architecture. This page covers setting up RabbitMQ, enabling necessary plugins, creating users, configuring permissions, and customizing settings for seamless integration with your deployment environment.

Prerequisites

Make sure the following things are in place, before installing:

- Root access to the VM.
- Redis is installed in the VM.

To install and configure RabbitMQ, follow the steps below:

1. Run the following bash command to Install necessary utilities for RabbitMQ setup.

```
dnf install socat logrotate -y
```

2. Download the Erlang package required for RabbitMQ.

```
curl -LO -C -
 https://github.com/rabbitmq/erlang-rpm/releases/download/v26.2.5/erlang-26.2.5-1.el8.x86_64.rpm
```

3. Use the following command to install the downloaded Erlang package.

```
sudo dnf install ./erlang-26.2.5-1.el8.x86_64.rpm
```

4. Run the following script to set up the RabbitMQ package repository.

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.rpm.sh | sudo bash
```

5. Install the RabbitMQ using the package manager.

```
dnf install -y rabbitmq-server
```

6. Enable the RabbitMQ service to start automatically on boot and then start the service.

```
sudo systemctl enable rabbitmq-server
sudo systemctl start rabbitmq-server
```

7. Enable the management plugin to provide a web-based interface for RabbitMQ.

```
rabbitmq-plugins enable rabbitmq_management
```

8. Now, RabbitMQ is now installed and configured. You can access the RabbitMQ Management UI at `http://<server-ip>:15672` using default credentials (`guest/guest`).

Post Configuration for RabbitMQ

After enabling RabbitMQ and the management plugin, to configure the system further follow these additional steps below:

1. Navigate to the RabbitMQ configuration directory to add configuration files:

```
cd /etc/rabbitmq/
```

2. Create the `definitions.json` file:

```
vi definitions.json
```

3. Paste the following content into `definitions.json` to set up a user, permissions, and virtual host:

```
{
    "users": [
        {
            "name": "celery",
            "password_hash": "fa604aceeeaae520de8742410200d620edc4a6101fb2971cd706b4cc19141dce",
            "hashing_algorithm": "rabbit_password_hashing_sha256",
            "tags": "administrator"
        }
    ],
    "vhosts": [
        {
            "name": "/"
        }
    ],
    "permissions": [
        {
            "user": "celery",
            "vhost": "/",
            "configure": ".*",
            "write": ".*",
            "read": ".*"
        }
    ],
    "queues": [],
    "exchanges": [],
    "bindings": []
}
```

4. Save and exit the file. Now, update the `rabbitmq.conf` file:

```
vi rabbitmq.conf
```

5. Add the following line to load the definitions file during RabbitMQ startup.

```
load_definitions = /etc/rabbitmq/definitions.json
```

6. Save and exit the file. Now, restart RabbitMQ to apply the changes:

```
sudo systemctl stop rabbitmq-server
sudo systemctl start rabbitmq-server
```

7. RabbitMQ is now configured with a predefined user, permissions, and virtual host.