HCLSoftware

HCL CDP v12.1.10 AWS Platform Installation Guide

Contents

Chapter 1. Installing HCL CDP Platform on AWS

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

Introduction

This section details the installation and configuration of HCL CDP on the Amazon Web Services (AWS) Cloud platform. It includes deployment of micro-services on Kubernetes (EKS), as well as VM-based components and supporting infrastructure.

Prerequisites

Before installation, ensure AWS account access with necessary permissions, VPC setup, subnets, KMS key, and installation of CLI tools like AWS CLI, kubectl, and Helm

The deployment consists of two parts:

- Microservices-based components to be deployed on EKS (Amazon Elastic Kubernetes Service)
- VM-based components to be deployed on EC2 instances

Please ensure access to AWS with appropriate permissions and the ability to provision infrastructure including IAM roles, VPC, subnets, and KMS keys.

Hardware Prerequisites

Below is the minimum hardware requirement for the HCL CDP deployment.

VMs

Server/Node	**CPU**	**Memory**
Bastion host	2 vCPUs	4 GB
Aerospike DB	4 vCPUs	8 GB (x2)
Druid server	r5.4xlarge: 16 vCPUs	128 GB (x2)
Mongo DB Server	m5ad.xlarge – 4 vCPU / 8 GB RAM (x4), t3a.small – 2 vCPU / 2 GB RAM (arbiter)4 vCPUs	8 GB (x4) 2 GB
Nifi DB	r5d.large – 2 vCPU	16 GB RAM (x3)
Postgres	m7i.large – 2 vCPU	8 GB RAM (x2)

Server/Node	CPU	Memory
Redis & RMQ for Celery	r5.large – 2 vCPU	16 GB RAM

EKS Cluster Nodes

Server/Node	Instance Type	CPU	Memory
3 Nodes	c7i.8xlarge	32 vCPUs	64.0 GiB
18 Nodes	m7i.xlarge	4 vCPUs	16.0 GiB

EMR Cluster Nodes

Server/Node	Instance Type	CPU	Memory
Primary Node	m5.xlarge	4 vCPUs	16 GiB
Core Node	m5.xlarge	4 vCPUs	16 GiB

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.

- Devtron
- Aerospike
- MongoDB
- Druid
- MSK Cluster
- EMR Cluster
- MWAA Environment
- Key Management Service (KMS)
- AWS Secrets Manager
- RDS Configuration
- eck-operator
- elasticsearch-kibana
- prometheus-grafana
- Fluentbit

Installing Devtron

Devtron is installed using Helm to manage the Kubernetes-based CDP microservices deployment.

Devtron is a Kubernetes-native CI/CD platform that is used to manage the deployment of microservices in the EKS cluster.

To install the aerospike, follow the steps below:

1. Install Devtron.

```
helm repo add devtron https://helm.devtron.ai
 helm repo
        update devtron
 helm install devtron devtron/devtron-operator \
  --create-namespace --namespace devtroncd \
  --set installer.arch=multi-arch
```

2. Check if Devtron service is available:

```
kubectl get svc -n devtroncd devtron-service -o
        jsonpath='{.status.loadBalancer.ingress}'
```

3. Retrieve the admin password:

```
kubectl -n devtroncd get secret devtron-secret -o
        jsonpath='{.data.ADMIN_PASSWORD}' | base64 -d
```

4. Ingress Configuration (Optional):

```
kubectl patch -n devtroncd svc devtron-service -p '{
  "spec": {
    "ports": [{
      "port": 80,
      "targetPort": "devtron",
      "protocol": "TCP",
      "name": "devtron",
      "nodePort": 30000
    }],
    "type": "NodePort",
    "selector": {
      "app": "devtron"
    }
  }
 }'
```

5. Apply ALB Ingress (if applicable):

```
kubectl apply -f Devtron-Ingress.yaml -n devtroncd
```

Note: If you are using any existing ALB, update ACM Arn & HTTPS Redirection Annotations. If it's the new ALB, create ALB with only HTTP and Add DNS & ACM Certs and upgrade.

Installing Aerospike

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

To install the aerospike, follow the steps below:

1. Install Aerospike on the target VM and configure it to run as a service.

```
wget -O aerospike.tgz
        https://enterprise.aerospike.com/.../el8_amd64
 tar -xzf aerospike.tgz
```

```
 cd
        aerospike-*/
 ./asinstall
```

2. Create systemd configuration:

```
vi /etc/systemd/system/aerospike.service.d/user.conf

 [Service]
 User=aerospike
 Group=aerospike
```

3. Configure disks and ownership:

```
lsblk  # check the instance store disk
 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
```

4. Update configuration:

```
cd /etc/aerospike/
vim aerospike.conf    # Update the Cluster host/port details and storage
vim features.conf   # Update the Licence Details
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/
```

5. Enable required ports and start Aerospike:

```
semanage port -a -t http_port_t -p tcp 13000-13003
systemctl start aerospike
 systemctl status aerospike
tail -F /var/log/aerospike/aerospike.log
```

6. Use the following to verify Aerospike service and cluster:

```
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

Node Exporter collects system metrics such as CPU, memory, disk, and network usage. These metrics are then scraped by Prometheus for monitoring and visualization.

1. Download and extract Node Exporter.

```
wget

 https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.0.linux-amd64.t
ar.
tar xfz node_exporter-1.7.0.linux-amd64.tar.gz
```

2. Move binary and create system user.

```
sudo mv node_exporter-1.7.0.linux-amd64/node_exporter
        /usr/local/bin/
 sudo useradd -rs /bin/false node_exporter
```

3. Create a systemd service unit.

```
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
```

4. Set permissions and SELinux contexts.

```
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
```

5. Start and enable service.

```
systemctl daemon-reload
 systemctl start node_exporter
 systemctl status node_exporter
 systemctl restart aerospike
 systemctl enable node_exporter
```

6. Validate metrics endpoint and exporter status.

```
/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
 curl -kv  http://localhost:9100/metrics
```

Aerospike Prometheus Exporter

Install the exporter to expose Aerospike metrics to Prometheus.

1. Install Aerospike Prometheus exporter.

```
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
 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

vim /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
```

2. Aerospike monitoring validation.

```
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

MongoDB is configured as a replica set across multiple nodes with an arbiter, and includes user and role setup. Monitoring is enabled using MongoDB Exporter.

Install MongoDB on the designated VMs and configure replica set with required ports and users.

To install the MongoDB, follow the steps below:

1. Configure MongoDB repository and install.

```
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

yum install -y mongodb-org
```

2. Update systemd configuration for replica set.

```
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
```

3. Configure data directories and SELinux permissions.

```
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

 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
```

4. Configure replica set.

```
mongosh --host <primary-host> --port 51090

 use admin;

 conf = {
  _id: "cdprsg",
  members: [
    { _id: 0, host: "<host1>:51090", tags: { usage:
        "schedule" } },
    { _id: 1, host: "<host2>:51090", tags: { usage:
        "schedule" } },
    { _id: 2, host: "<host3>:51090", tags: { usage:
        "segloader" } },
    { _id: 3, host: "<host4>:51090", tags: { usage:
        "analytics" } }
  ]
 }
 rs.initiate(conf)
 rs.addArb("<arbiter-host>:51090")
 conf = rs.conf()
```

```
 conf.members[3].hidden = true
 rs.status()
```

5. Create required MongoDB users.

```
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: "<password>", roles: [
  { role: "backup", db: "admin" },
  { role: "clusterMonitor", db: "admin" },
  { role: "read", db: "local" },
  { role: "readWrite", db: "admin" },
  { role: "restore", db: "admin" }
 ]
        });
  openssl rand -base64 16
```

Node Exporter

Node Exporter installed on MongoDB virtual machines collect systems metrics such as CPU, memory, disk, and network usage. These metrics are then scraped by Prometheus for monitoring and visualization.

To install node exporter, follow the steps below:

1. Download and extract Node Exporter.

```
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-1.7.0.linux-amd64.tar.gz
```

2. Move binary and create system user.

```
sudo mv node_exporter-1.7.0.linux-amd64/node_exporter /usr/local/bin/
 sudo useradd -rs /bin/false node_exporter
```

3. Create a systemd service unit.

```
vi /etc/systemd/system/node_exporter.service
```

Paste the following contents:

```


 [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
```

4. Start and enable service.

```
restorecon -Rv /usr/local/bin
systemctl daemon-reload
 systemctl start node_
 systemctl status node_exporter
 systemctl enable node_exporter
```

5. To validate metrics endpoint and exporter status, run this command to confirm the service is up.

```
  curl -kv http://localhost:9100/metrics
```

MongoDB Prometheus Exporter

This section details how to install and configure **Percona's MongoDB Exporter** to expose MongoDB metrics to Prometheus for scraping and monitoring. To install Prometheus exporter, follow the steps below:

1. Install the Exporter Package.

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

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

2. Create a systemd Unit File.

```
vi /etc/systemd/system/mongodb_exporter.service
```

Paste the following contents:

```
 [Unit]
 Description=Prometheus MongoDB Exporter
 After=network.target

 [Service]
 Type=simple
 User=mongodb_exporter
 Group=mongodb_exporter

        Environment="OPTIONS=--mongodb.uri=mongodb://pmm:<password>@<mongo-host>:51090
        --compatible-mode"
 ExecStart=/usr/bin/mongodb_exporter $OPTIONS
 Restart=always

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

Note: Replace `<password>` and `<mongo-host>` with actual credentials and hostname.

3. Reload and Start the Service.

```
systemctl daemon-reload
 systemctl start mongodb_exporter
```

```
 systemctl status mongodb_exporter
 systemctl enable mongodb_exporter
```

4. Run this command to validate and confirm the service is up.

```
curl -kv http://localhost:9216/metrics
```

Installing Druid

Apache Druid is deployed for real-time analytics, using MySQL for metadata and S3 for deep storage. Configuration includes connecting storage and starting Druid services.

Apache Druid is a real-time analytics database used in HCL CDP. The following steps detail its installation using MySQL for metadata storage and S3 for segment storage.

1. Install MySQL and create metadata database.

```
yum install mysql-server
 systemctl start mysqld

 mysql -u root -p
 CREATE USER 'druid'@'localhost' IDENTIFIED BY '<password>';
 CREATE DATABASE druid
        DEFAULT CHARACTER SET utf8mb4;
 GRANT ALL PRIVILEGES ON druid.* TO
        'druid'@'localhost';
 FLUSH PRIVILEGES;
```

2. Download and set up Druid.

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

 wget https://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.49/mysql-connector-java-5.1.49.jar
 cp mysql-connector-java-5.1.49.jar /sdd/apache-druid-25.0.0/extensions/mysql-metadata-storage/
```

3. Configure metadata and segment storage in Druid.

```
vi /sdd/apache-druid-25.0.0/conf/druid/single-server/small/_common/common.runtime.properties

 # Load extensions
 druid.extensions.loadList=["druid-hdfs-storage", "druid-kafka-indexing-service", "druid-datasketches",
 "druid-multi-stage-query","mysql-metadata-storage", "druid-s3-extensions"]
```

4. Comment Derby configurations and update the following properties in.

```
 common.runtime.properties
# Metadata storage
 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>
```

5. Update the following in S3 to store segments.

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

6. Add the following at the end of the file.

```
druid.storage.disableAcl=true
```

7. Start Druid.

```
cd /sdd/apache-druid-25.0.0
 nohup ./bin/start-single-server-small &
 ps -eaf | grep supervise
```

Installing MSK

AWS Managed Streaming for Kafka (MSK) is configured with IAM-based and SASL authentication. Secrets are managed in AWS Secrets Manager and linked to MSK.

AWS MSK provides a highly available Kafka cluster managed by AWS. This section details its configuration.

1. In the AWS MSK page, click **Create cluster**. In the **Create cluster** page, select Custom create, and enter the following details to create a provisioned cluster.
    - Cluster Name: hclsw-hss-hxcd-dev-msk
    - Cluster Type: Provisioned
    - Kafka Version: 3.5.1
    - Broker Type: kafka.m7g.large
    - Number of broker per zone:1
    - EBS storage volume per broker: 300 GiB

Step 1: Cluster settings

Edit

Cluster name

Name
hclsw-hss-hxcd-dev-msk

Cluster type

Cluster type
Provisioned

Apache Kafka version

Apache Kafka version
3.5.1

Brokers

Broker type	Brokers per zone
kafka.m7g.large	1

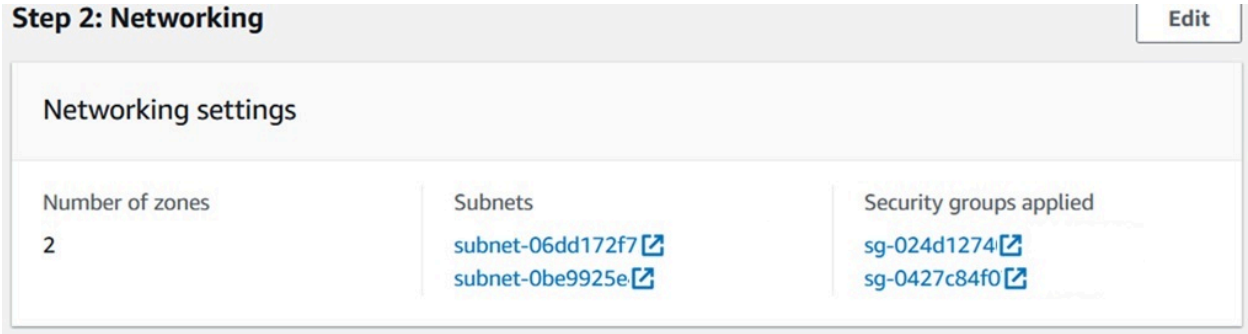
Storage

EBS storage volume per broker	Provisioned storage throughput per broker	Cluster storage mode
300 GiB	Not enabled	EBS storage

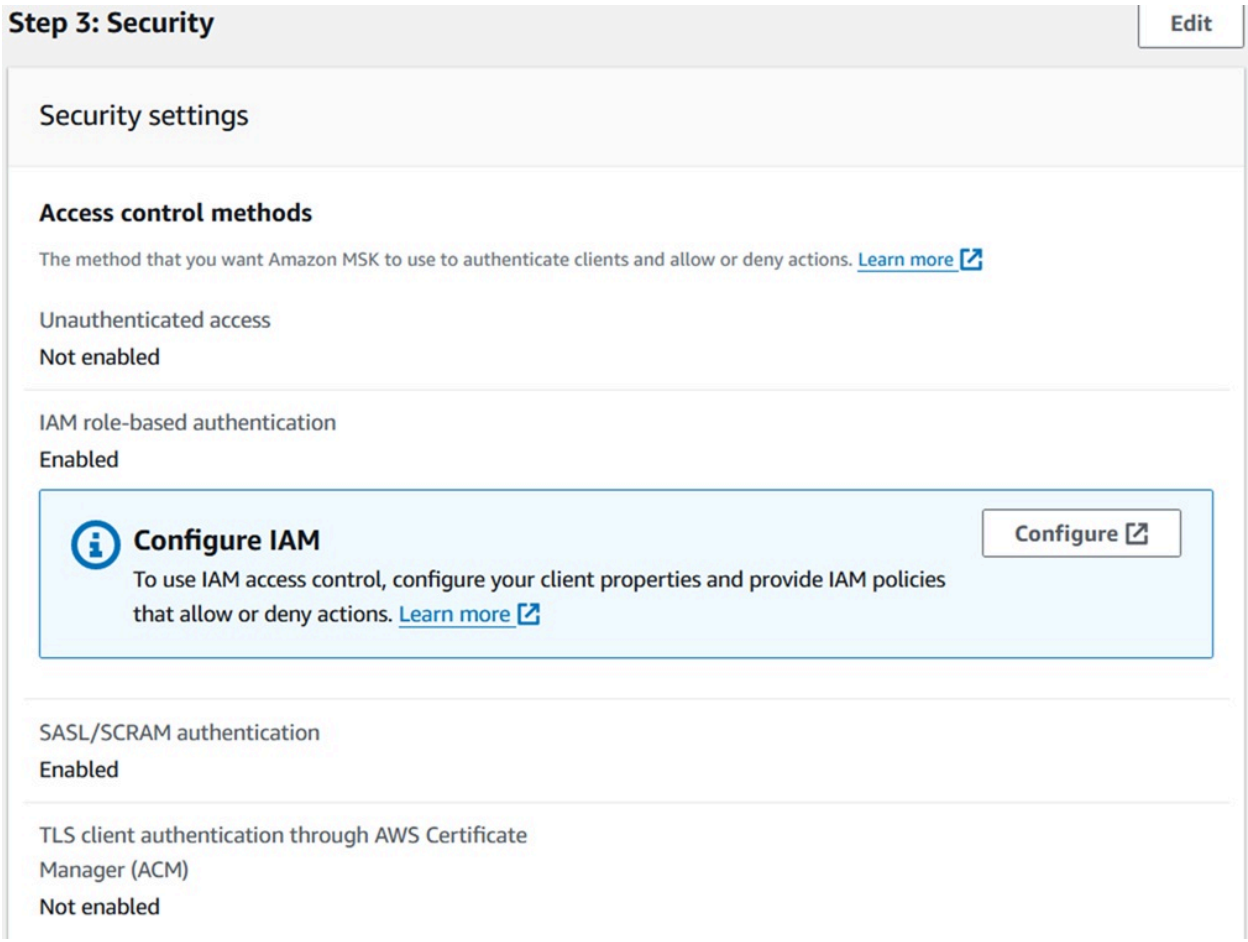
Cluster configuration

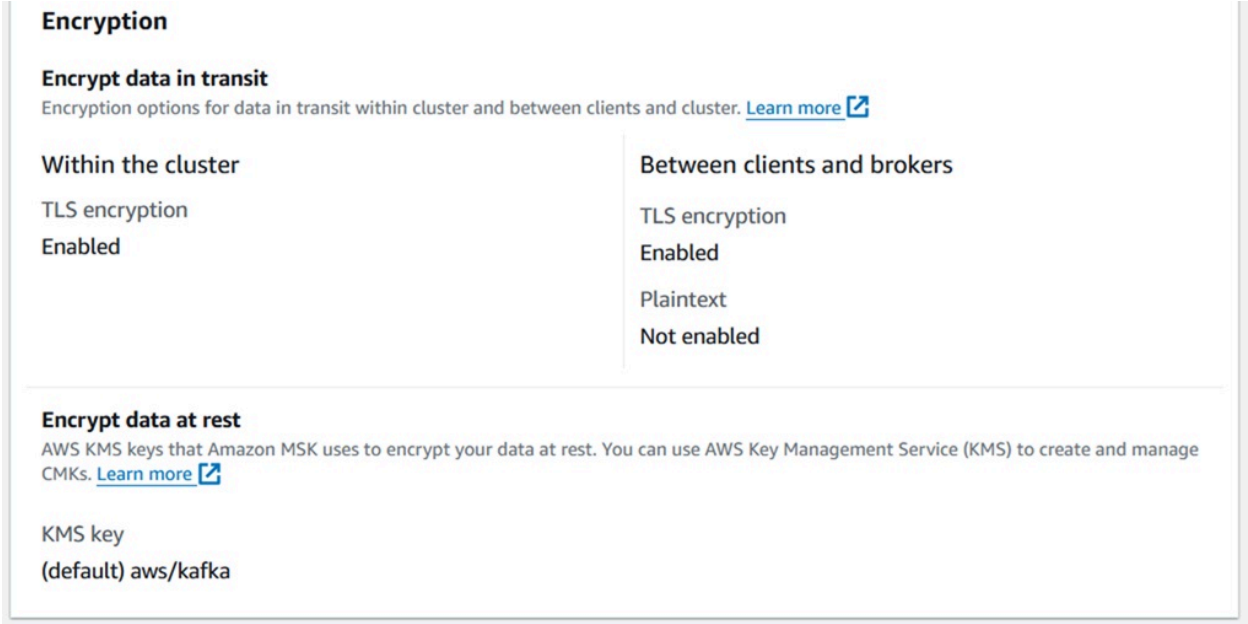
Cluster configuration
MSK default configuration

2. Click **Next**, and in the **Networking** page, select number of zones as 2.
3. Configure the subnets as subnet-06dd172f732b25926 and subnet-0be9925e46f6ef44a.
4. Assign MSK security group (sg-024d127404d8fd9d5 and sg-0427c84f077122631) and appropriate inbound rules.

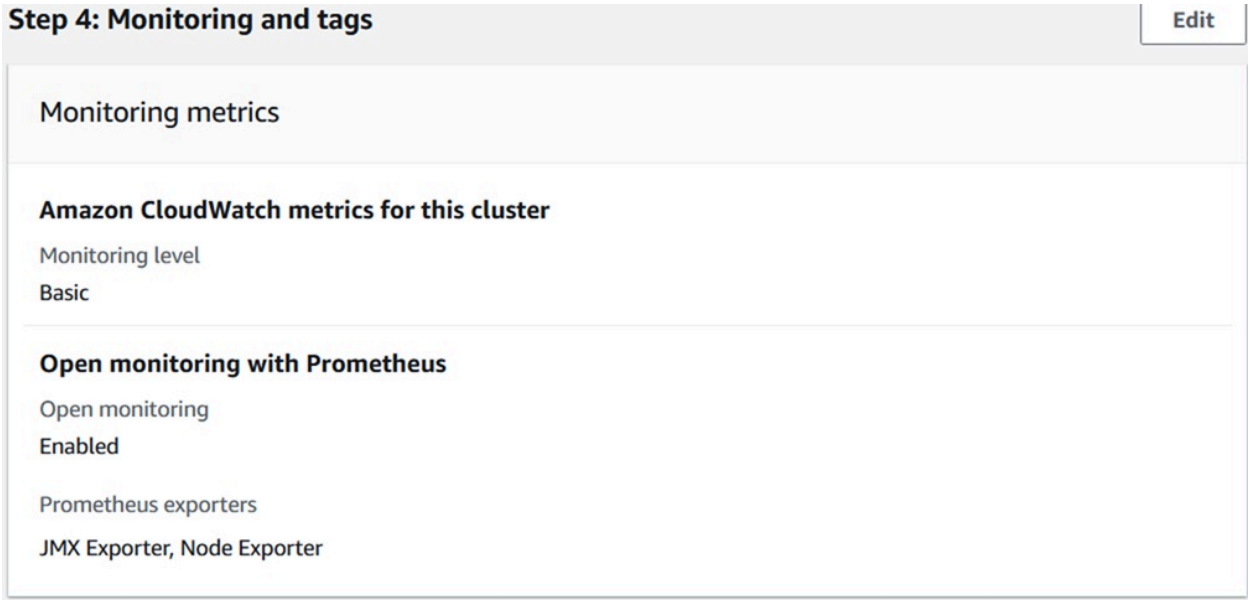


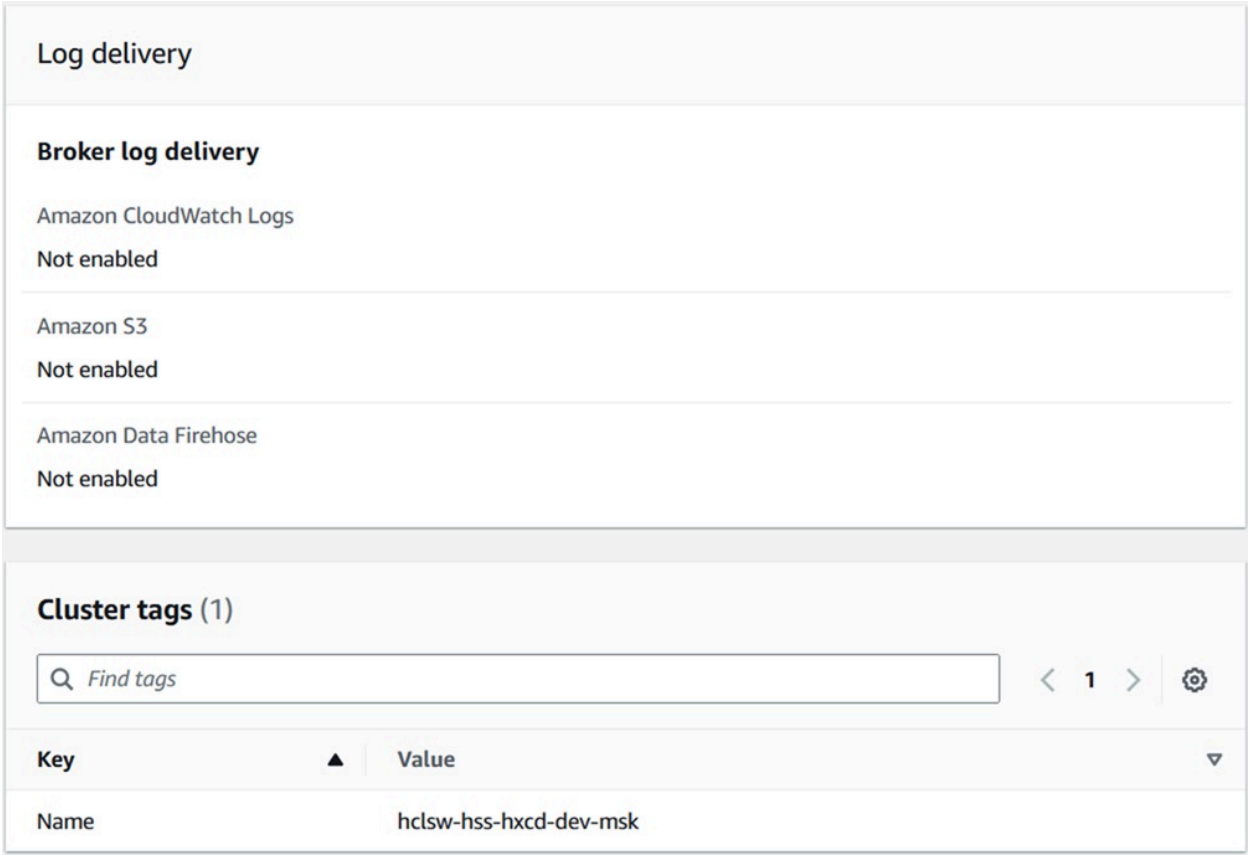
5. After updating the network configuration, click **Next**. In the **Security** page, configure the following:
   a. Enable IAM-based and SASL/SCRAM authentication.
   b. Enable TLS encryption for data in transit and between clients and brokers.
   c. Use a custom or AWS-managed KMS key to encrypt data at rest.



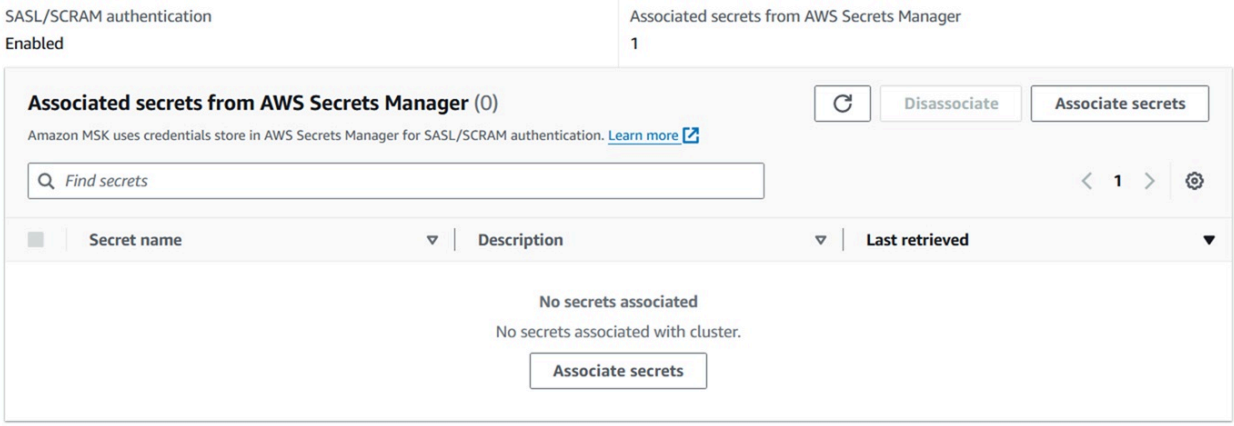


6. Click **Next**, and in the **Monitoring and tags** section, make the changes as shown below.
    a. In the **Amazon CloudWatch metrics for this cluster** section, select the **Basic monitoring** option.
    b. In the **Open monitoring with Prometheus** section, select the **Enable open monitoring with Prometheus** option. Where, Prometheus exporters are JMX Exporter and Node Exporter.
    c. In the **Cluster tags** section, enter the **Key** field as Name, and the **Value** field as hclsw-hss-hxcd-dev-msk.

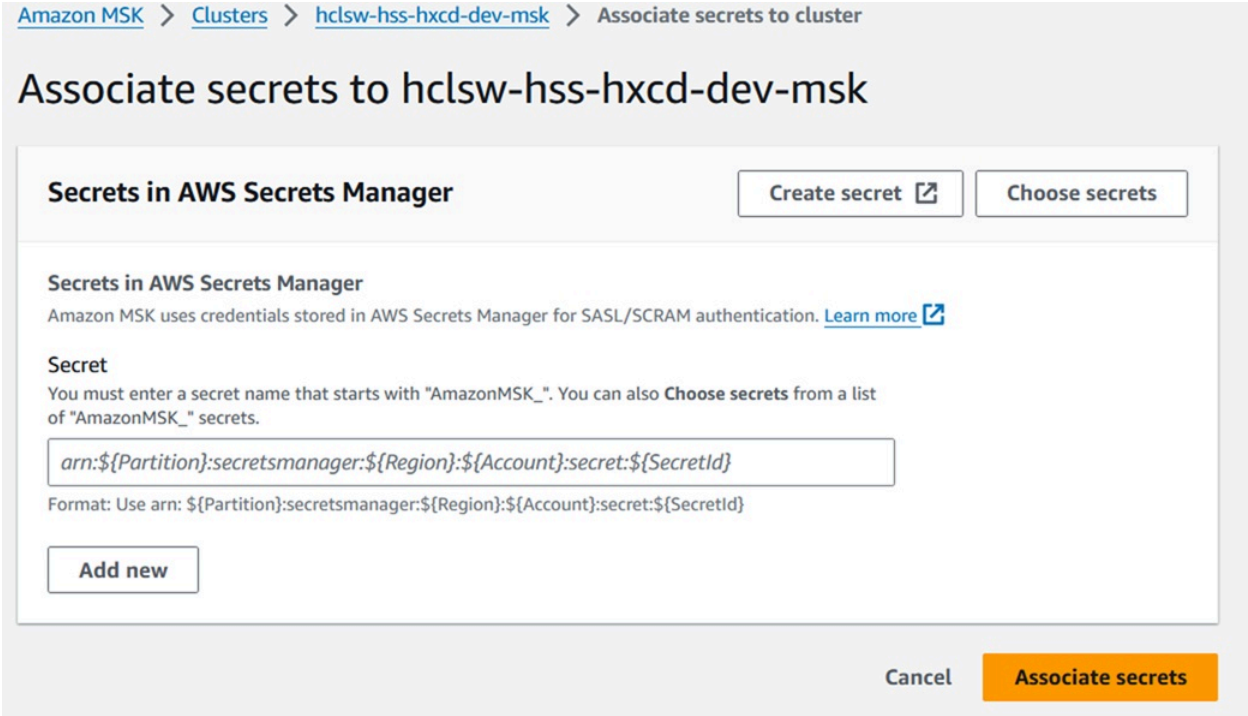


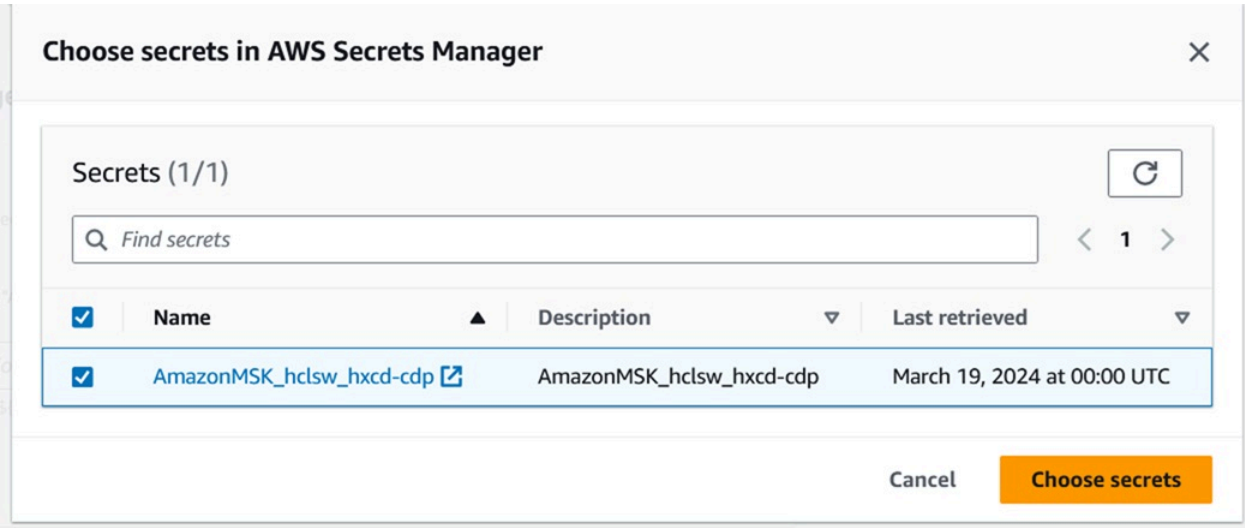


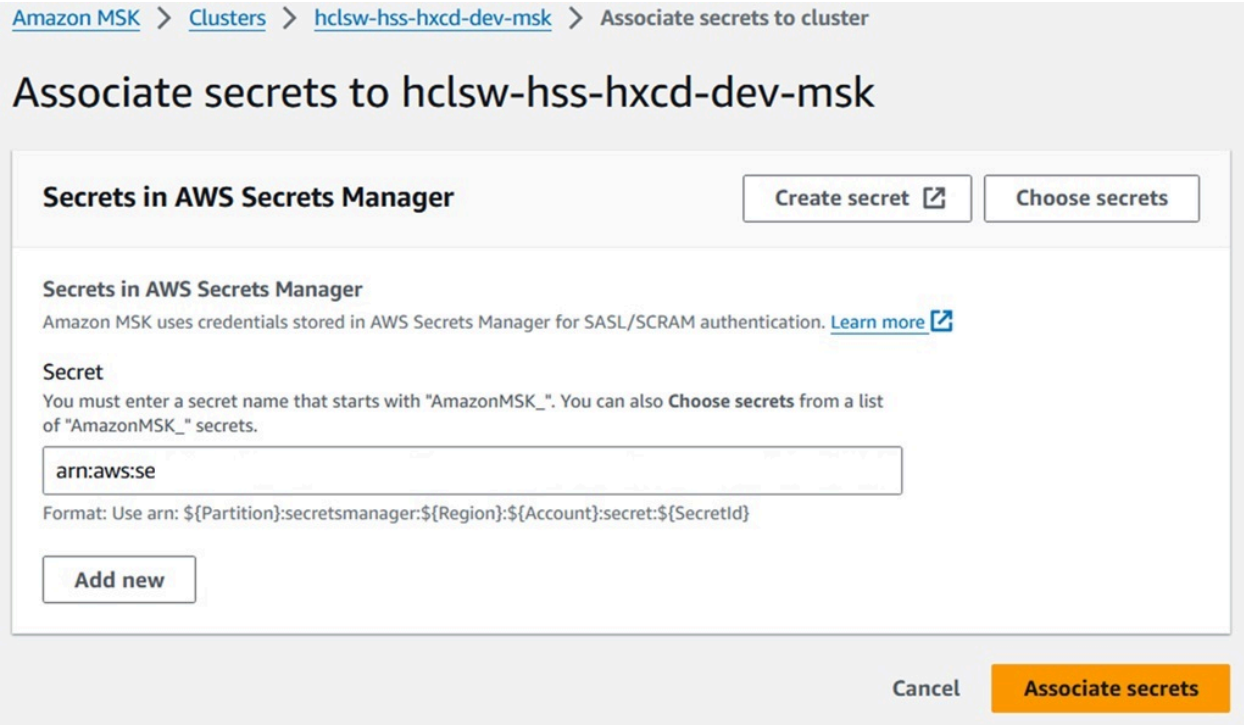
7. Click **Next**, and in the **Review and create** section, review and click **Create cluster**.
8. After creating the clusters, navigate to **Cluster** > **Properties** > **Security Settings** > **Associate Secrets**, and choose secrets starting with 'AmazonMSK' created in the Secrets Manager.



9. Click **Choose secrets** and select the Secrete Name starts with "**AmazonMSK**".


Amazon MSK > Clusters > hclsw-hss-hxcd-dev-msk > Associate secrets to cluster
Associate secrets to hclsw-hss-hxcd-dev-msk
Secrets in AWS Secrets Manager
Create secret
Choose secrets
Secrets in AWS Secrets Manager
Amazon MSK uses credentials stored in AWS Secrets Manager for SASL/SCRAM authentication. Learn more
Secret
You must enter a secret name that starts with "AmazonMSK_". You can also Choose secrets from a list of "AmazonMSK_" secrets.
arn:${Partition}:secretsmanager:${Region}:${Account}:secret:${SecretId}
Format: Use arn: ${Partition}:secretsmanager:${Region}:${Account}:secret:${SecretId}
Add new
Cancel
Associate secrets





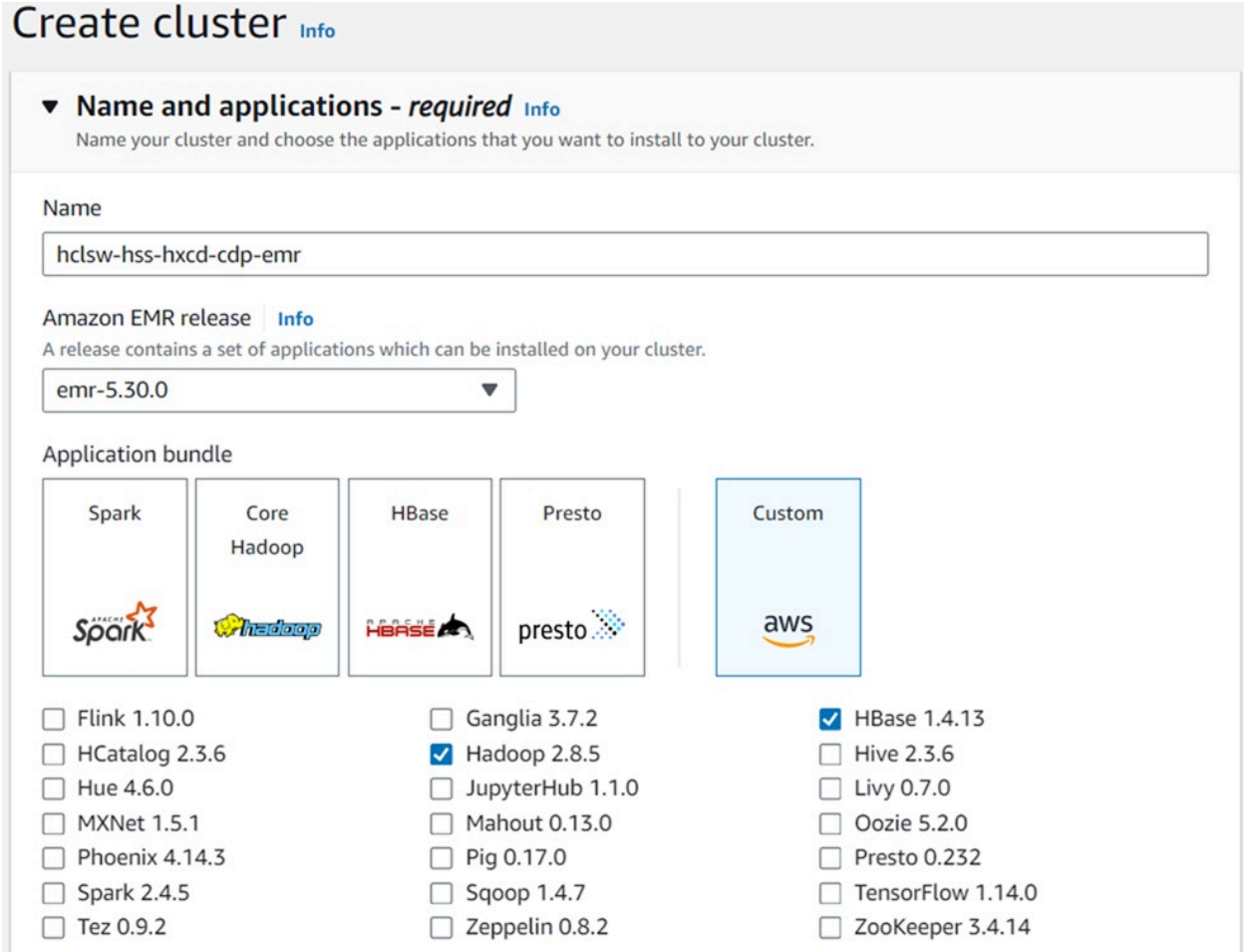
Installing EMR

Amazon Elastic MapReduce (EMR) is set up with Hadoop and HBase to handle distributed data processing. Logs and data are stored in S3 buckets.

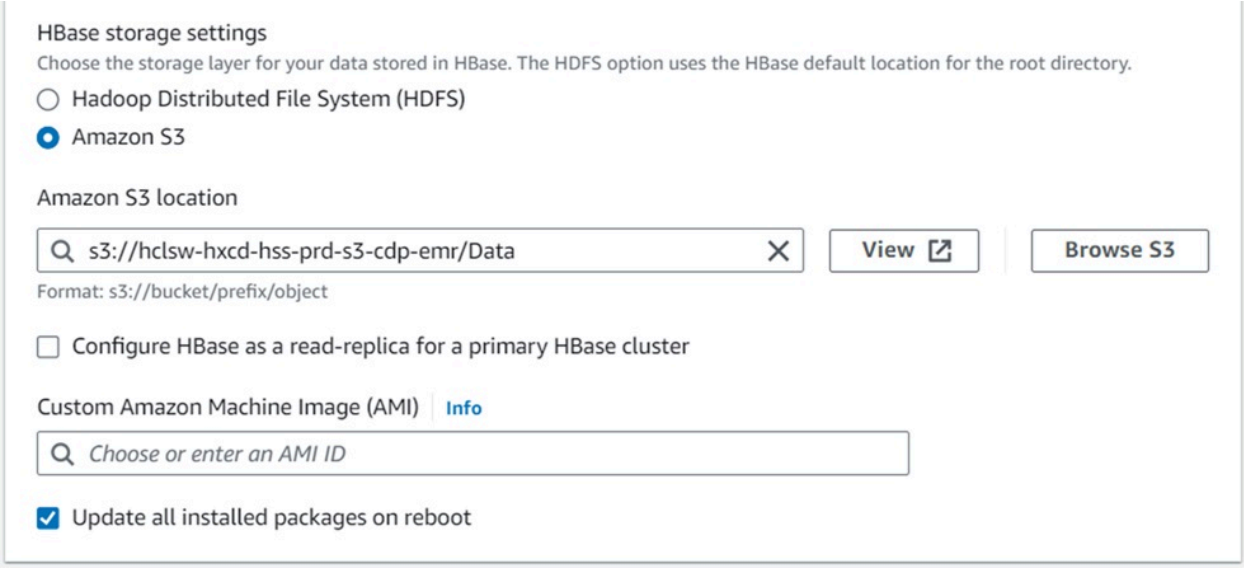
AWS EMR is used for distributed data processing using Hadoop and HBase.

To create Amazon EMR cluster, follow the steps below:

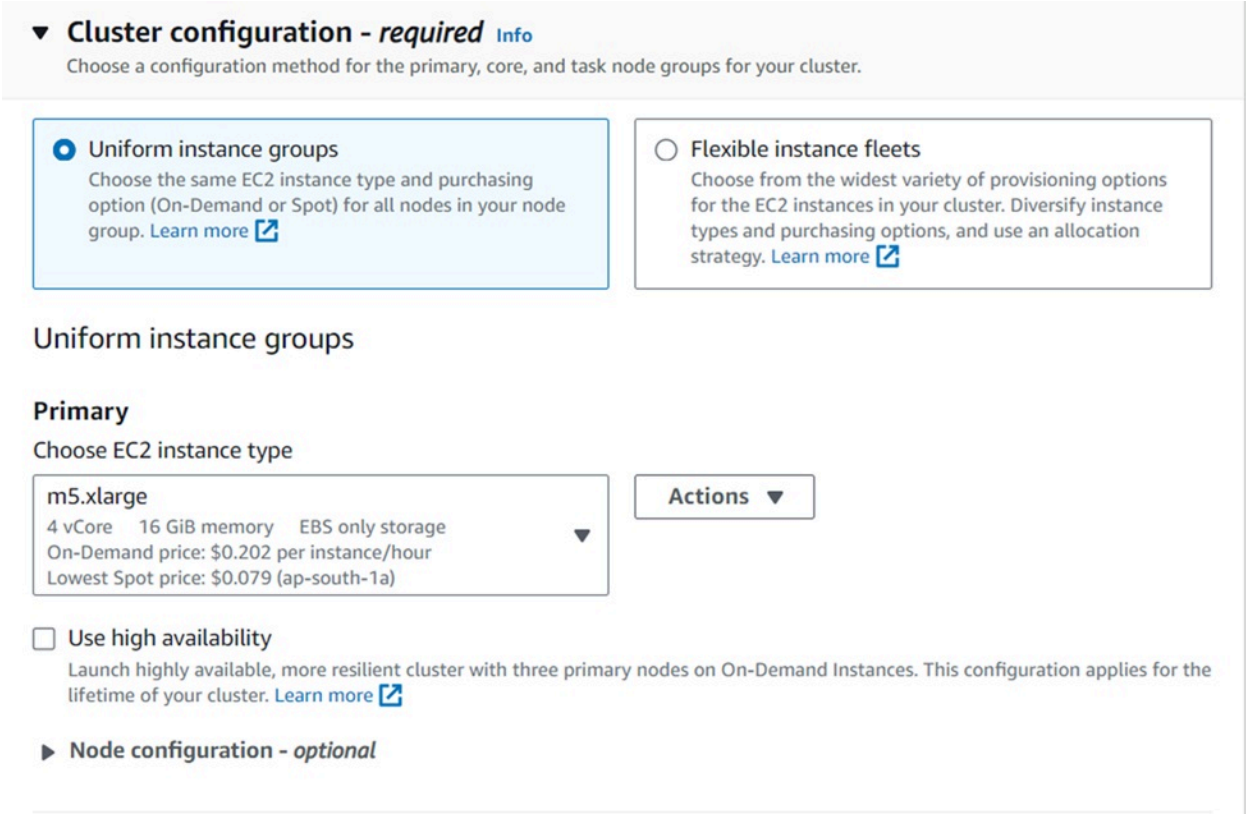
1. In the Amazon EMR page, click Create cluster.

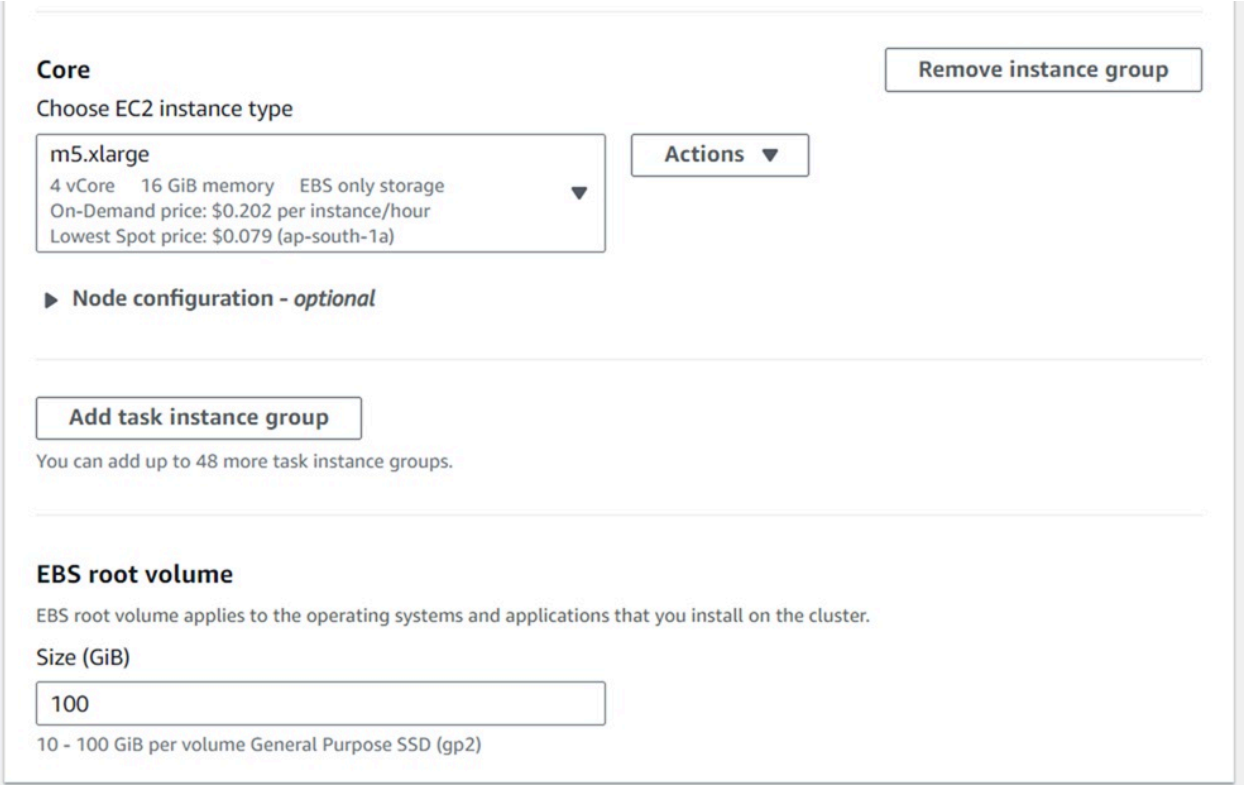


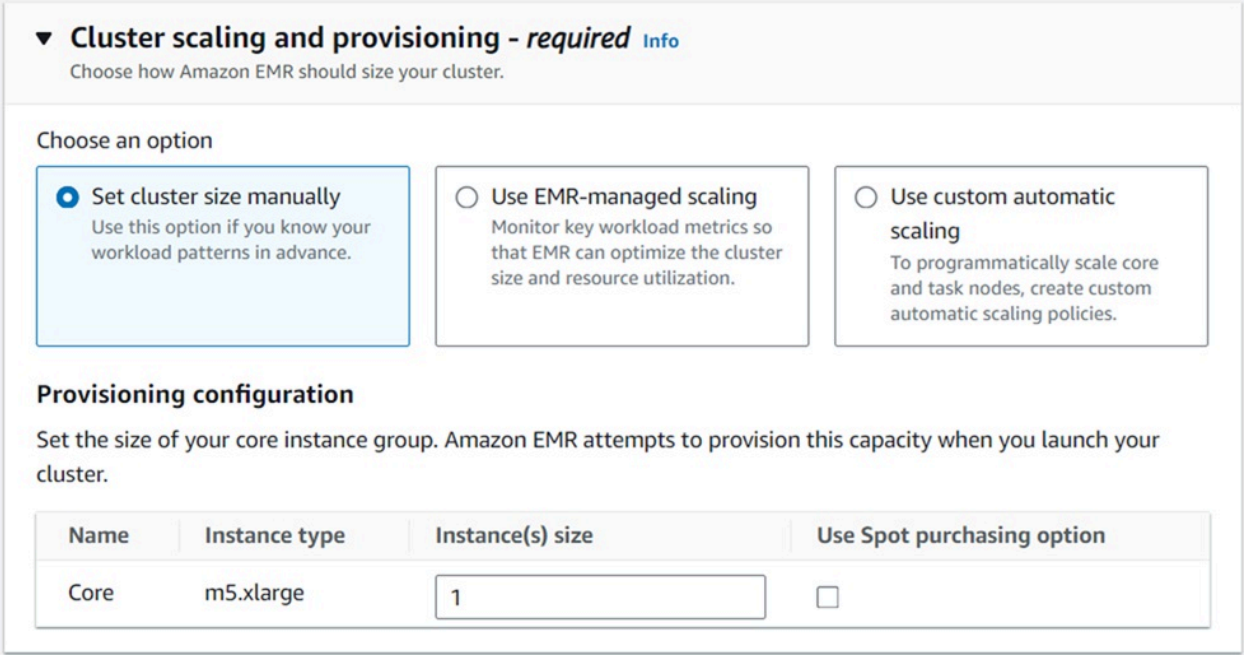
2. In the **Name** field, enter the cluster name, and in the Amazon EMR-release drop-down list, select the EMR Release Version 5.36.0 option.
3. From the Application bundles, select Hadoop and Hbase options.
4. In the Add S3 storage location field, enter the value as "s3://hclsw-hxcd-hss-prd-s3-cdp-emr/Data".



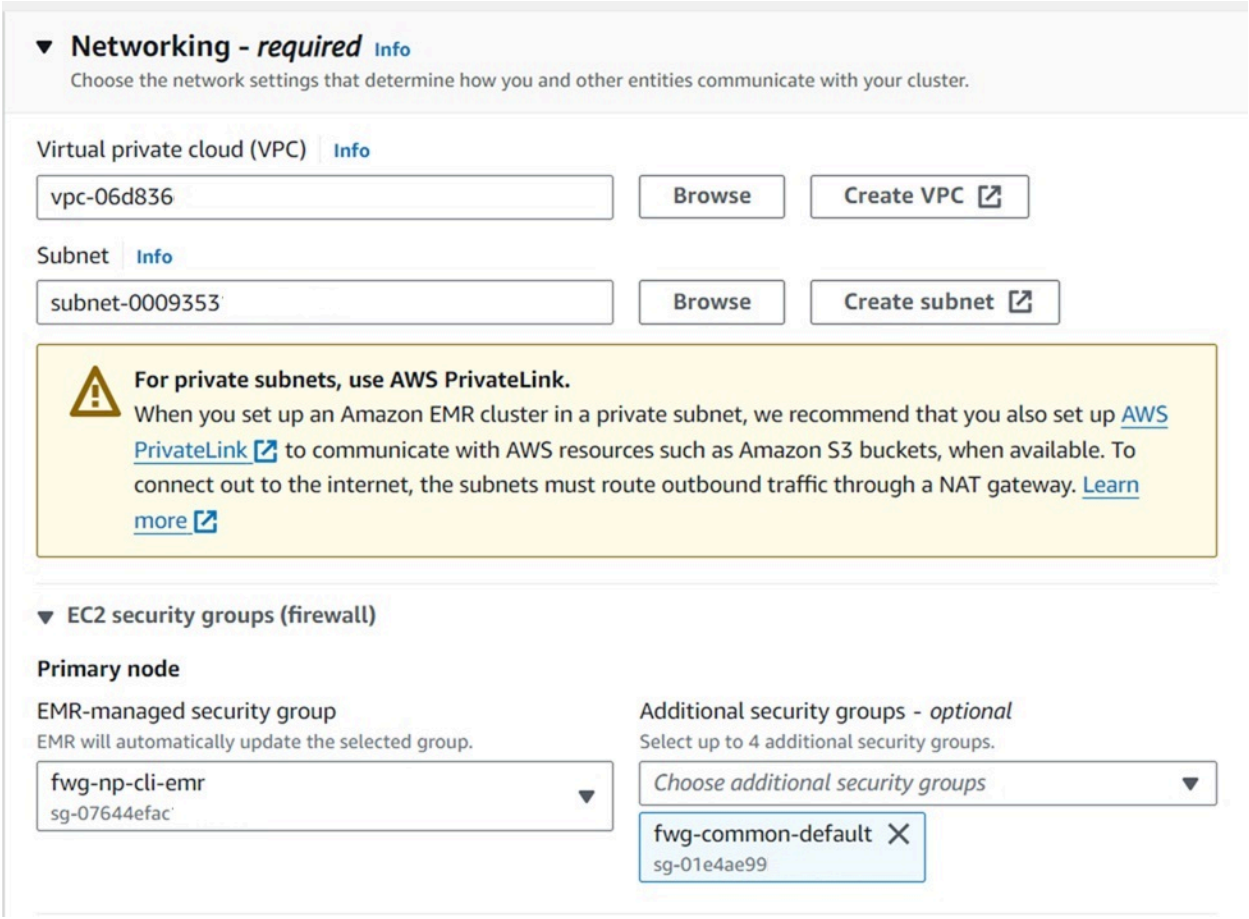
5. In the **Cluster configuration** section, select **Uniform instance groups** option, and for the **Uniform instance groups**, select primary and core options.

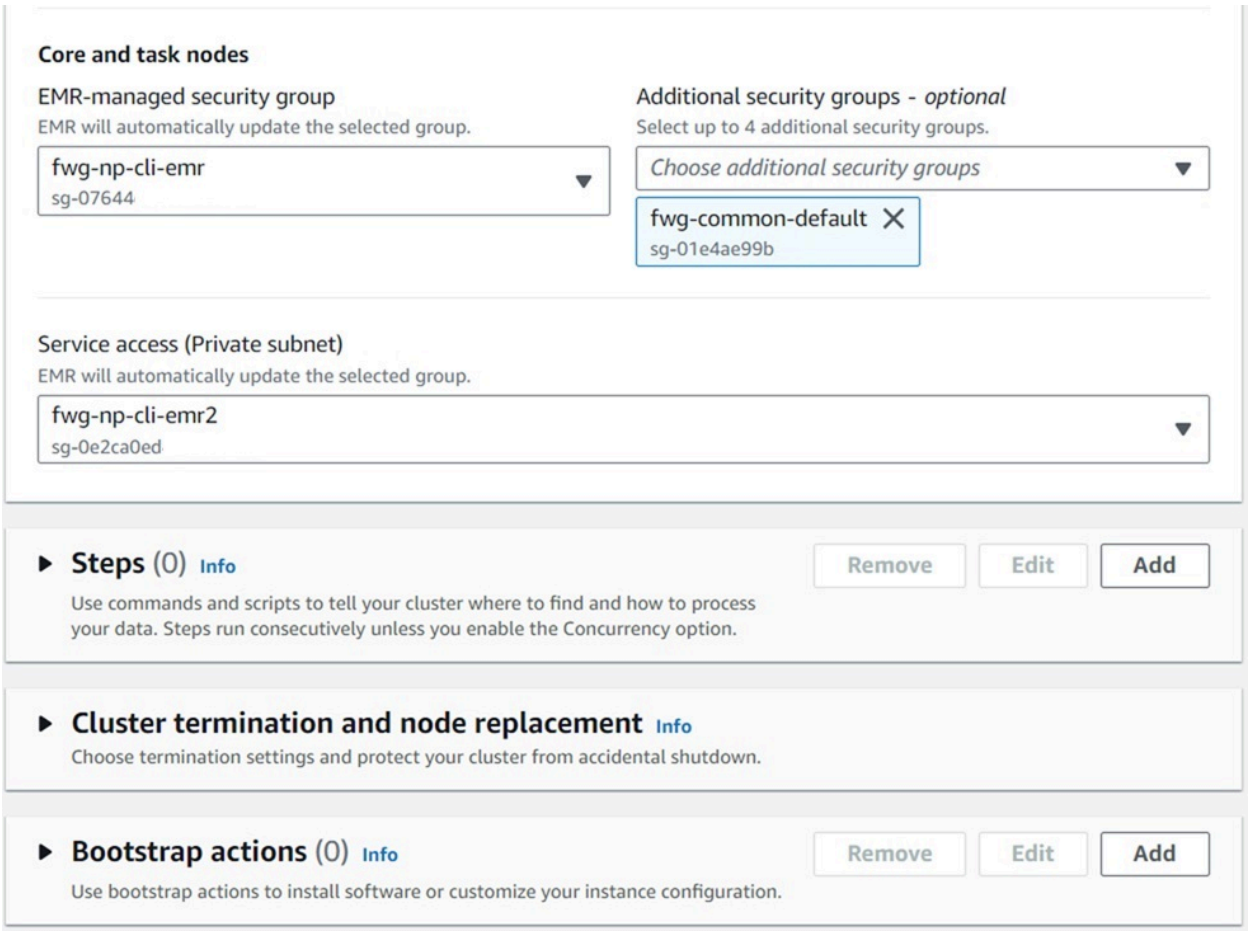




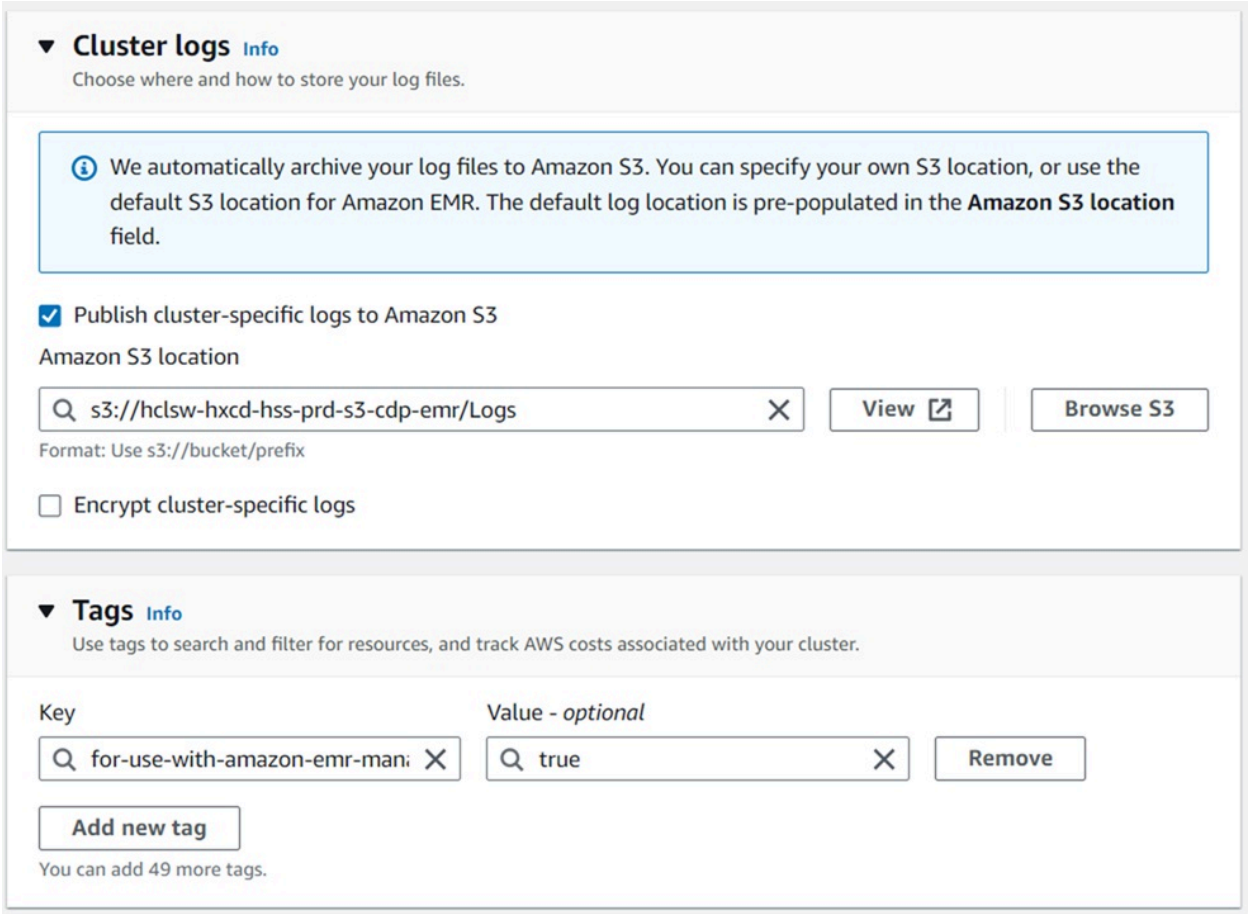


6. Remove Task Instance Group, and update the **EBS Volume size** field to 100 GB.
7. Add the Virtual private cloud (VPC), Subnets and Security Groups as shown below.

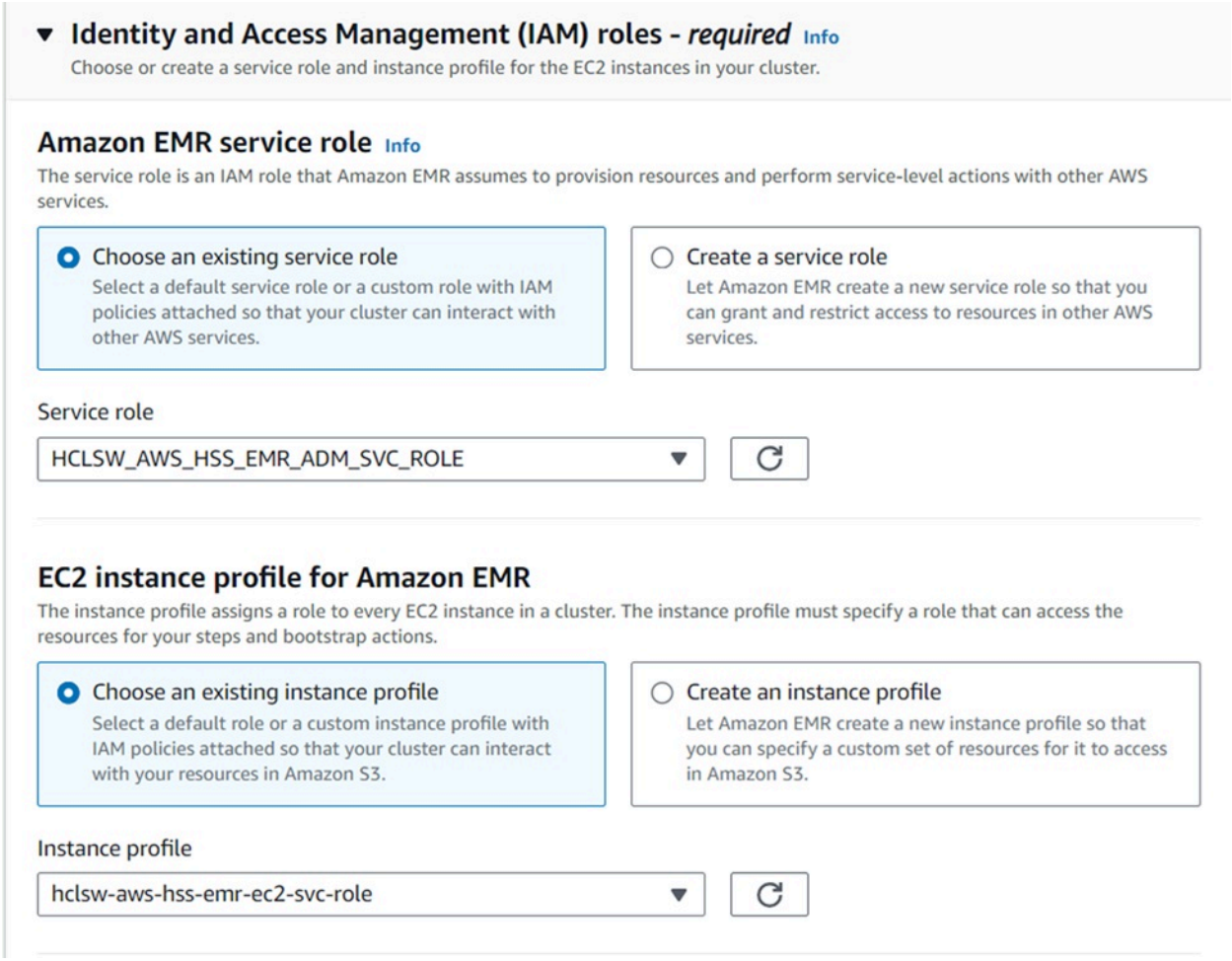

Networking - required Info
Choose the network settings that determine how you and other entities communicate with your cluster.
Virtual private cloud (VPC) Info
vpc-06d836
Browse
Create VPC
Subnet Info
subnet-0009353
Browse
Create subnet
For private subnets, use AWS PrivateLink.
When you set up an Amazon EMR cluster in a private subnet, we recommend that you also set up AWS PrivateLink to communicate with AWS resources such as Amazon S3 buckets, when available. To connect out to the internet, the subnets must route outbound traffic through a NAT gateway. Learn more
EC2 security groups (firewall)
Primary node
EMR-managed security group
EMR will automatically update the selected group.
fwg-np-cli-emr
sg-07644efac
Additional security groups - optional
Select up to 4 additional security groups.
Choose additional security groups
fwg-common-default
sg-01e4ae99



8. In the Cluster logs section, add s3 path for logs as S3://hclsw-hxcd-hss-prd-s3-cdp-emr/Logs.
9. Similarly, in the Tags section, add key and value as shown below.
    - Key : for-use-with-amazon-emr-managed-policies
    - Value : true


Cluster logs Info
Choose where and how to store your log files.
We automatically archive your log files to Amazon S3. You can specify your own S3 location, or use the default S3 location for Amazon EMR. The default log location is pre-populated in the Amazon S3 location field.
Publish cluster-specific logs to Amazon S3
Amazon S3 location
s3://hclsw-hxcd-hss-prd-s3-cdp-emr/Logs
View
Browse S3
Format: Use s3://bucket/prefix
Encrypt cluster-specific logs
Tags Info
Use tags to search and filter for resources, and track AWS costs associated with your cluster.
Key
Value - optional
for-use-with-amazon-emr-man
true
Remove
Add new tag
You can add 49 more tags.

10. Select the Custom EMR Service Role and EC2 instance Profile Role as shown below, and click **Create cluster**.

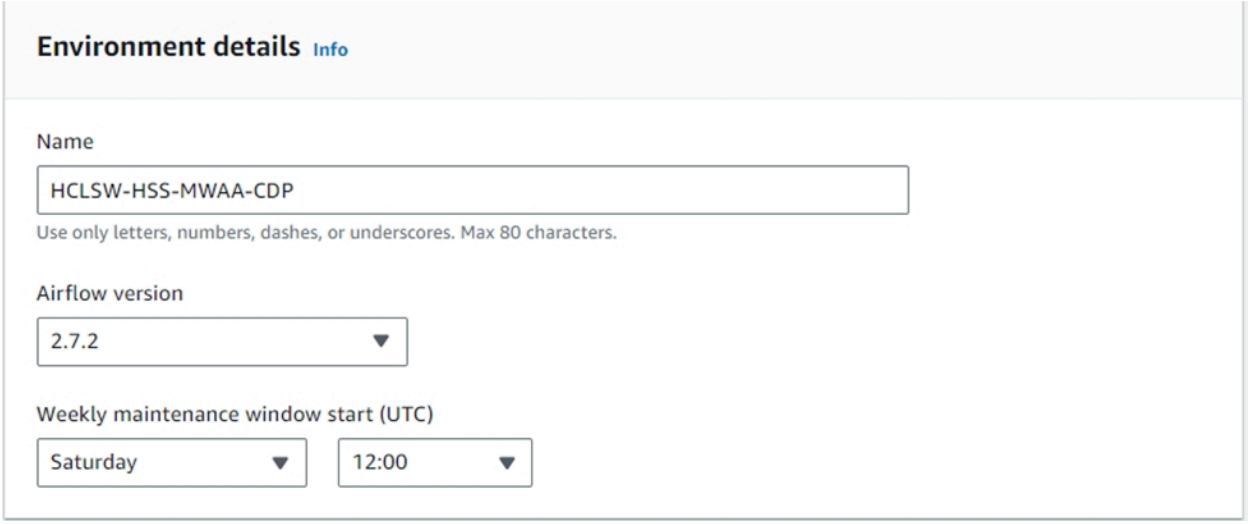


Installing MWAA

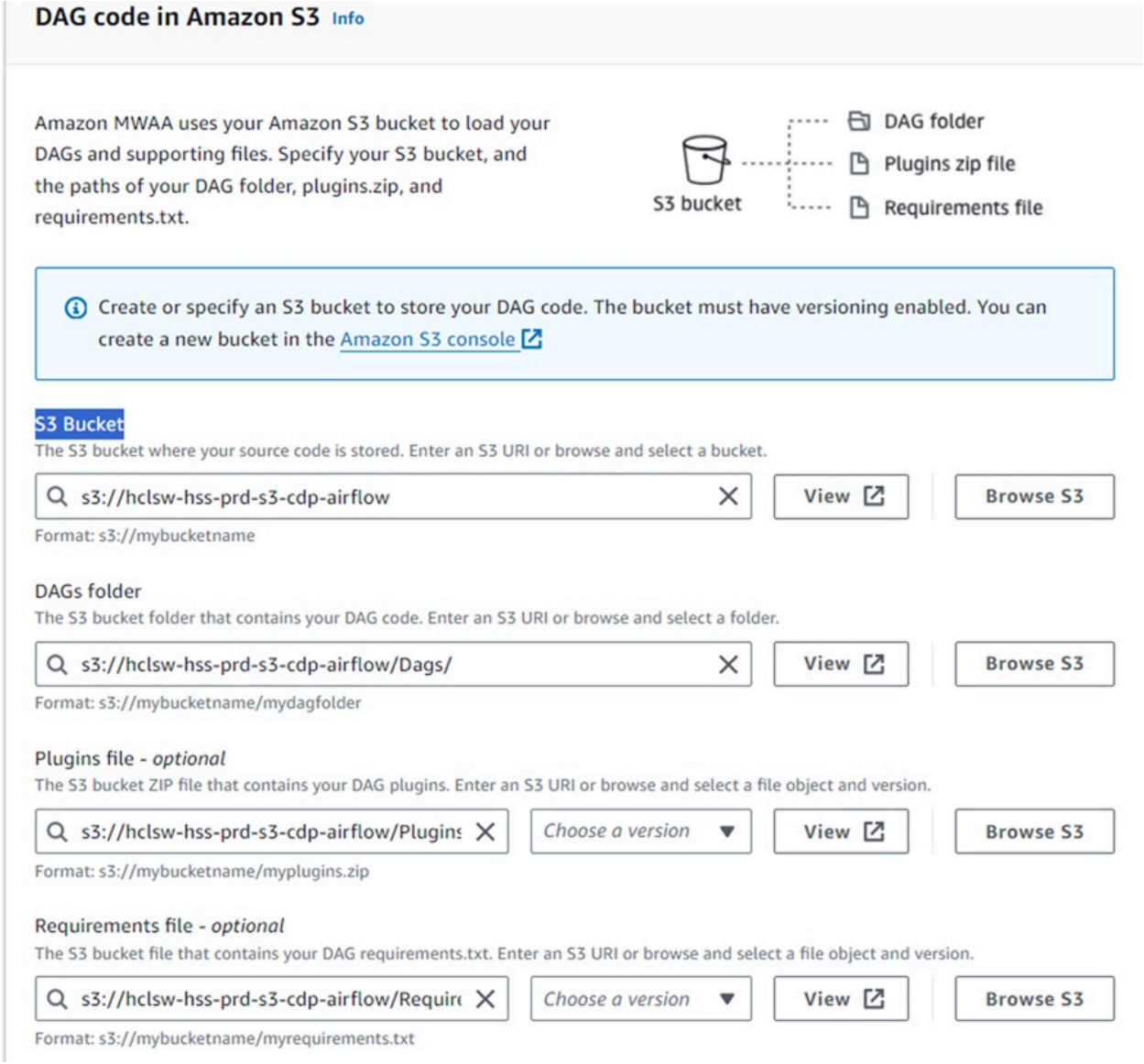
Amazon Managed Workflows for Apache Airflow (MWAA) provides Airflow orchestration for CDP jobs. The environment is configured with custom DAGs and logging enabled.

MWAA is used for orchestrating DAG-based workflows in HCL CDP. To create MWAA environment, follow the steps below:

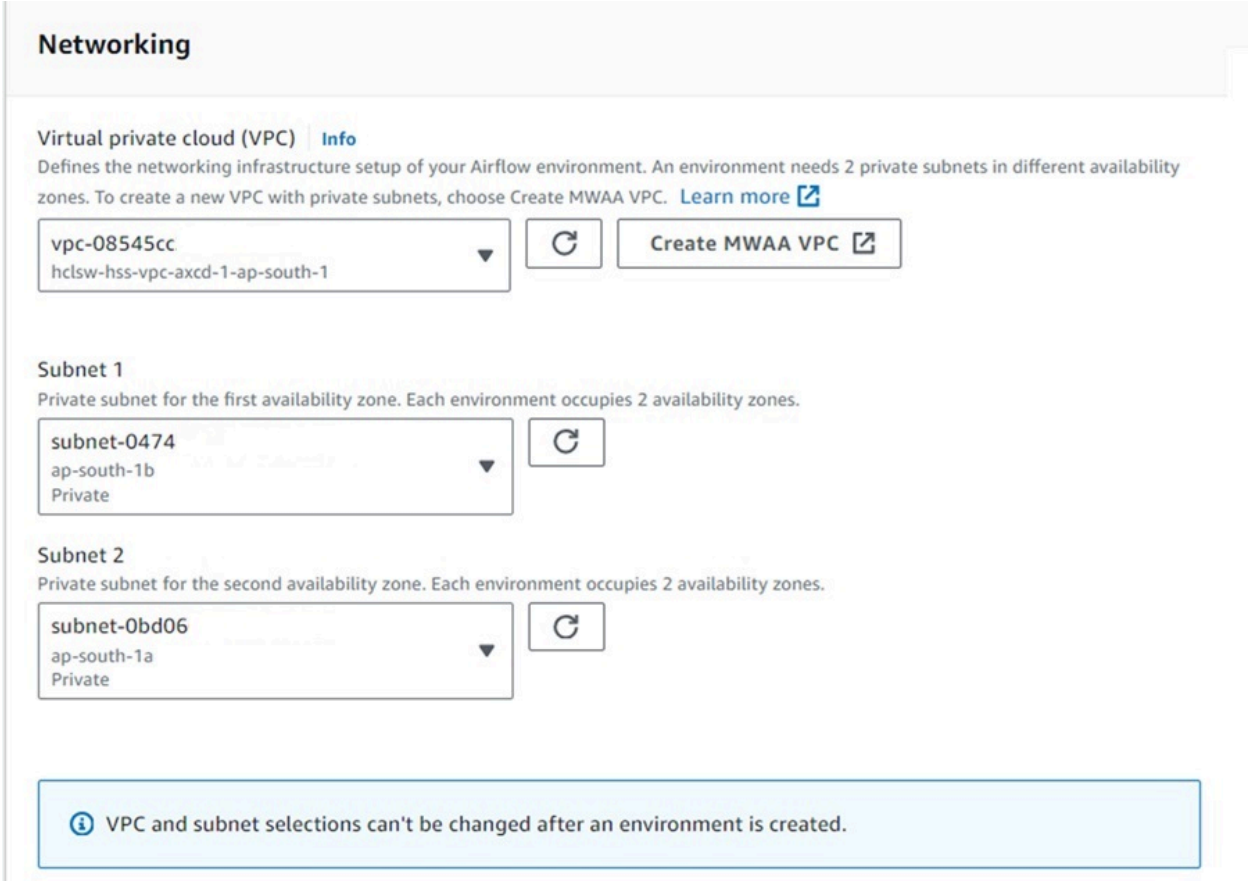
1. Open the Amazon MWAA console. Use the AWS Region selector to select your region.
2. Choose **Create environment**. On the **Specify details** page, under **Environment details**:
    a. Type a unique name for your environment in **Name**.
    b. Choose the Apache Airflow version in **Airflow version**.

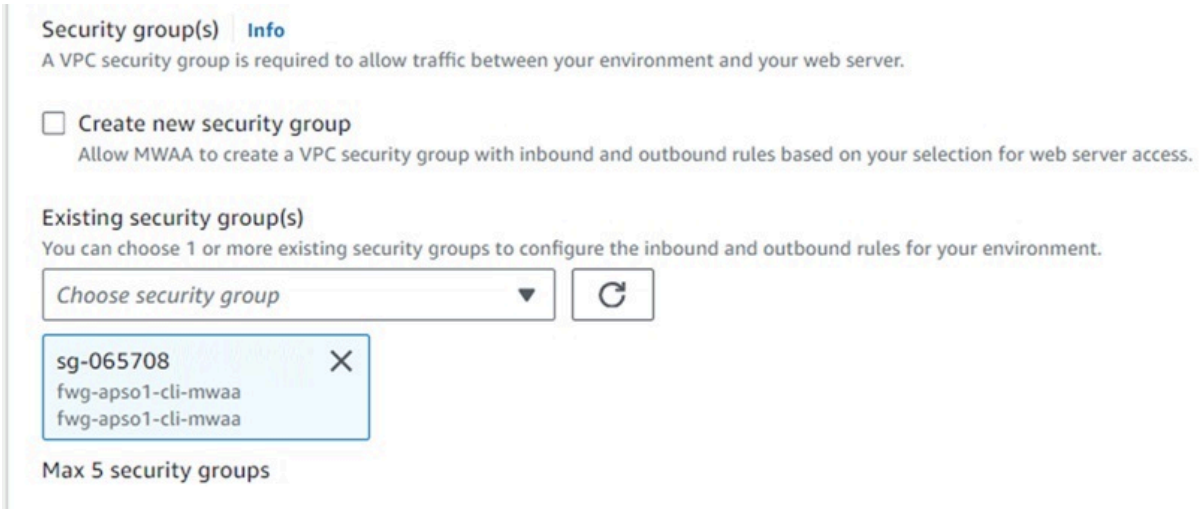


3. In the DAG code in S3 section, update the S3 Bucket location, DAGs, logs, and plugins.

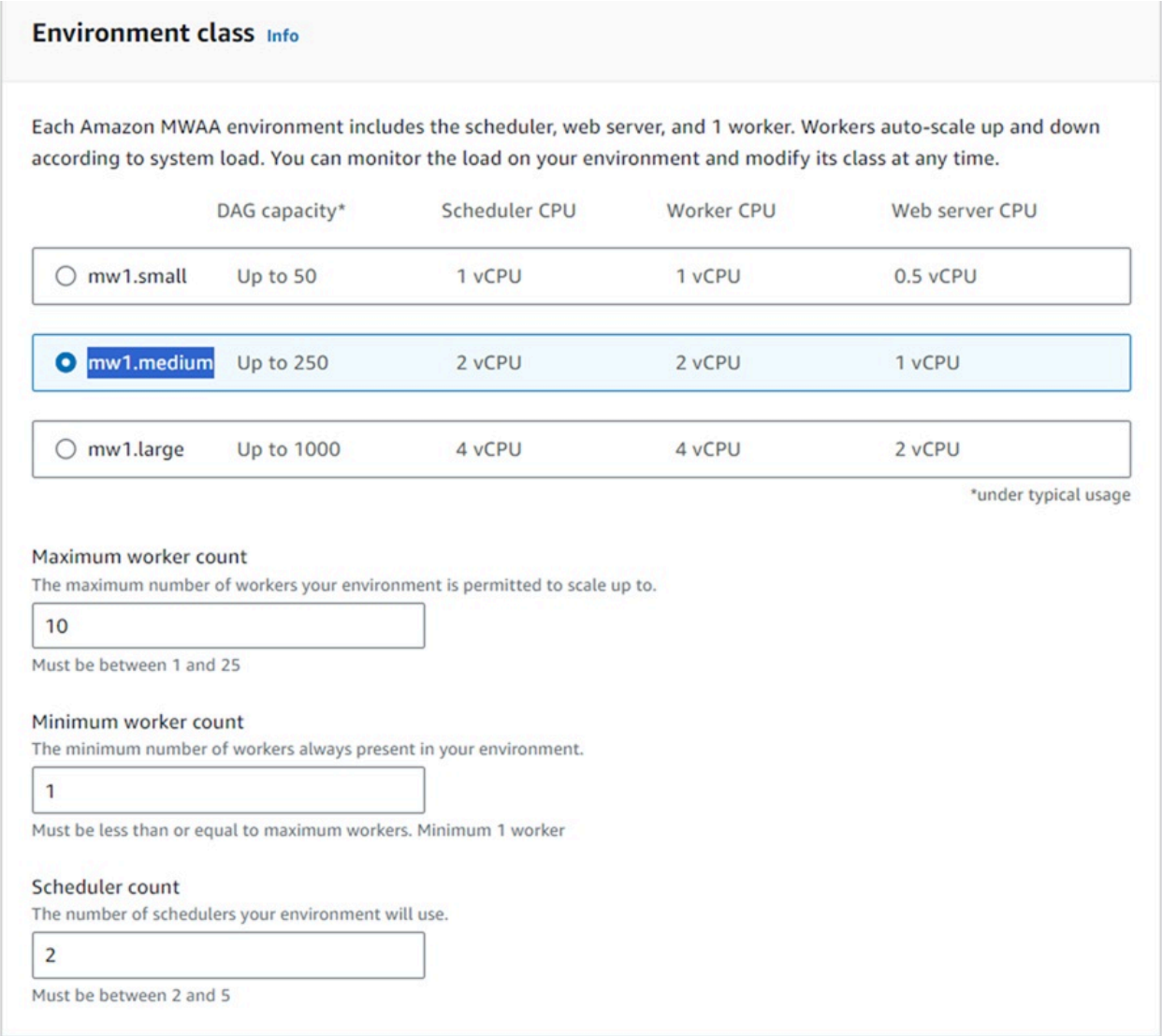


4. In the Networking section, select appropriate VPC, private subnets, and security groups.
5. In the Endpoint management, choose Service managed endpoints. This option is recommended.

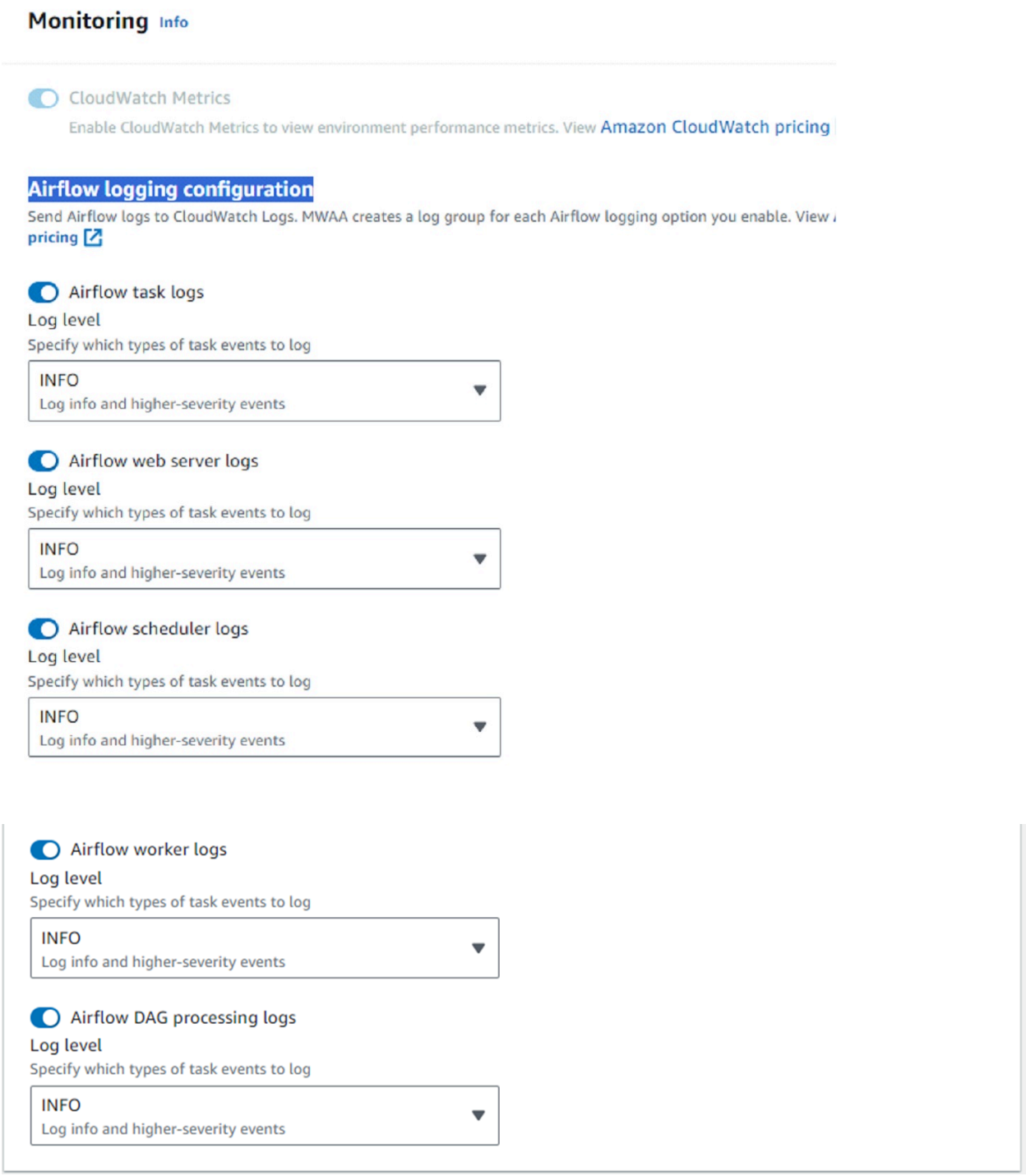




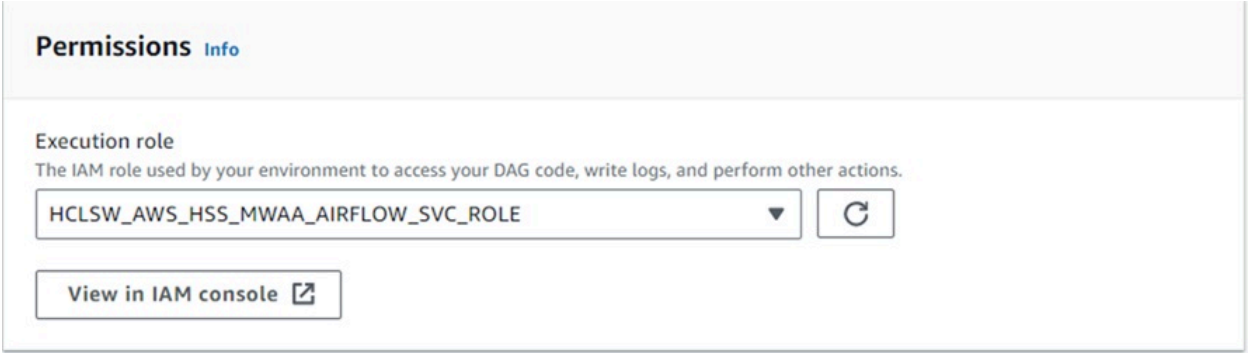
6. In the Environment class section, choose mw1.medium like shown below.



7. In the Monitoring section, enable all the task, worker, scheduler, and web server logs.



8. In the Permissions section, choose Execution Role as shown below and attach IAM role with S3, MWAA, CloudWatch, and KMS permissions.



9. Review and click **Create environment**.

Installing Postgres

PostgreSQL is deployed with master-slave replication using Docker Compose. Schema setup includes essential tables and indexes.

PostgreSQL is used as the transactional database in HCL CDP. The setup includes master-slave replication, schema creation, and Prometheus monitoring integration.

Installing PostgreSQL on Master VM

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

1. Install Docker and compose.

```
yum update -y
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
yum install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
systemctl start docker
systemctl enable 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
```

2. Directory Structure.

```
systemctl status docker
ln -s /data /disk1
mkdir -p /disk1/postgress
mkdir -p /disk1/postgress/data
```

3. Create the Docker Compose File as shown below.

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

```
version: "3"
services:
  postgres:
    image: postgres:15.3
```

```
    restart: always
    environment:
      POSTGRES_USER: <username>
      POSTGRES_PASSWORD: <password>
      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://<username>:<password>@<vmip>:5432/analytics?sslmode=disable"
    links:
      - postgres

volumes:
  db-data:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /disk1/postgress/data
```

4. Start Services.

```
cd /home/kgunda/postgress
docker-compose up -d
docker ps
```

Installation on Slave Host

Repeat the same Docker install steps and create the same volume and directory structure. Then, use this `docker-compose.yaml` with adjusted passwords and memory.

```
postgres:
  shm_size: 58gb
postgres-exporter:
  DATA_SOURCE_NAME: "postgresql://<username>:<password>@<master-ip>:5432/analytics?sslmode=disable"
```

```
cd /home/kgunda/postgress
docker-compose up -d
```

Replication Configuration

On Master:

```
psql -U <username> -W -d analytics -h localhost -p 5432
```

On Slave

1. System updates and dependencies.

```
yum update -y
yum install -y yum-utils
```

2. Add Docker repo and install Docker and Compose.

```
yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
yum install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

3. Start and verify Docker.

```
systemctl start docker
systemctl enable docker
systemctl status docker
```

4. Install Docker Compose.

```
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
```

5. Create directory structure for PostgreSQL.

```
ln -s /data /disk1
mkdir -p /disk1/postgress
mkdir -p /disk1/postgress/data
```

6. Create a docker composer file as shown below.

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

```
version: "3"
services:

  postgres:
    image: postgres:15.3
    restart: always
    environment:
      POSTGRES_USER: <username>
      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://<username>: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
```

7. Start PostgreSQL and Exporter.

```
cd /home/kgunda/postgress/
docker-compose up -d
docker ps
```

8. Access the PostgreSQL Container.

```
psql -U <username> -W -d analytics -h localhost -p 5432
```

9. After the Postgress installation is completed on both Master and Slave, whitelist the 5432 and 9187 ports from Master and Slave hosts (Inbound and Outbound).

PostgreSQL Replication and Schema Setup

1. Replication Steps - Master.

```
psql -U <username> -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;

# Edit the /disk1/postgress/data/postgresql.conf file:
max_connections =  1000
wal_level = replica
max_wal_size = 100GB
max_wal_senders = 10
wal_keep_size = 3000

# Edit the /disk1/postgress/data/pg_hba.conf file:
host    replication
         replication     <Slave Host IP>/0   md5

cd /home/kgunda/postgress/
systemctl stop docker; systemctl start docker

       docker-compose down
docker-compose up -d
docker ps
```

2. Replication Steps - Slave.

```
psql -U <username> -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;

# Edit the /disk1/postgress/data/postgresql.conf:
max_connections = 1000
```

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

# Edit /disk1/postgress/data/pg_hba.conf:
host    replication     replication     <Master Host IP>/0   md5

cd /disk1/postgress/data
rm -rfv *
```

3. 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

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

4. Restart the postgress DB on Slave.

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

docker-compose down
docker-compose up -d
```

5. Validate Replication.

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

6. Create required tables and indexes.

```
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 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 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 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 SEQUENCE source_outbound_id_seq;

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 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 SEQUENCE source_failure_id_seq;

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 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 SEQUENCE destination_outbound_id_seq;

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)
);

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);
```

RDS DB Configuration

This step sets up the MySQL database `vrm` on your RDS instance (or VM-based MySQL running on port `3307`).

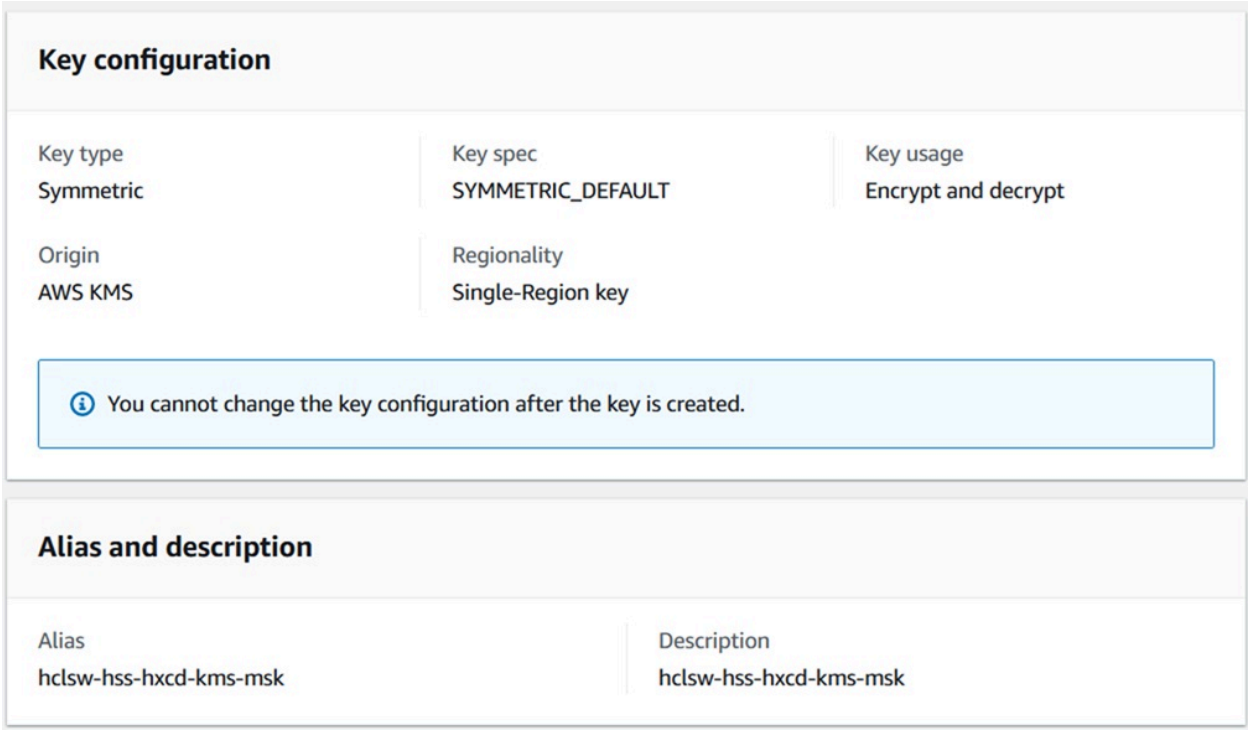
```
# Connect to MySQL using admin credentials
mysql -u admin -h <vm_ip> -P 3307

# Create the database
create database vrm;
```

Configuring AWS KMS

AWS Key Management Service (KMS) is used to create and manage encryption keys. These are used with Secrets Manager and other encrypted services.

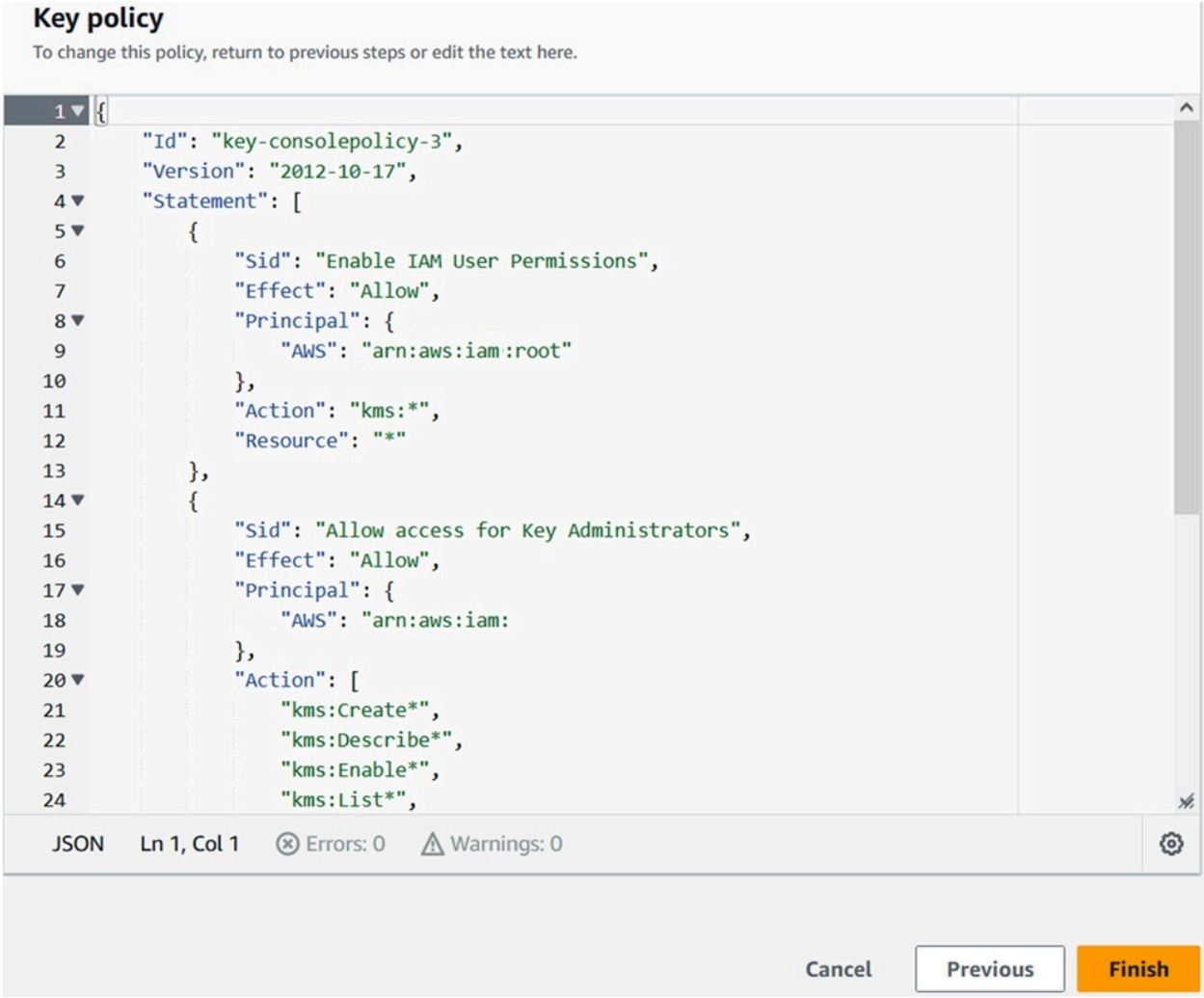
AWS KMS is used to manage encryption keys for secure storage and communication within CDP. To configure KMS, follow the steps below:

1. In the AWS KMS console, to change the AWS Region, use the Region selector in the upper-right corner of the page.
2. In the navigation pane, choose **Customer managed keys**.
3. Choose **Create key**. To create a symmetric encryption KMS key, for **Key type** choose **Symmetric**.
4. In **Key usage**, the **Encrypt and decrypt** option is selected for you.
5. Choose **Next**, and enter alias name and description for the key.



6. Choose **Next**, and select the IAM users and roles that can administer the KMS key.\

7. Choose **Next**, and review the key policy statements for the key. To make changes to the key policy, select **Edit**.



8. Click **Next** and Finish.

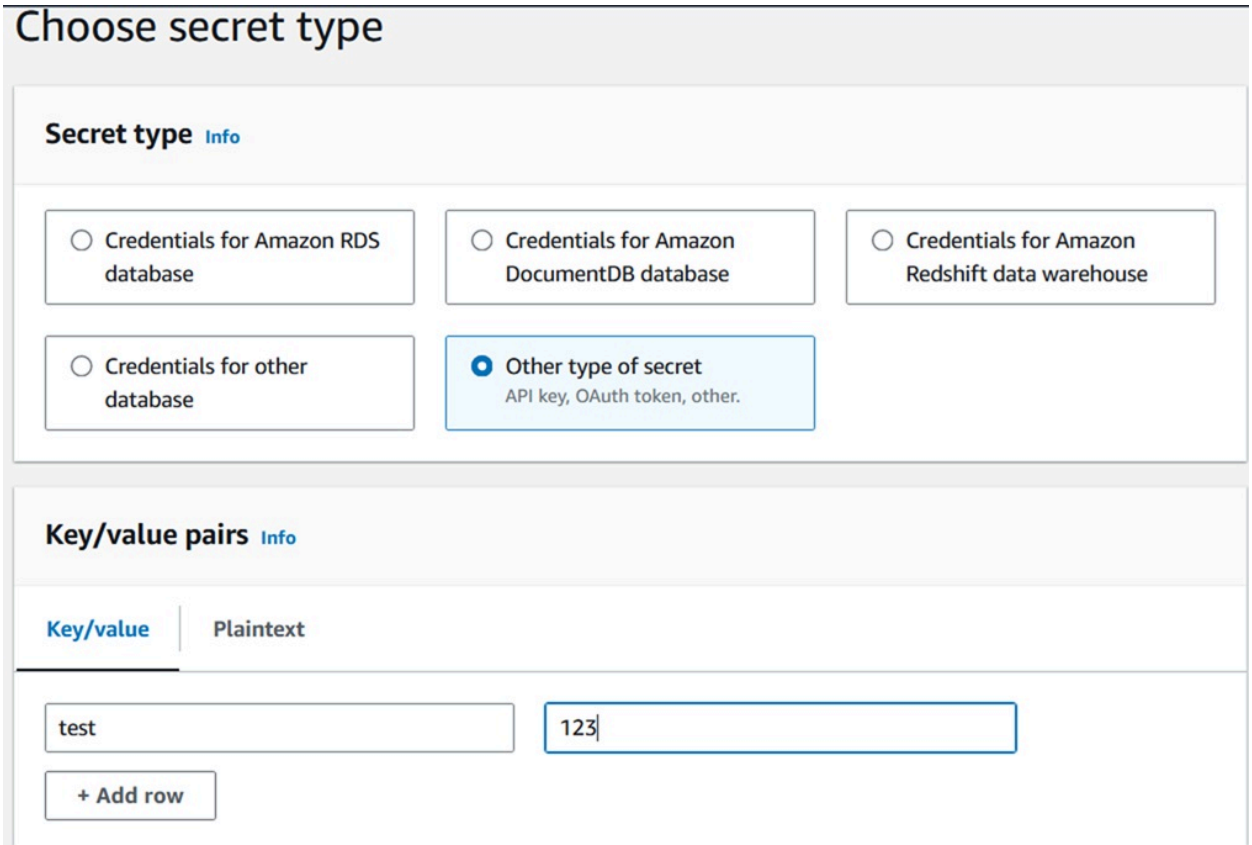
Configuring AWS Secrets

AWS Secrets Manager stores sensitive credentials like DB passwords and Kafka tokens, encrypted with KMS.

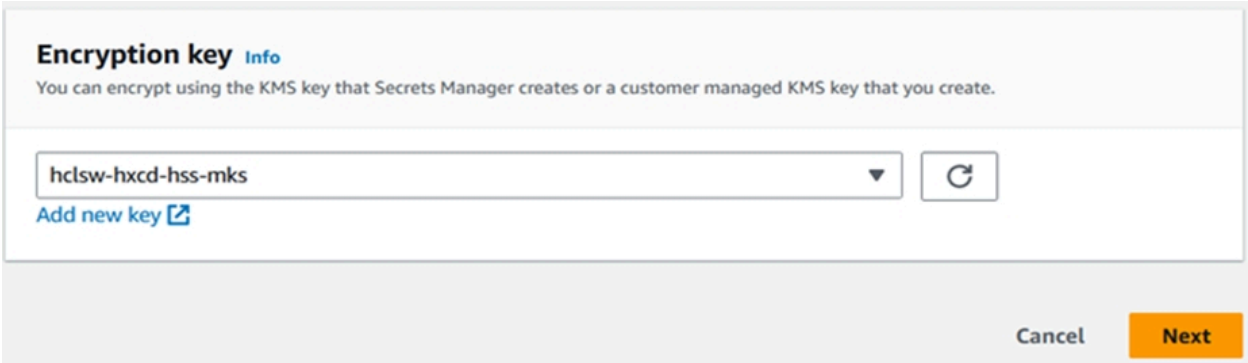
Secrets Manager stores sensitive information such as database passwords and Kafka credentials.

To store custom secrets, follow the steps below:

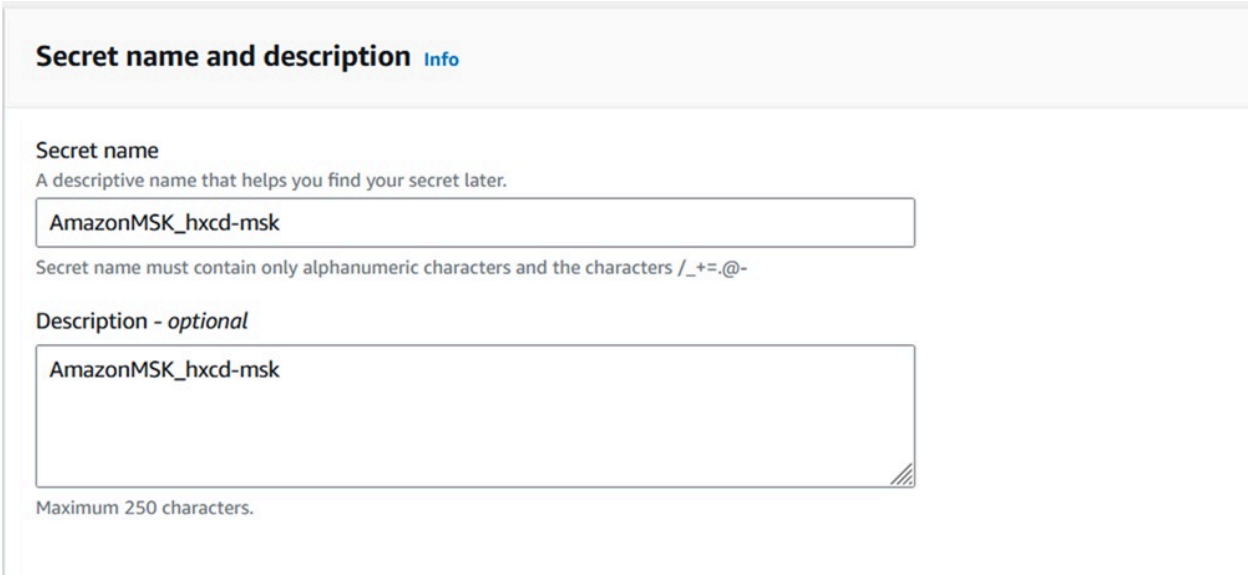
1. In the console, choose **Store a new secret**, select the secret type as Other type of secrets.
2. In **Key/value pairs**, either enter your secret in JSON **Key/value** pairs, or choose the **Plaintext** tab and enter the secret in any format.



3. Select custom KMS key as shown below.



4. Enter secret name and description as shown below.



5. On the **Review** page, review your secret details, and then choose **Store**.

DP Monitoring Applications Configuration

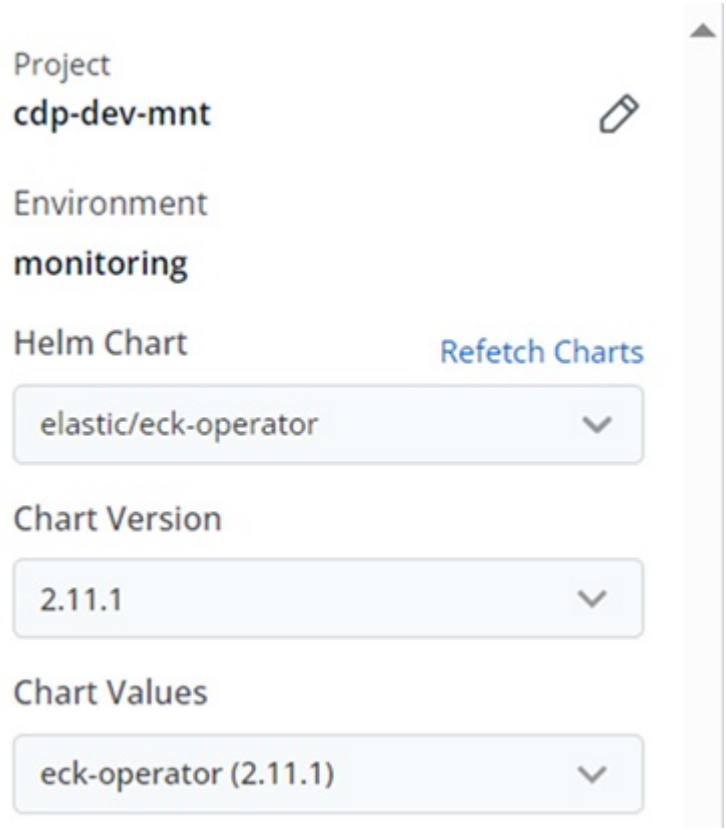
Monitoring includes Prometheus, Grafana, Node Exporter, Mongo Exporter, and Fluentbit for log aggregation.

These applications are installed using **Devtron's Chart Store** to support observability across your CDP stack.

eck-operator

- **Chart**: `elastic/eck-operator`
- **Version**: `2.11.1`
- **Steps**:

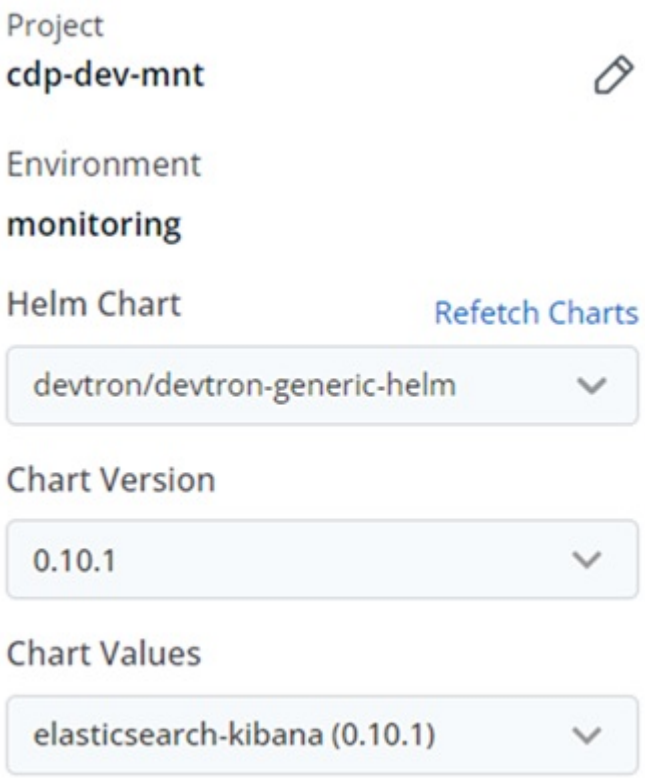
- Navigate to: `Chart Store → elastic → eck-operator`.
- Click **Configure & Deploy**.



elasticsearch-kibana

- **Chart**: `generic`
- **Steps**:
  - Go to `Chart Store → generic`.
  - Select the Elasticsearch-Kibana chart.

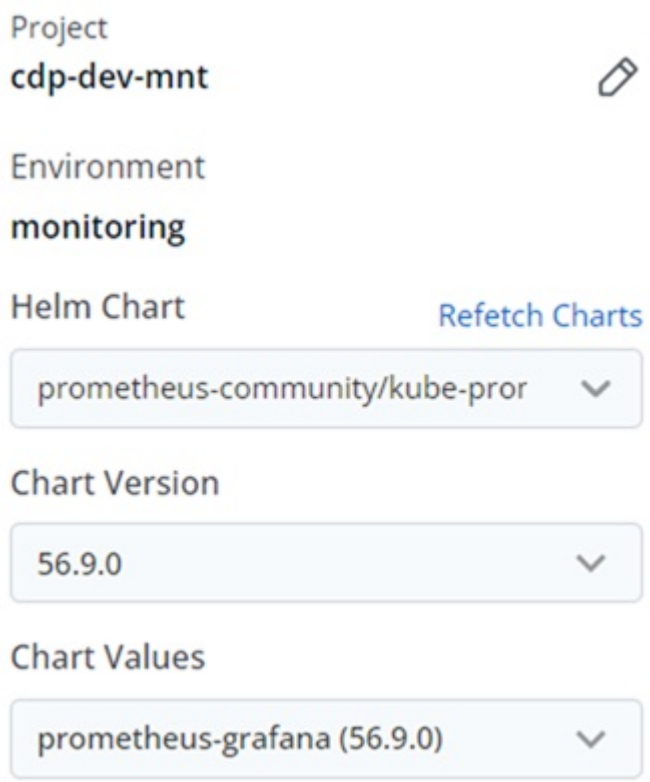
- Click **Configure & Deploy**.



prometheus-grafana

- **Chart**: `prometheus-community/kube-prometheus-stack`
- **Version**: `56.9.0`
- **Steps**:
  - Go to `Chart Store → prometheus-community`.
  - Select `kube-prometheus-stack`.

- Click **Configure & Deploy**.



Fluentbit

- **Chart**: `generic`
- **Steps**:
  - Navigate to `Chart Store → generic`.
  - Select `Fluentbit` chart.
  - Click **Deploy Chart**.

Chapter 2. Deploying HCL CDP Core Components on AWS

This section details the deploying core components required for HCL CDP on the AWS platform.

Introduction

This section details the installation and configuration of the services and components required by HCL CDP core components deployment using devtron, on the AWS platform.

Prerequisites

Micro-services based HCL CDP core components, to be deployed on containerized platform using Devtron CI/CD tool.

Before proceeding with the installation of the required components, which have mention in deployment guide.

Below tools are expected to present in the deployment to facilitate the installation:

- **Kubernetes CLI:** Verify the availability of the `kubectl` command.
- **Helm CLI:** Ensure the `helm` command-line tool is installed.
- **Devtron Deployment Tool:** Confirm that Devtron is set up for managing deployments.

List of CDP Core components for AWS EKS

List of HCL CDP Core modules components to be deployed and configured. CDP utilizes the functionalities from these modules to perform the end to end operations.

Application/Services	**Deployment Type**
CDP Admin UI	Helm Chart
CDP Admin UI Backend	Helm Chart
CDP Core API	Helm Chart
CDP Dash Backend	Helm Chart
CDP Dash Iframe	Helm Chart
CDP DMP SST	Helm Chart
CDP DMP SST Maps	Helm Chart
CDP DMP SST API	Helm Chart
CDP DMP SST Zookeeper	Helm Chart
CDP Web App	Helm Chart
CDP Live Events	Helm Chart
CDP Manager	Helm Chart

Application/Services	Deployment Type
CDP Pixel Listener	Helm Chart
CDP RTS	Helm Chart
CDP RTS MS	Helm Chart
CDP DI API	Helm Chart
CDP Psql API	Helm Chart
CDP S3 Sink Connector	Helm Chart
CDP MongoDB Sink Connector	Helm Chart
CDP KMS Service	Helm Chart
DMP Producer	Helm Chart
NIFI	VM Based Deployment

Installing Devtron

This section provides detailed instructions on how to deploy Devtron to your AWS EKS cluster using Helm charts.

To install Devtron, follow these steps:

1. Add the Devtron repository to the helm chart sources.

```
helm repo add devtron https://helm.devtron.ai
helm repo update devtron
```

2. Install the Devtron on the AWS EKS Cluster.

```
helm install devtron devtron/devtron-operator --create-namespace --namespace devtroncd --set
 installer.arch=multi-arch
```

Note: Before installing the devtron, make sure to allow appropriate permissions in SCC.

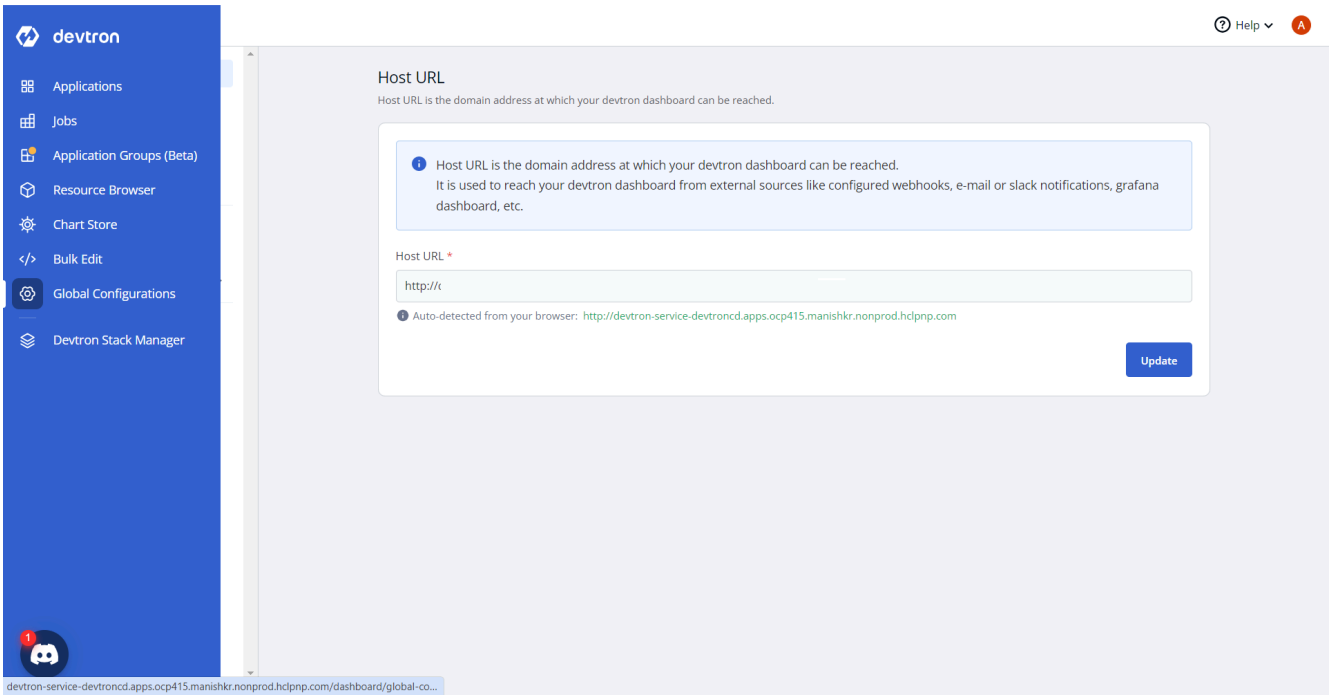
3. Get the Devtron login credentials. By default, the username will be admin.

```
kubectl -n devtroncd get secret devtron-secret -o jsonpath='{.data.ADMIN_PASSWORD}' | base64 -d
```

Configuring Devtron

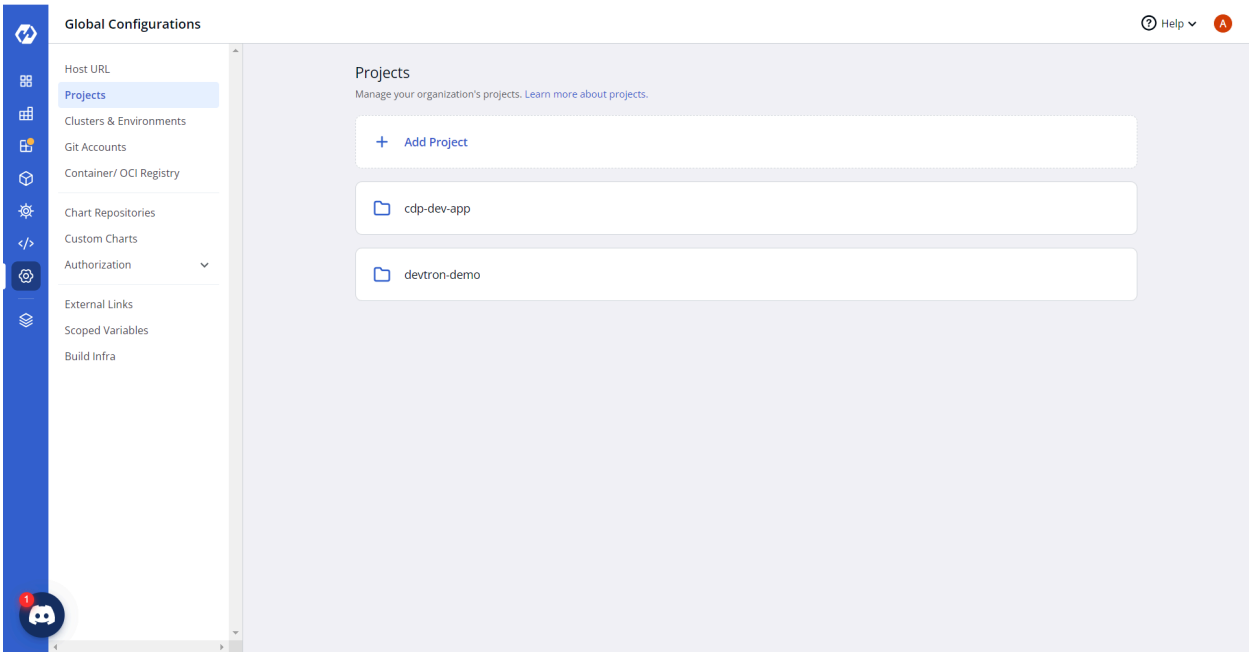
Prerequisites:

Before you start configuring the Devtron, verify the Global Configurations.

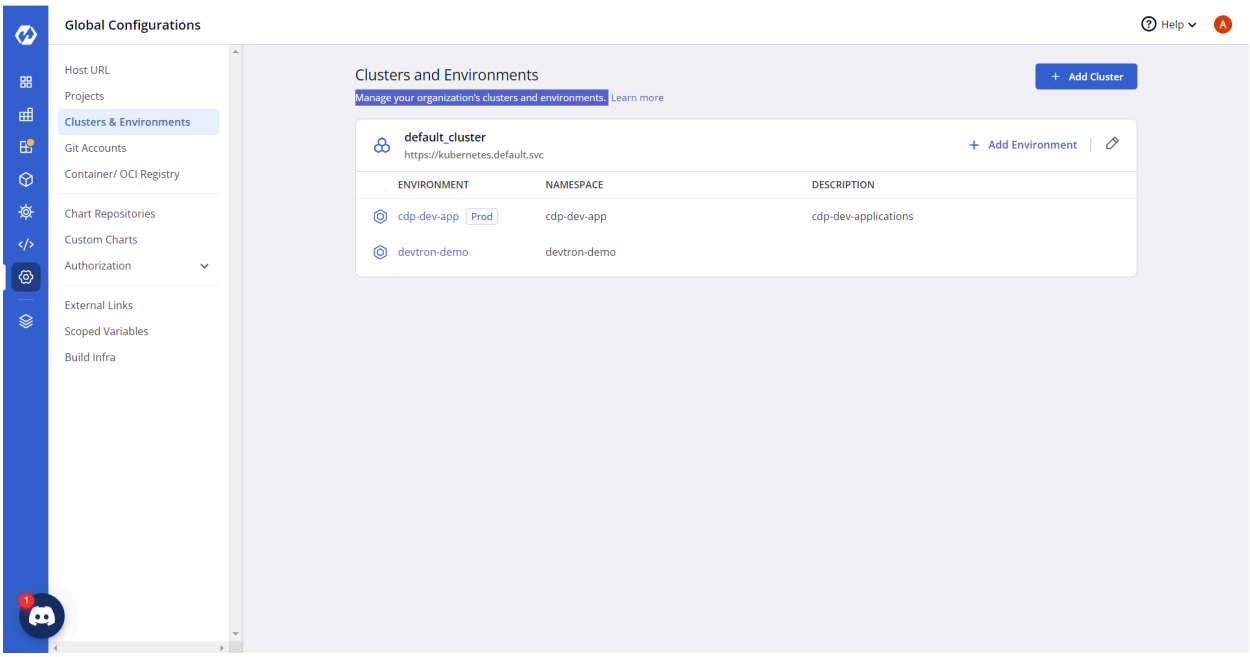


To configure the Devtron, follow the steps below:

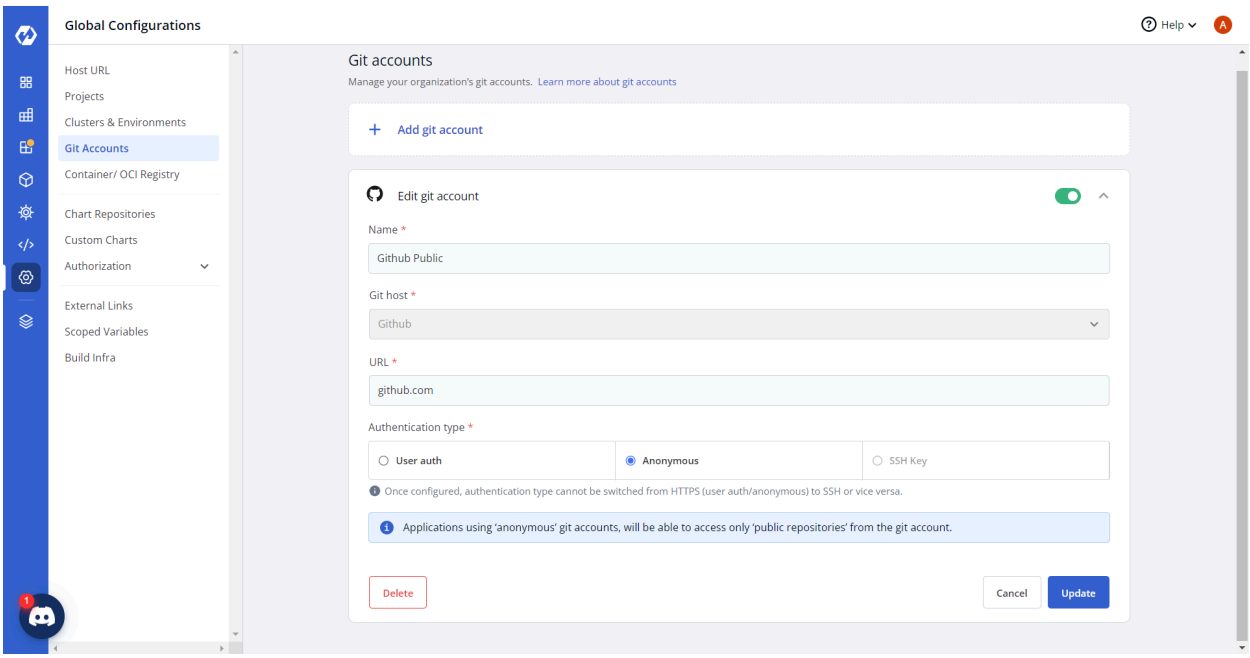
1. In the Devtron console, select **Global Configurations** > **Projects**. Click **Add Projects** and enter projects details.



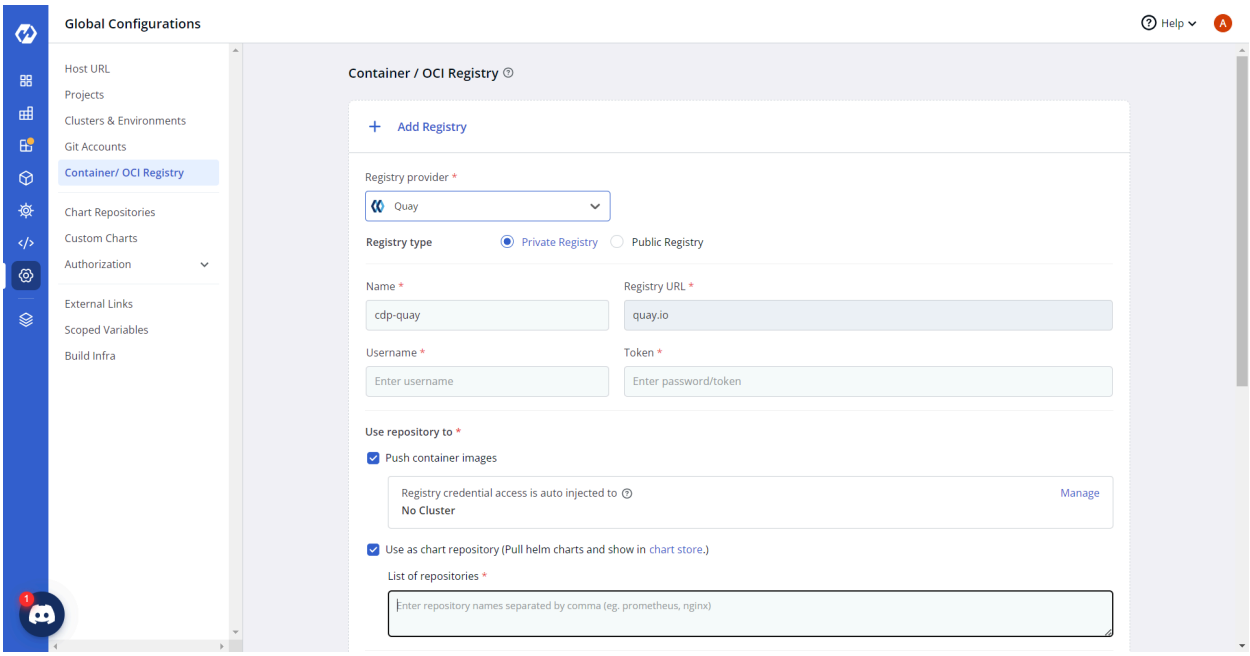
2. From the left pane, click **Clusters and Environment**, and manage your organizations clusters and environments details.



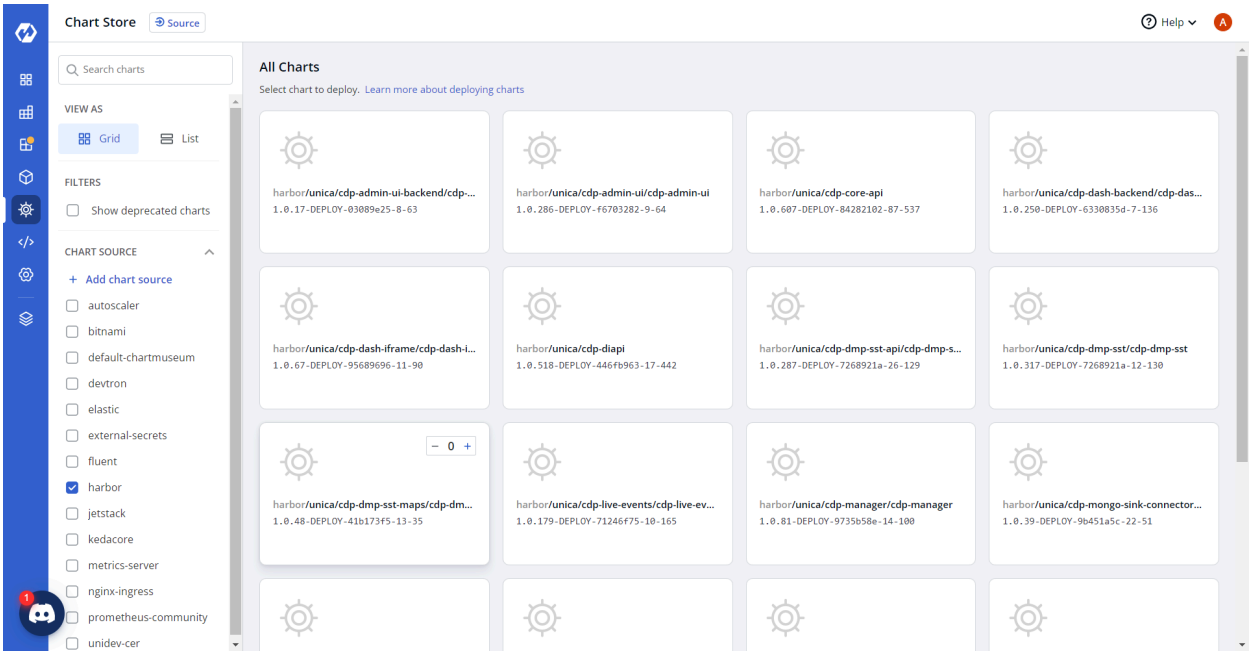
3. Click **Git Accounts**, and manage your organizations git accounts.



4. Click **Container/OCI Registry** > **Add Registry**, and update registry details and push helm charts and docker images to the configured registry.



5. In the registry configuration, update the list of repositories, and verify helm charts are synced to Devrton's chart store as shown below.



Configuring Ingress

To configure the ingress, change the LoadBalancer service to NodePort using the following:

```
kubectl patch -n devtroncd svc devtron-service -p '{"spec": {"ports": [{"port": 80,"targetPort":
 "devtron","protocol": "TCP","name": "devtron","nodePort": 30000}],"type": "NodePort","selector": {"app":
 "devtron"}}}'
```

Similarly, configure the ALB Ingress using:

```
kubectl apply -f Devtron-Ingress.yaml -n devtroncd
```

Note: If you are using any existing ALB, update the ACM Arn and HTTPS Redirection Annotations. If it is a new ALB, create ALB with only HTTP, add DNS and ACM Certs and upgrade them.

Deploying Core Components

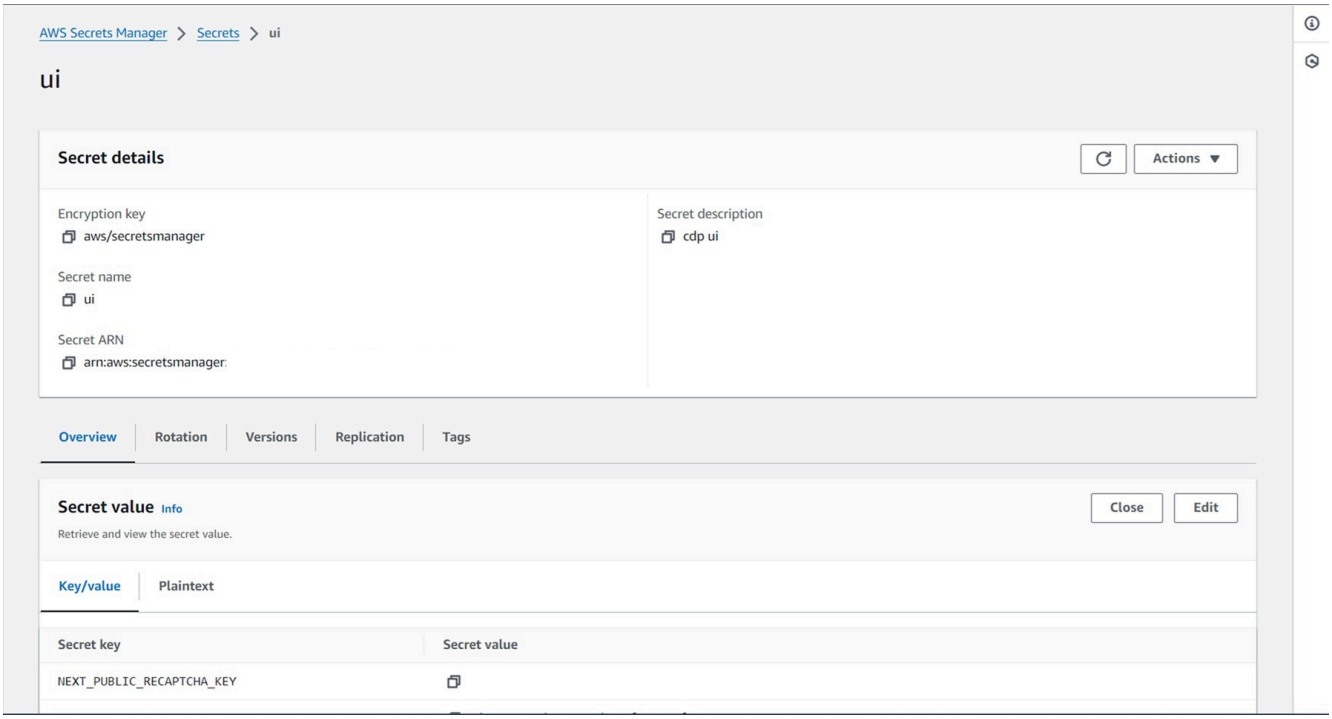
This section provides step-by-step instructions for deploying each service, component, and module required for HCL CDP Core Components. Follow the guidelines in each subsection to ensure the successful deployment and configuration of the respective elements.

Deploying CDP Admin UI

This section provides detailed instructions on how to deploy HCL CDP Admin UI using the Devtron in the AWS.

Prerequisites:

Make sure to create UI secret with required data in the AWS Secrets Manager before deploying CDP admin UI.



To create the UI secret in the AWS Secrets Manager, follow the steps below:

1. Create a UI secret sample key and value in the UI secret shown below, and update the actual values.

```
{
  "AKAMAI_HOST": "",
  "AKAMAI_KEY": "",
```

```
  "AKAMAI_KEY_NAME": "",
  "AKAMAI_PATH": "",
  "AKAMAI_PURGE_AUTH_KEY": "",
  "API_KEY_1": "",
  "API_KEY_2": "",
  "API_KEY_3": "",
  "API_KEY_4": "",
  "API_KEY_5": "",
  "API_KEY_6": "",
  "API_SECRET_1": "",
  "API_SECRET_2": "",
  "API_SECRET_3": "",
  "API_SECRET_4": "",
  "API_SECRET_5": "",
  "API_SECRET_6": "",
  "AUTH_TOKEN": "",
  "NEXT_PUBLIC_CLIENT_ID": "",
  "NEXT_PUBLIC_CLIENT_SECRET": "",
  "NEXT_PUBLIC_FROALA_SECRET": "",
  "NEXT_PUBLIC_GOOGLE_CLIENT_ID": "",
  "NEXT_PUBLIC_GOOGLE_CLIENT_SECRET": "",
  "NEXT_PUBLIC_RECAPTCHA_KEY": "",
  "REACT_APP_CLIENT_ID": "",
  "REACT_APP_S3_ACCESS_KEY_ID": "",
  "REACT_APP_S3_SECRET_KEY": "",
  "auAuthKey": "",
  "auEndpoint": "",
  "host": "",
  "muAuthKey": "",
  "muEndpoint": "",
  "password": "",
  "sesEmail": "",
  "sesPassword": "",
  "smtpFrom": "",
  "smtpHost": "",
  "smtpPassword": "",
  "smtpPort": "",
  "smtpUser": "",
  "usAuthKey": "",
  "usEndpoint": "",
  "user": ""
}
```

2. Create an AdminUI Secret to fetch the secrets from the vault.

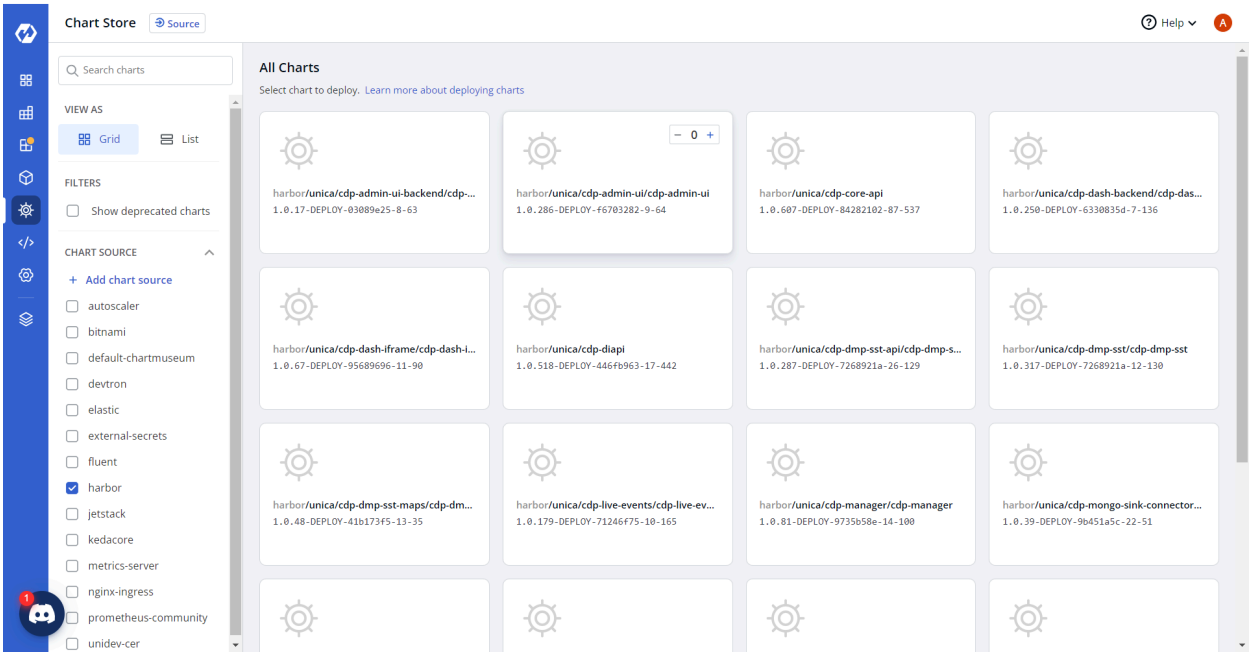
```
{
  "NEXT_PUBLIC_RECAPTCHA_KEY": "",
  "NEXT_PUBLIC_CLIENT_ID": "",
  "NEXT_PUBLIC_CLIENT_SECRET": "",
  "NEXT_PUBLIC_GOOGLE_CLIENT_ID": "",
  "NEXT_PUBLIC_GOOGLE_CLIENT_SECRET": "",
  "NEXT_PUBLIC_FROALA_SECRET": "",
  "REACT_APP_CLIENT_ID": "",
  "host": "",
  "user": "",
  "password": "",
  "sesEmail": "",
  "sesPassword": "",
```

```
  "REACT_APP_S3_ACCESS_KEY_ID": "",
  "REACT_APP_S3_SECRET_KEY": "",
  "AKAMAI_KEY": "",
  "AKAMAI_KEY_NAME": "",
  "AKAMAI_HOST": "",
  "AKAMAI_PATH": "",
  "AKAMAI_PURGE_AUTH_KEY": "",
  "usEndpoint": "",
  "muEndpoint": "",
  "muAuthKey": "",
  "auAuthKey": "",
  "auEndpoint": "",
  "usAuthKey": "",
  "API_KEY_1": "",
  "API_SECRET_1": "",
  "API_KEY_2": "",
  "API_SECRET_2": "",
  "API_KEY_3": "",
  "API_SECRET_3": "",
  "API_KEY_4": "",
  "API_SECRET_4": "",
  "API_KEY_5": "",
  "API_SECRET_5": "",
  "API_KEY_6": "",
  "API_SECRET_6": "",
  "smtpUser": "",
  "smtpPassword": ":tn",
  "smtpHost": "",
  "smtpPort": "",
  "smtpFrom": "",
  "AUTH_TOKEN": ""
}
```

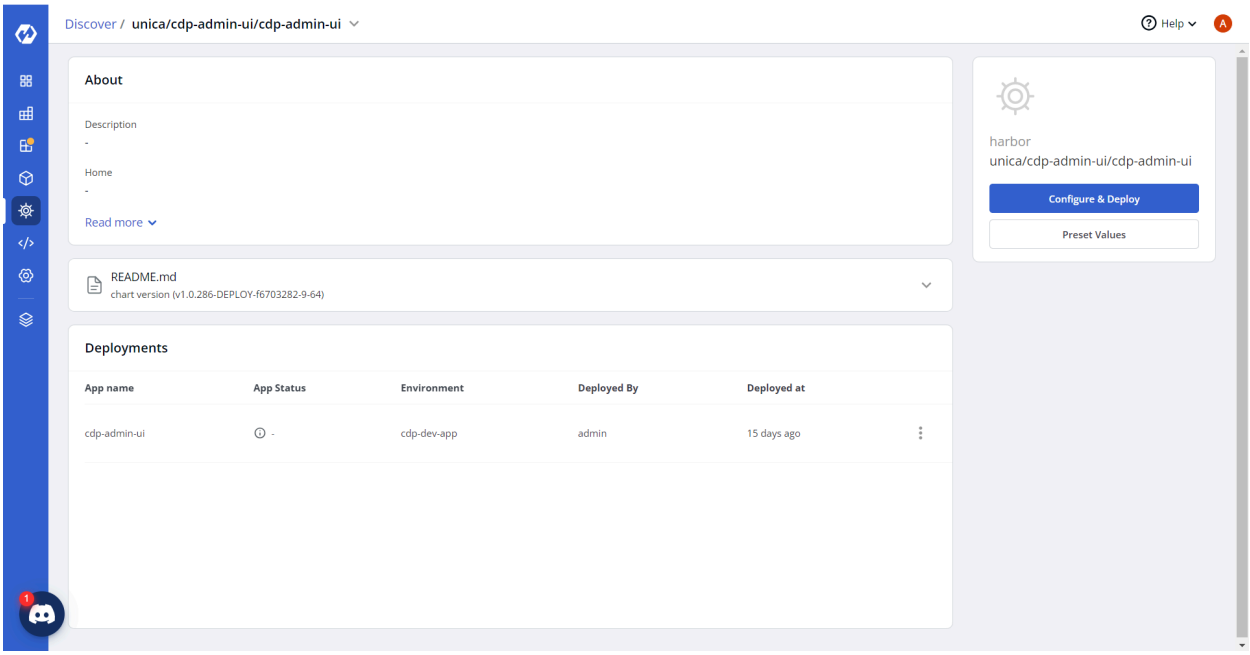
Deploying CDP Admin UI

To deploy the CDP admin UI, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-admin-ui chart to deploy.

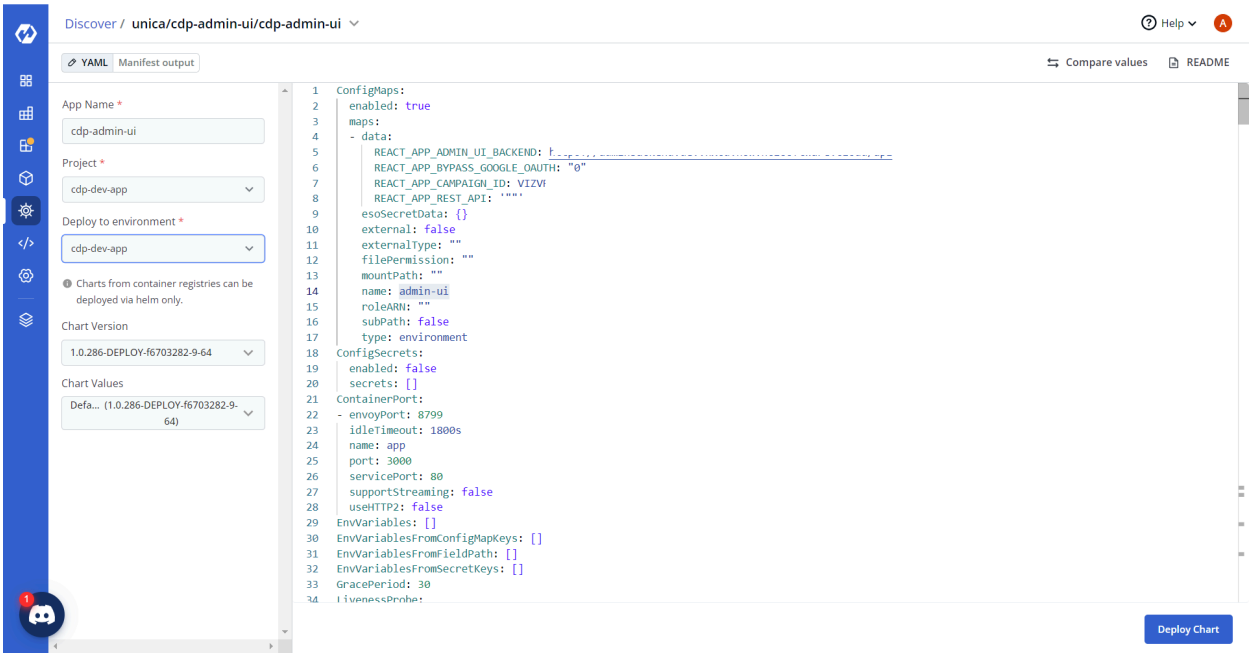


2. Now, configure and deploy the admin UI charts.

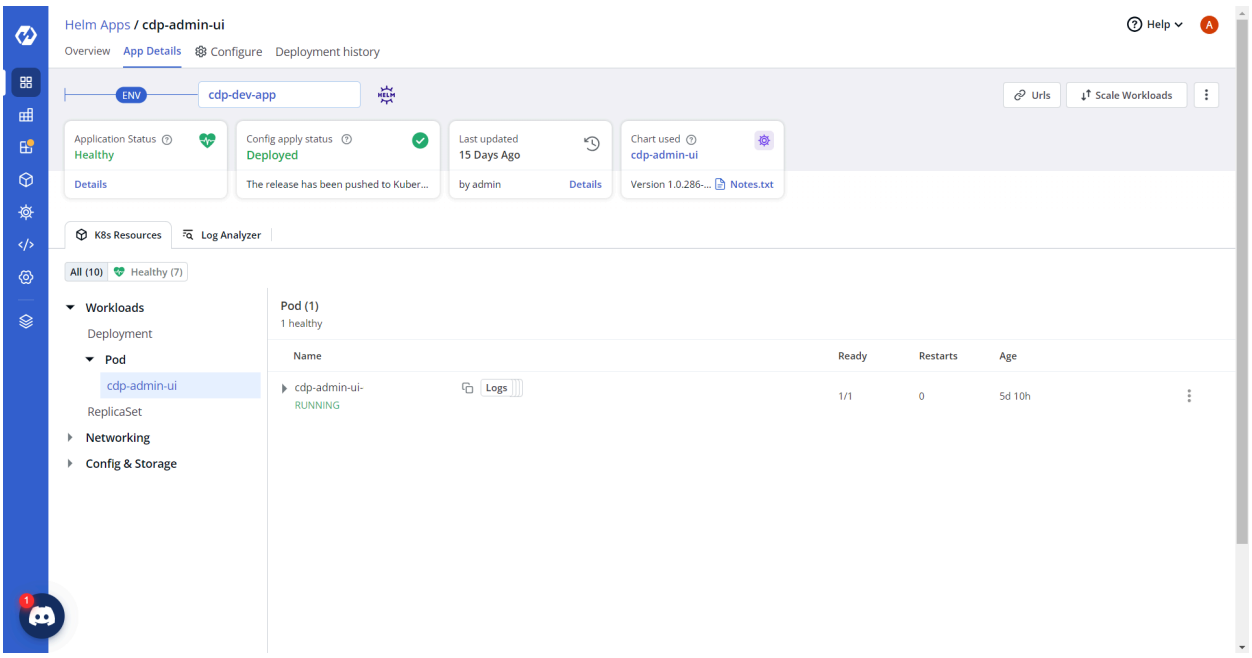


3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
REACT_APP_ADMIN_UI_BACKEND: <ADMIN_UI_BACKEND_URL>
REACT_APP_BYPASS_GOOGLE_OAUTH: "0"
REACT_APP_CAMPAIGN_ID: <VIZVRM6053>
REACT_APP_REST_API: '""'
```



4. On successful deployment, validate the deployment as shown below.

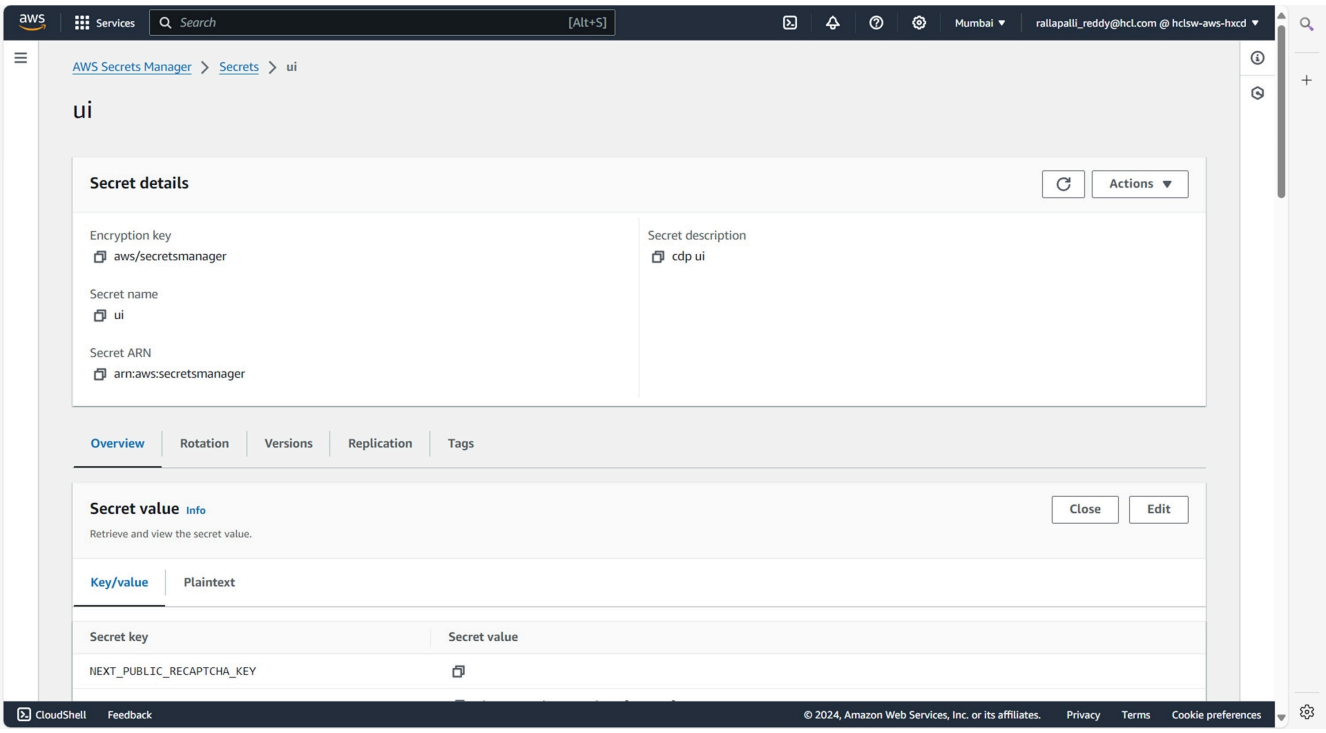


Deploying CDP Admin UI Backend

This section provides detailed instructions on how to deploy HCL CDP Admin UI Backend using the Devtron in the AWS.

Prerequisites:

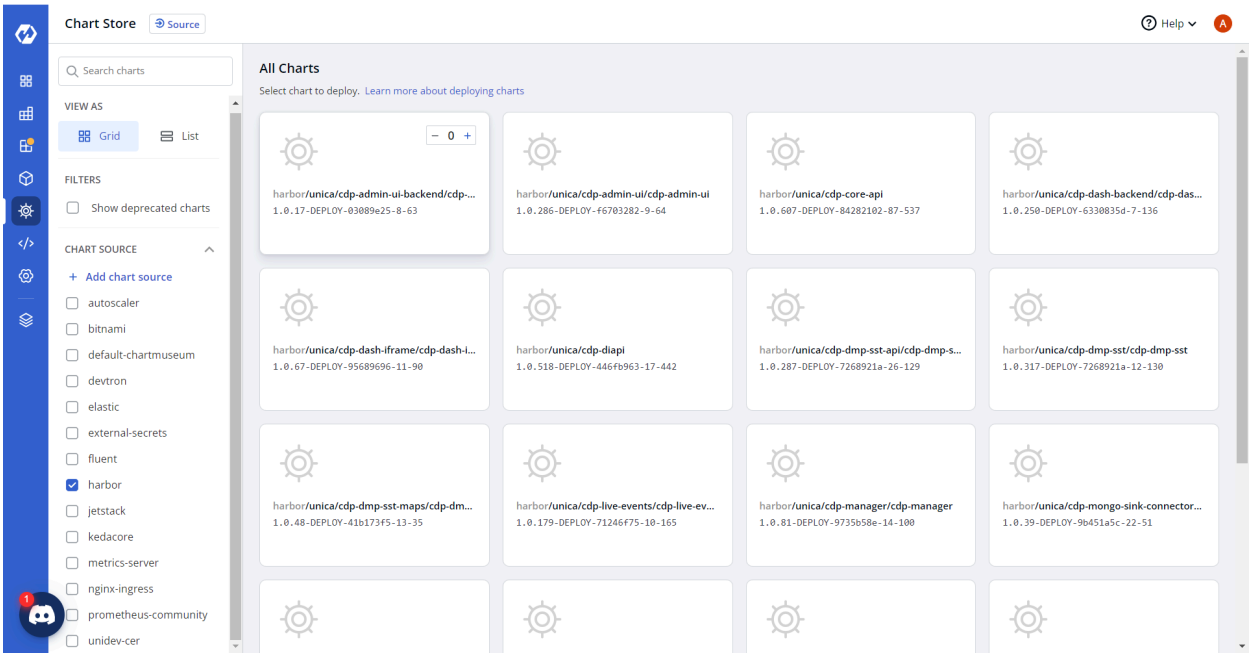
Make sure to create UI secret with required data in the AWS Secrets Manager before deploying CDP admin UI.



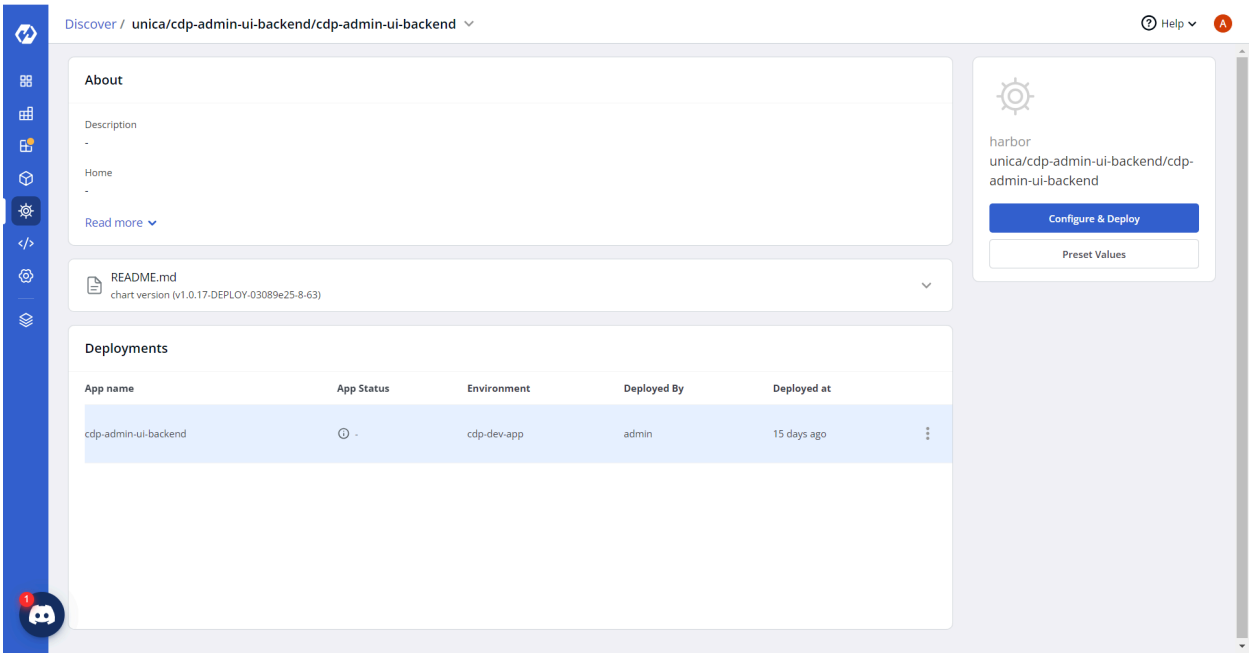
Deploying CDP Admin UI Backend

To deploy the CDP admin UI backend, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-admin-ui-backend chart to deploy.



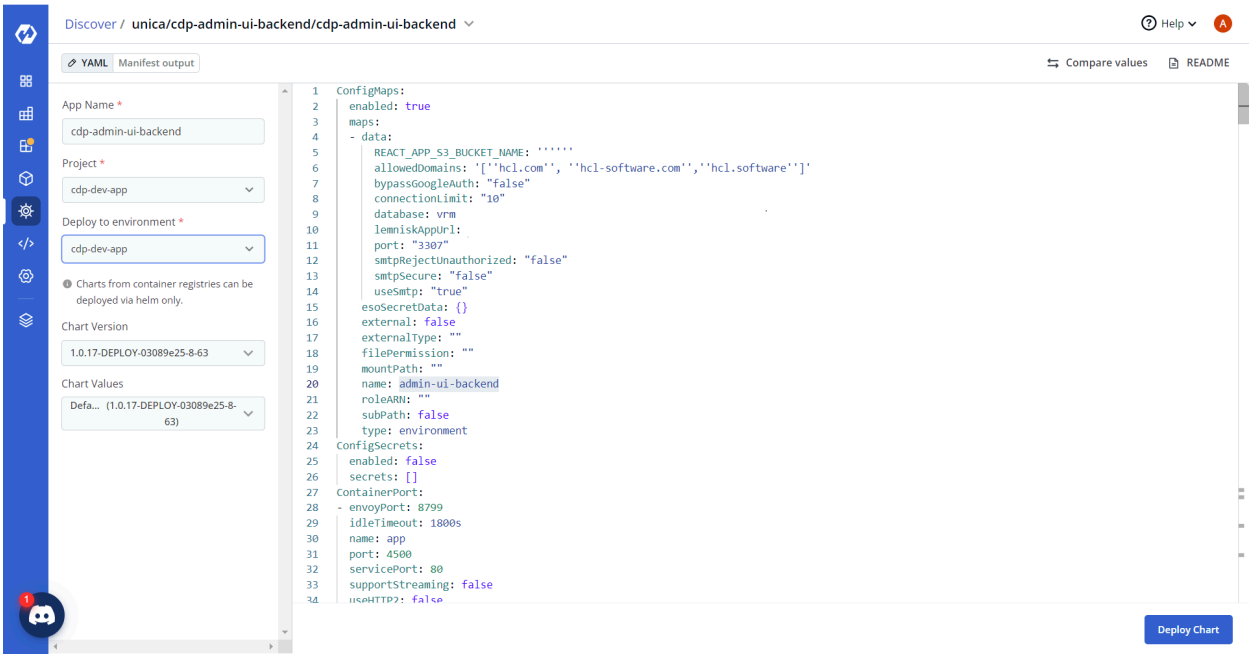
2. Now, configure and deploy the admin UI charts.



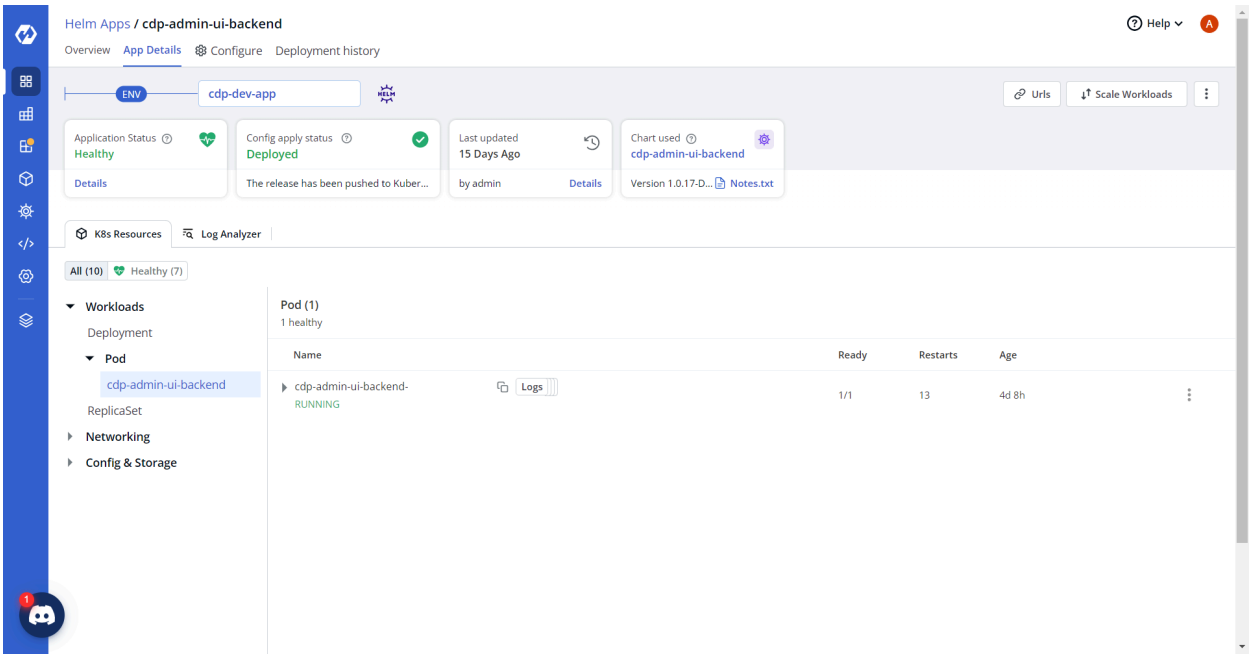
3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
REACT_APP_S3_BUCKET_NAME: ''''''
allowedDomains: '[''hcl.com'', ''hcl-software.com'',''hcl.software'',''hclpnp.com'']'
bypassGoogleAuth: "false"
connectionLimit: "10"
database:< MySQL_DB_Name>
lemniskAppUrl: <WebApp_URL>
port: "<3306>"
smtpRejectUnauthorized: "false"
```

```
smtpSecure: "false"
useSmtp: "true"
```



4. On successful deployment, validate the deployment as shown below.

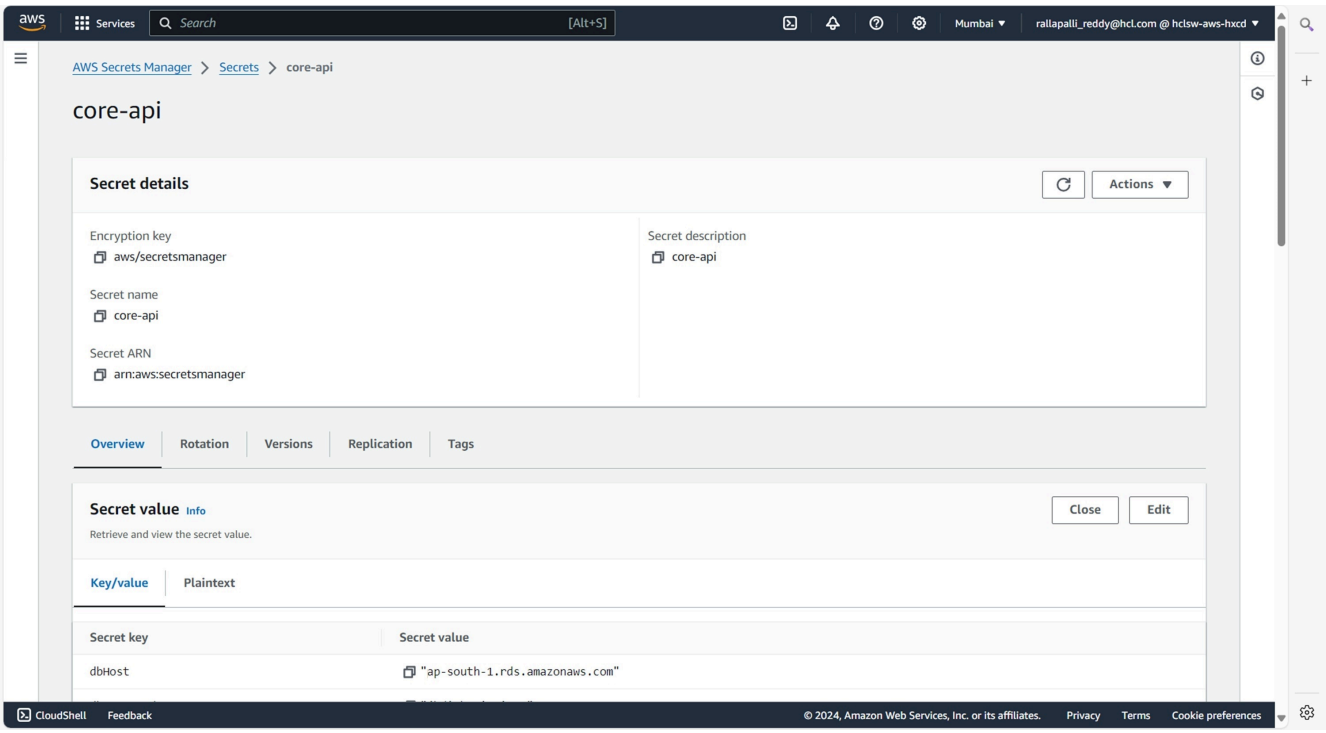


Deploying CDP Core API

This section provides detailed instructions on how to deploy HCL CDP Core API using the Devtron in the AWS.

Prerequisites:

Make sure to create core-api secret with required data in the AWS Secrets Manager before deploying CDP core-api.



To create the core-api secret in the AWS secret manager, follow the steps below:

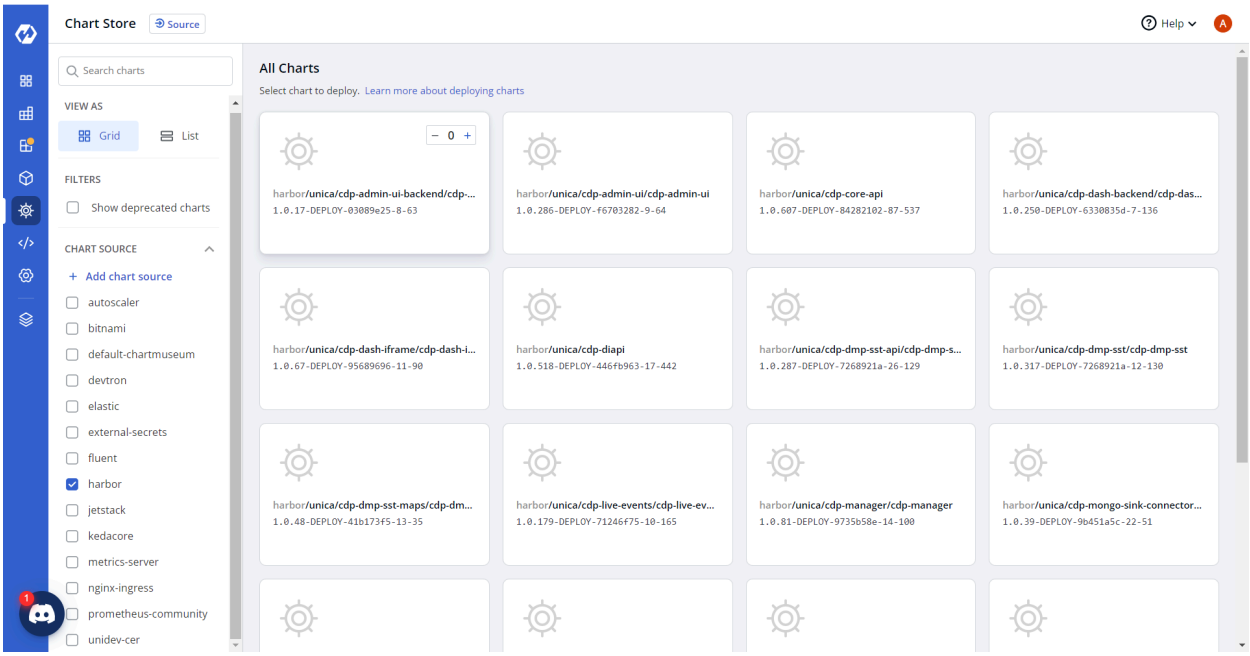
1. Create a core-api secret sample key and value in the core-api secret, and update ConfigMaps data with actual values.

```
{
  "X_API_KEY": "\"\"",
  "X_API_SECRET": "\"\"",
  "akamaiHost": "\"\"",
  "akamaiKey": "\"\"",
  "akamaiKeyName": "\"\"",
  "akamaiPath": "\"\"",
  "akamaiPurgeauthkey": "\"\"",
  "amqpAuthMechanism": "\"\"",
  "amqpHost": "\"\"",
  "amqpLogin": "\"\"",
  "amqpPassword": "\"\"",
  "amqpPort": "\"\"",
  "dbConnectionLimit": "\"80\"",
  "dbDatabase": "\"<dbName>\"",
  "dbHost": "\"<dbHost>\"",
  "dbPassword": "\"<dbPassword>\"",
  "dbPort": "\"<dbPort>\"",
  "dbUser": "\"<dbUser>\"",
  "lemnisk_mailId": "\"\"",
  "lemnisk_mailPassword": "\"\"",
  "mailId": "\"\"",
  "mailPassword": "\"\"",
  "stagingS3AccessKey": "\"\"",
  "stagingSecretAccessKey": "\"\"",
  "userIdentifierAuthKey": "\"<userIdentifierAuthKey>\""
}
```

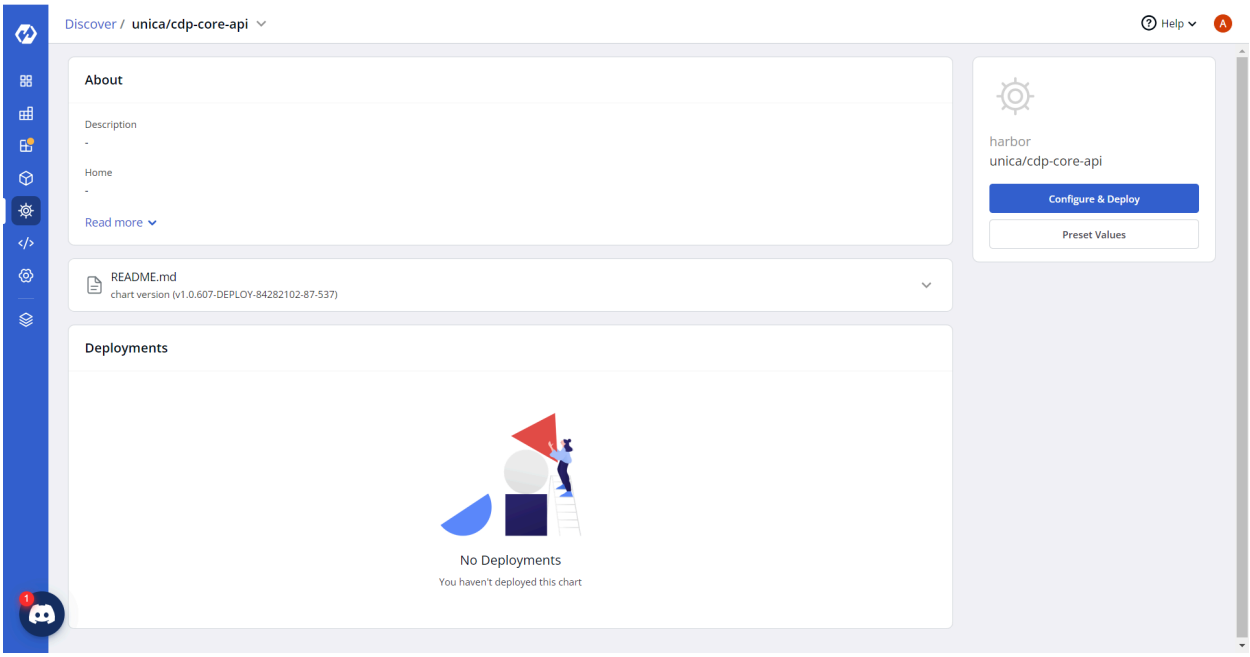
Deploying CDP CORE API

To deploy the CDP core API, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-core-api chart to deploy.



2. Now, configure and deploy the cdp core api charts.

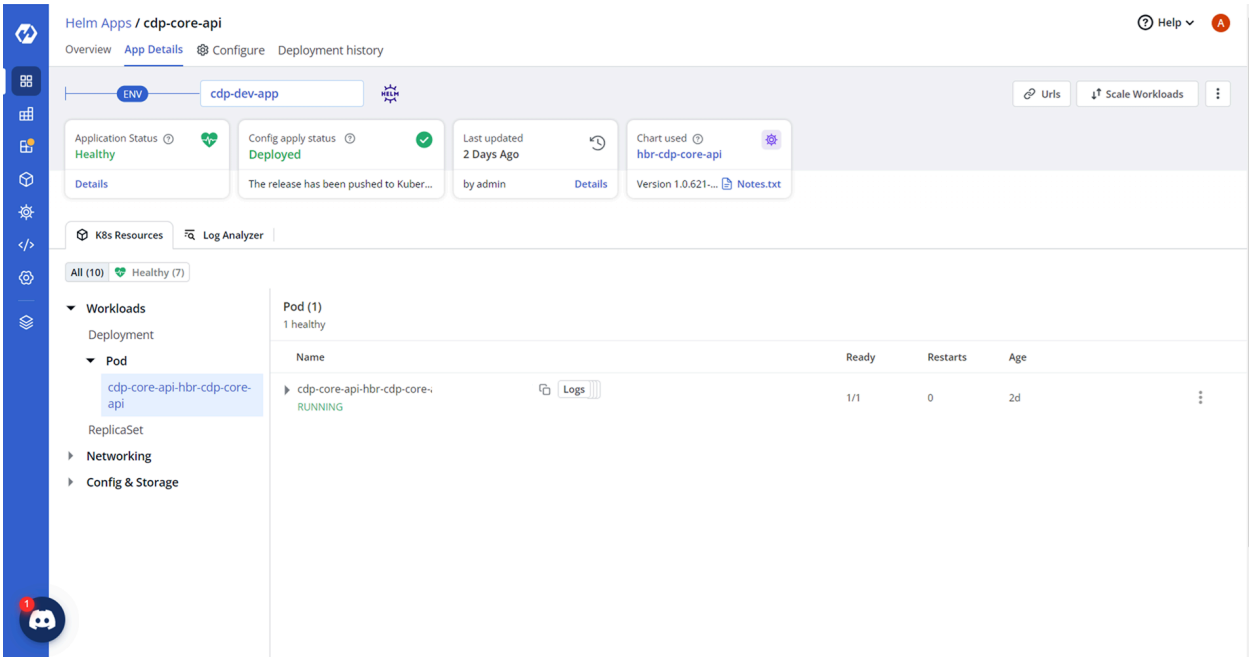


3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
druidURL: "' '"
tsdbURL: "''"
uploadsBucket: "' '"
uploadsFolder: "' '"
segmentUserListuploadsBucket: "''"
segmentUserListuploadsFolder: "''"
TPPActivationUrl: "''"
hasSegRefreshQueueEnabled: "[]"
whiteListCrudAlertMails: "[]"
enablePrometheus: "true"
suppressionList: '"'"
productCatalog: '"'"
rts_endpoint: "''"
disableActivation: |
  [5459]
s3Bucket: "''"
DSN: "''"
cdpDashboardService: "''"
rmq: "'jb_triggers_us'"
javascript_sdk_endpoint: "''"
sendRabbitmqMessageDisabled: "true"
API: "''"
cdnImagePrefixWithoutProtocol: '"'"
cdnImageWithProtocol: '"'"
cdnFallbackImageWithoutProtocol: '"'"
urlShortnerConfigDomain: '"'"
sendEmailDomain: '"'"
uibase: "''"
lemnisk_uibase: "''"
reportAutomationApiKeyUrl: '"'"
accessTokenLifeTime: "'86400'"
segRefreshQueue: "''"
enableCrudAlertForCampaigns: |
  []
isKmsEncryptDisabled: "false"
disablePageres: "false"
tableauProxy: "''"
channelQueue: "''"
engagementScheduleQueue: "''"
journeyDisableQueue: "''"
webPushQueue: "''"
adhocBucket: "''"
adhocQueue: "''"
adHocFileLoc: "''"
adHocCancelQueue: "''"
sslEnabled: "false"
offlineProfileUploadBucket: "''"
offlineProfileUploadFolder: "''"
disableNotification: "''"
tableauURL: "''"
kms_encrypt: "''"
kms_decrypt: "''"
kms_region: "'aps1'"
allowedTags: "['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'div', 'hr', 'li', 'ol', 'p', 'pre', 'ul', 'a', 'b',
 'br', 'em', 'i', 'span', 'strong', 'sub', 'sup', 'u']"
allowedAttributes: "{ '*': ['href', 'align', 'target', 'alt', 'style', 'class', 'name', 'id',
 'fr-original-style']}"
tawuniyaCoreApi: "''"
```

```
psql_endpoint: "'"
psql_host: "'"
fileUploadTimeout: "40000"
CampaignRegionList:
 "{aps1:['6525','6532','6533','6534','6530','6529','6528','6527','6526','6536','6537','6538']}"
muUserIdentifierDomain: "'"
sftpFilePath: "'"
usUserIdentifierDomain: "'"
sstApiVersion: "'v1'"
devMode: "false"
FROM_EMAIL_ID: "'"
ALERT_TO_EMAIL_ID: "'"
```

4. On successful deployment, validate the deployment as shown below.

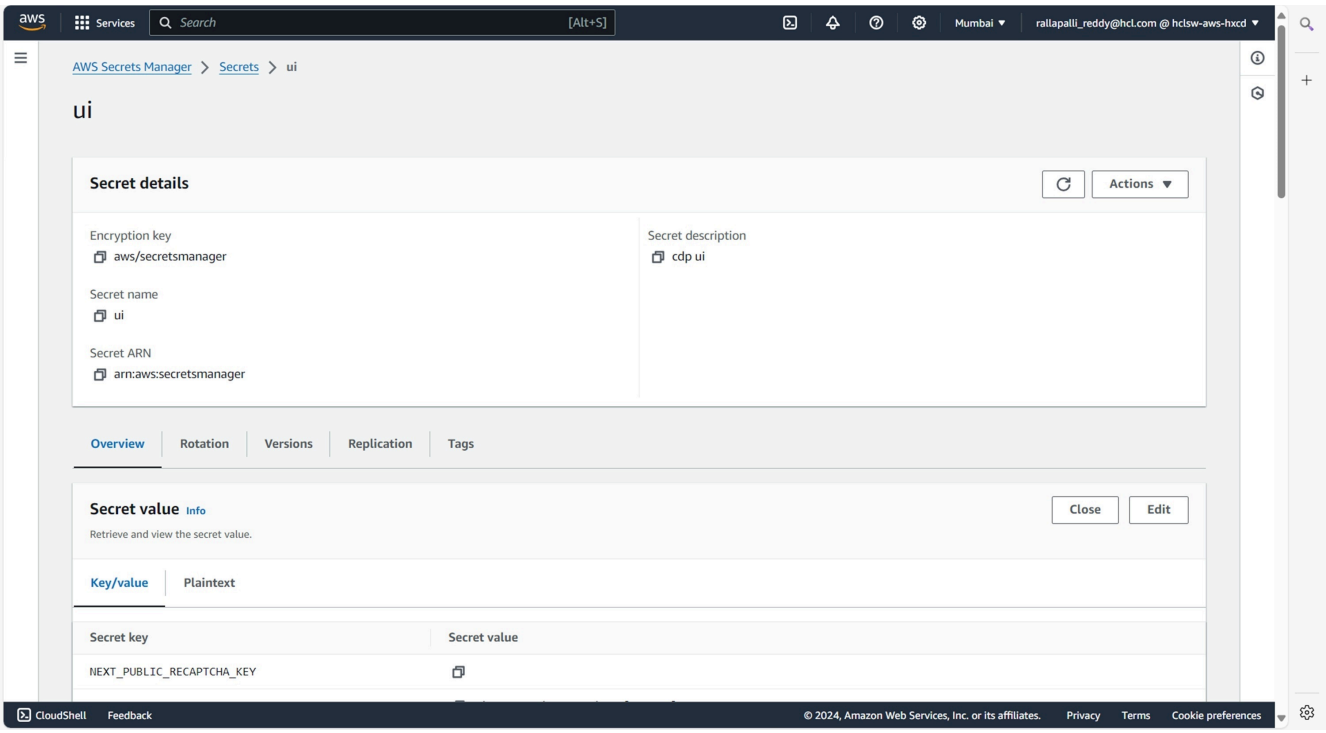


Deploying CDP Dash Backend

This section provides detailed instructions on how to deploy HCL CDP Dash Backend using the Devtron in the AWS.

Prerequisites:

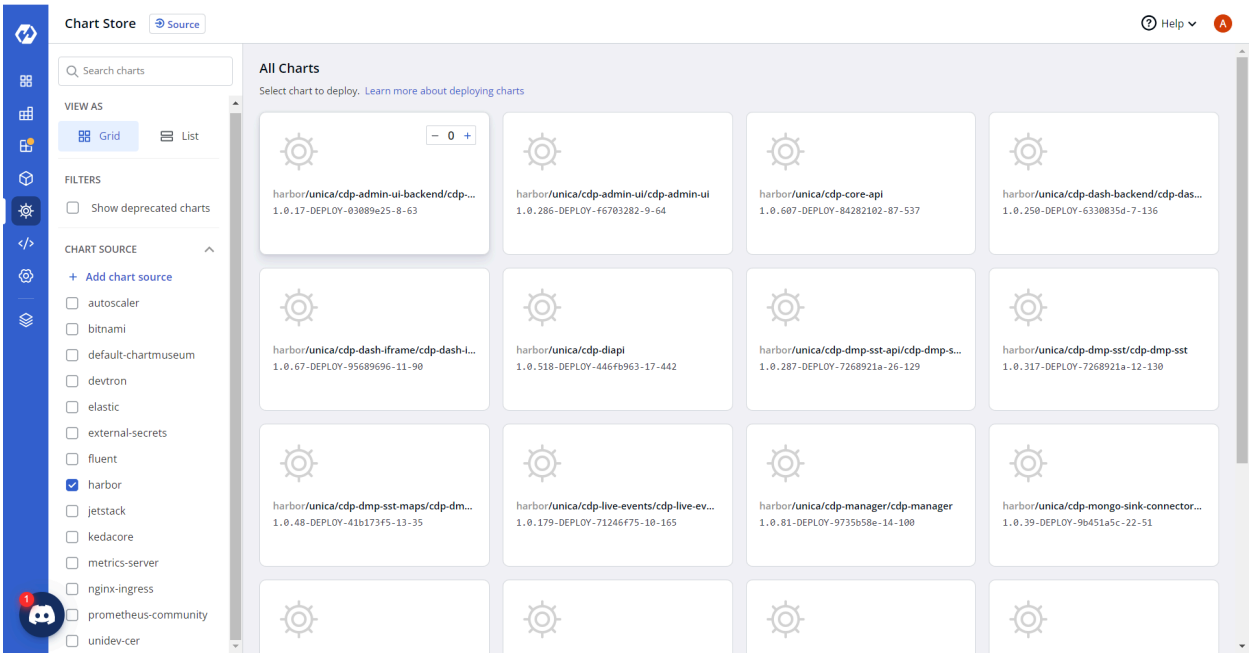
Make sure to create UI secret with required data in the AWS Secrets Manager before deploying CDP Dash Backend.



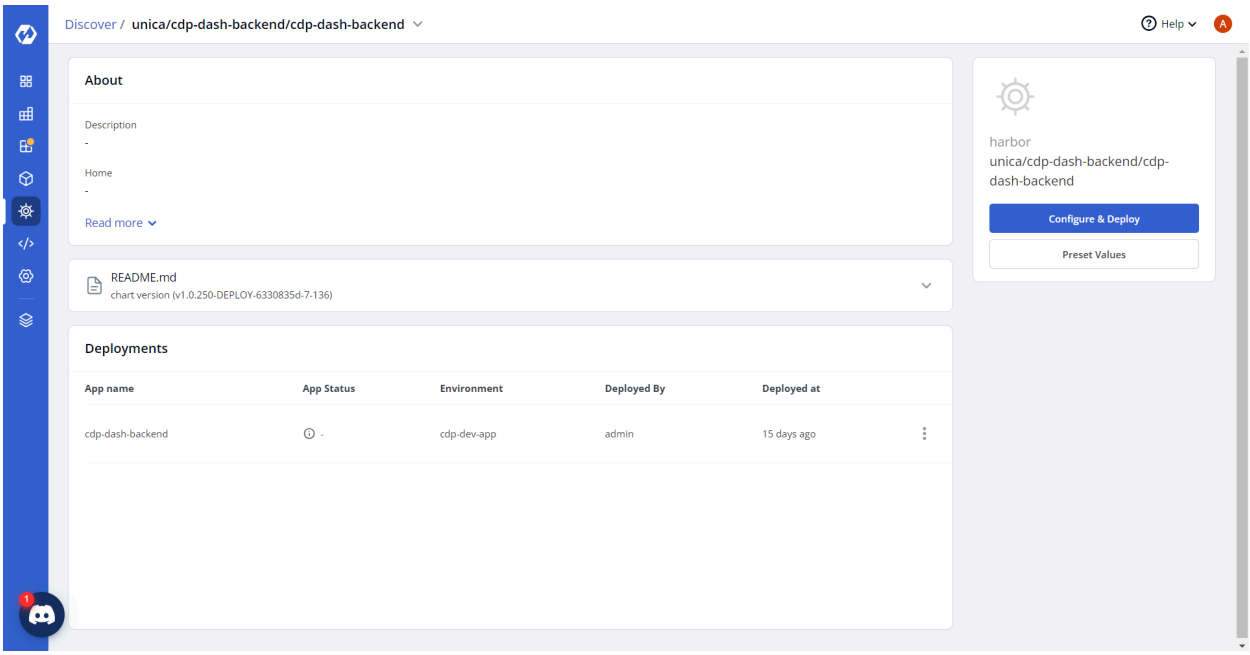
Deploying CDP Dash Backend

To deploy the CDP Dash Backend, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-dash-backend chart to deploy.

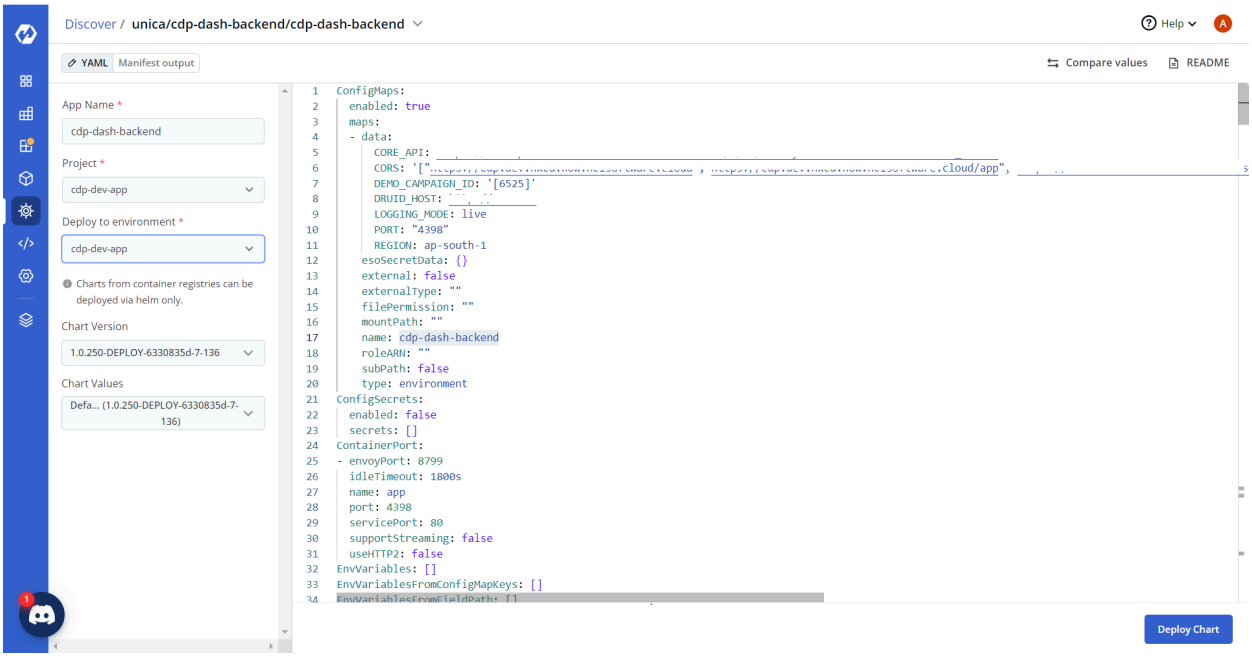


2. Now, configure and deploy the CDP Dash Backend charts.

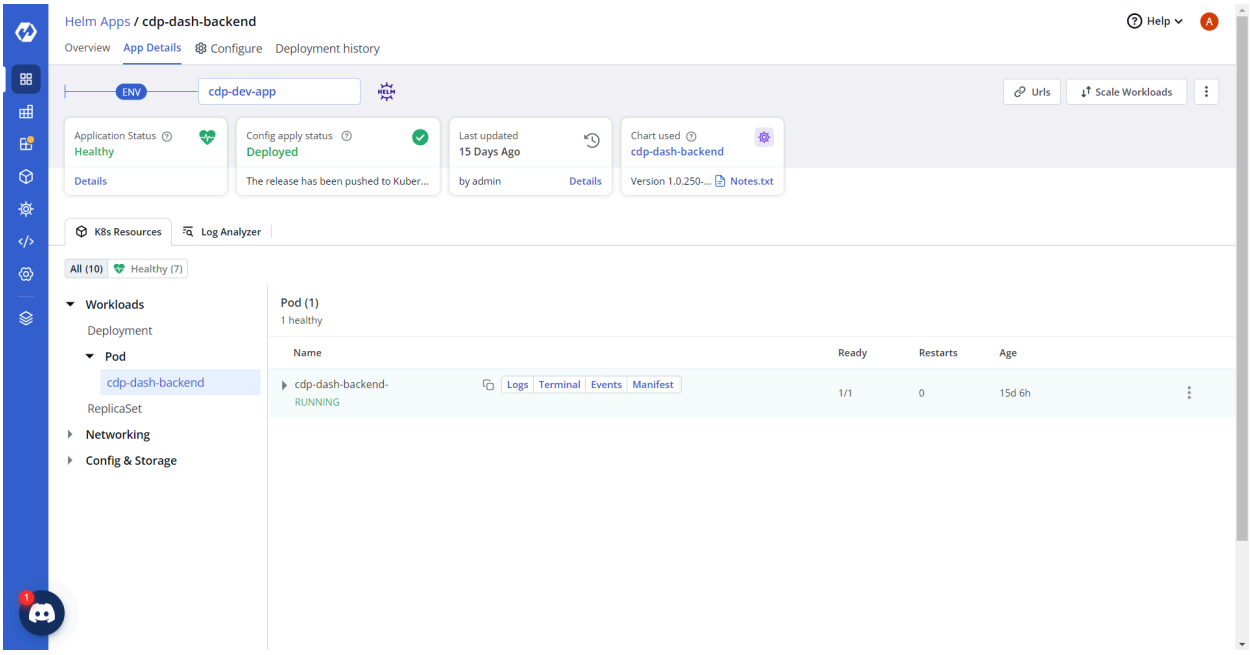


3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
CORE_API:<http://core_api_url/-/v1/verifyUserAuthentication?access_token=>
CORS: '[cors_urls]'
DEMO_CAMPAIGN_ID: '[6525]'
DRUID_HOST: http://<druid_host:port>/druid/v2/sql
LOGGING_MODE: <live>
PORT: "<port>"
REGION: <region>
ANALYTICS_BUCKET: ""
```



4. On successful deployment, validate the deployment as shown below.

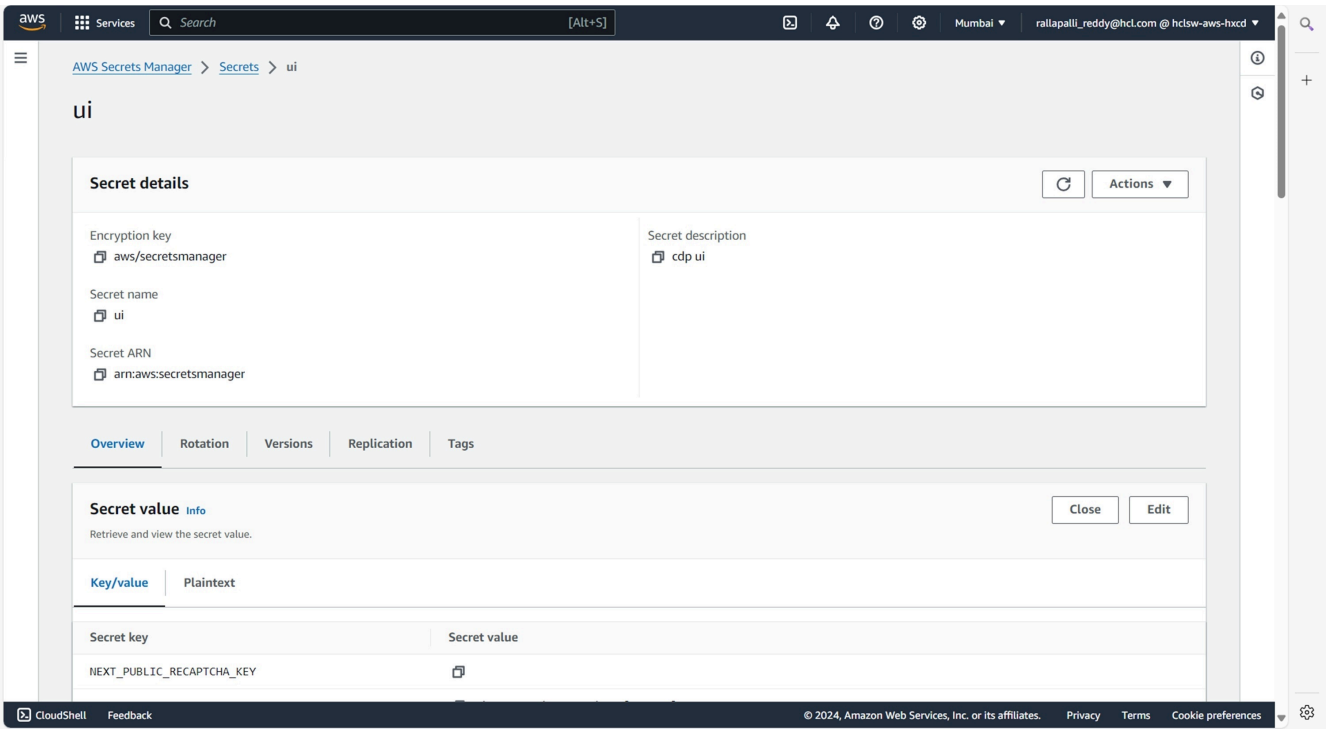


Deploying CDP Dash IFrame

This section provides detailed instructions on how to deploy HCL CDP Dash IFrame using the Devtron in the AWS.

Prerequisites:

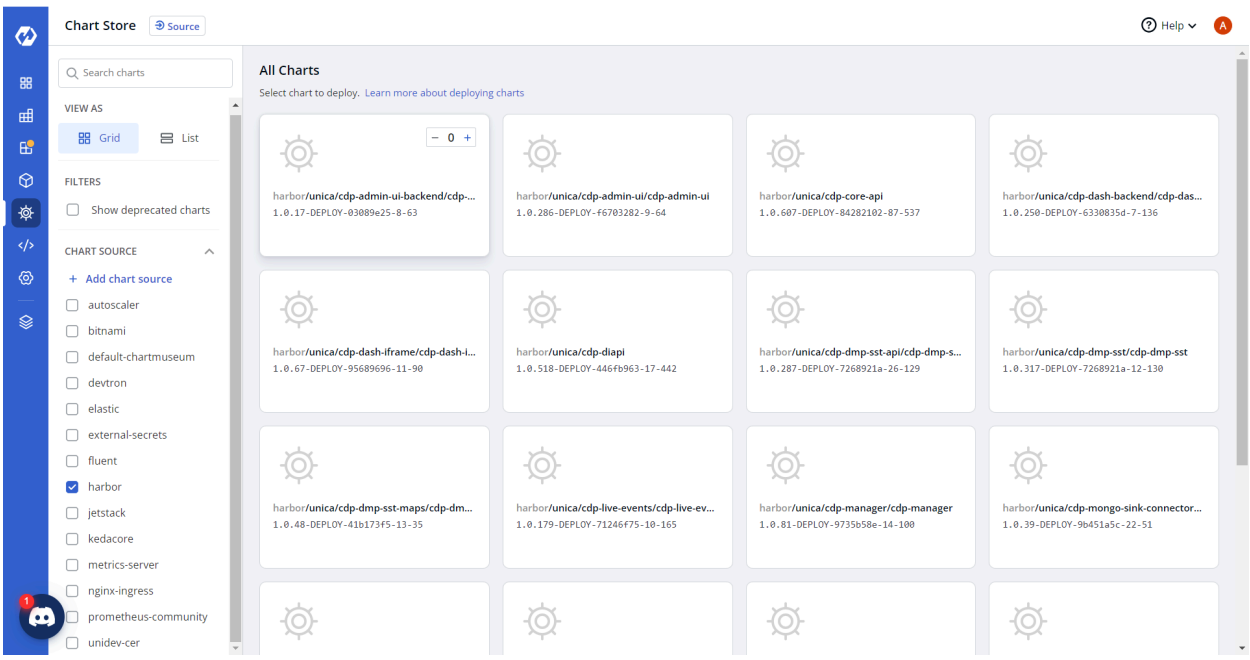
Make sure to create UI secret with required data in the AWS Secrets Manager before deploying CDP Dash IFrame.



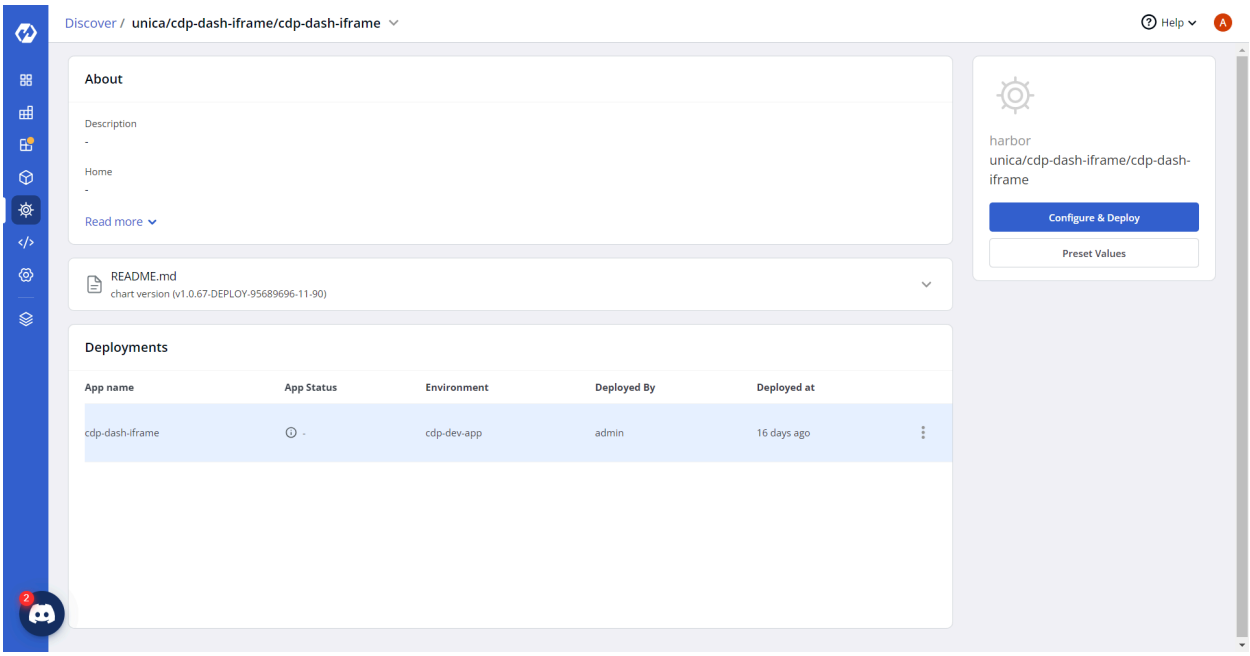
Deploying CDP Dash IFrame

To deploy the CDP Dash IFrame, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-dash-iframe chart to deploy.

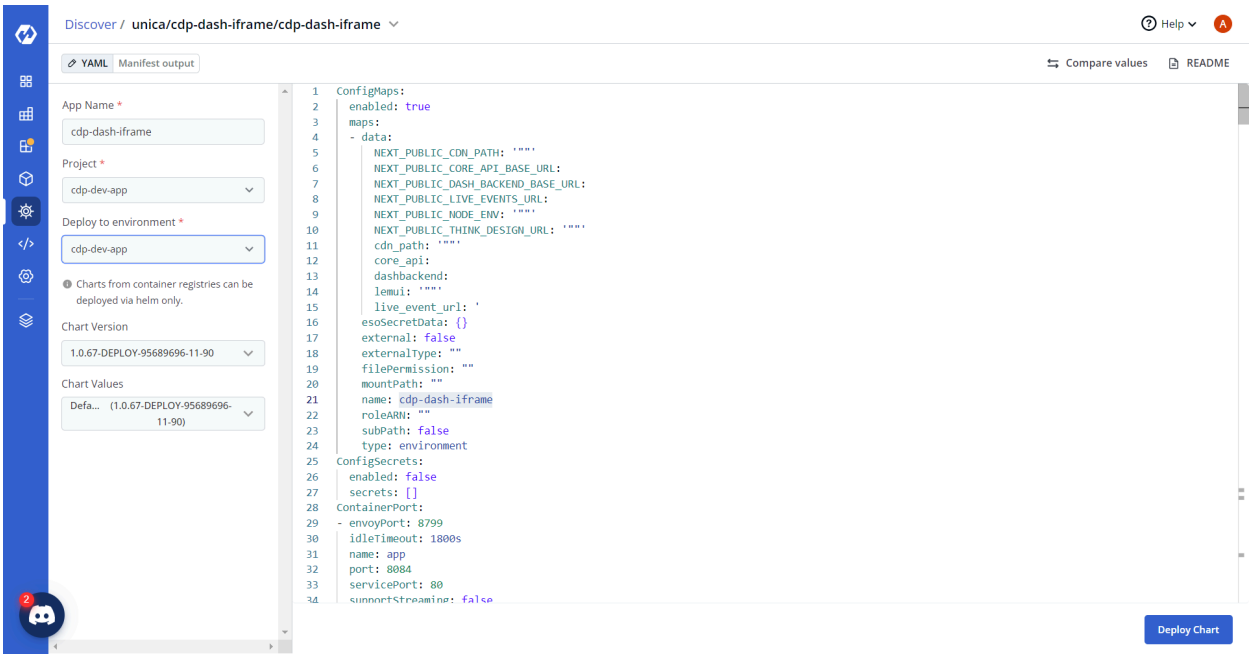


2. Now, configure and deploy the CDP Dash IFrame charts.

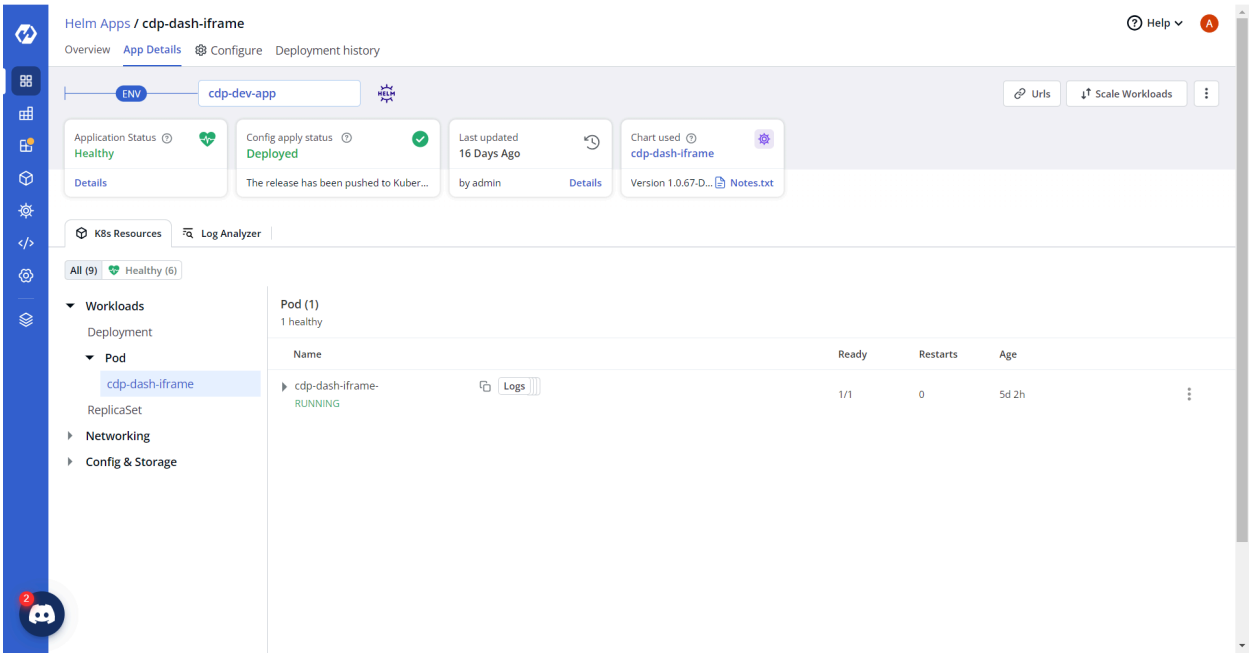


3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
NEXT_PUBLIC_CDN_PATH: '"'
 NEXT_PUBLIC_CORE_API_BASE_URL:<CORE_API_BASE_URL>
NEXT_PUBLIC_DASH_BACKEND_BASE_URL: <DASH_BACKEND_BASE_URL>
NEXT_PUBLIC_LIVE_EVENTS_URL: <LIVE_EVENTS_URL>
NEXT_PUBLIC_NODE_ENV: '"'
NEXT_PUBLIC_THINK_DESIGN_URL: '"'
cdn_path: '"'
core_api:<core_api_url>
dashbackend:<dashbackend_url>
lemui: '"'
live_event_url: <live_event_url>
```



4. On successful deployment, validate the deployment as shown below.

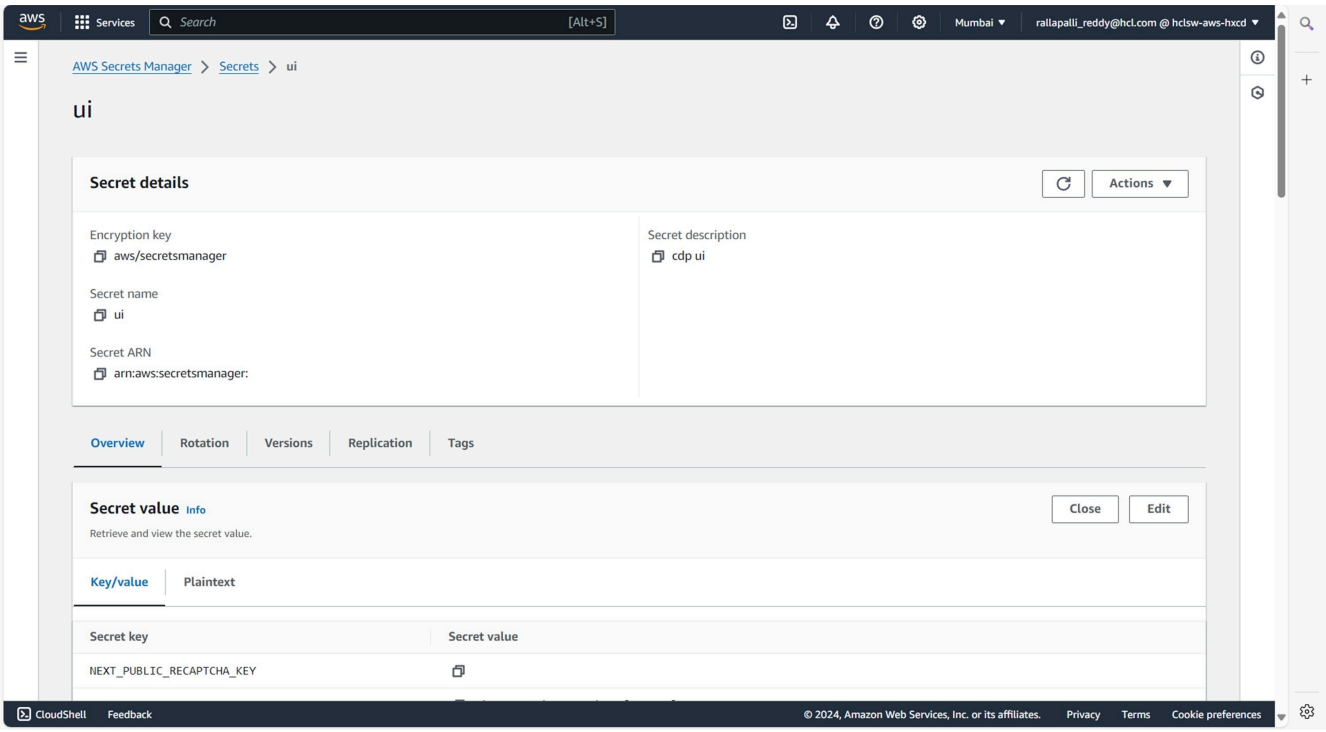


Deploying CDP Web App

This section provides detailed instructions on how to deploy HCL CDP Web App using the Devtron in the AWS.

Prerequisites:

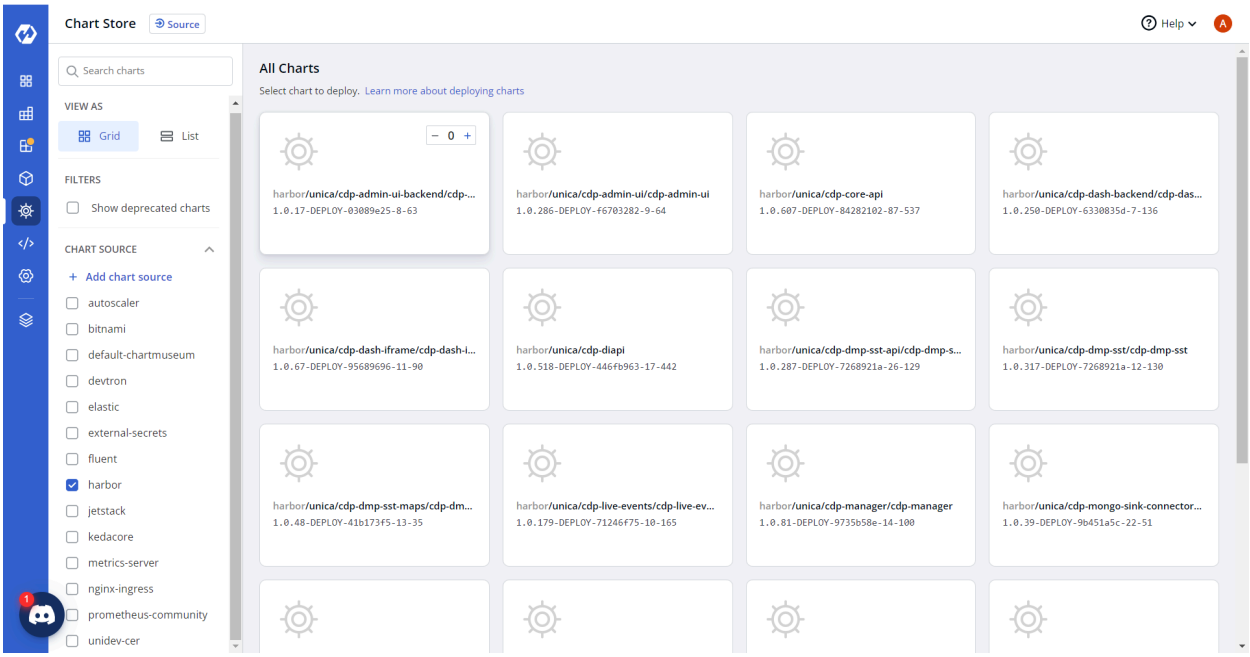
Make sure to create UI secret with required data in the AWS Secrets Manager before deploying CDP Web App.



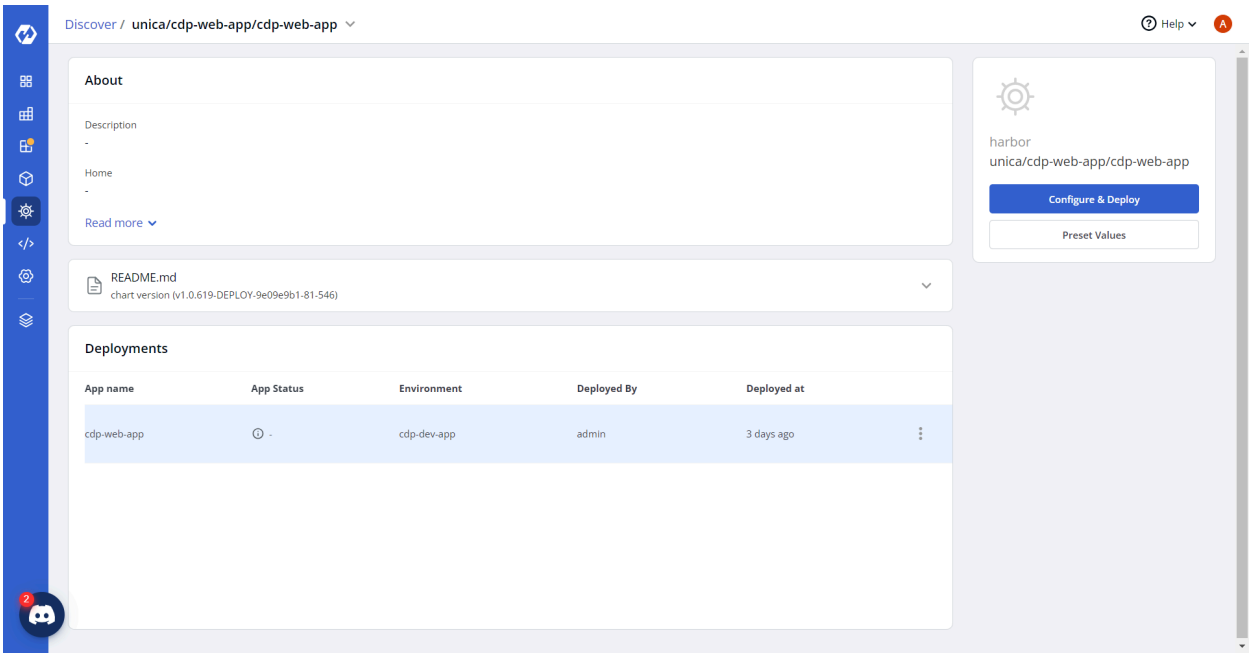
Deploying CDP Web App

To deploy the CDP Web App, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-web-app chart to deploy.



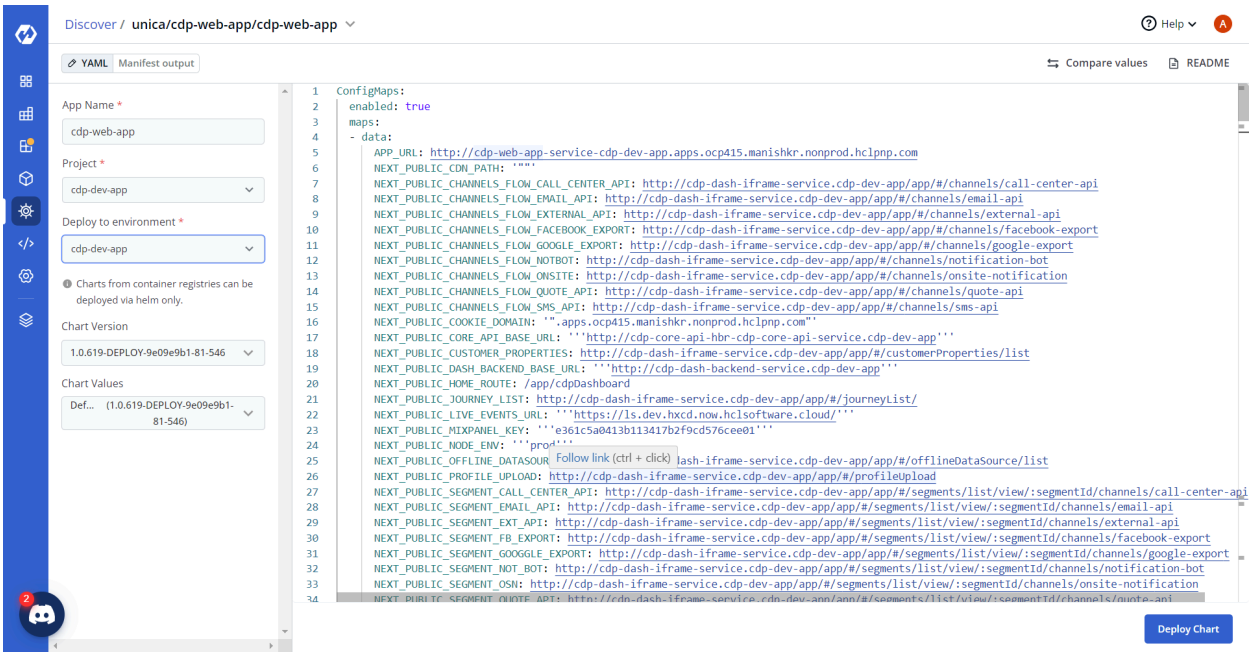
2. Now, configure and deploy the CDP Web App charts.



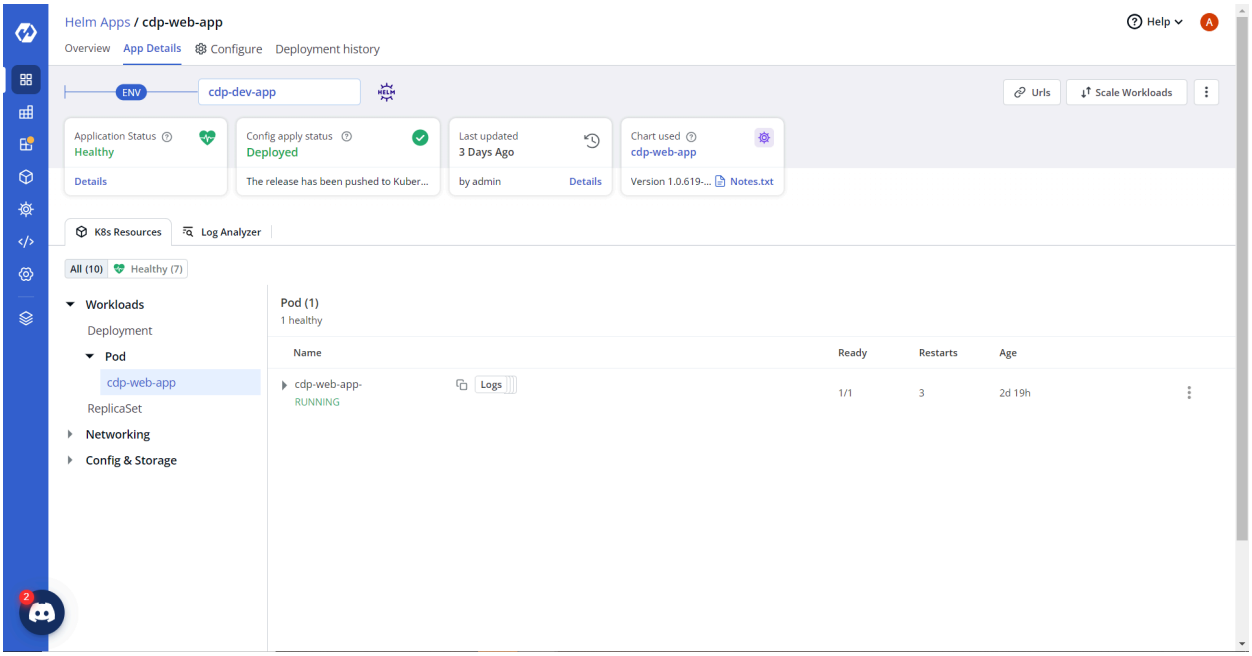
3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
APP_URL: <WebApp_URL>
NEXT_PUBLIC_CDN_PATH: '"'
NEXT_PUBLIC_CHANNELS_FLOW_CALL_CENTER_API: <dash_iframe_url>/app/#/channels/call-center-api
NEXT_PUBLIC_CHANNELS_FLOW_EMAIL_API: <dash_iframe_url>/app/#/channels/email-api
NEXT_PUBLIC_CHANNELS_FLOW_EXTERNAL_API: <dash_iframe_url>/app/#/channels/external-api
NEXT_PUBLIC_CHANNELS_FLOW_FACEBOOK_EXPORT: <dash_iframe_url>/app/#/channels/facebook-export
NEXT_PUBLIC_CHANNELS_FLOW_GOOGLE_EXPORT: <dash_iframe_url>/app/#/channels/google-export
NEXT_PUBLIC_CHANNELS_FLOW_NOTBOT: <dash_iframe_url>/app/#/channels/notification-bot
NEXT_PUBLIC_CHANNELS_FLOW_ONSITE: <dash_iframe_url>/app/#/channels/onsite-notification
NEXT_PUBLIC_CHANNELS_FLOW_QUOTE_API: <dash_iframe_url>/app/#/channels/quote-api
NEXT_PUBLIC_CHANNELS_FLOW_SMS_API: <dash_iframe_url>/app/#/channels/sms-api
NEXT_PUBLIC_COOKIE_DOMAIN: <COOKIE_DOMAIN>
NEXT_PUBLIC_CORE_API_BASE_URL: <CORE_API_BASE_URL>
NEXT_PUBLIC_CUSTOMER_PROPERTIES: <dash_iframe_url>/app/#/customerProperties/list
NEXT_PUBLIC_DASH_BACKEND_BASE_URL: <DASH_BACKEND_BASE_URL>
NEXT_PUBLIC_HOME_ROUTE: /app/cdpDashboard
NEXT_PUBLIC_JOURNEY_LIST: <dash_iframe_url>/app/#/journeyList/
NEXT_PUBLIC_LIVE_EVENTS_URL: <LIVE_EVENTS_URL>
NEXT_PUBLIC_MIXPANEL_KEY: <MIXPANEL_KEY>
NEXT_PUBLIC_NODE_ENV: '''prod'''
NEXT_PUBLIC_OFFLINE_DATASOURCE_LIST: <dash_iframe_url>/app/#/offlineDataSource/list
NEXT_PUBLIC_PROFILE_UPLOAD: <dash_iframe_url>/app/#/profileUpload
NEXT_PUBLIC_SEGMENT_CALL_CENTER_API:
 <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/call-center-api
NEXT_PUBLIC_SEGMENT_EMAIL_API: <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/email-api
NEXT_PUBLIC_SEGMENT_EXT_API: <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/external-api
NEXT_PUBLIC_SEGMENT_FB_EXPORT:
 <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/facebook-export
NEXT_PUBLIC_SEGMENT_GOOGGLE_EXPORT:
 <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/google-export
NEXT_PUBLIC_SEGMENT_NOT_BOT:
 <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/notification-bot
```

```
NEXT_PUBLIC_SEGMENT_OSN:
 <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/onsite-notification
NEXT_PUBLIC_SEGMENT_QUOTE_API: <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/quote-api
NEXT_PUBLIC_SEGMENT_SMS_API: <dash_iframe_url>/app/#/segments/list/view/:segmentId/channels/sms-api
NEXT_PUBLIC_SENTRY_DSN: ''''''
NEXT_PUBLIC_THINK_DESIGN_URL: '''<dash_iframe_url>'''
SENTRY_PROJECT: '''us-tkd'''
SERVER_NAME: <WebApp_server_name>
```



4. On successful deployment, validate the deployment as shown below.

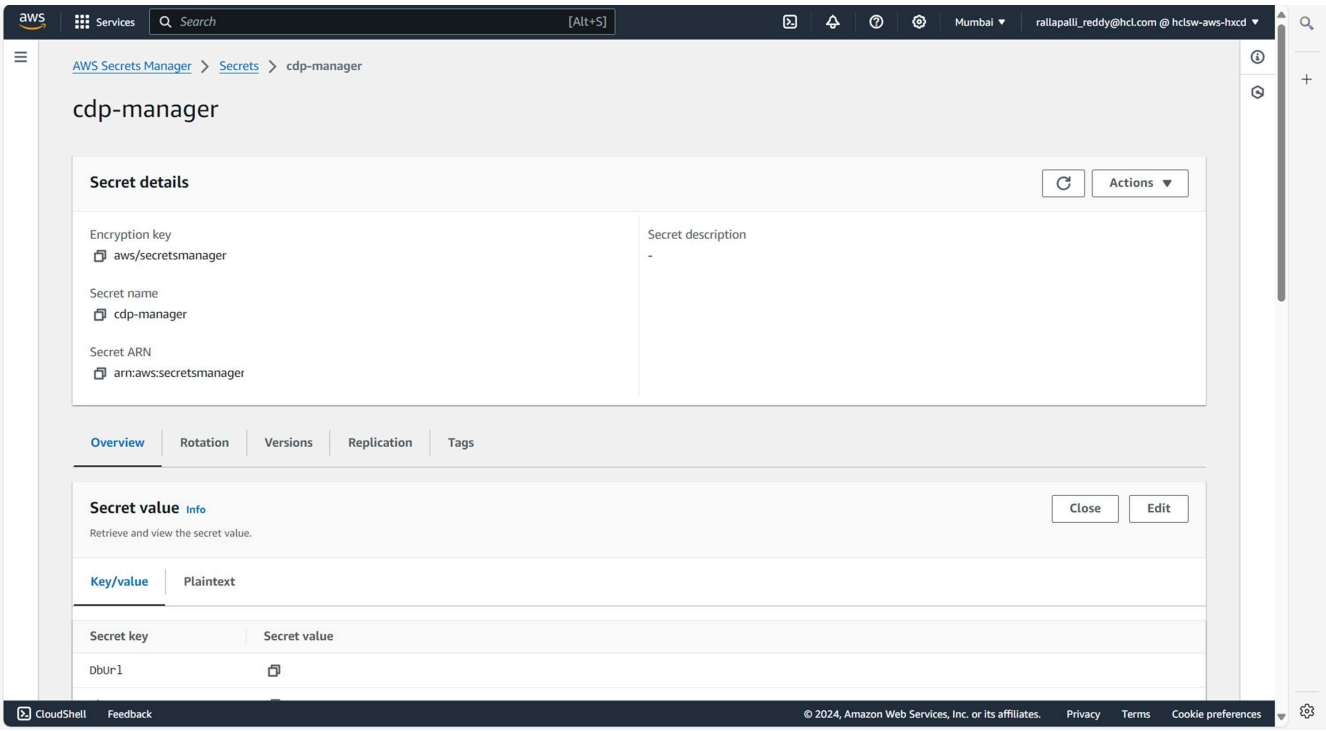


Deploying CDP Manager

This section provides detailed instructions on how to deploy HCL CDP Manager using the Devtron in the AWS.

Prerequisites:

Make sure to create cdp-manager secret with required data in the AWS Secrets Manager before deploying CDP Manager.



To create the cdp-manager secret in the AWS secret manager, follow the steps below:

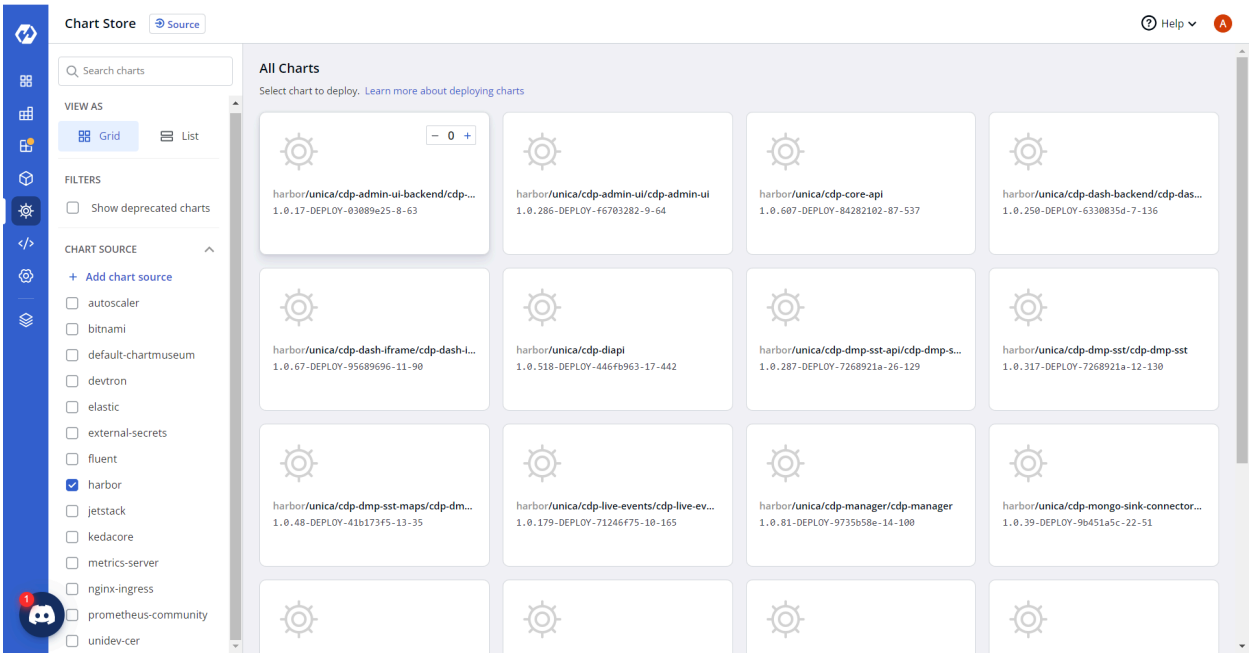
1. Create a cdp-manager secret sample key and value in the cdp-manager secret, and update ConfigMaps data with actual values.

```
{
  "AerospikeHost": "<AerospikeHost>",
  "AerospikePort": "<AerospikePort>",
  "DbName": "<DbName>",
  "DbPassword": "<DbPassword>",
  "DbUrl": "<ip:port>",
  "DbUsername": "<DbUsername>",
  "KafkaBrokers": "<KafkaBrokers>",
  "KafkaPassword": "<KafkaPassword>",
  "KafkaSaslMechanism": "<KafkaSaslMechanism>",
  "KafkaSecurityProtocol": "<KafkaSecurityProtocol>",
  "KafkaUsername": "<KafkaUsername>"
}
```

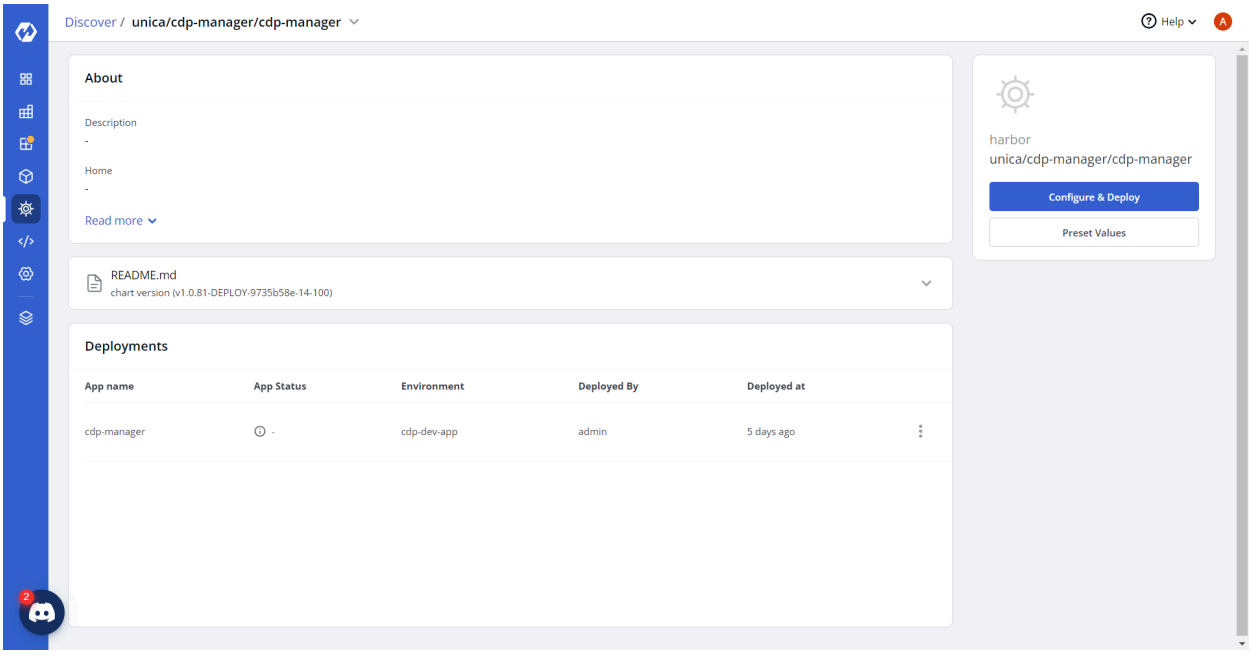
Deploying CDP Manager

To deploy the CDP Manager, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-manager chart to deploy.



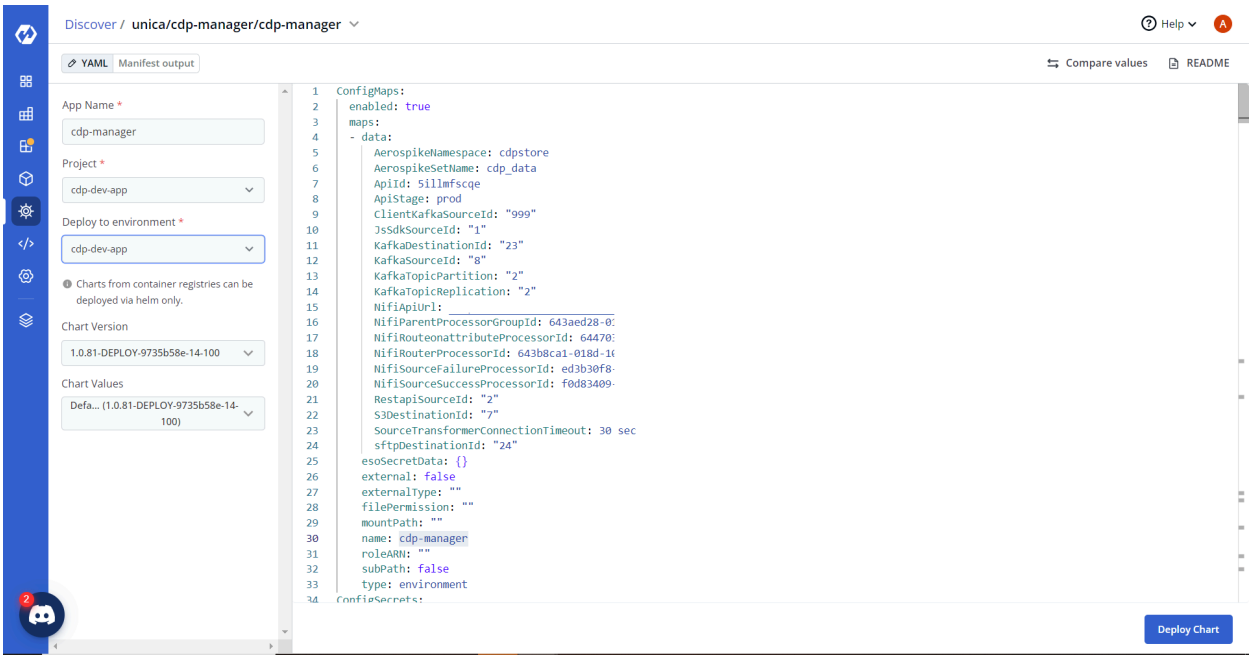
2. Now, configure and deploy the CDP Manager charts.



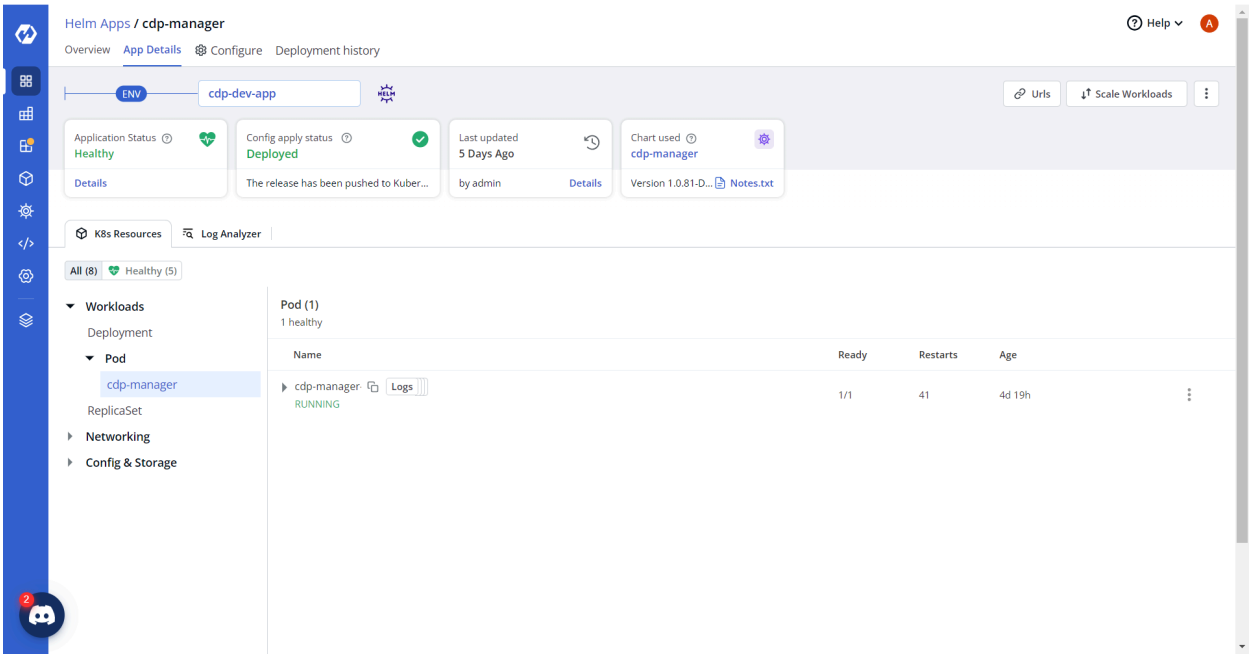
3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
AerospikeNamespace: <AerospikeNamespace>
AerospikeSetName: <AerospikeSetName>
NifiApiUrl: http://<ip:port>/nifi-api
NifiParentProcessorGroupId: <NifiParentProcessorGroupId>
NifiRouteonattributeProcessorId: <NifiRouteonattributeProcessorId>
NifiRouterProcessorId: <NifiRouterProcessorId>
```

```
NifiSourceFailureProcessorId: <NifiSourceFailureProcessorId>
NifiSourceSuccessProcessorId: <NifiSourceSuccessProcessorId>
```



4. On successful deployment, validate the deployment as shown below.

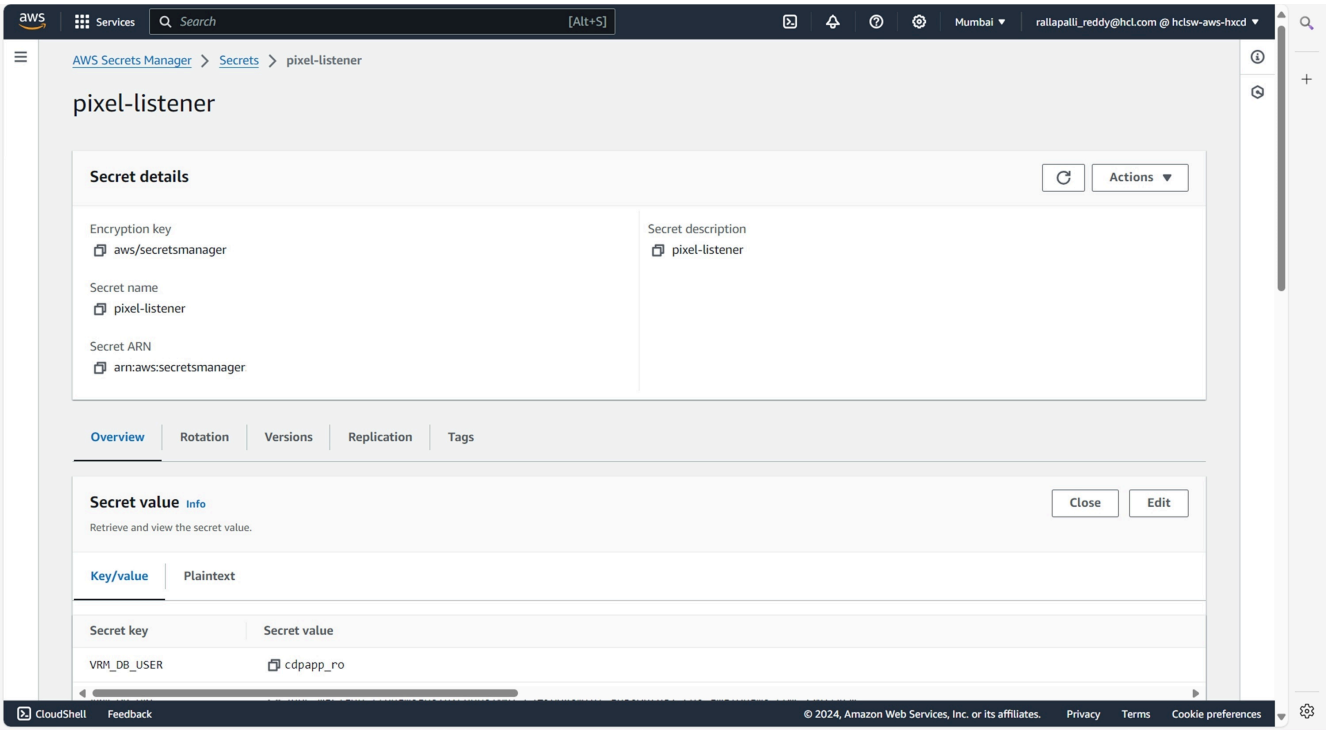


Deploying CDP Pixel Listener

This section provides detailed instructions on how to deploy HCL CDP Pixel Listener using the Devtron in the AWS.

Prerequisites:

Make sure to create pixel-listener secret with required data in the AWS Secrets Manager before deploying CDP Pixel Listener.



To create the pixel-listener secret in the AWS secret manager, follow the steps below:

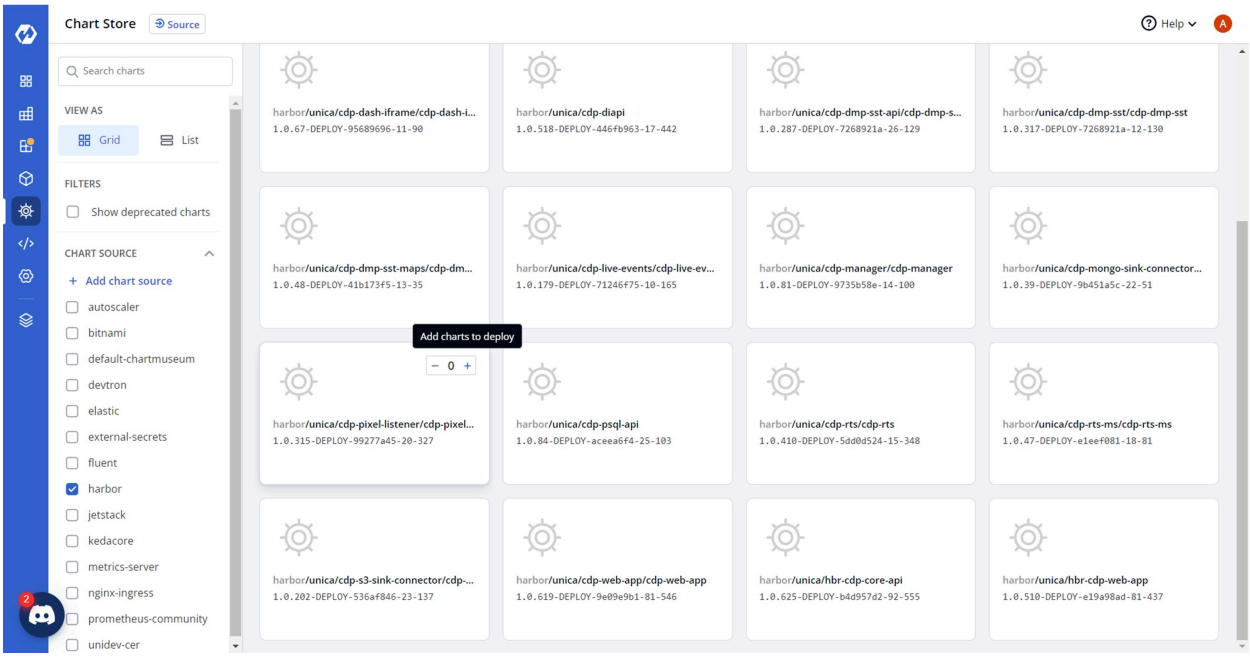
1. Create a pixel-listener secret sample key and value in the pixel-listener secret, and update ConfigMaps data with actual values.

```
{
  "ADCB_AERO_CLUSTER": "<Aerospike_ip:port>",
  "KAFKA_PRODUCER_CONFIG": "<KAFKA_PRODUCER_CONFIG>,
  "PII_AERO_APS1_HOST": "<Aerospike_ip>",
  "PII_AERO_APS1_PORT": "<Aerospike_port>",
  "PII_AERO_HOST": "<Aerospike_ip>",
  "PII_AERO_PORT": "<Aerospike_port>",
  "PII_AERO_USE1_HOST": "<Aerospike_ip>",
  "PII_AERO_USE1_PORT": "<Aerospike_port>",
  "TSDB_HOST": "<TSDB_HOST>",
  "TSDB_PORT": "<TSDB_PORT>",
  "VRM_DB_PASSWORD": "<VRM_DB_PASSWORD>",
  "VRM_DB_URL": "jdbc:mariadb://<ip:port>/<dbName>",
  "VRM_DB_USER": "<VRM_DB_USER>"
}
```

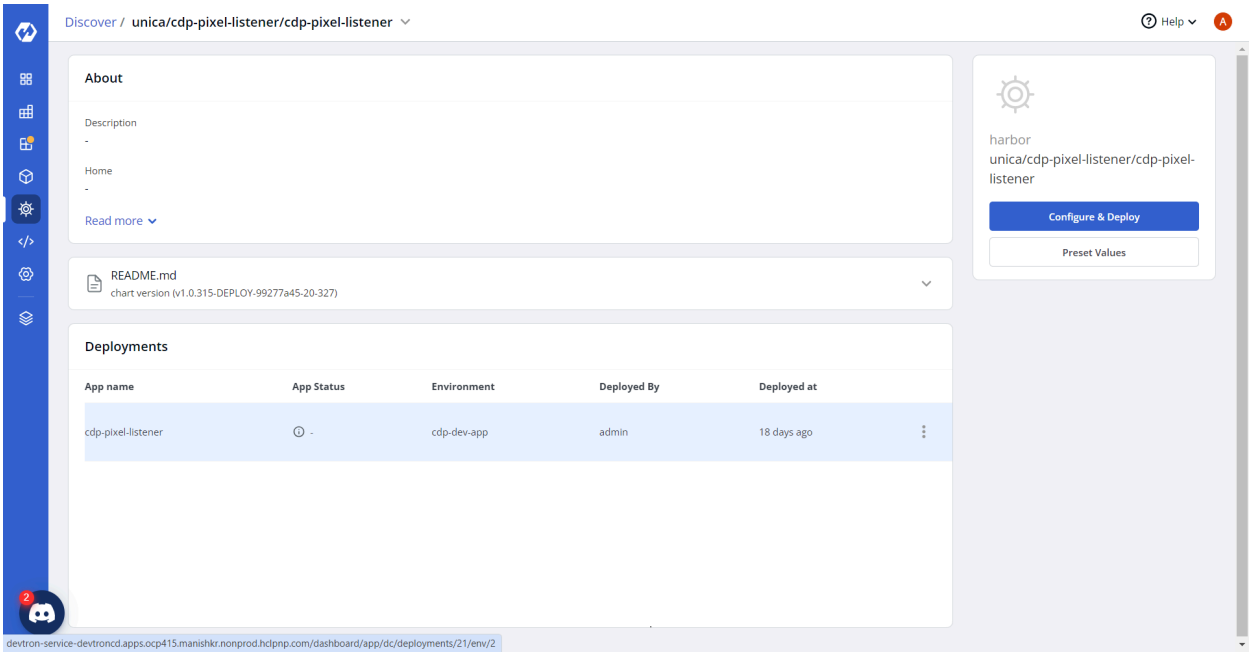
Deploying CDP Pixel Listener

To deploy the CDP Pixel Listener, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-pixel-listener chart to deploy.

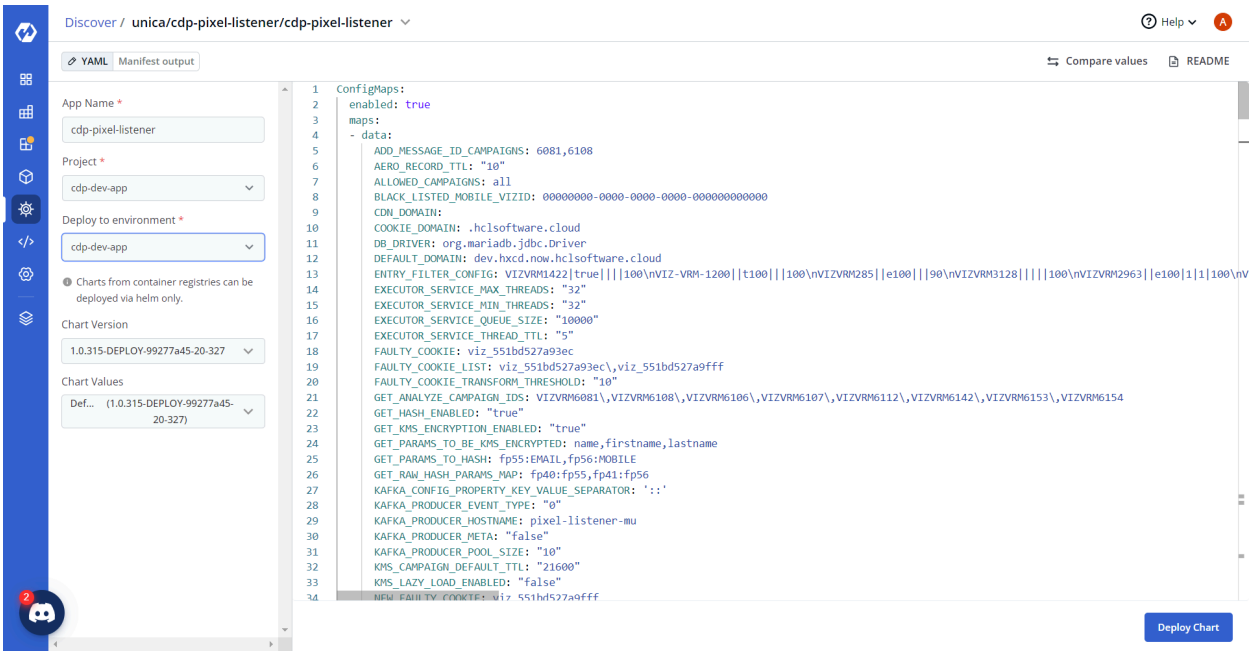


2. Now, configure and deploy the CDP Pixel Listener charts.

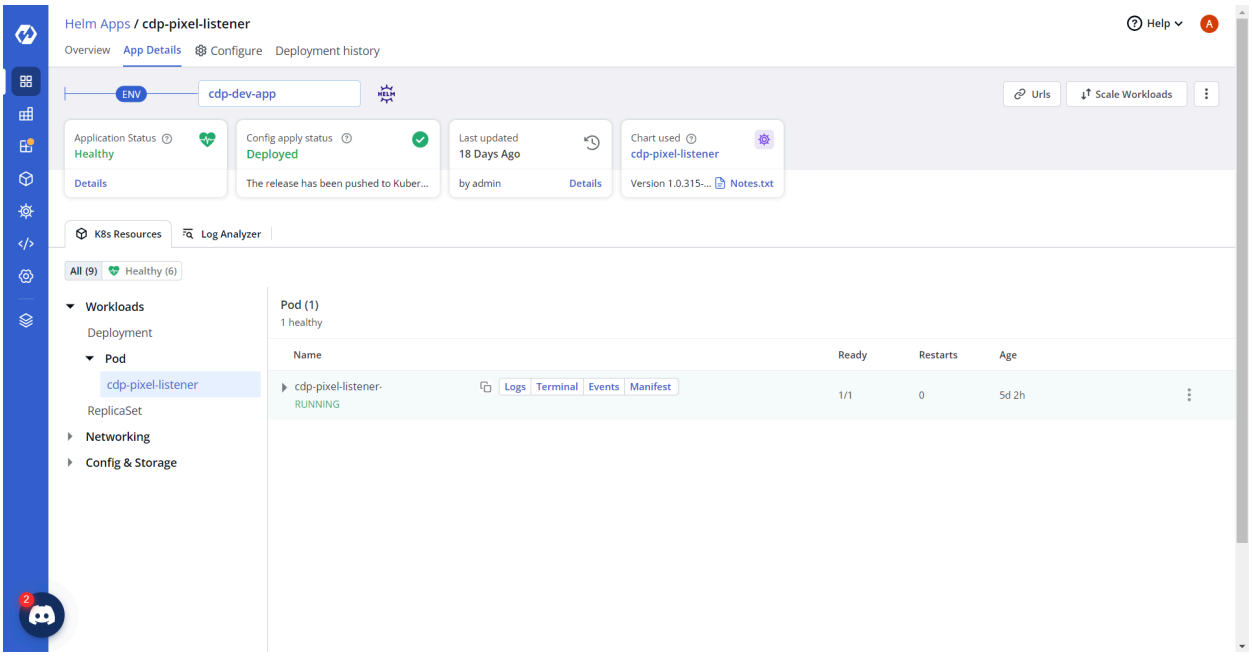


3. In the **YAML** section, update the ConfigMap using below details and deploy the charts.

```
ALLOWED_CAMPAIGNS: all
CDN_DOMAIN: <CDN_DOMAIN>
COOKIE_DOMAIN: <COOKIE_DOMAIN>
DB_DRIVER: org.mariadb.jdbc.Driver
DEFAULT_DOMAIN: <DEFAULT_DOMAIN>
```



4. On successful deployment, validate the deployment as shown below.

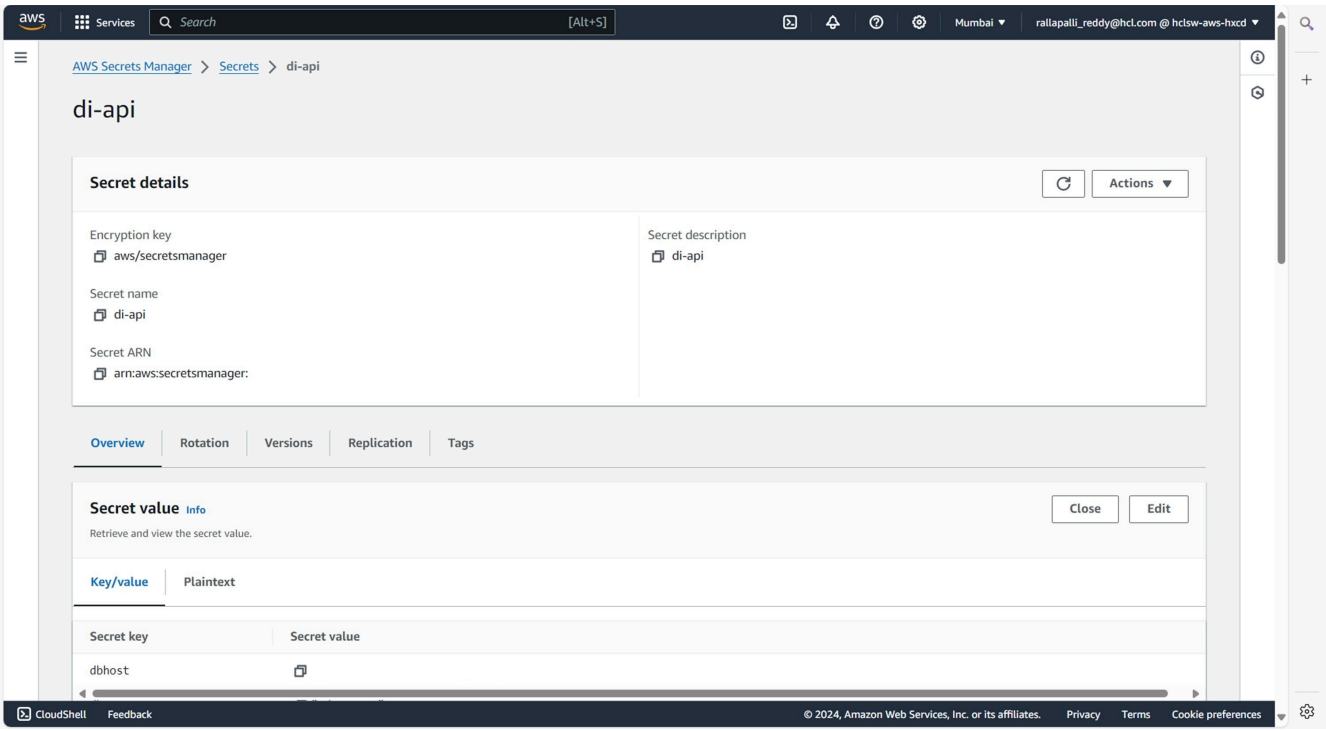


Deploying CDP DI API

This section provides detailed instructions on how to deploy HCL CDP DI API using the Devtron in the AWS.

Prerequisites:

Make sure to create di-api secret with required data in the AWS Secrets Manager before deploying CDP DI API.



To create the di-api secret in the AWS secret manager, follow the steps below:

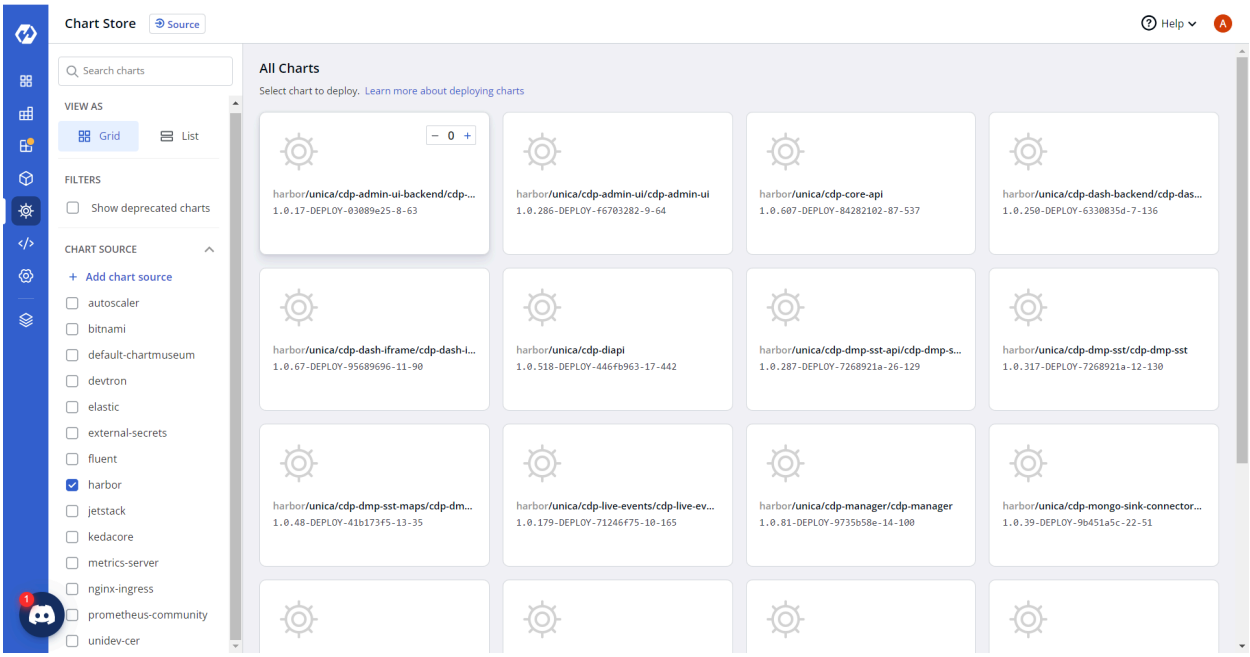
1. Create a di-api secret sample key and value in the di-api secret, and update ConfigMaps data with actual values.

```
{
  "ADV_6022_SECRET": "\"\"",
  "ADV_6022_URL": "\"\"",
  "ADV_6282_SECRET": "\"\"",
  "ADV_6282_URL": "\"\"",
  "aerospikeHostaps1": "\"<aerospikeHostaps1>\"",
  "aerospikeHostuse1": "\"\"",
  "aps1KafkaPassword": "\"<aps1KafkaPassword>\"",
  "aps1KafkaUsername": "\"<aps1KafkaUsername>\"",
  "dbhost": "\"<dbhost>\"",
  "dbpassword": "\"<dbpassword>\"",
  "dbusername": "\"<dbusername>\"",
  "kafkaconnectionStringaps1": "\"<kafkaconnectionStringaps1>\"",
  "kafkaconnectionStringuse1": "\"\"",
  "kmsUrl": "\"<kmsUrl>\"",
  "tdsbHost": "'<tdsbHost>'",
  "use1KafkaPassword": "\"\"",
  "use1KafkaUsername": "\"\""
}
```

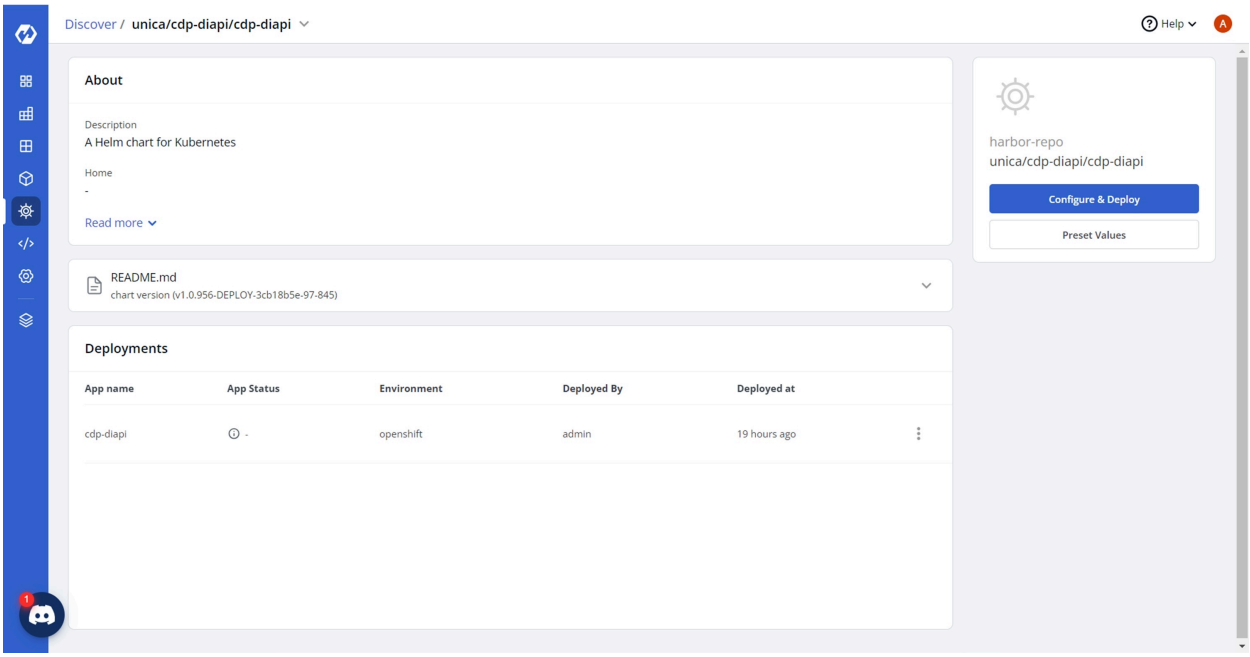
Deploying CDP DI API

To deploy the CDP DI API, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-di-api chart to deploy.



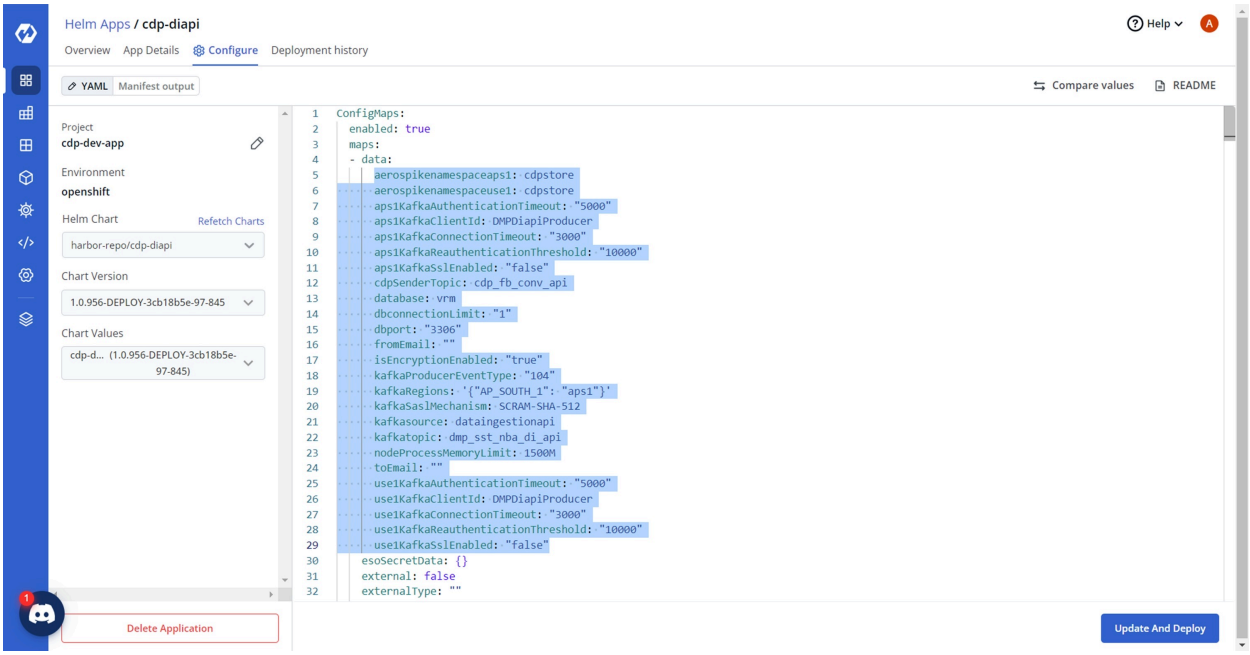
2. Now, configure and deploy the CDP DI API charts.



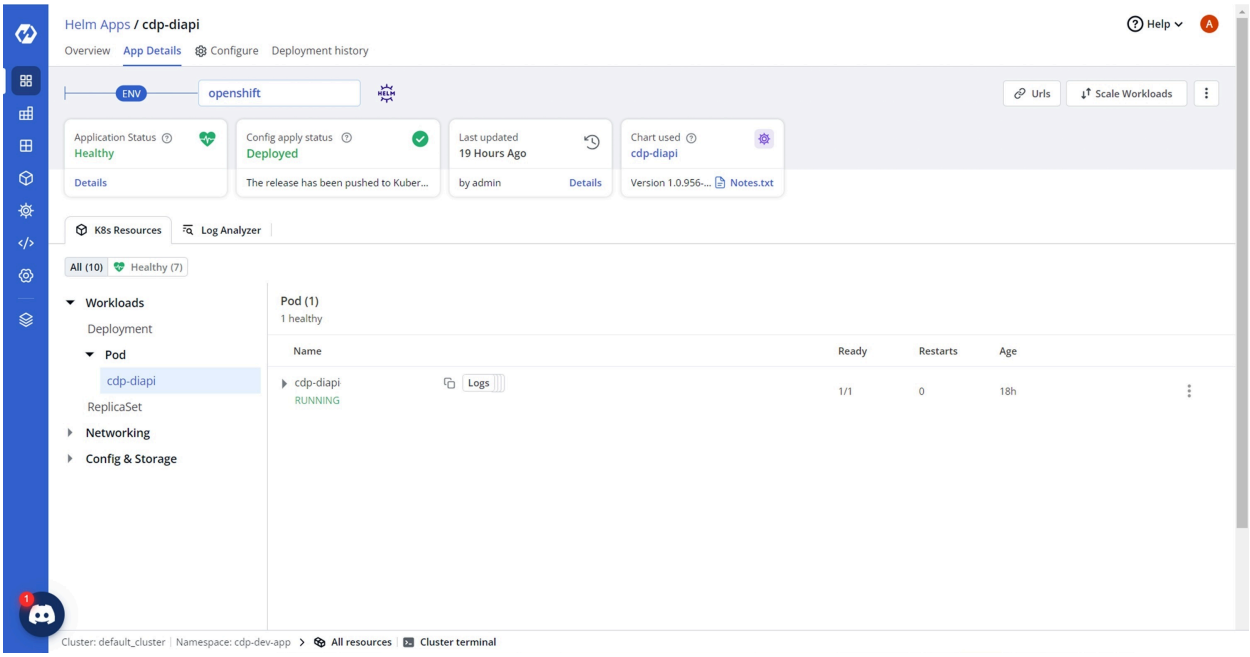
3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
aerospikenamespaceaps1: cdpstore
      aerospikenamespaceuse1: cdpstore
      aps1KafkaAuthenticationTimeout: "5000"
      aps1KafkaClientId: DMPDiapiProducer
      aps1KafkaConnectionTimeout: "3000"
      aps1KafkaReauthenticationThreshold: "10000"
```

```
    aps1KafkaSslEnabled: "false"
    cdpSenderTopic: cdp_fb_conv_api
    database: vrm
    dbconnectionLimit: "1"
    dbport: "3306"
    fromEmail: ""
    isEncryptionEnabled: "true"
    kafkaProducerEventType: "104"
    kafkaRegions: '{"AP_SOUTH_1": "aps1"}'
    kafkaSaslMechanism: SCRAM-SHA-512
    kafkasource: dataingestionapi
    kafkatopic: dmp_sst_nba_di_api
    nodeProcessMemoryLimit: 1500M
    toEmail: ""
    use1KafkaAuthenticationTimeout: "5000"
    use1KafkaClientId: DMPDiapiProducer
    use1KafkaConnectionTimeout: "3000"
    use1KafkaReauthenticationThreshold: "10000"
    use1KafkaSslEnabled: "false"
```



4. On successful deployment, validate the deployment as shown below.

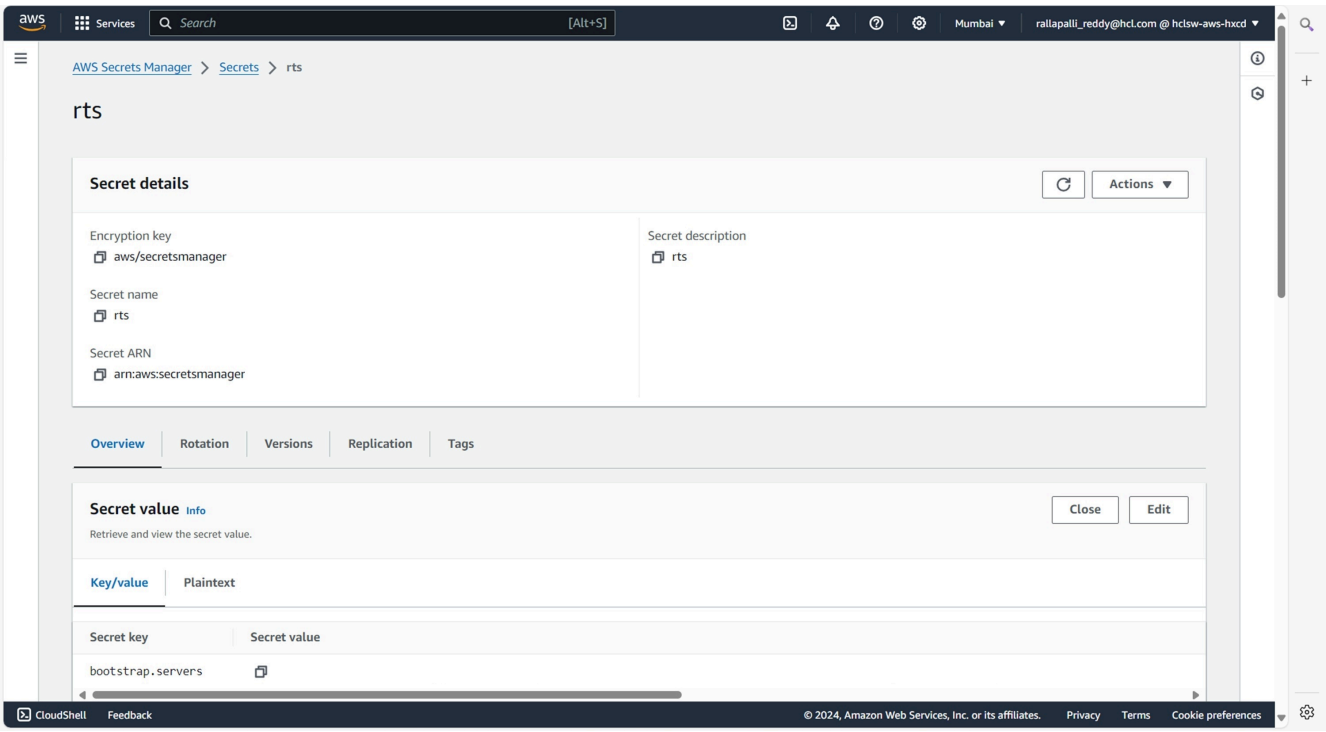


Deploying CDP RTS

This section provides detailed instructions on how to deploy HCL CDP RTS using the Devtron in the AWS.

Prerequisites:

Make sure to create rts secret with required data in the AWS secret manager before deploying CDP RTS.



To create the rts secret in the AWS secret manager, follow the steps below:

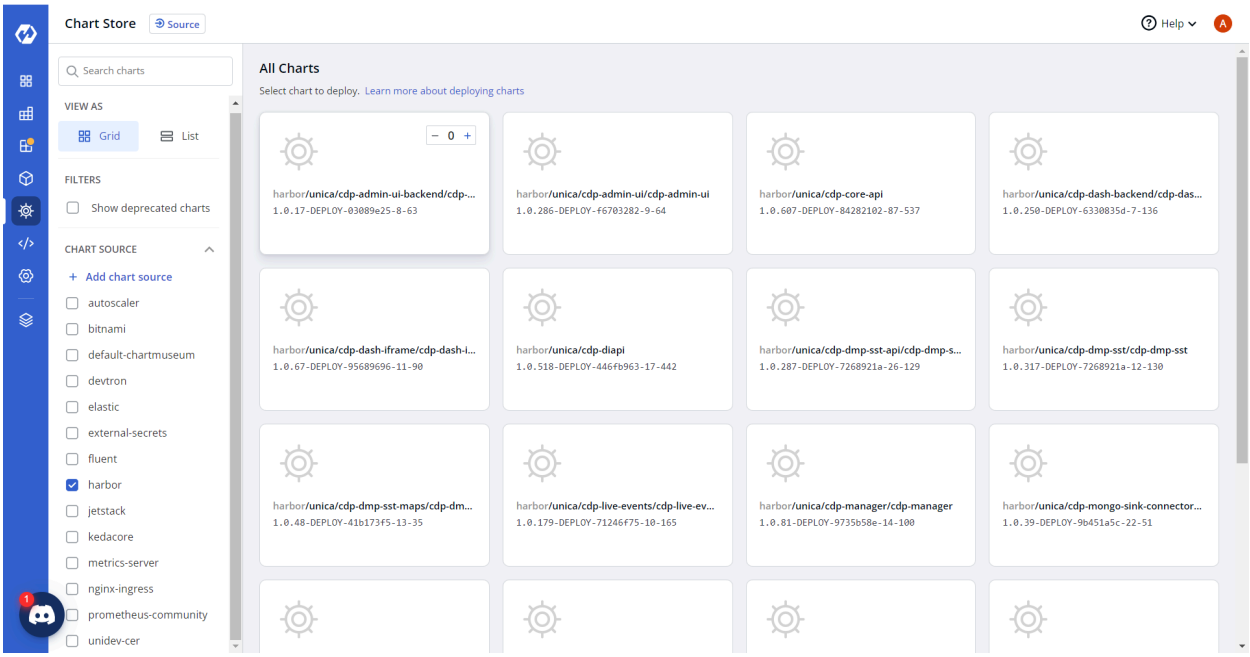
1. Create a rts secret sample key and value in the rts secret, and update ConfigMaps data with actual values.

```
{
  "KafkaConsumerConfig": "<KafkaConsumerConfig>",
  "KafkaProducerConfig": "<KafkaProducerConfig>",
  "MysqlDb": "<MysqlDb>",
  "MysqlHost": "<MysqlHost>",
  "MysqlPass": "<MysqlPass>",
  "MysqlPort": "<MysqlPort>",
  "MysqlUser": "<MysqlUser>",
  "bootstrap.servers": "<bootstrap.servers>",
  "sasl.mechanism": "<sasl.mechanism>",
  "security.protocol": "<security.protocol>"
}
```

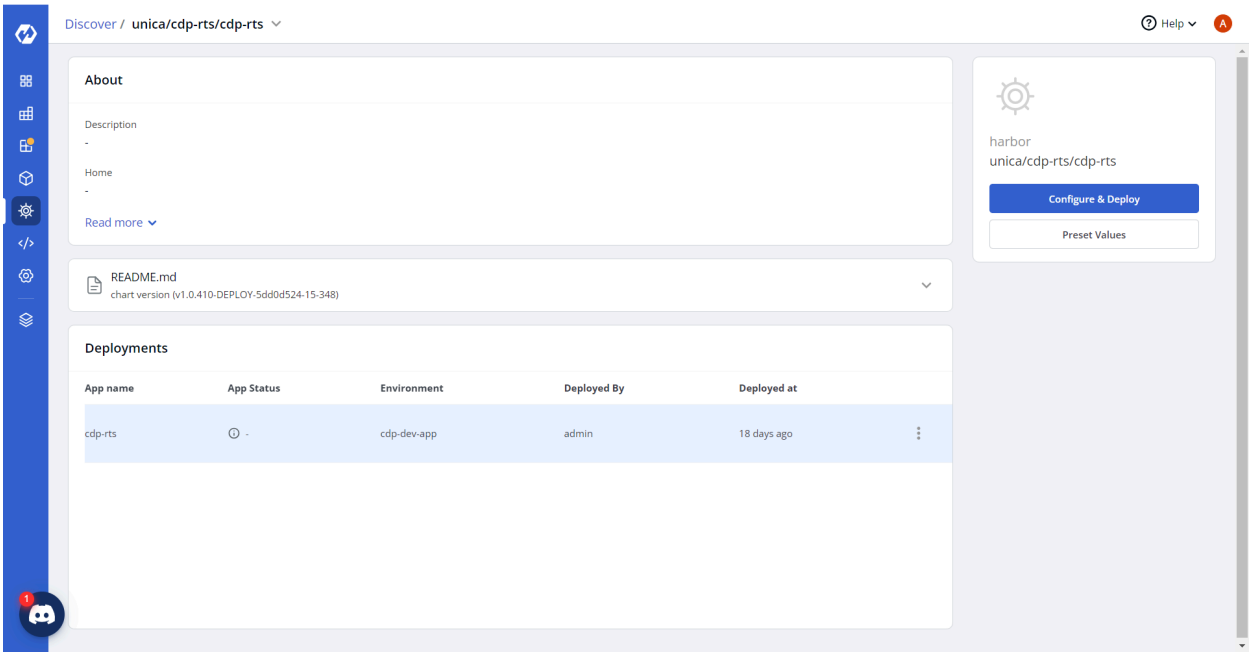
Deploying CDP RTS

To deploy the CDP RTS, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-rts chart to deploy.



2. Now, configure and deploy the CDP RTS charts.

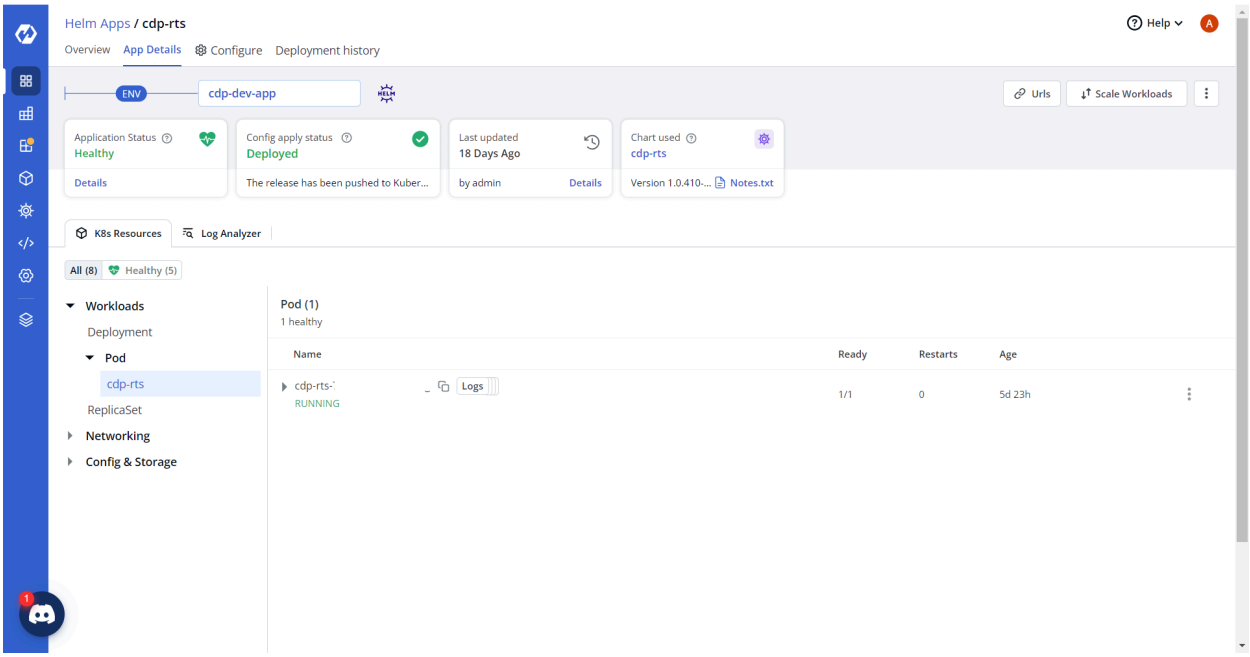


3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
FROM_EMAIL: <FROM_EMAIL>
KAFKA_CONSUMER_GROUP_ID: <k8s-rts-prod>
KAFKA_PRODUCER_FAILURE_TOPIC: <dmp_rt_failure>
KAFKA_PRODUCER_RAMANUJAN_TOPIC: <dmp_rt_ramanujan>
KAFKA_PRODUCER_TOPIC: <dmp_rt_segments>
KAFKA_SST_TOPIC: <dmp_sst_rt_profile>
```

```
KAFKA_TOPIC_LIST: <dmp_sst_rt_profile>
MODEL_CONFIG_LOCAL_PATH: /disk1/rts/config/predictiveSegmentationModel.ini
MODEL_CONFIG_S3_PATH: s3://<s3_bucket_name>/sst/predictivesegments/rtson/predictiveSegmentationModel.ini
MODEL_CUTOFFS_LOCAL_PATH: /disk1/rts/config/predictiveSegmentationCutoffs.ini
MODEL_CUTOFFS_S3_PATH:
 s3://<s3_bucket_name>/sst/predictivesegments/rtson/predictiveSegmentationCutoffs.ini
PMML_MODEL_CONFIG_LOCAL_PATH: /disk1/rts/config/predictivesegments/pmml_model_config/
PMML_MODEL_CONFIG_S3_PATH: s3://<s3_bucket_name>/sst/predictivesegments/pmml_model_config/rtson/
PMML_MODEL_DIRECTORY_LOCAL_PATH: /disk1/rts/config/predictivesegments/pmml_models/
PMML_MODEL_DIRECTORY_S3_PATH: s3://<s3_bucket_name>/sst/predictivesegments/pmml_models/rtson/
TO_EMAIL: <TO_EMAIL>
TSDB_HOST: <TSDB_HOST>
TSDB_PORT: <TSDB_PORT>
```

4. On successful deployment, validate the deployment as shown below.

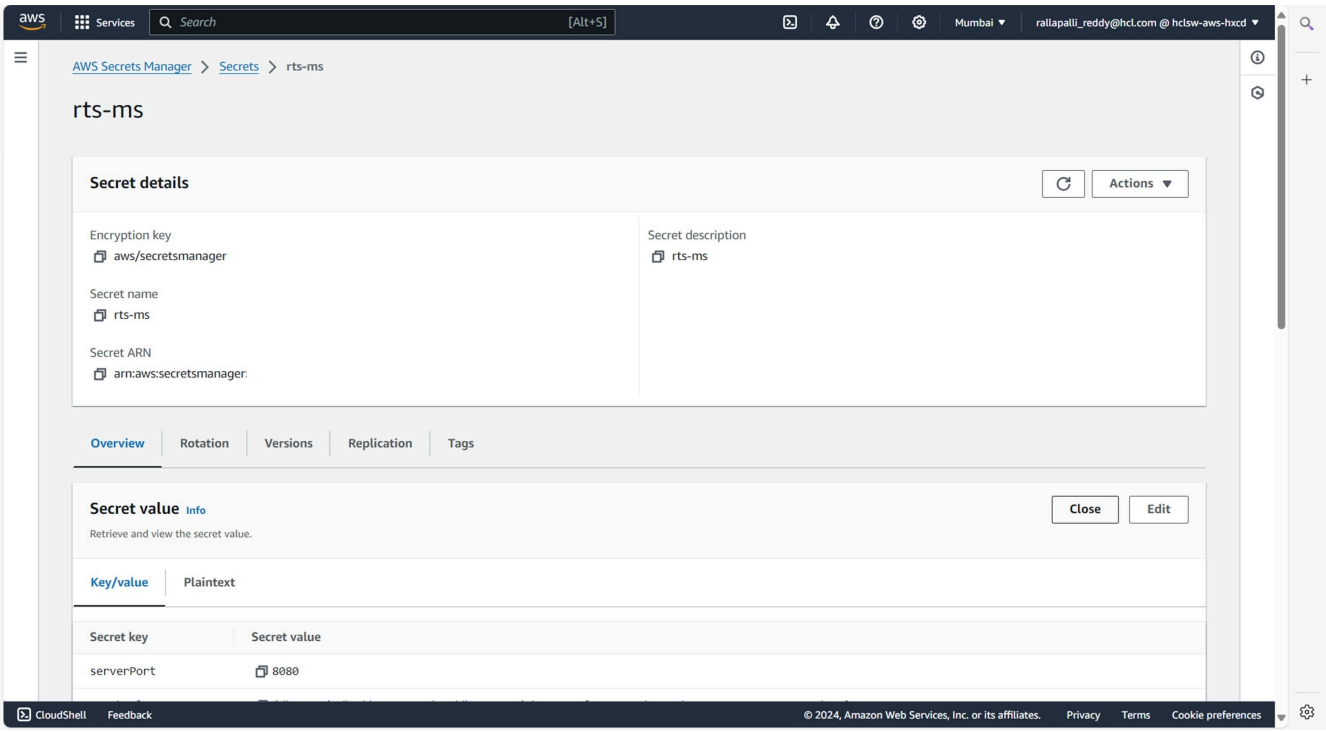


Deploying CDP RTS MS

This section provides detailed instructions on how to deploy HCL CDP RTS MS using the Devtron in the AWS.

Prerequisites:

Make sure to create rts-ms secret with required data in the AWS secret manager before deploying CDP RTS MS.



To create the rts-ms secret in the AWS secret manager, follow the steps below:

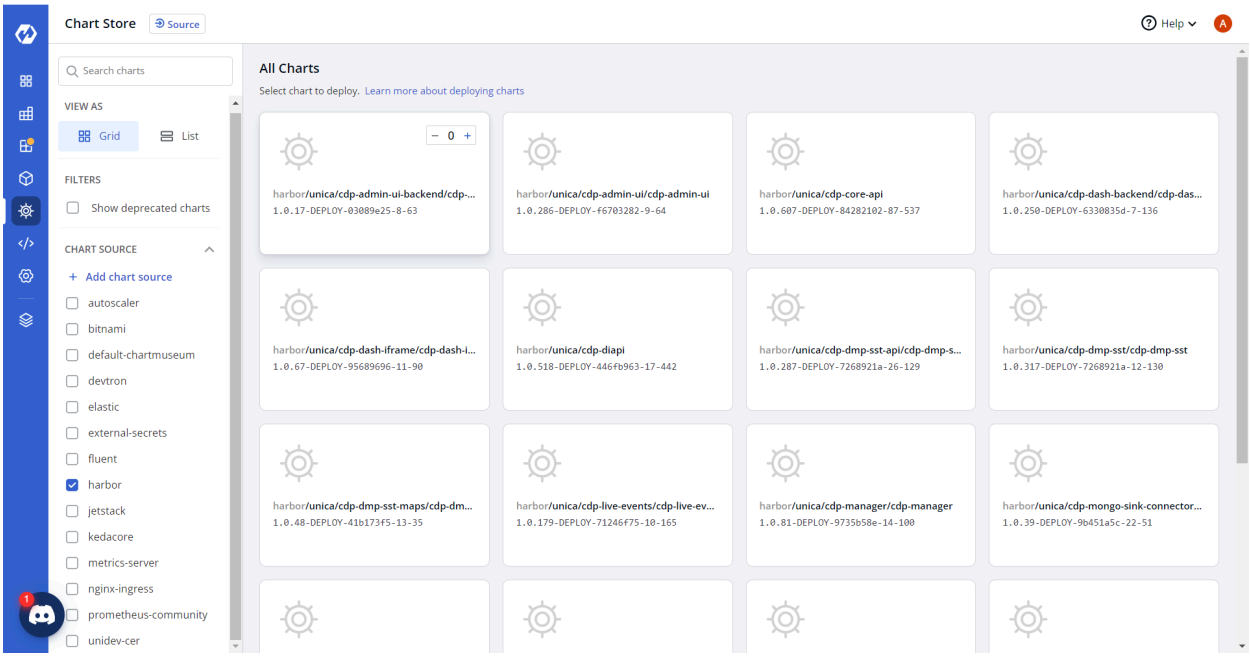
1. Create a rts-ms secret sample key and value in the rts-ms secret, and update ConfigMaps data with actual values.

```
{
  "MongoDbDatabase": "<MongoDbDatabase>",
  "MongoDbUri": "mongodb://<user>:<password>@<ip>:<port>/?directConnection=true",
  "TsdbHost": "<TsdbHost>",
  "VrmDbPassword": "<TsdbHost>",
  "VrmDbUrl": "jdbc:mariadb://<ip>:<port>/<dbName>?autoReconnect=true",
  "VrmDbUsername": "<VrmDbUsername>",
  "serverPort": "<serverPort>"
}
```

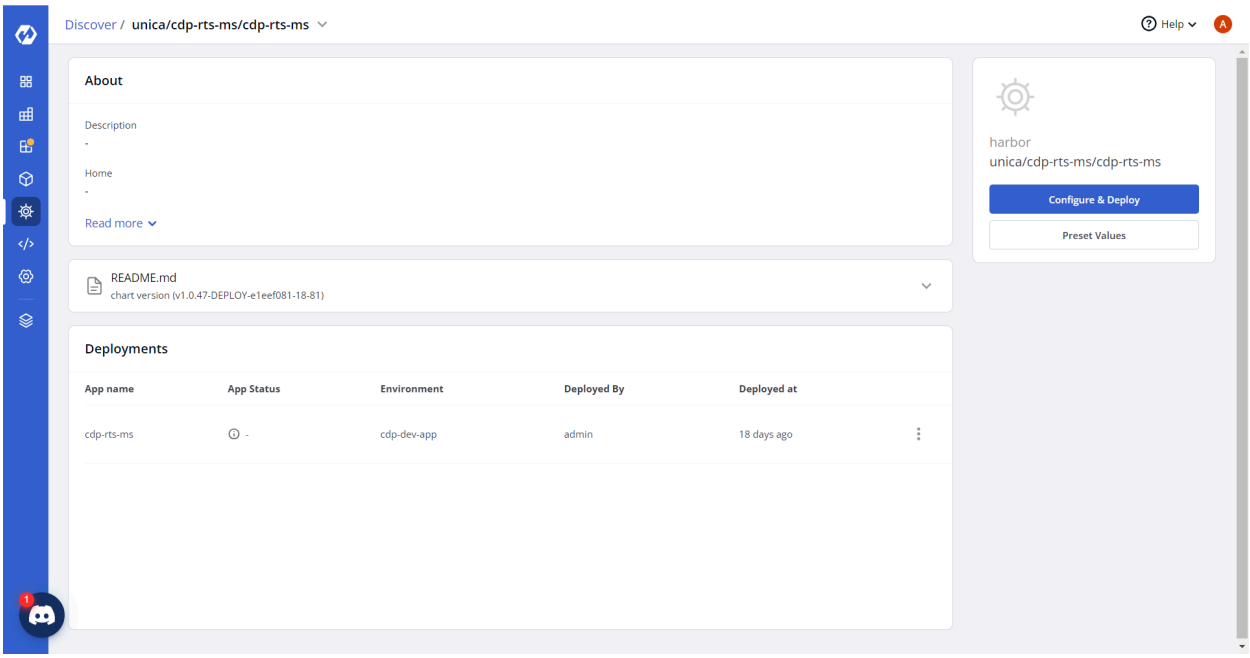
Deploying CDP RTS MS

To deploy the CDP RTS MS, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-rts-ms chart to deploy.



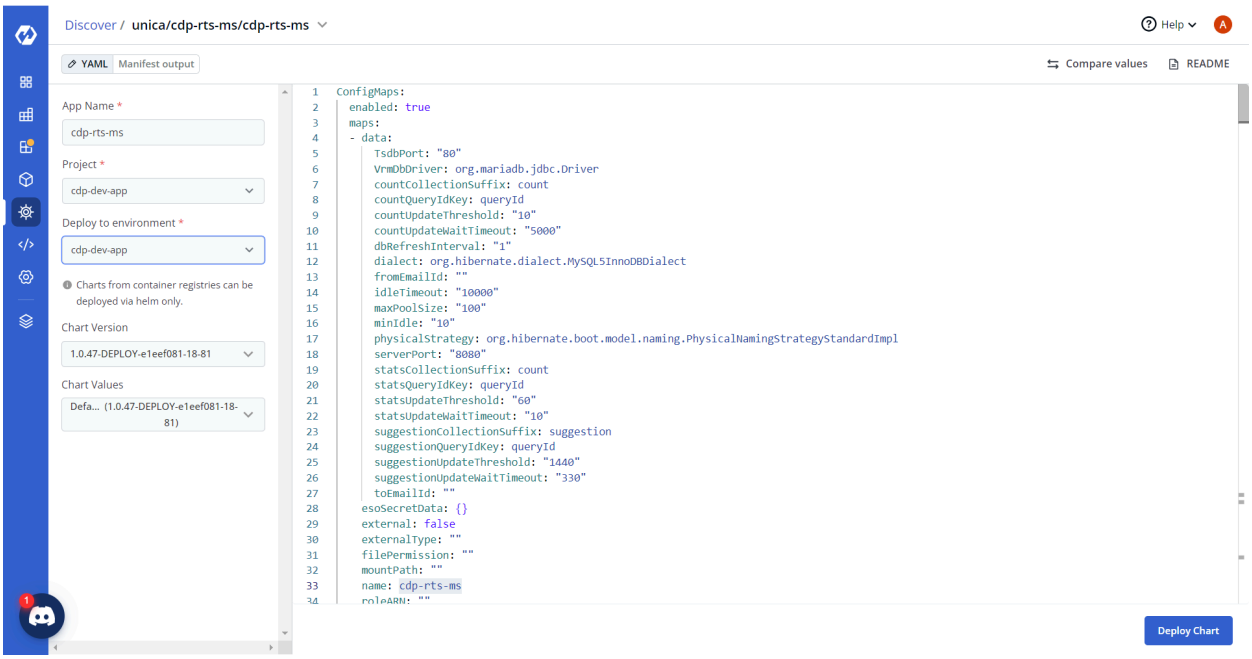
2. Now, configure and deploy the CDP RTS MS charts.



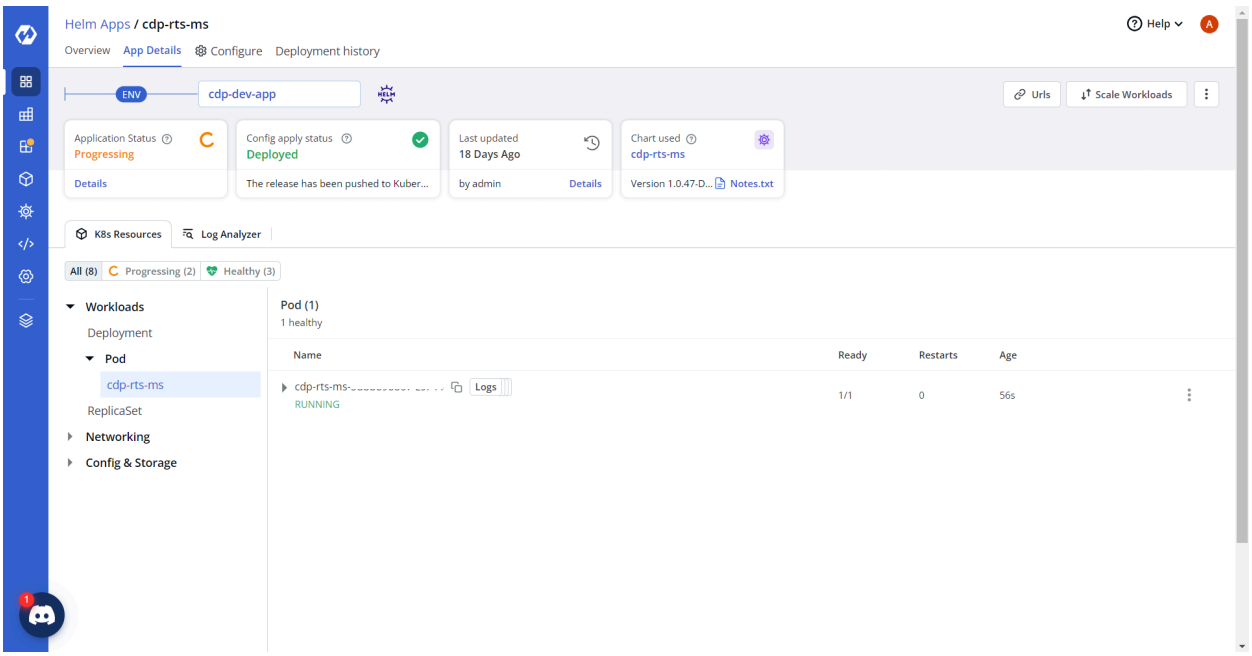
3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
FROM_EMAIL: <FROM_EMAIL>
TO_EMAIL: <TO_EMAIL>
TSDB_HOST: <TSDB_HOST>
TSDB_PORT: <TSDB_PORT>
VrmDbDriver: <org.mariadb.jdbc.Driver>
dialect: <org.hibernate.dialect.MySQL5InnoDBDialect>
```

```
physicalStrategy: <org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl>
```



4. On successful deployment, validate the deployment as shown below.

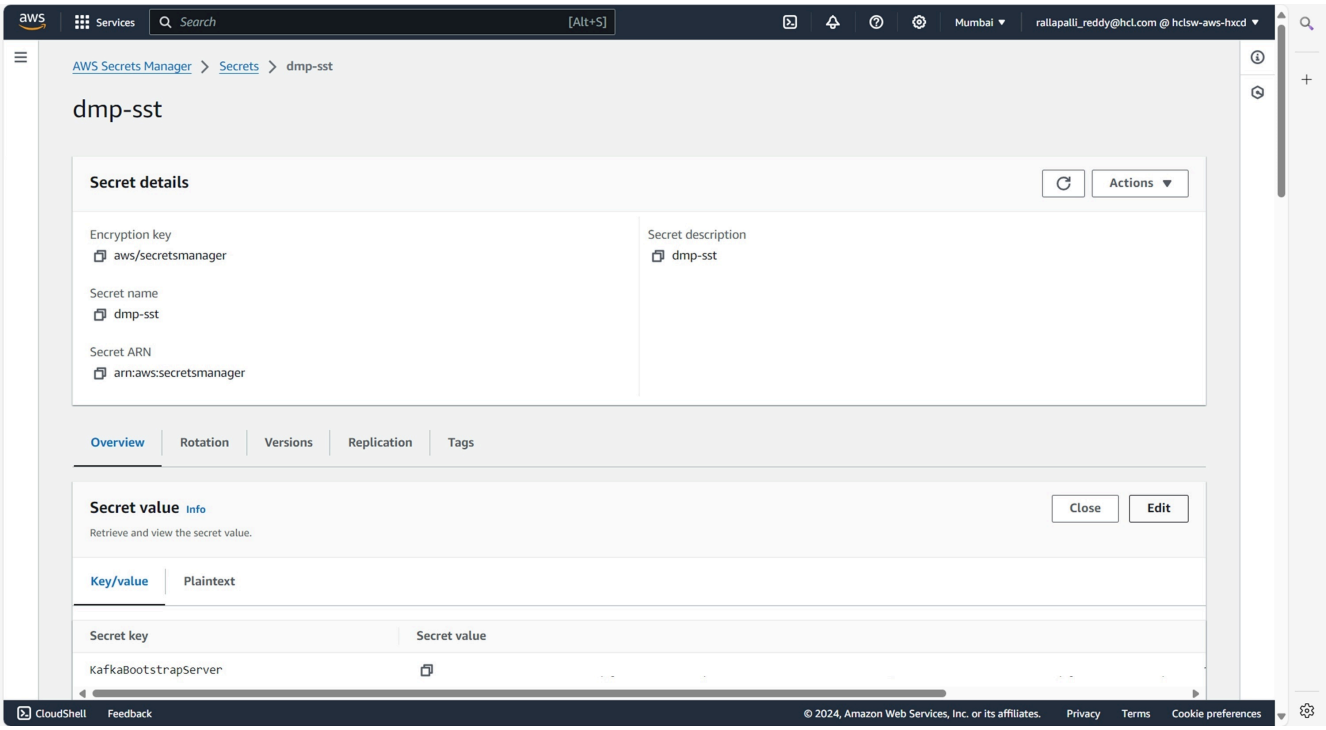


Deploying CDP DMP SST

This section provides detailed instructions on how to deploy HCL CDP DMP SST using the Devtron in the AWS.

Prerequisites:

Make sure to create dmp-sst secret with required data in the AWS secret manager before deploying CDP DMP SST.



To create the dmp-sst secret in the AWS secret manager, follow the steps below:

1. Create a dmp-sst secret sample key and value in the dmp-sst secret, and update ConfigMaps data with actual values.

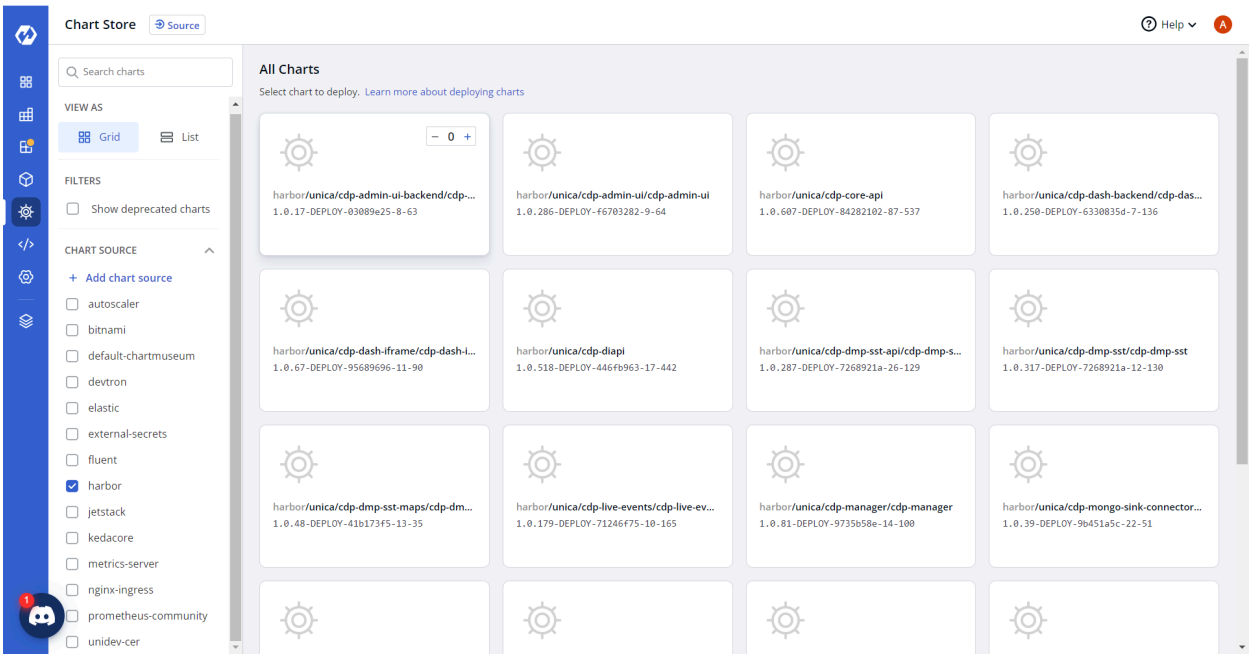
```
{
  "AWSRegion": "<AWSRegion>",
  "AerospikeNamespace": "<AerospikeNamespace>",
  "AerospikeServerAddress": "<AerospikeServerAddress>",
  "AerospikeServerPort": "<AerospikeServerPort>",
  "AerospikeTTL": "15552000",
  "AerospikeTimeOut": "10000",
  "AlertEmail": "",
  "AlertFromEmail": "",
  "CookieCacheThreshold": "60",
  "DbDriver": "<DbDriver>",
  "DbHost": "<DbHost>",
  "DbPassword": "<DbPassword>",
  "DbPort": "<DbPort>",
  "DbUrl": "jdbc:mariadb://<ip:port>/dbName",
  "DbUsername": "<DbUsername>",
  "GraphDBEnable": "false",
  "GraphDBEndpoint": "<GraphDBEndpoint>",
  "GraphDBPort": "<GraphDBPort>",
  "HBaseDBEnable": "false",
  "HbaseZookeeperClientPort": "",
  "HbaseZookeeperQuorum": "",
  "KafkaBootstrapServer": "<KafkaBootstrapServer>",
```

```
  "KafkaConsumerGroup": "<dmp-sst-kafka-k8>",
  "KafkaPassword": "<KafkaPassword>",
  "KafkaProducerHostName": "<KafkaPassword>",
  "KafkaProducerTopic": "<dmp_sst_rt_profile>",
  "KafkaProfileProducerTopic": "<dmp_sst_profile>",
  "KafkaTopicConsumerBatchSize": "<10,10,10,10>",
  "KafkaTopicConsumerDefaultBatchTimeoutMS": "<5000,5000,5000,5000>",
  "KafkaTopicConsumerDefaultBatchTimeoutTrigger": "<60000,10000,10000,10000>",
  "KafkaTopicConsumers": "<6,6,6,6>",
  "KafkaTopicList": "<dmp_sst_nba,dmp_sst_nba_di_api,analyze_post,dmp_sst_data>",
  "KafkaUsername": "<KafkaUsername>",
  "SstApiAuthKeys": "<HdxDeMMHvUyrqlSBHyaCaQGA>",
  "TsdbHost": "<TsdbHost>",
  "TsdbPort": "<TsdbPort>",
  "ZookeeperClients": "<15>",
  "ZookeeperLockAddress": "<cdp-dmp-sst-zookeeper.cdp-dev-app:2181>",
  "ZookeeperLockAddressOld": "",
  "ZookeeperRetrySleepTime": "<1000>",
  "region": "<region>",
  "type": "<realtime>"
}
```

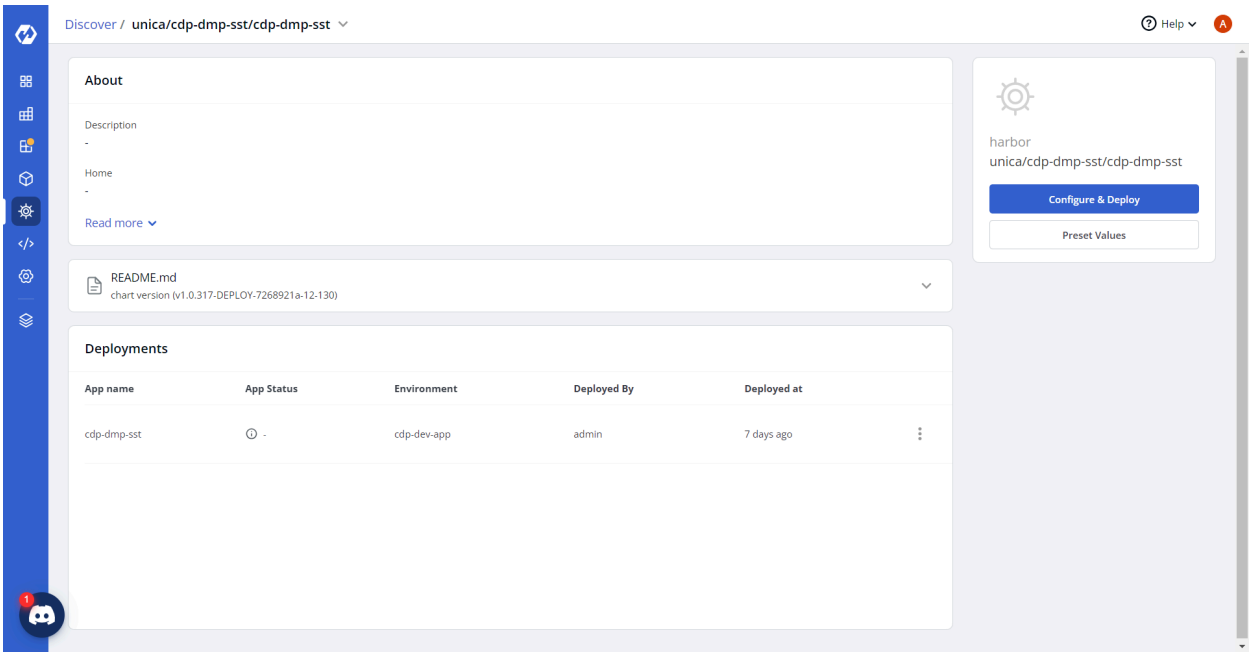
Deploying CDP DMP SST

To deploy the CDP DMP SST, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-dmp-sst chart to deploy.

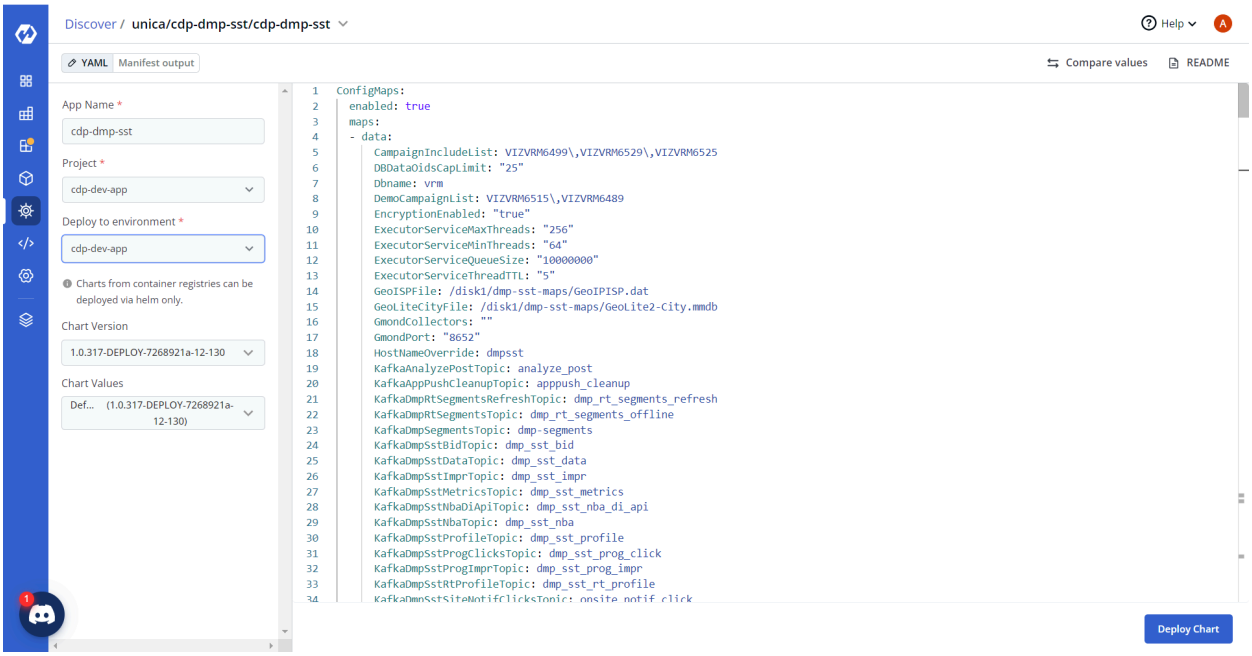


2. Now, configure and deploy the CDP DMP SST charts.

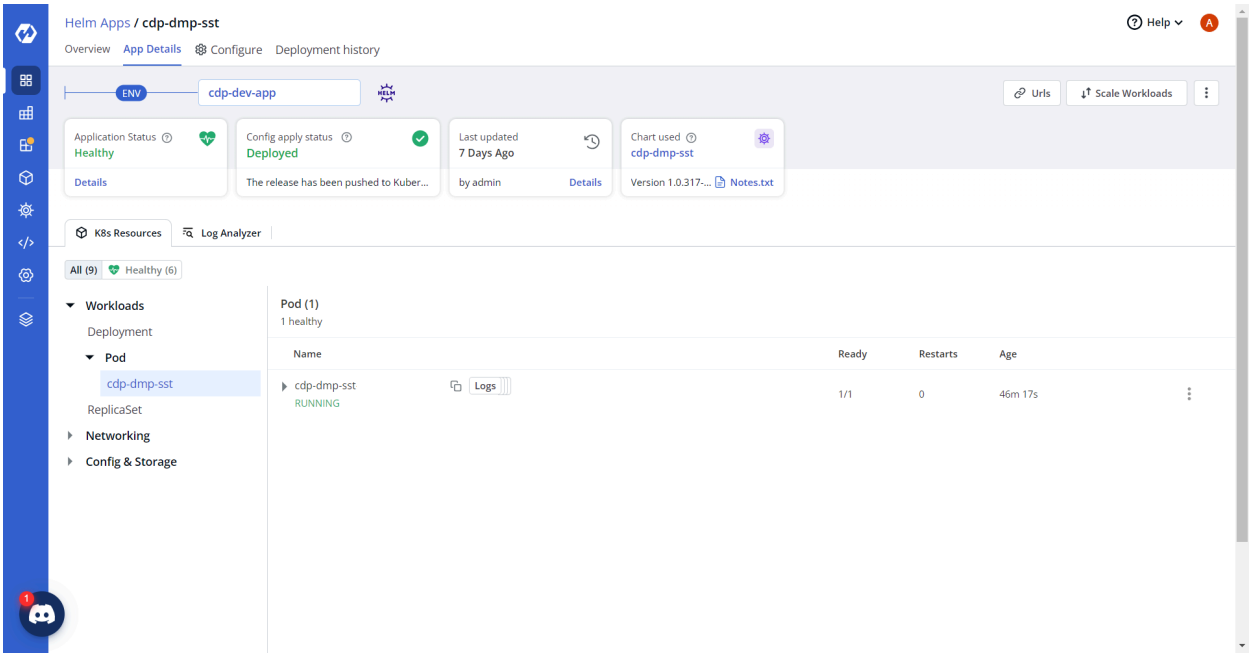


3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
ValidValuesConfig: /disk1/config/dmp-sst-maps/valid_values.json
ValidValuesS3Path: s3://<bucket_name>/dmp/config/predictivesegments/validvalues.tsv
ValuesMappingConfig: /disk1/config/dmp-sst-maps/values_mapping.json
ValuesMappingS3Path: s3://<bucket_name>/dmp/config/predictivesegments/mappingvalues.tsv
```



4. On successful deployment, validate the deployment as shown below.

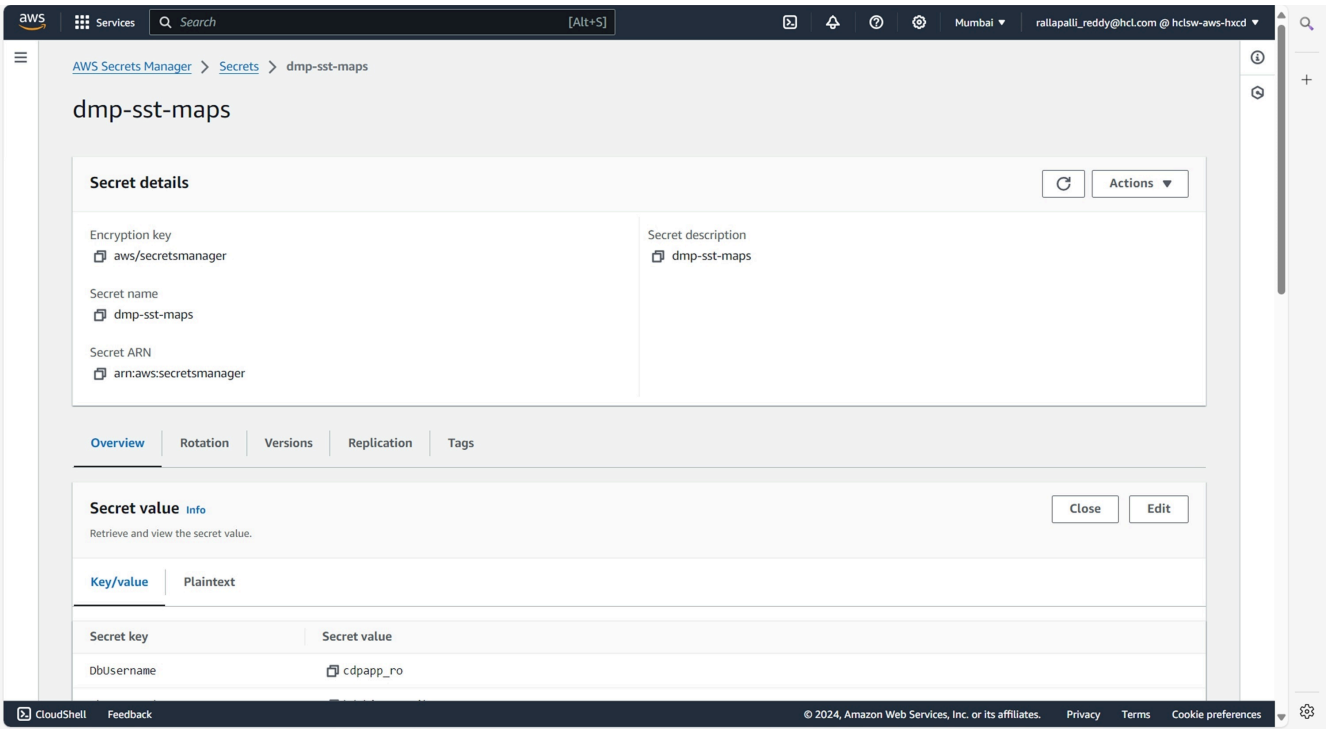


Deploying CDP DMP SST Maps

This section provides detailed instructions on how to deploy HCL CDP DMP SST Maps using the Devtron in the AWS.

Prerequisites:

Make sure to create a dmp-sst-maps secret with required data in the AWS secret manager before deploying CDP DMP SST Maps.



To create the dmp-sst-maps secret in the AWS secret manager, follow the steps below:

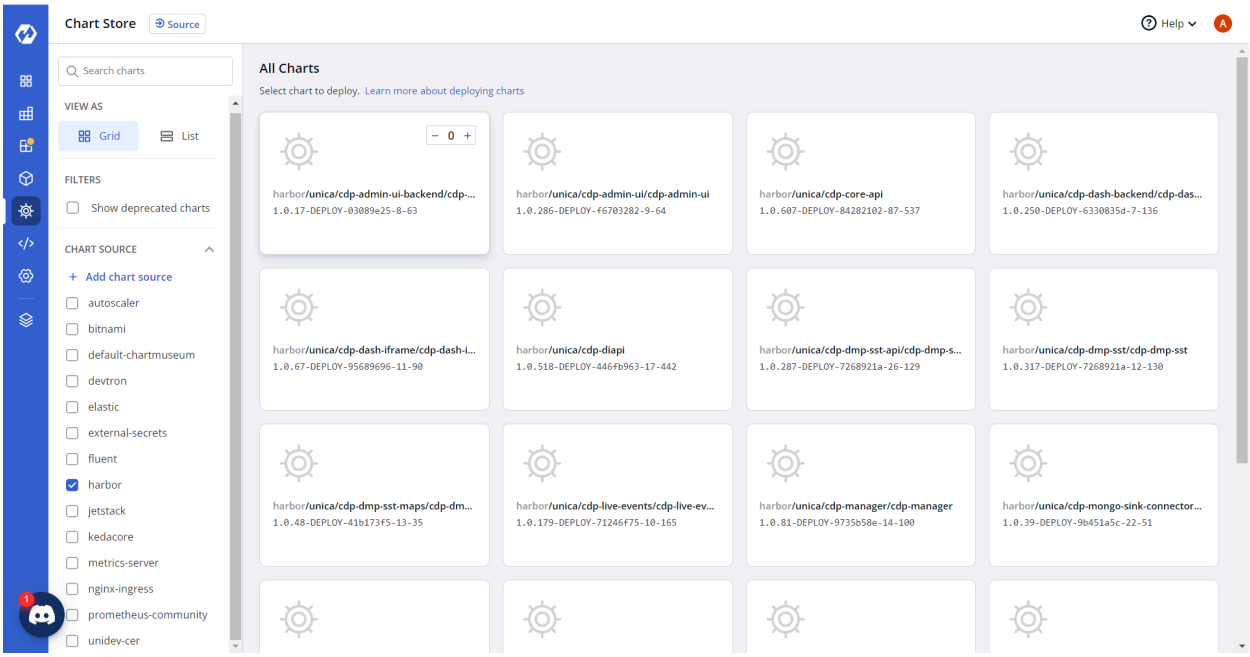
1. Create a dmp-sst-maps secret sample key and value in the dmp-sst-maps secret, and update ConfigMaps data with actual values.

```
{
  "AWSRegion": "<AWSRegion>",
  "AlertEmail": "",
  "AlertFromEmail": "",
  "DbDriver": "<com.mysql.cj.jdbc.Driver>",
  "DbHost": "<DbHost>",
  "DbPassword": "<DbPassword>",
  "DbPort": "<DbPort>",
  "DbUrl": "jdbc:mysql://ip:port/dbName",
  "DbUsername": "<DbUsername>",
  "TsdbHost": "<TsdbHost>",
  "TsdbPort": "<TsdbPort>",
  "region": "<region>",
  "type": "<b-aero-7-nodes>"
}
```

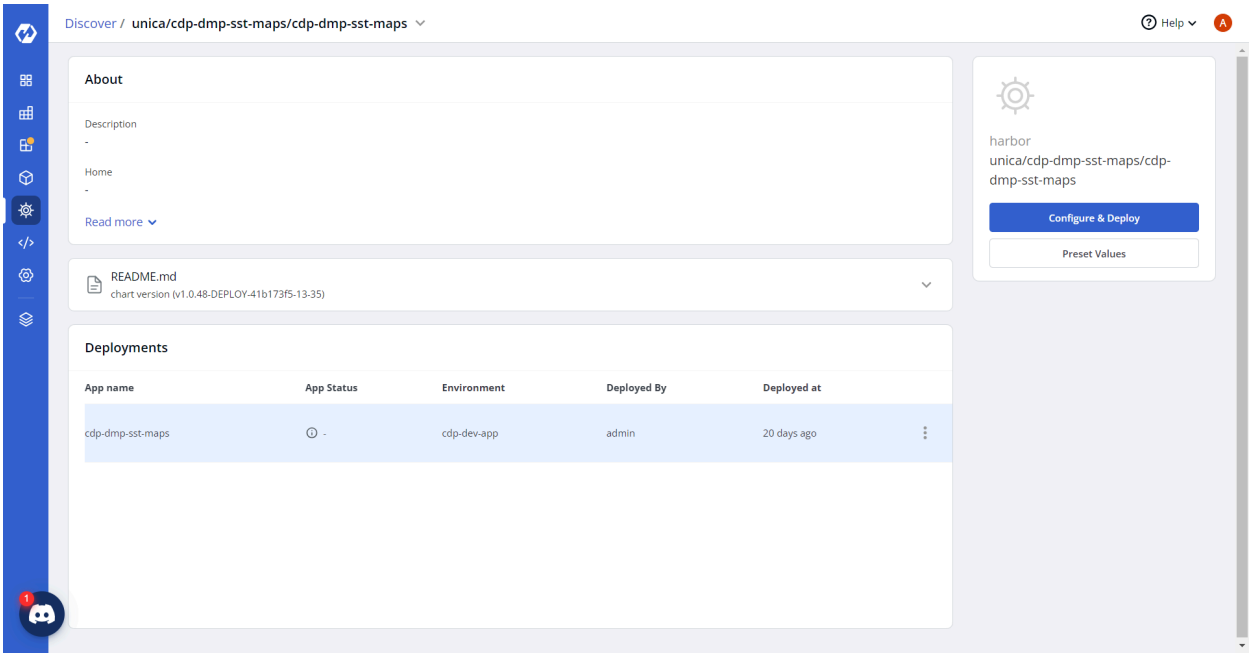
Deploying CDP DMP SST Maps

To deploy the CDP DMP SST Maps, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-dmp-sst-maps chart to deploy.

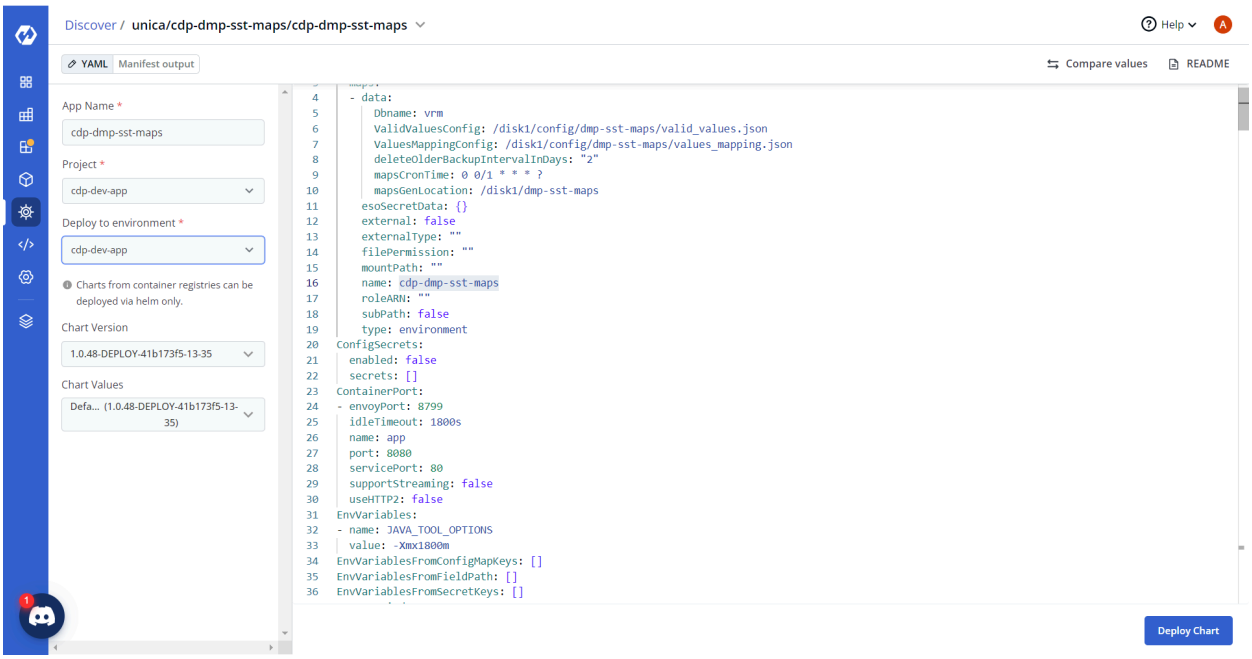


2. Now, configure and deploy the CDP DMP SST Maps charts.

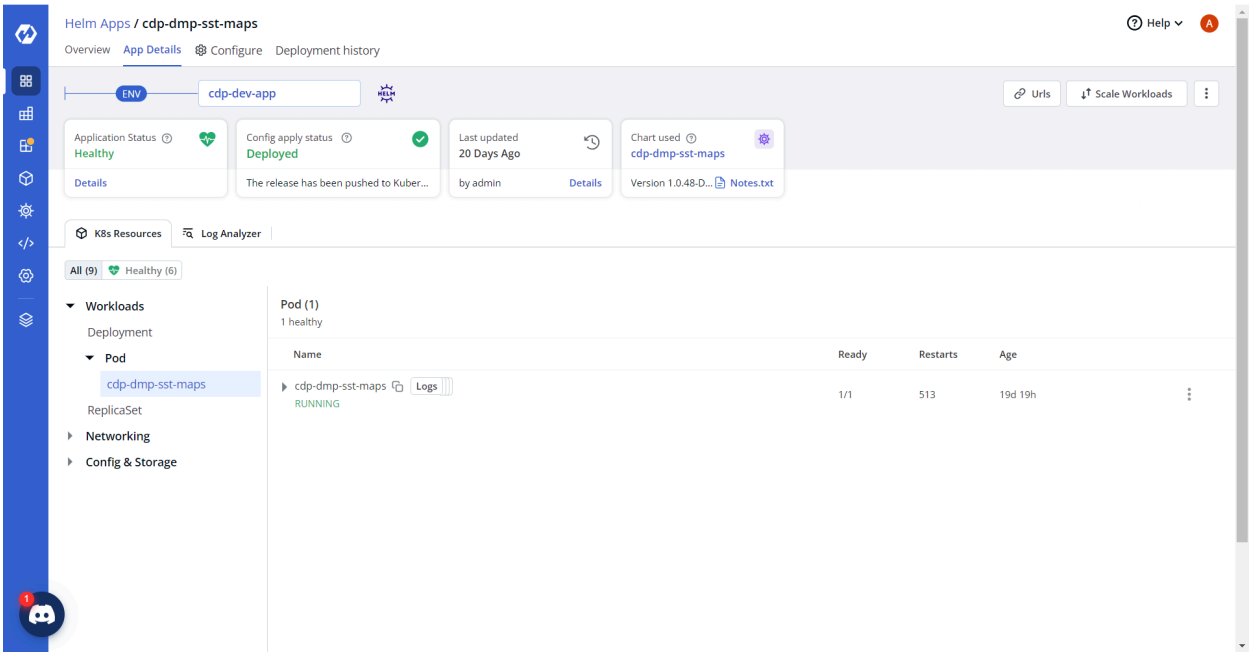


3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
ValidValuesConfig: /disk1/config/dmp-sst-maps/valid_values.json
ValuesMappingConfig: /disk1/config/dmp-sst-maps/values_mapping.json
mapsGenLocation: /disk1/dmp-sst-maps
DbName: <DbName>
```



4. On successful deployment, validate the deployment as shown below.

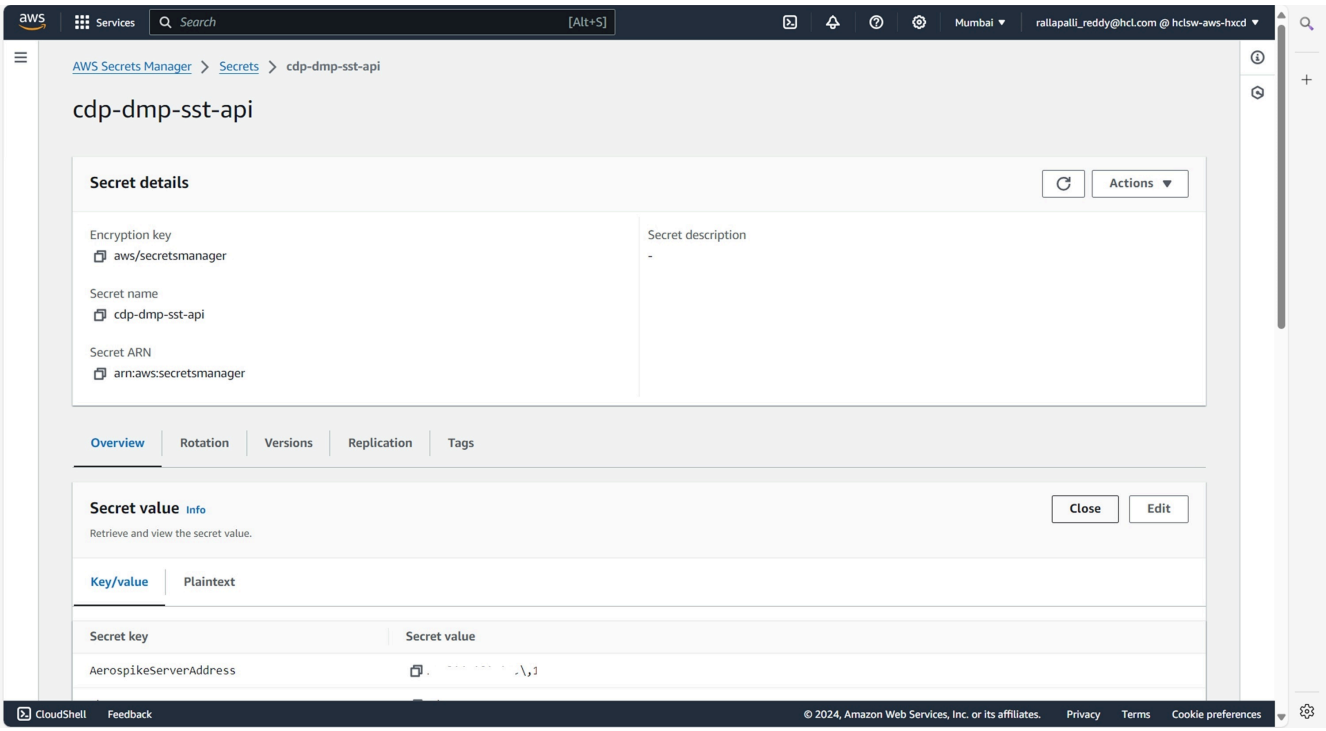


Deploying CDP DMP SST API

This section provides detailed instructions on how to deploy HCL CDP DMP SST API using the Devtron in the AWS.

Prerequisites:

Make sure to create a cdp-dmp-sst-api secret with required data in the AWS secret manager before deploying CDP DMP SST API.



To create the cdp-dmp-sst-api secret in the AWS secret manager, follow the steps below:

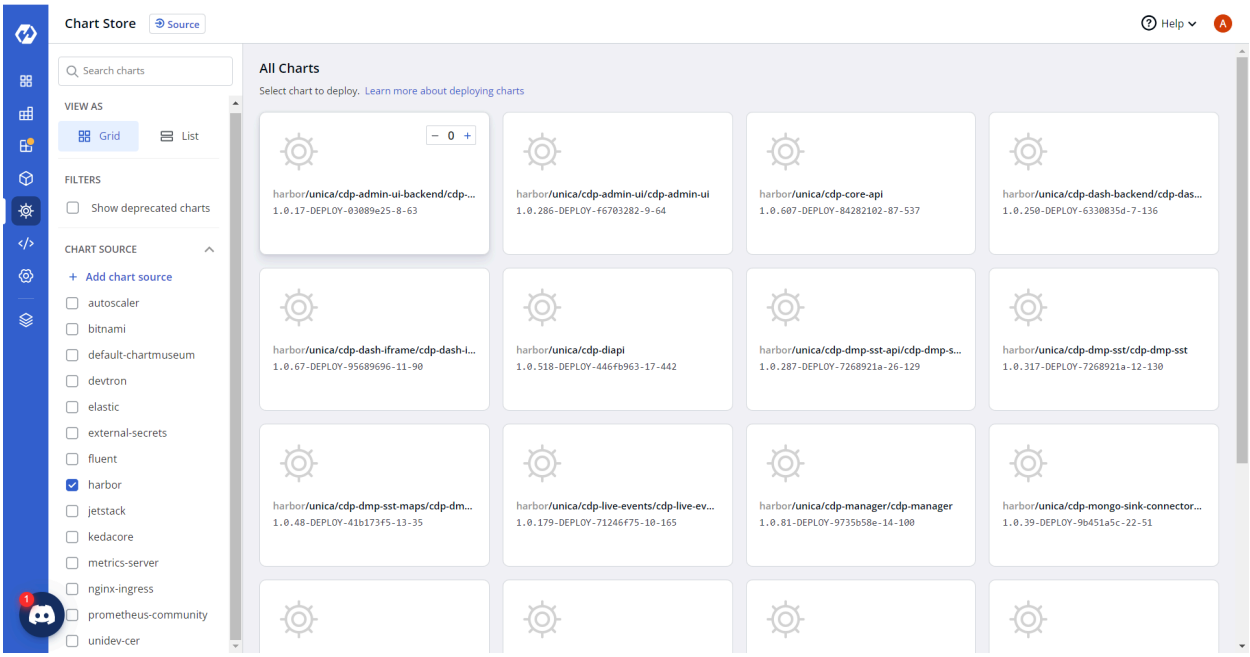
1. Create a cdp-dmp-sst-api secret sample key and value in the cdp-dmp-sst-api secret, and update ConfigMaps data with actual values.

```
{
  "AerospikeServerAddress": "<AerospikeServerAddress>",
  "DbHost": "<DbHost>",
  "DbPassword": "<DbPassword>",
  "DbUsername": "<DbUsername>",
  "GraphDBEndpoint": "",
  "HbaseZookeeperQuorum": "",
  "SstApiAuthKeys": "<SstApiAuthKeys>"
}
```

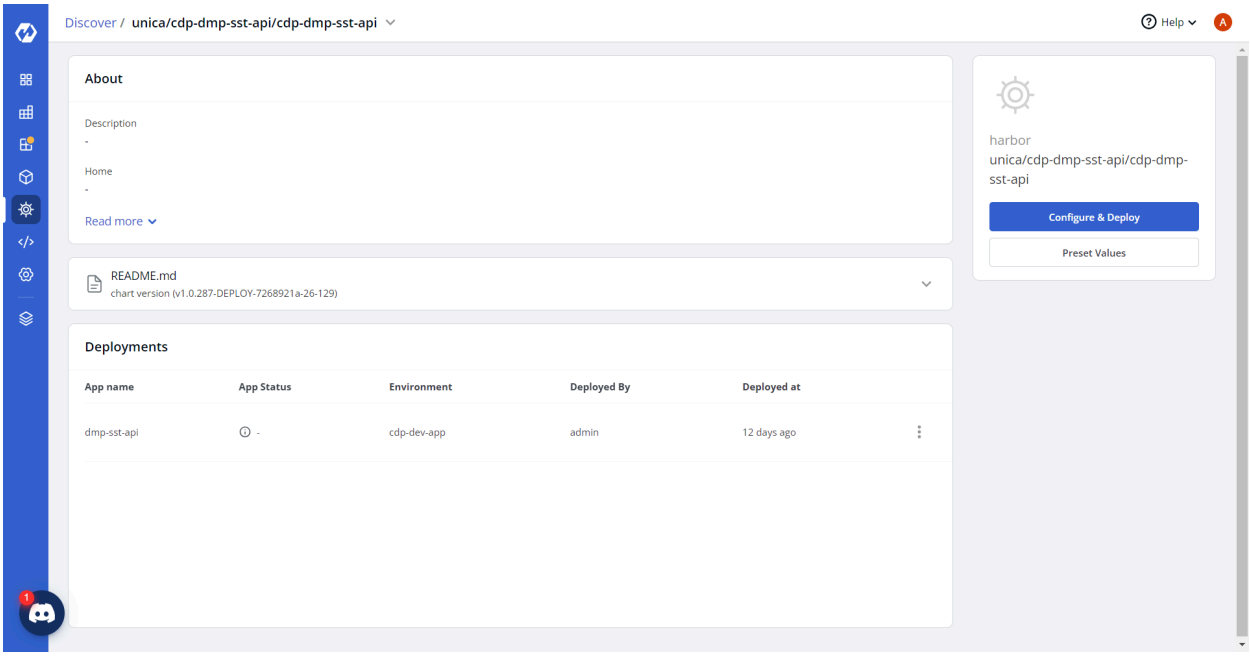
Deploying CDP DMP SST API

To deploy the CDP DMP SST API, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-dmp-sst-api chart to deploy.

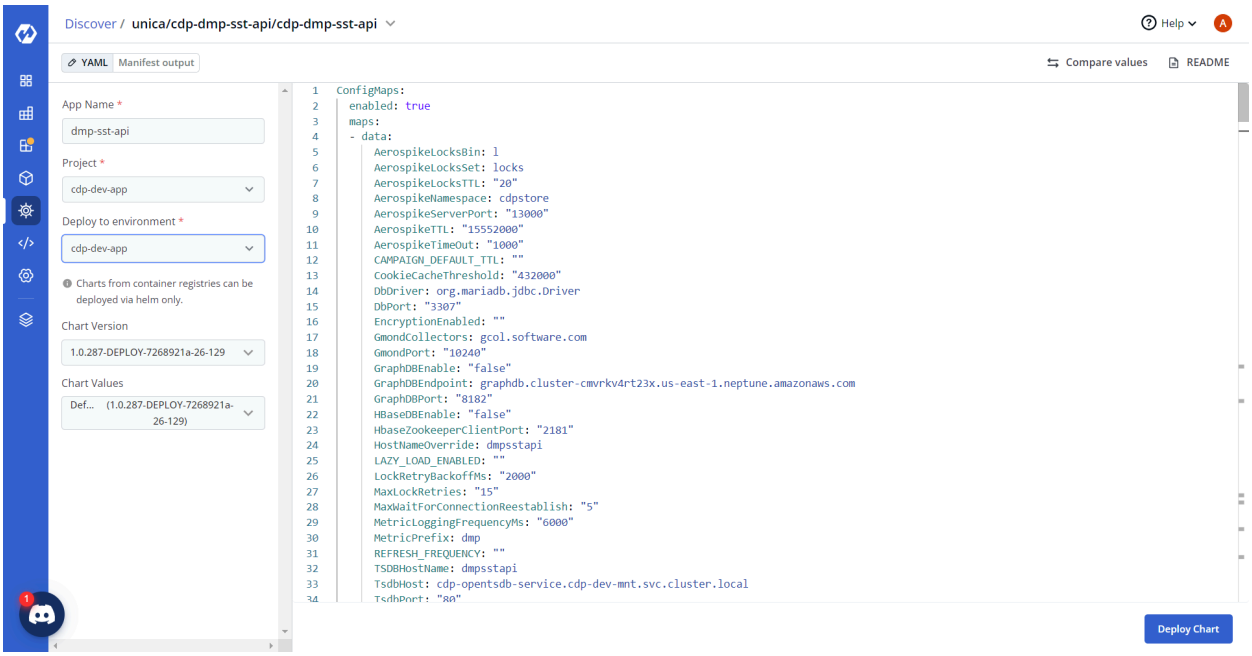


2. Now, configure and deploy the CDP DMP SST API charts.

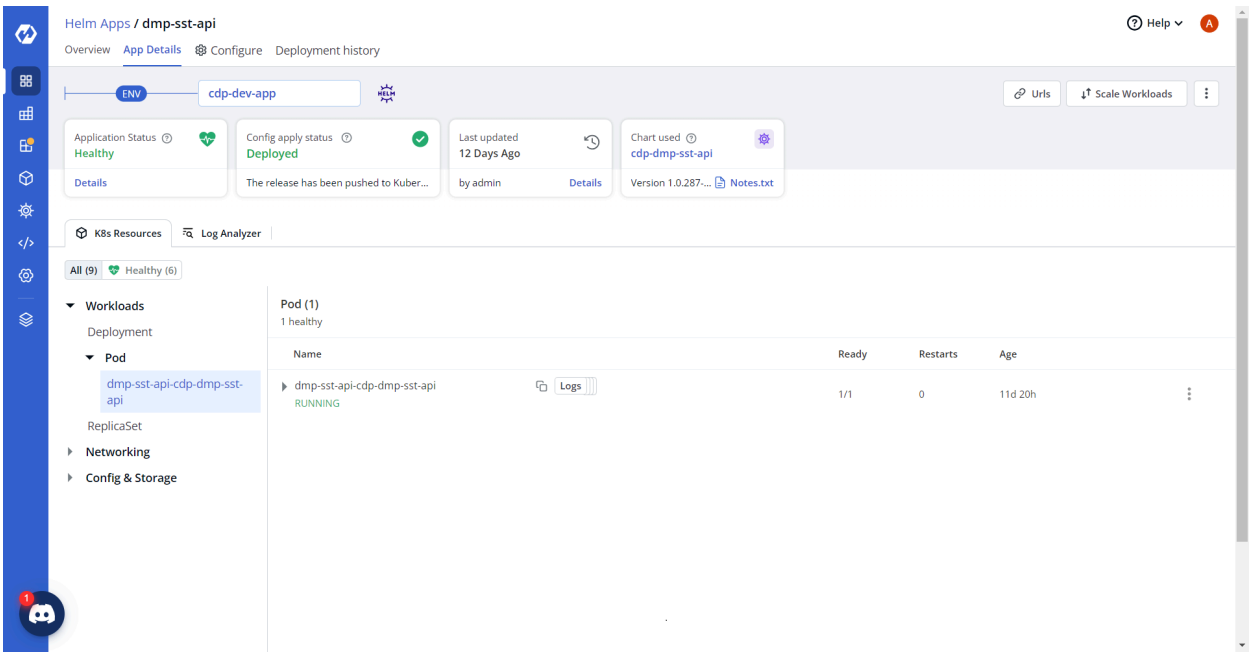


3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
DbDriver: <DbDriver>
DbPort: <DbPort>
EncryptionEnabled: ""
TsdbHost: <TsdbHost>
TsdbPort: <TsdbPort>
```



4. On successful deployment, validate the deployment as shown below.



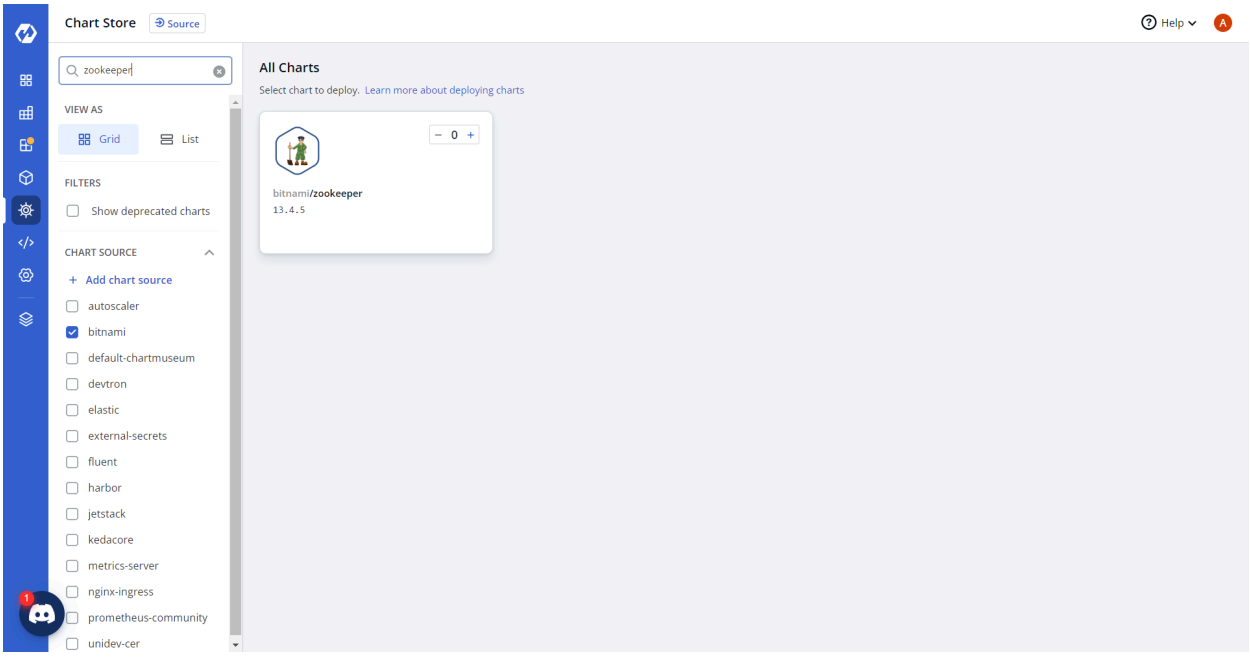
Deploying CDP DMP SST Zookeeper

This section provides detailed instructions on how to deploy HCL CDP DMP SST Zookeeper using the Devtron in the AWS.

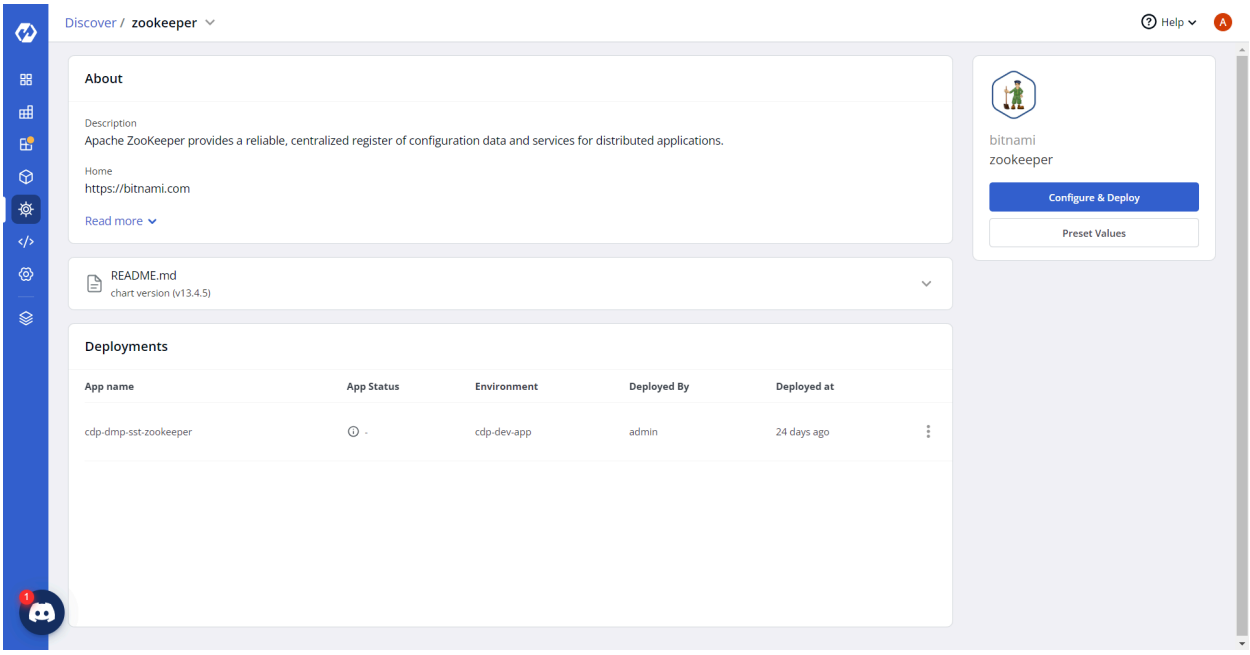
Deploying CDP DMP SST Zookeeper

To deploy the CDP DMP SST Zookeeper, follow the steps below:

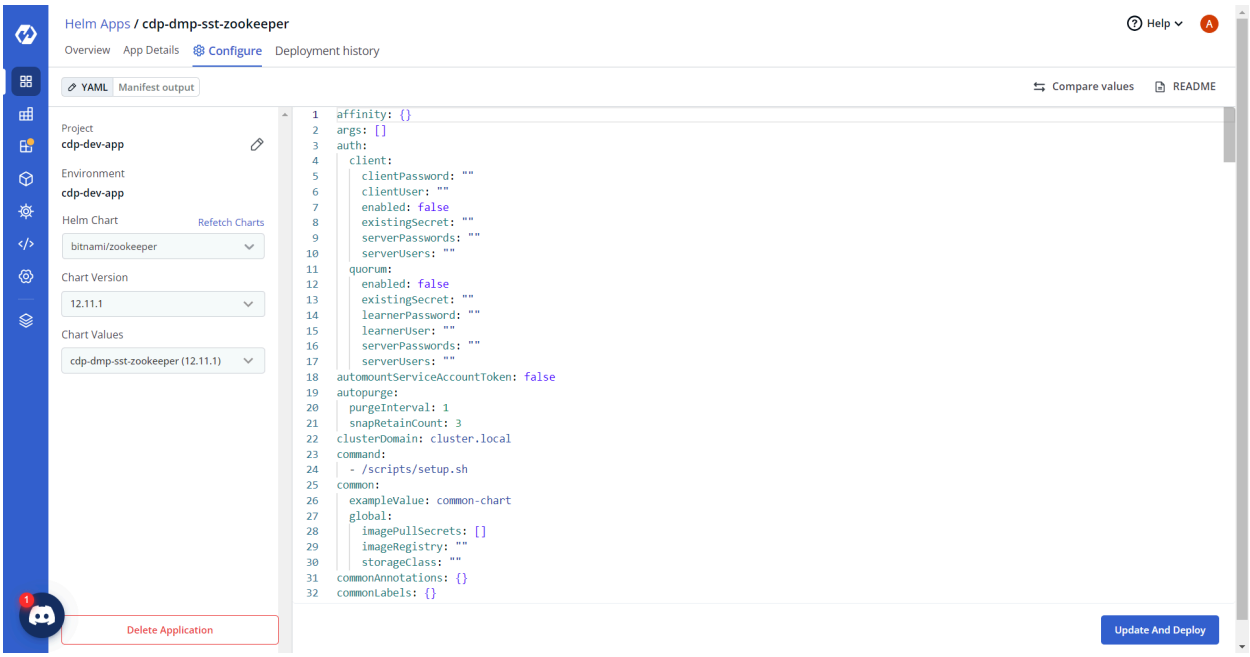
1. Navigate to the Devtron **Chart Store**, and select the bitnami/zookeeper chart to deploy.



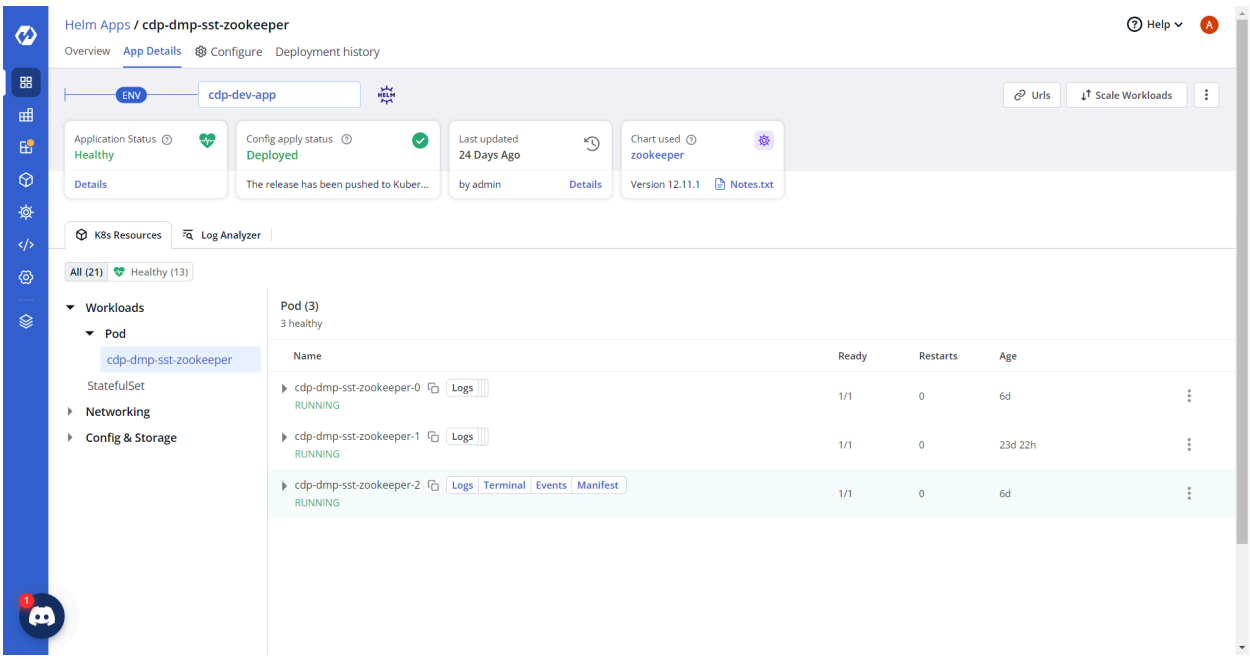
2. Now, configure and deploy the CDP DMP SST Zookeeper charts.



3. In the **YAML** section, update the configuration, and deploy the charts.



4. On successful deployment, validate the deployment as shown below.

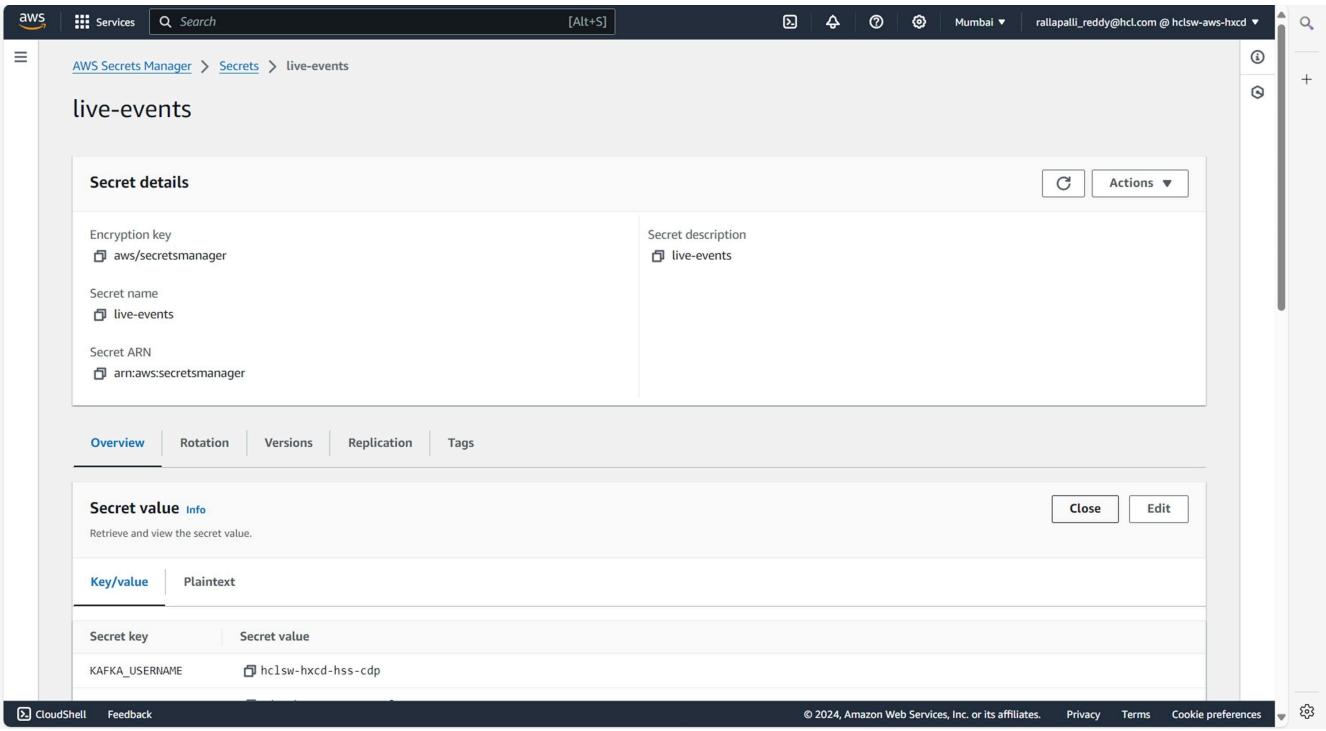


Deploying CDP Live Events

This section provides detailed instructions on how to deploy HCL CDP Live Events using the Devtron in the AWS.

Prerequisites:

Make sure to create a live-events secret with required data in the AWS Secrets Manager before deploying CDP Live Events.



To create the live-events secret in the AWS secret manager, follow the steps below:

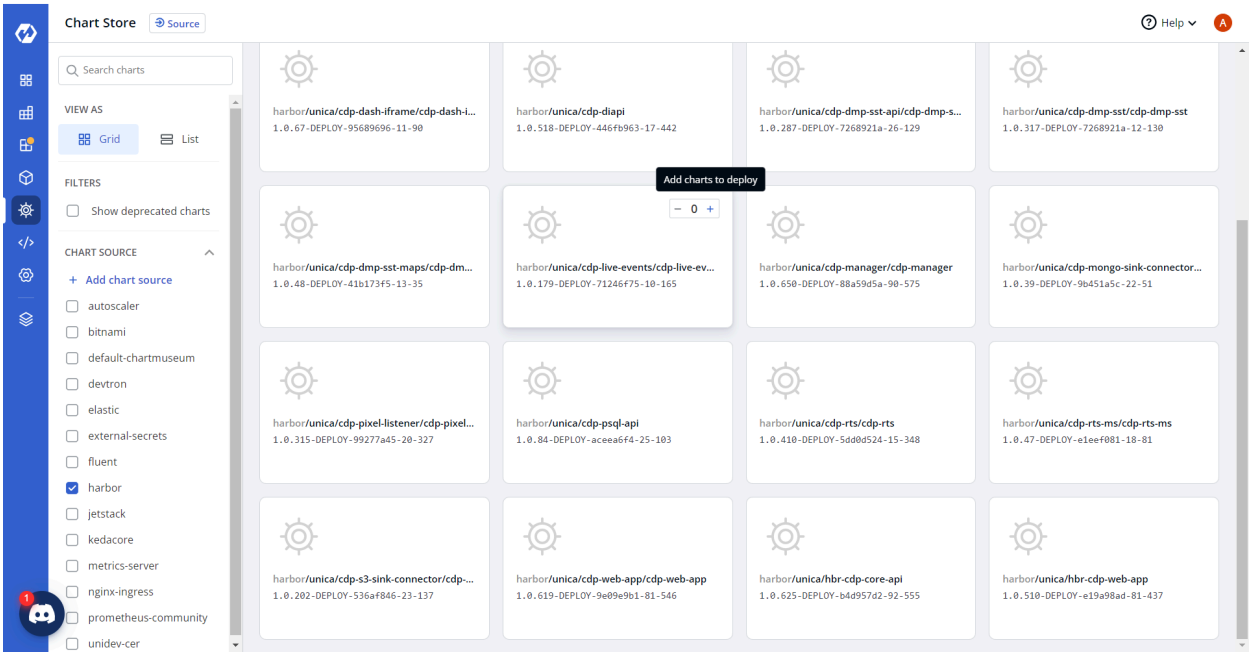
1. Create a live-events secret sample key and value in the live-events secret, and update ConfigMaps data with actual values.

```
{
  "KAFKA_BROKER_LIST": "<KAFKA_BROKER_LIST>",
  "KAFKA_GROUP": "<KAFKA_GROUP>",
  "KAFKA_PASSWORD": "<KAFKA_PASSWORD>",
  "KAFKA_USERNAME": "<KAFKA_USERNAME>",
  "KMS_HOST_STRING": "<KMS_HOST_STRING>",
  "MSYQL_DB": "<MSYQL_DB>",
  "MYSQL_HOST": "<MYSQL_HOST>",
  "MYSQL_PASSWORD": "<MYSQL_PASSWORD>",
  "MYSQL_PORT": "<MYSQL_PORT>",
  "MYSQL_USERNAME": "<MYSQL_USERNAME>"
}
```

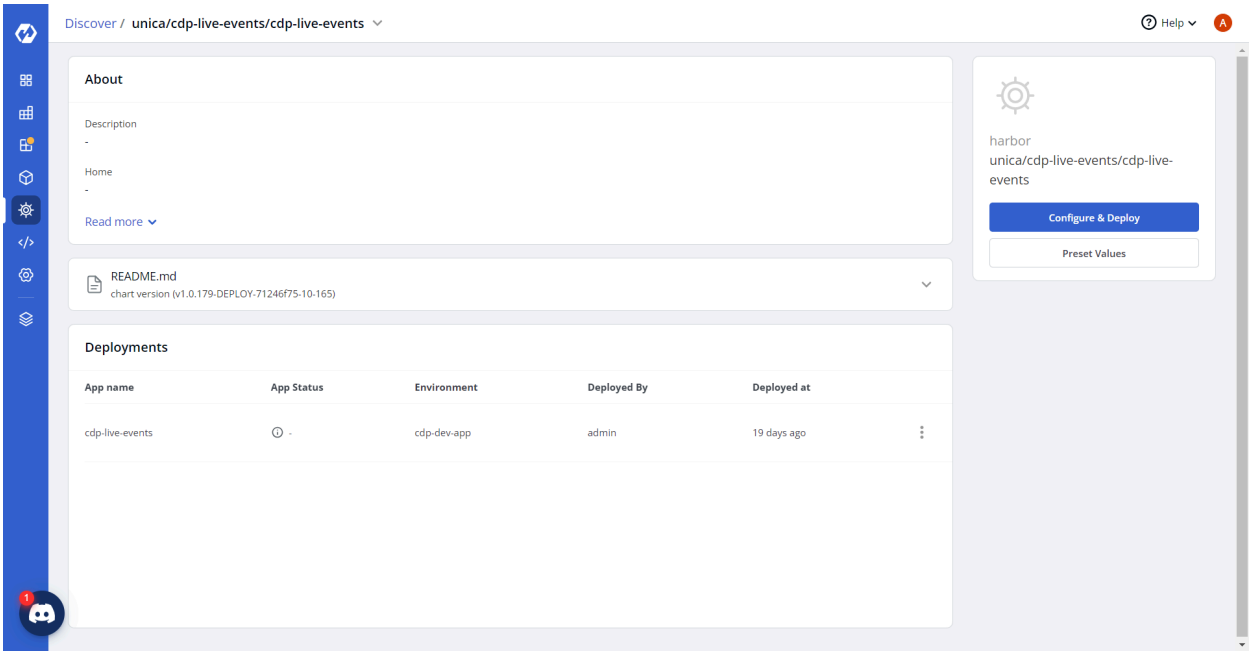
Deploying CDP Live Events

To deploy the CDP Live Events, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the bitnami/zookeeper chart to deploy.

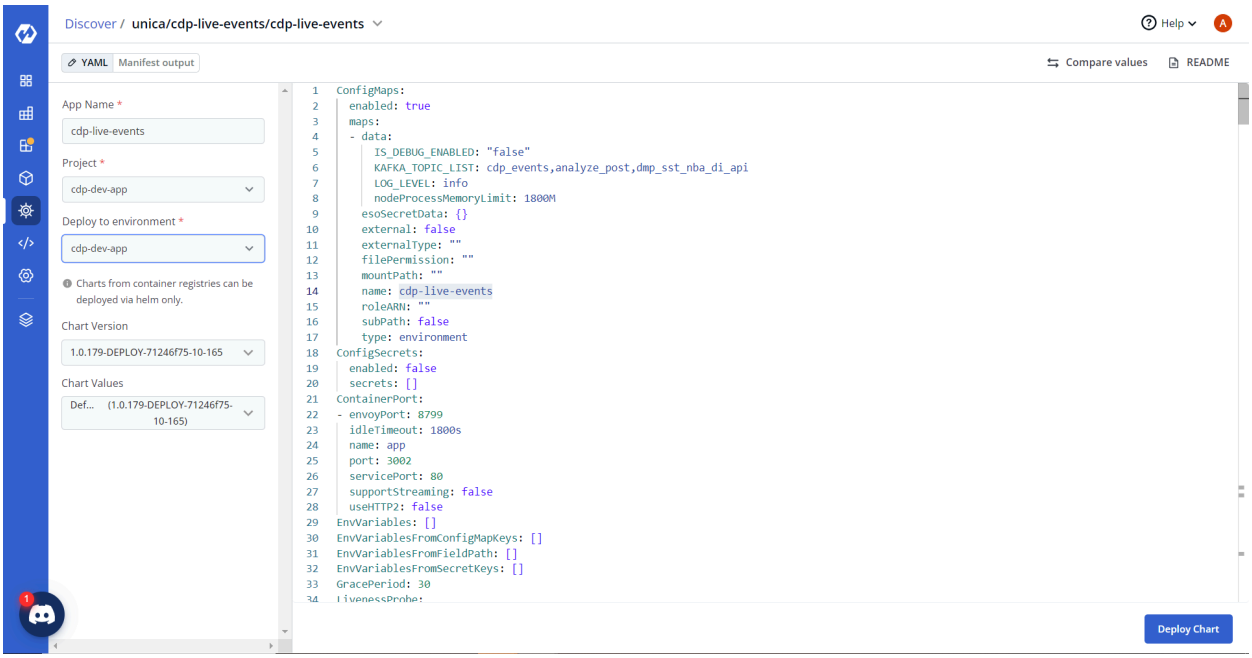


2. Now, configure and deploy the CDP Live Events charts.

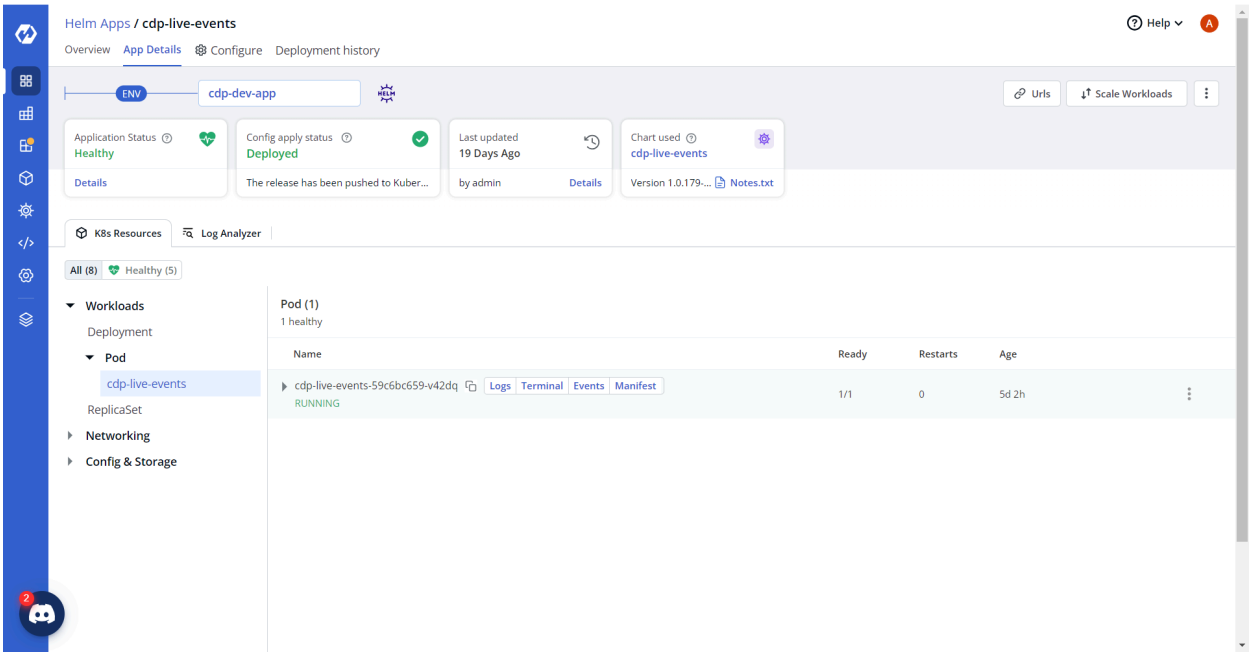


3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
IS_DEBUG_ENABLED: "false"
KAFKA_TOPIC_LIST: <cdp_events,analyze_post,dmp_sst_nba_di_api>
LOG_LEVEL: <info>
nodeProcessMemoryLimit: <1800M>
```



4. On successful deployment, validate the deployment as shown below.

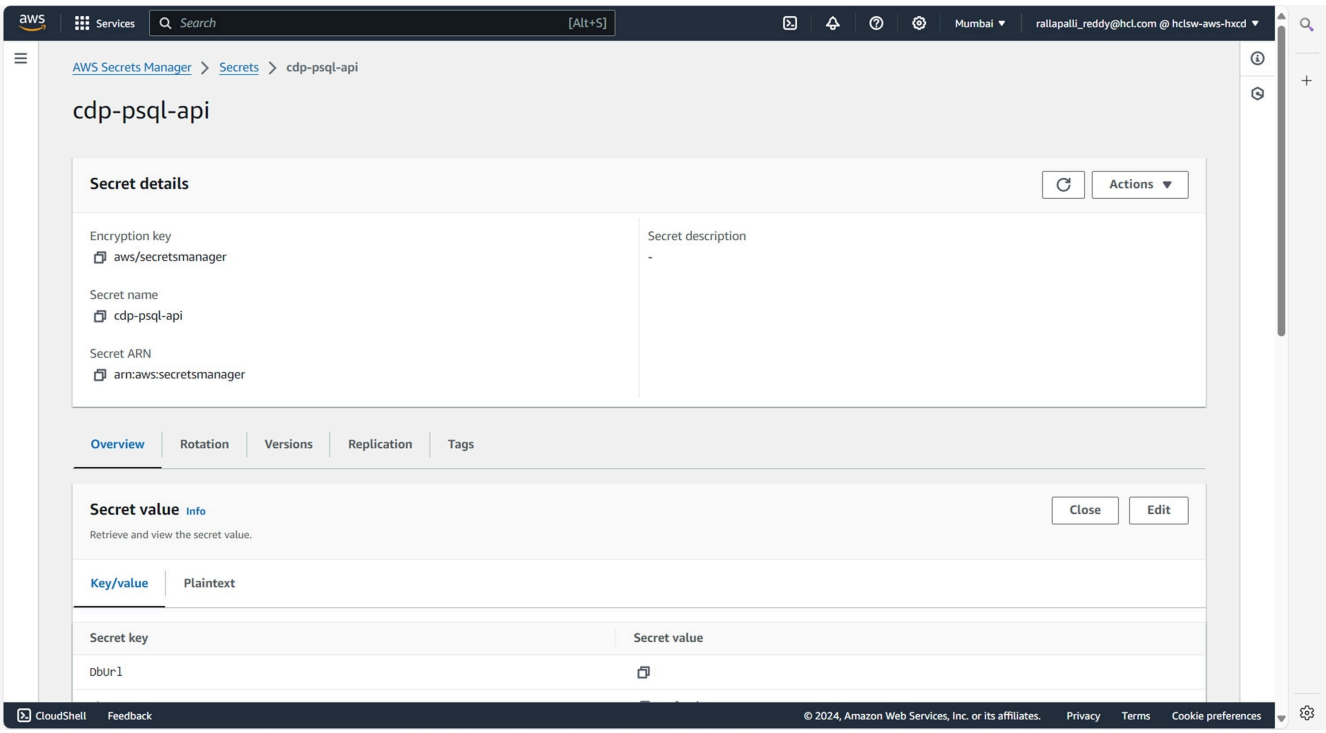


Deploying CDP PostgreSQL API

This section provides detailed instructions on how to deploy HCL CDP PSQL API using the Devtron in the AWS.

Prerequisites:

Make sure to create a cdp-psql-api secret with required data in the AWS Secrets Manager before deploying CDP PSQL API.



To create the cdp-psql-api secret in the AWS secret manager, follow the steps below:

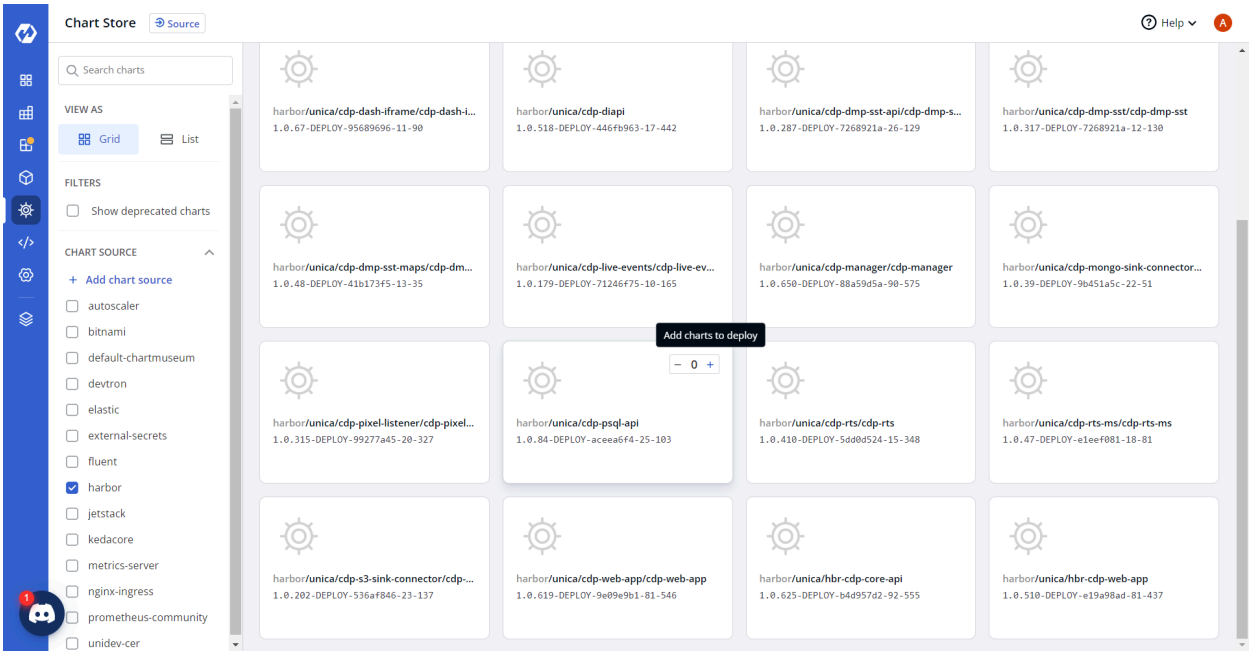
1. Create a cdp-psql-api secret sample key and value in the cdp-psql-api secret, and update ConfigMaps data with actual values.

```
{
  "ActivitiesQueryLimit": "20",
  "DbName": "<DbName>",
  "DbPassword": "<DbPassword>",
  "DbUrl": "<ip:port>",
  "DbUsername": "<DbUsername>",
  "DemoCampaignList": ""
}
```

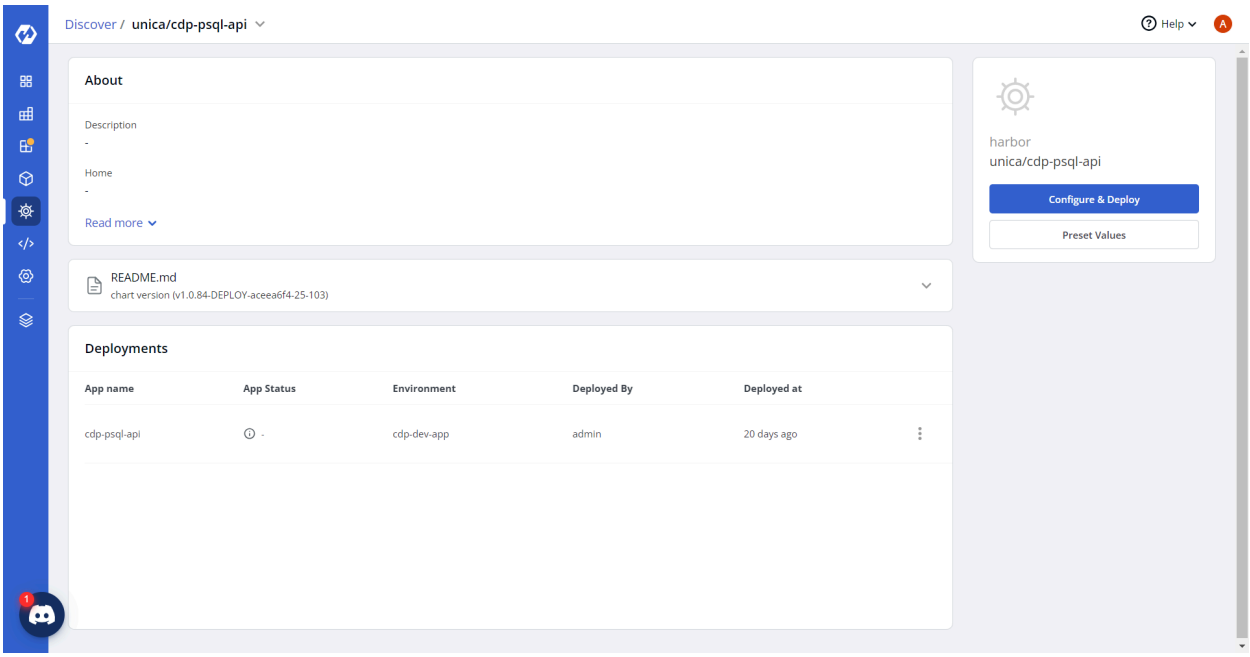
Deploying CDP PSQL API

To deploy the CDP PSQL API, follow the steps below:

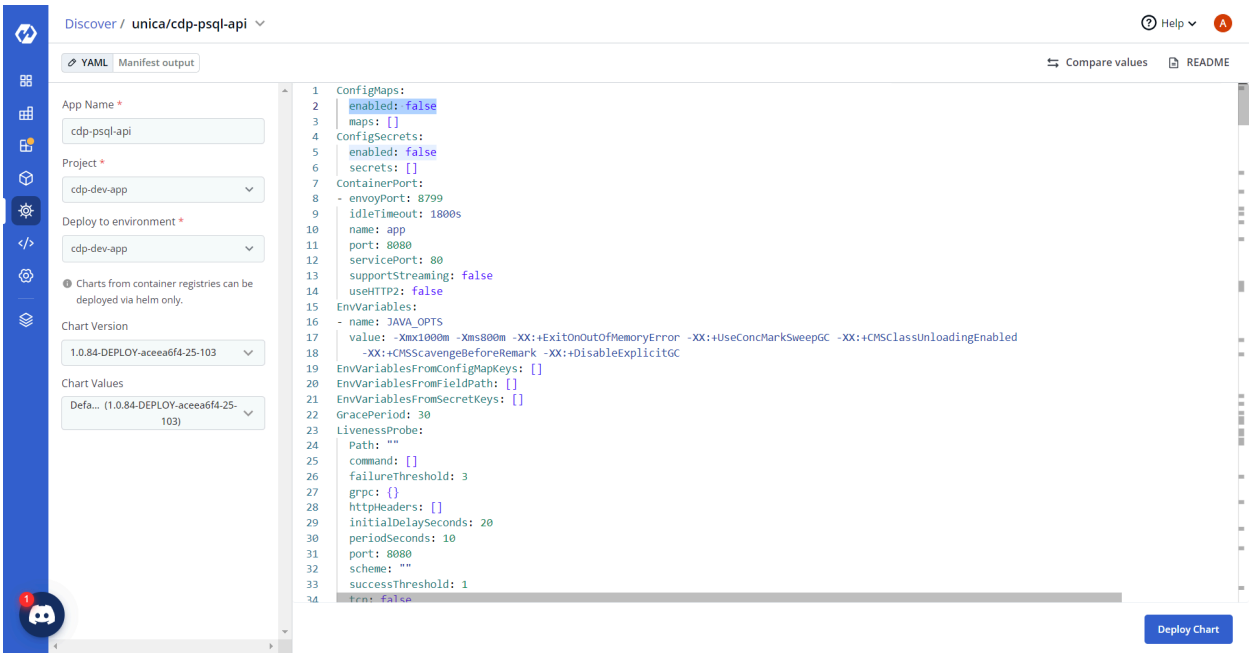
1. Navigate to the Devtron **Chart Store**, and select the cdp-psql-api chart to deploy.



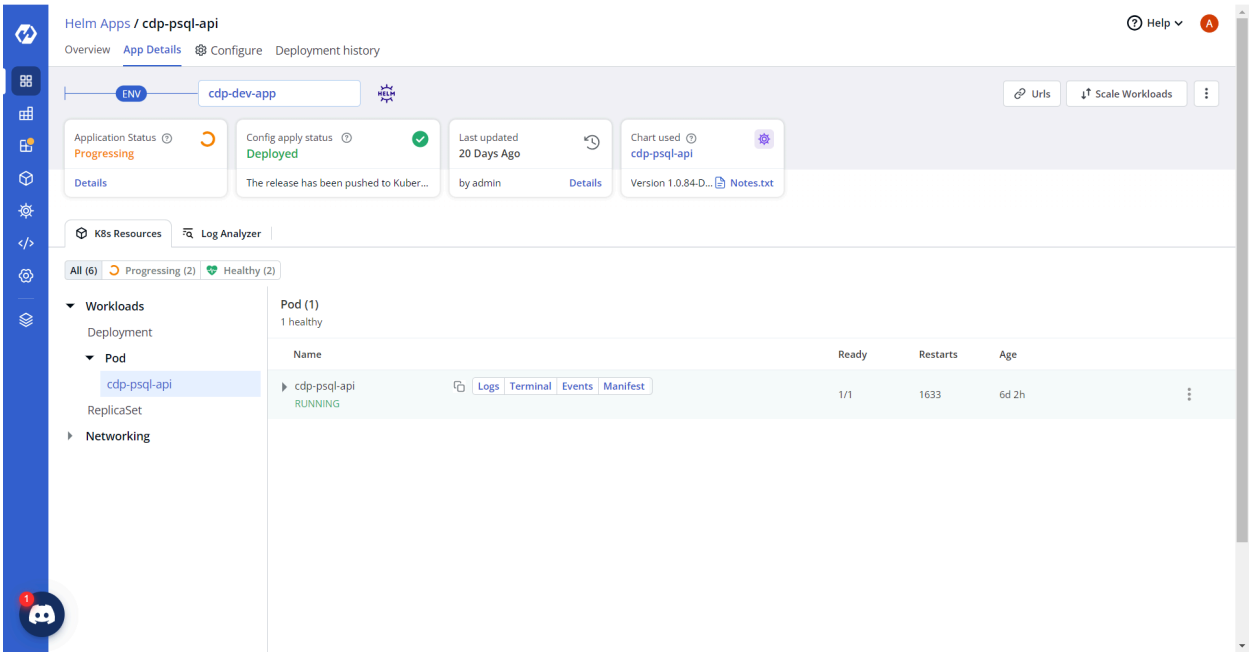
2. Now, configure and deploy the CDP PSQL API charts.



3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.



4. On successful deployment, validate the deployment as shown below.

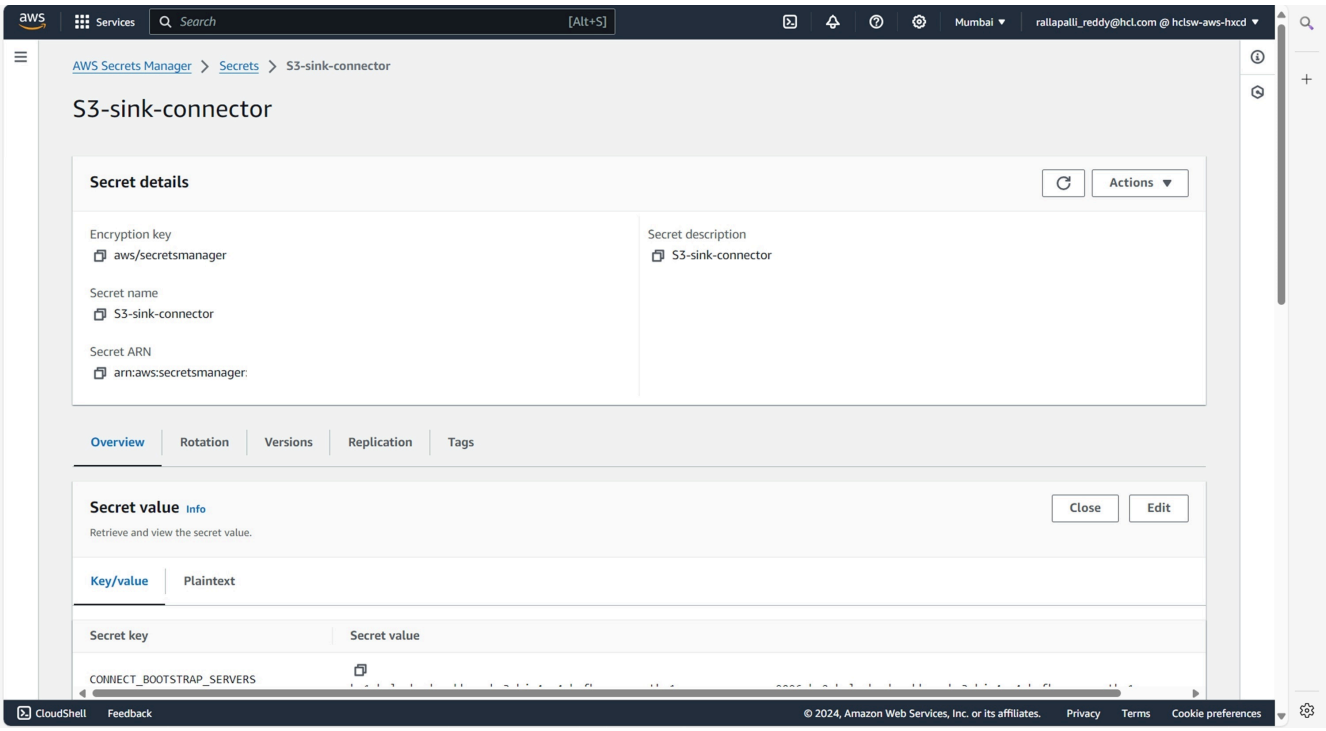


Deploying CDP S3 Sink Connector

This section provides detailed instructions on how to deploy HCL CDP S3 Sink Connector using the Devtron in the AWS.

Prerequisites:

Make sure to create a s3-sink-connector secret with required data in the AWS Secrets Manager before deploying CDP S3 Sink Connector.



To create the s3-sink-connector secret in the AWS secret manager, follow the steps below:

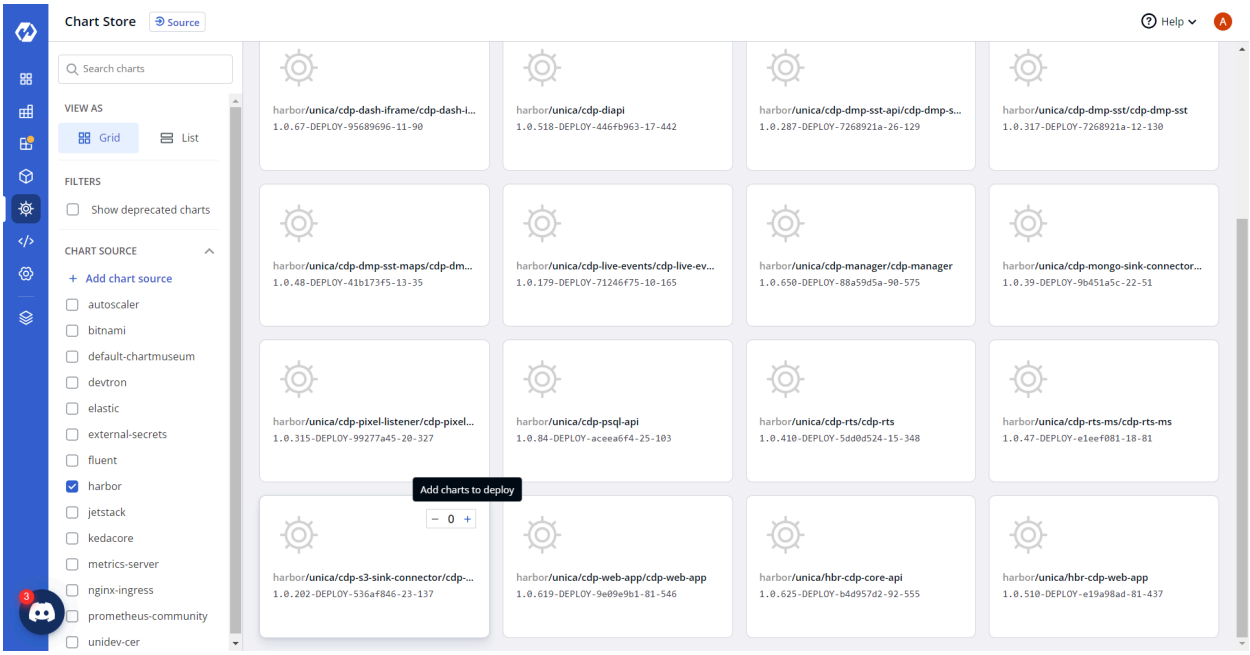
1. Create a s3-sink-connector secret sample key and value in the s3-sink-connector secret, and update ConfigMaps data with actual values.

```
{
  "CONNECT_BOOTSTRAP_SERVERS": <CONNECT_BOOTSTRAP_SERVERS>,
  "CONNECT_CONSUMER_SASL_JAAS_CONFIG": <CONNECT_CONSUMER_SASL_JAAS_CONFIG>,
  "CONNECT_CONSUMER_SASL_MECHANISM": <CONNECT_CONSUMER_SASL_MECHANISM>,
  "CONNECT_CONSUMER_SECURITY_PROTOCOL": <CONNECT_CONSUMER_SECURITY_PROTOCOL>,
  "CONNECT_PRODUCER_SASL_JAAS_CONFIG": <CONNECT_PRODUCER_SASL_JAAS_CONFIG>,
  "CONNECT_PRODUCER_SASL_MECHANISM": <CONNECT_PRODUCER_SASL_MECHANISM>,
  "CONNECT_PRODUCER_SECURITY_PROTOCOL": <CONNECT_PRODUCER_SECURITY_PROTOCOL>,
  "CONNECT_SASL_JAAS_CONFIG": <CONNECT_SASL_JAAS_CONFIG>,
  "CONNECT_SASL_MECHANISM": <CONNECT_SASL_MECHANISM>,
  "CONNECT_SECURITY_PROTOCOL": <CONNECT_SECURITY_PROTOCOL>
}
```

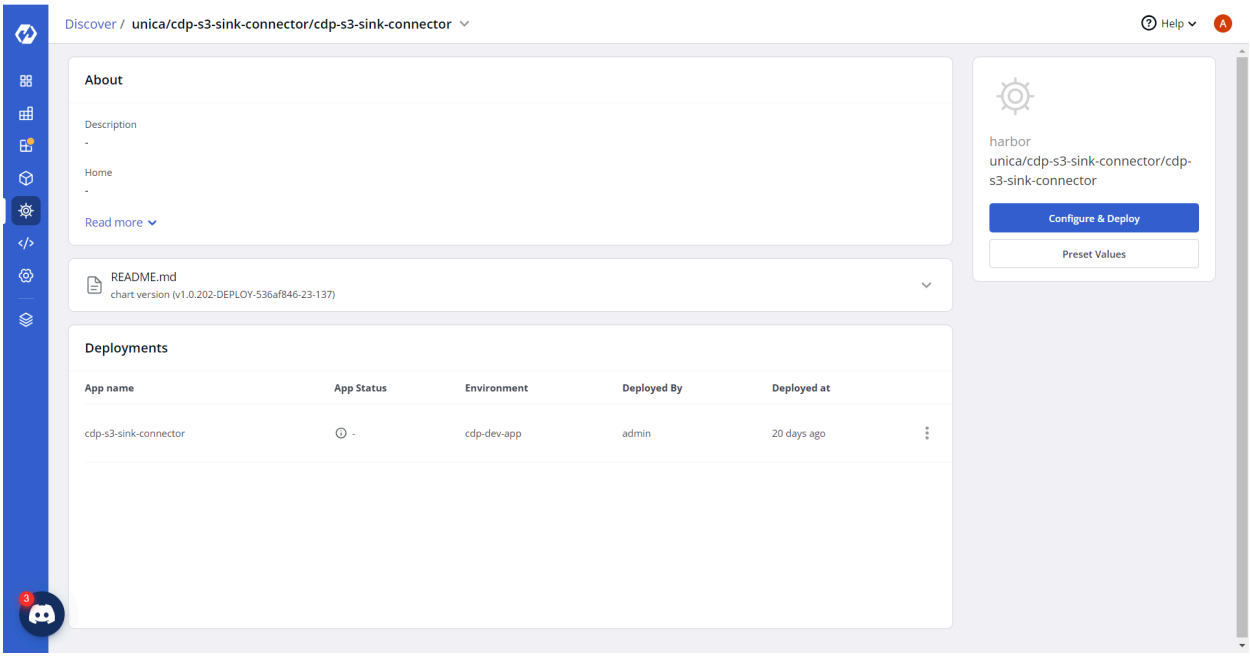
Deploying CDP S3 Sink Connector

To deploy the CDP S3 Sink Connector, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the s3-sink-connector chart to deploy.



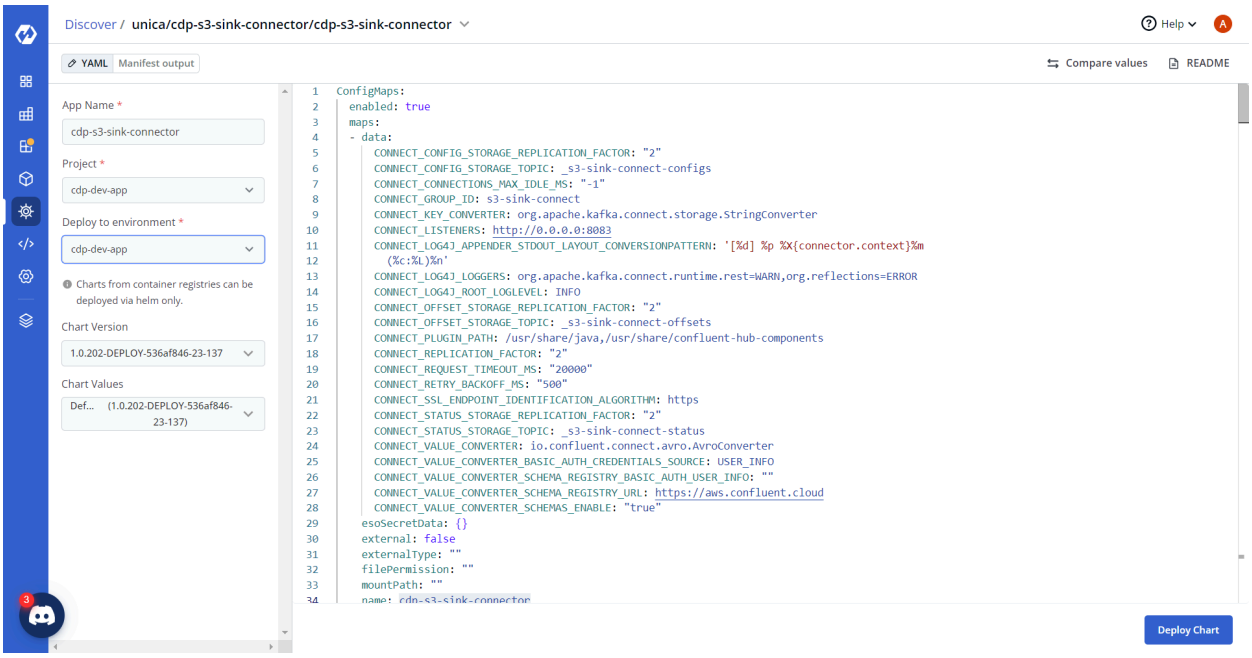
2. Now, configure and deploy the CDP S3 Sink Connector charts.



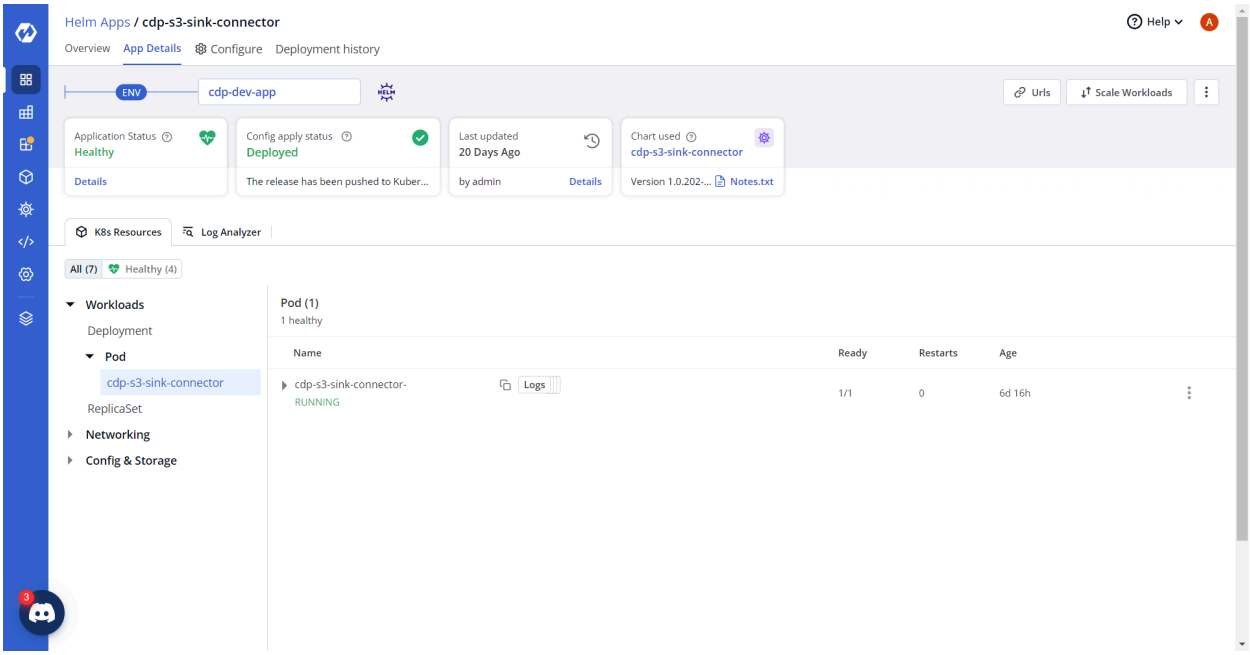
3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR: "2"
CONNECT_CONFIG_STORAGE_TOPIC: _s3-sink-connect-configs
CONNECT_CONNECTIONS_MAX_IDLE_MS: "-1"
CONNECT_GROUP_ID: s3-sink-connect
CONNECT_KEY_CONVERTER: org.apache.kafka.connect.storage.StringConverter
CONNECT_LISTENERS: http://0.0.0.0:8083
```

```
CONNECT_LOG4J_APPENDER_STDOUT_LAYOUT_CONVERSIONPATTERN: '[%d] %p %X{connector.context}%m
        (%c:%L)%n'
CONNECT_LOG4J_LOGGERS: org.apache.kafka.connect.runtime.rest=WARN,org.reflections=ERROR
CONNECT_LOG4J_ROOT_LOGLEVEL: INFO
CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR: "2"
CONNECT_OFFSET_STORAGE_TOPIC: _s3-sink-connect-offsets
CONNECT_PLUGIN_PATH: /usr/share/java,/usr/share/confluent-hub-components
CONNECT_REPLICATION_FACTOR: "2"
CONNECT_REQUEST_TIMEOUT_MS: "20000"
CONNECT_RETRY_BACKOFF_MS: "500"
CONNECT_SSL_ENDPOINT_IDENTIFICATION_ALGORITHM: https
CONNECT_STATUS_STORAGE_REPLICATION_FACTOR: "2"
CONNECT_STATUS_STORAGE_TOPIC: _s3-sink-connect-status
CONNECT_VALUE_CONVERTER: io.confluent.connect.avro.AvroConverter
CONNECT_VALUE_CONVERTER_BASIC_AUTH_CREDENTIALS_SOURCE: USER_INFO
CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_BASIC_AUTH_USER_INFO: ""
CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL: https://aws.confluent.cloud
CONNECT_VALUE_CONVERTER_SCHEMAS_ENABLE: "true"
```



4. On successful deployment, validate the deployment as shown below.

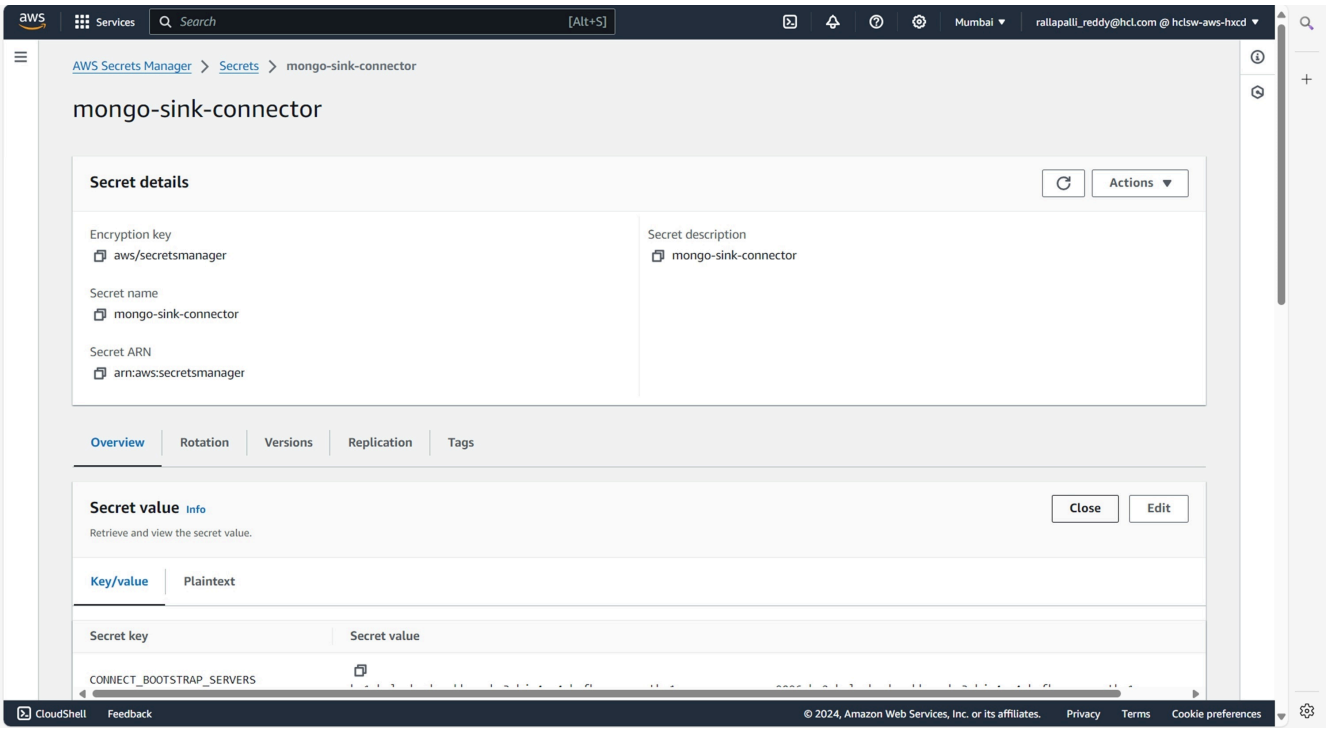


Deploying CDP Mongo Sink Connector

This section provides detailed instructions on how to deploy HCL CDP Mongo Sink Connector using the Devtron in the AWS.

Prerequisites:

Make sure to create a mongo-sink-connector UI secret with required data in the AWS secret manager before deploying CDP Mongo Sink Connector.



To create the mongo-sink-connector secret in the AWS secret manager, follow the steps below:

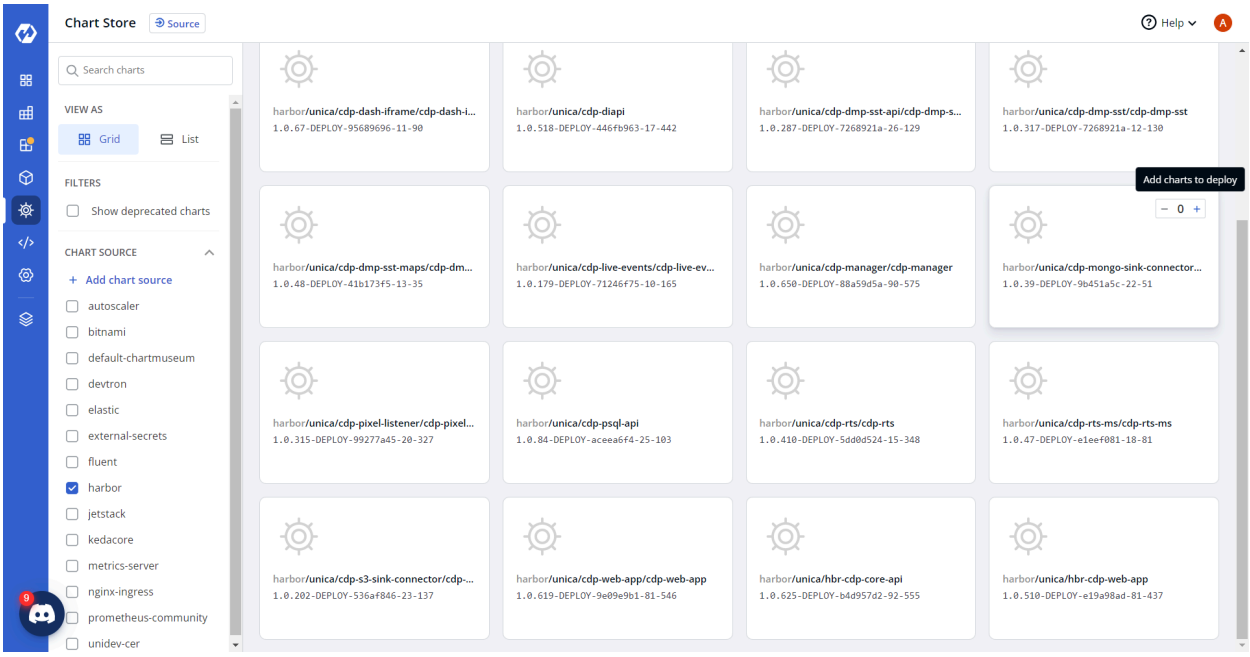
1. Create a mongo-sink-connector secret sample key and value in the mongo-sink-connector secret, and update ConfigMaps data with actual values.

```
{
  "CONNECT_BOOTSTRAP_SERVERS": <CONNECT_BOOTSTRAP_SERVERS>,
  "CONNECT_CONSUMER_SASL_JAAS_CONFIG": <CONNECT_CONSUMER_SASL_JAAS_CONFIG>,
  "CONNECT_CONSUMER_SASL_MECHANISM": <CONNECT_CONSUMER_SASL_MECHANISM>,
  "CONNECT_CONSUMER_SECURITY_PROTOCOL": <CONNECT_CONSUMER_SECURITY_PROTOCOL>,
  "CONNECT_PRODUCER_SASL_JAAS_CONFIG": <CONNECT_PRODUCER_SASL_JAAS_CONFIG>,
  "CONNECT_PRODUCER_SASL_MECHANISM": <CONNECT_PRODUCER_SASL_MECHANISM>,
  "CONNECT_PRODUCER_SECURITY_PROTOCOL": <CONNECT_PRODUCER_SECURITY_PROTOCOL>,
  "CONNECT_SASL_JAAS_CONFIG": <CONNECT_SASL_JAAS_CONFIG>,
  "CONNECT_SASL_MECHANISM": <CONNECT_SASL_MECHANISM>,
  "CONNECT_SECURITY_PROTOCOL": <CONNECT_SECURITY_PROTOCOL>
}
```

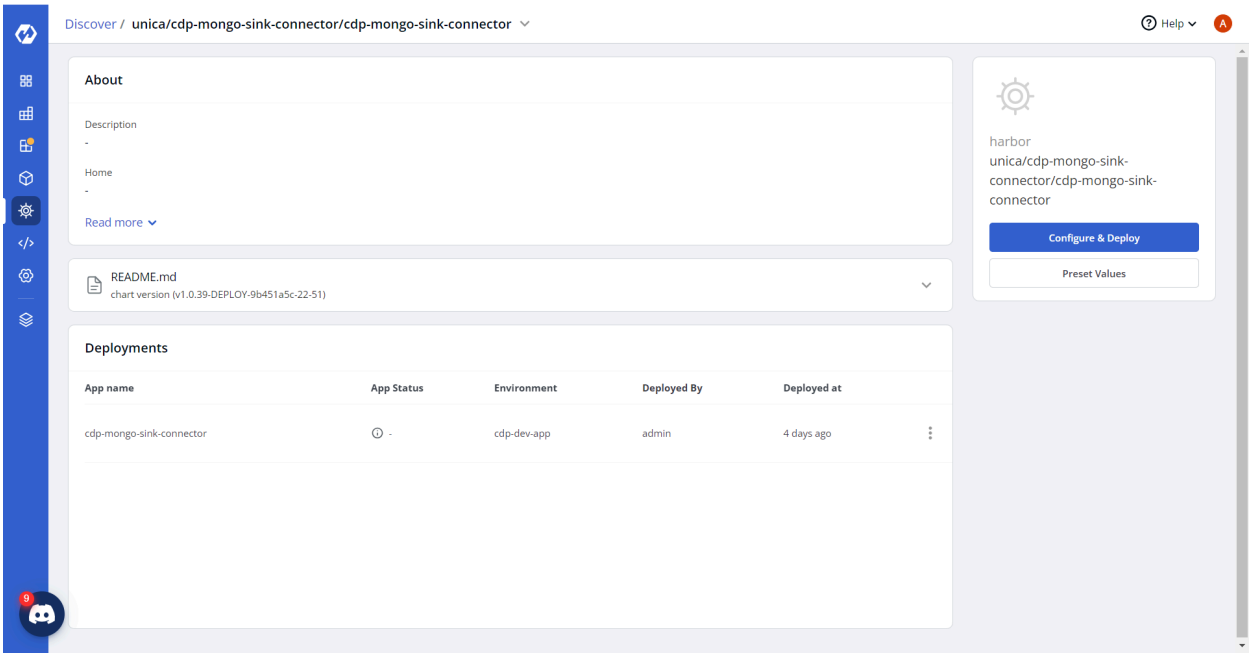
Deploying CDP Mongo Sink Connector

To deploy the CDP Mongo Sink Connector, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-mongo-sink-connector chart to deploy.



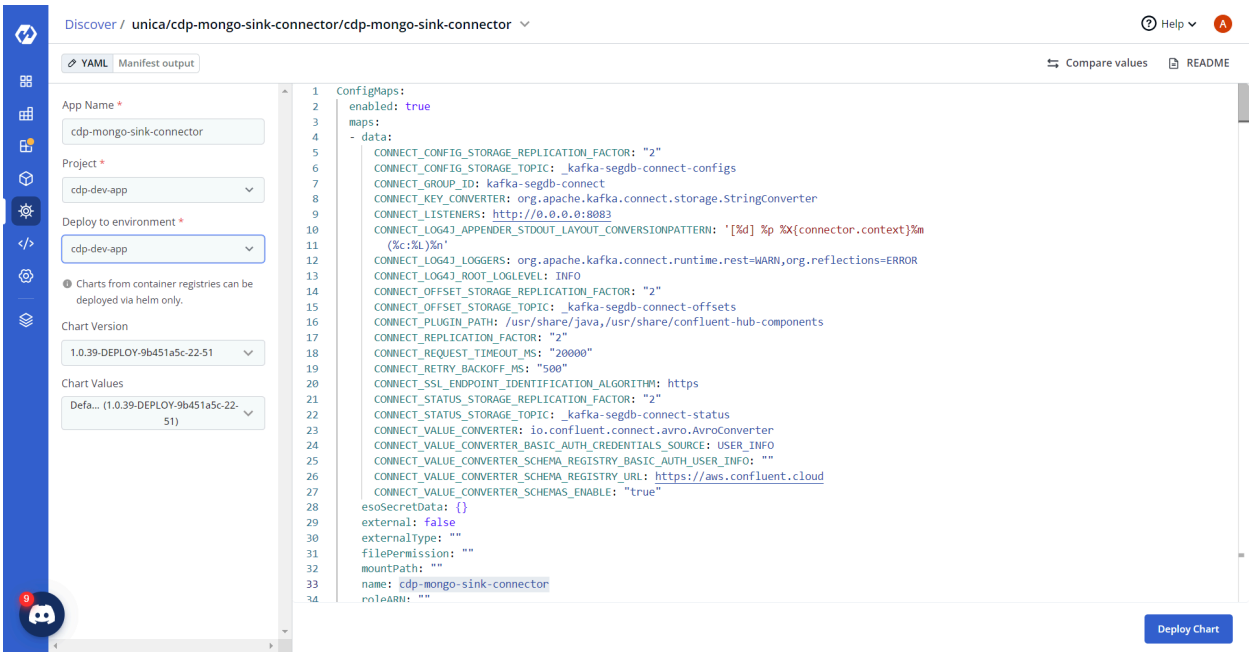
2. Now, configure and deploy the CDP Mongo Sink Connector.



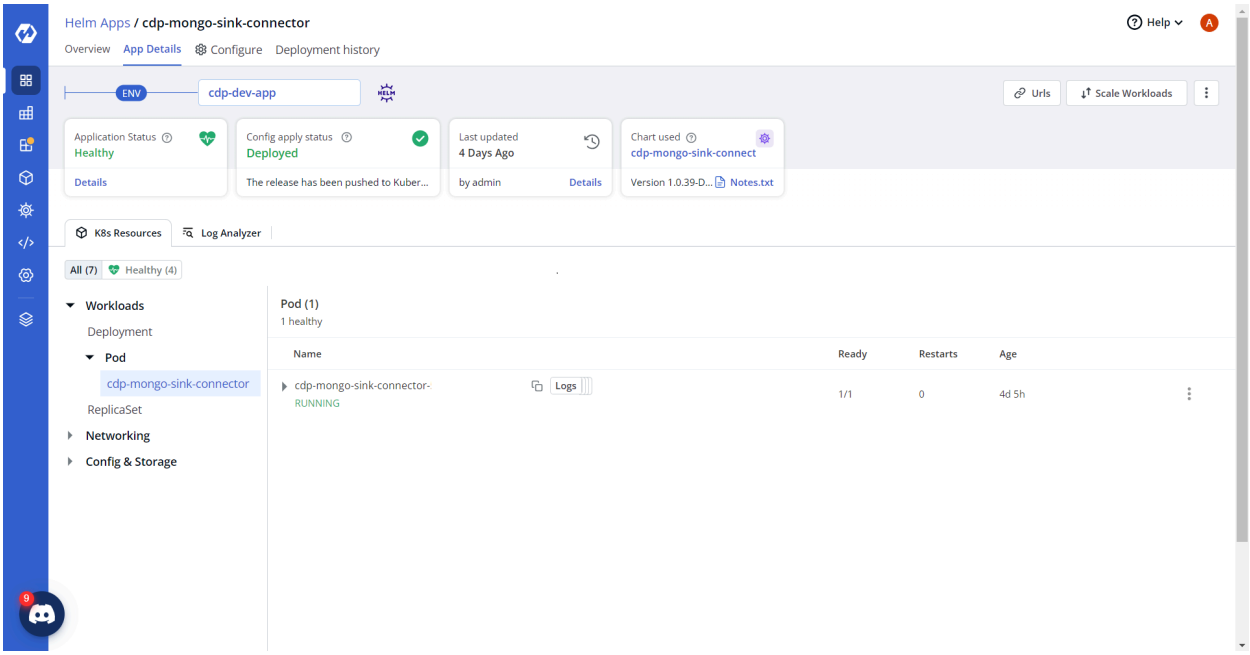
3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.

```
CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR: "2"
CONNECT_CONFIG_STORAGE_TOPIC: _kafka-segdb-connect-configs
CONNECT_GROUP_ID: kafka-segdb-connect
CONNECT_KEY_CONVERTER: org.apache.kafka.connect.storage.StringConverter
CONNECT_LISTENERS: http://0.0.0.0:8083
CONNECT_LOG4J_APPENDER_STDOUT_LAYOUT_CONVERSIONPATTERN: '[%d] %p %X{connector.context}%m
```

```
  (%c:%L)%n'
CONNECT_LOG4J_LOGGERS: org.apache.kafka.connect.runtime.rest=WARN,org.reflections=ERROR
CONNECT_LOG4J_ROOT_LOGLEVEL: INFO
CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR: "2"
CONNECT_OFFSET_STORAGE_TOPIC: _kafka-segdb-connect-offsets
CONNECT_PLUGIN_PATH: /usr/share/java,/usr/share/confluent-hub-components
CONNECT_REPLICATION_FACTOR: "2"
CONNECT_REQUEST_TIMEOUT_MS: "20000"
CONNECT_RETRY_BACKOFF_MS: "500"
CONNECT_SSL_ENDPOINT_IDENTIFICATION_ALGORITHM: https
CONNECT_STATUS_STORAGE_REPLICATION_FACTOR: "2"
CONNECT_STATUS_STORAGE_TOPIC: _kafka-segdb-connect-status
CONNECT_VALUE_CONVERTER: io.confluent.connect.avro.AvroConverter
CONNECT_VALUE_CONVERTER_BASIC_AUTH_CREDENTIALS_SOURCE: USER_INFO
CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_BASIC_AUTH_USER_INFO: ""
CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL: https://aws.confluent.cloud
CONNECT_VALUE_CONVERTER_SCHEMAS_ENABLE: "true"
```



4. On successful deployment, validate the deployment as shown below.

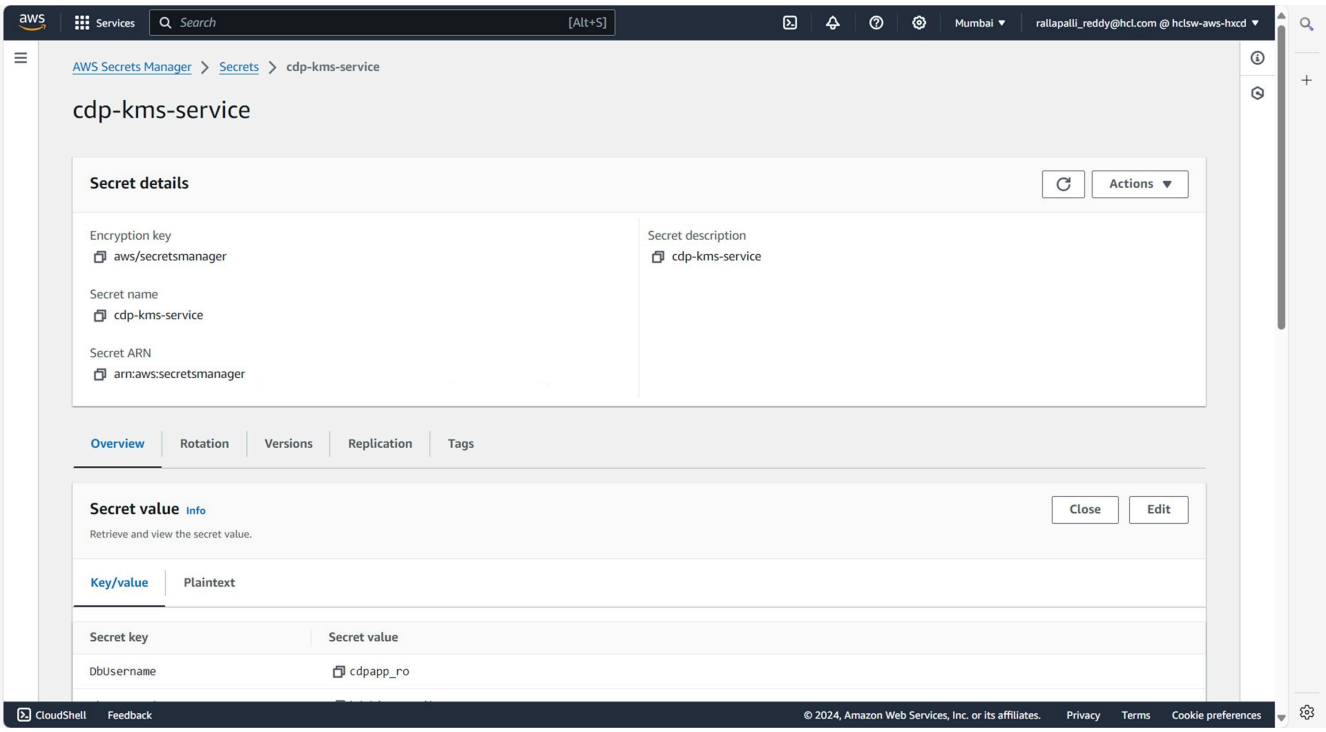


Deploying CDP KMS Service

This section provides detailed instructions on how to deploy HCL CDP KMS Service using the Devtron in the AWS.

Prerequisites:

Make sure to create a cdp-kms-service secret with required data in the AWS Secrets Manager before deploying CDP KMS Service.



To create the cdp-kms-service secret in the AWS secret manager, follow the steps below:

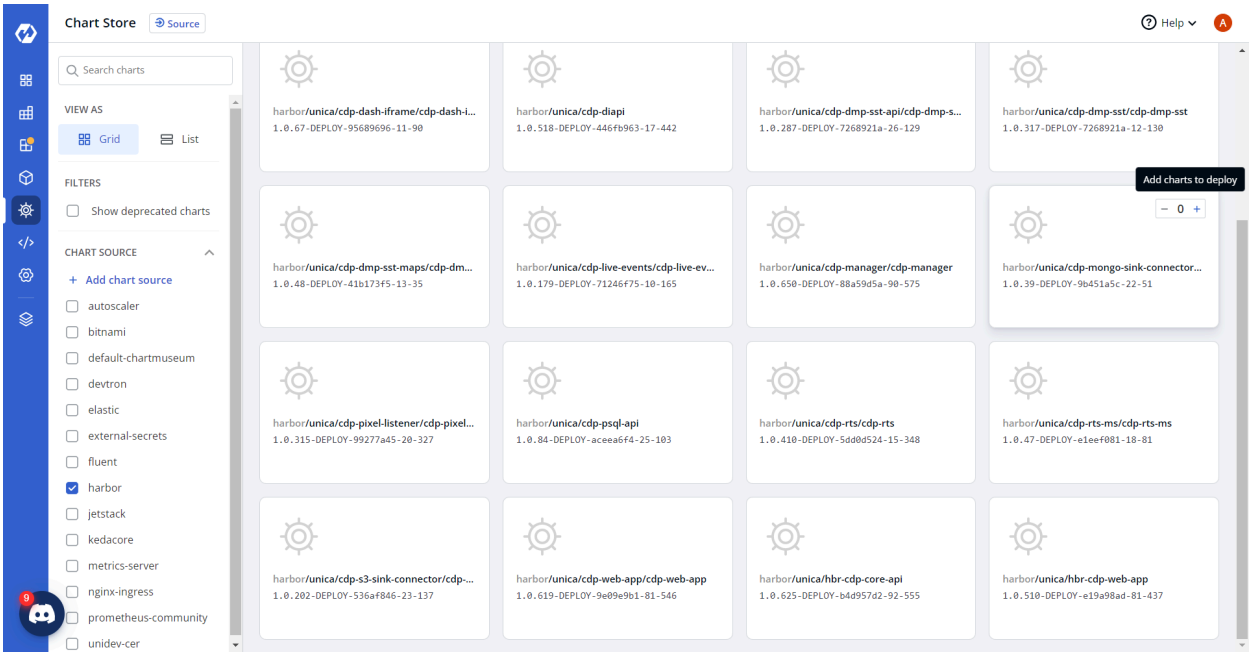
1. Create a cdp-kms-service secret sample key and value in the cdp-kms-service secret, and update ConfigMaps data with actual values.

```
{
  "CAMPAIGN_DEFAULT_TTL": "21600",
  "DbDriver": "<DbDriver>",
  "DbPassword": "<DbPassword>",
  "DbUrl": "<DbUrl>",
  "DbUsername": "<DbUsername>",
  "KMS_HASHICORPVAULT_ENDPOINT": "<KMS_HASHICORPVAULT_ENDPOINT>",
  "KMS_HASHICORPVAULT_ENDPOINT_TOKEN": "<KMS_HASHICORPVAULT_ENDPOINT_TOKEN>",
  "KMS_IMPLEMENTATION": "HashiCorpVault",
  "awsRegions": "ap-south-1",
  "supportedRegions": "aps1"
}
```

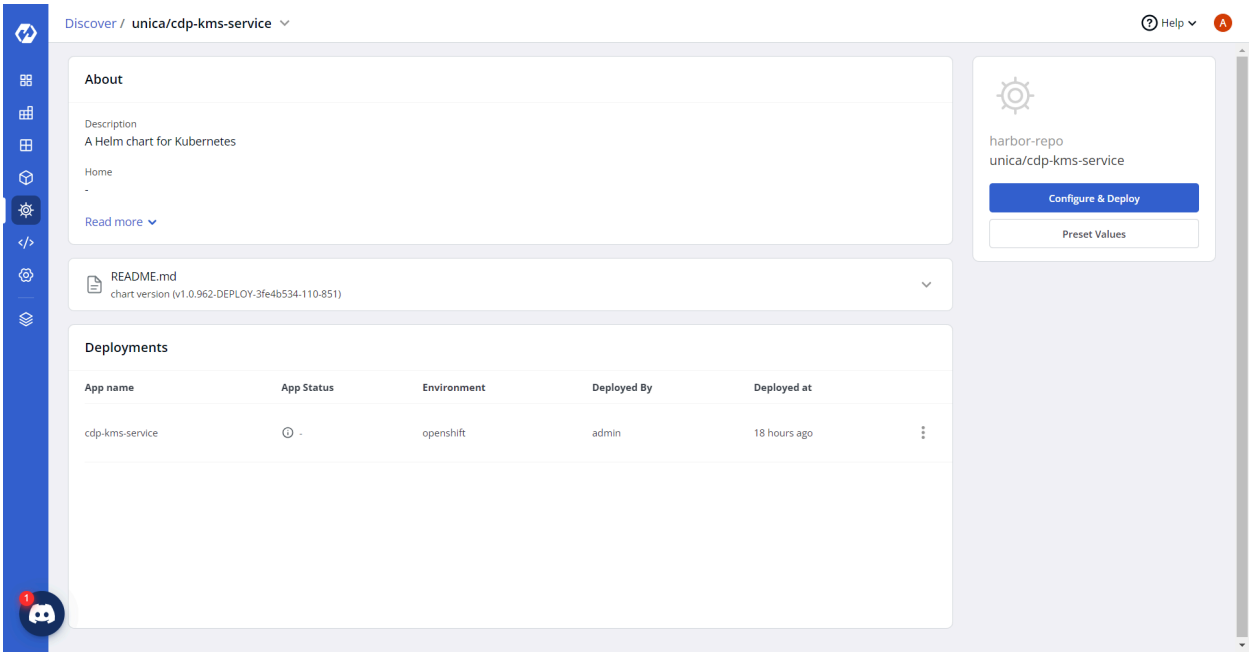
Deploying CDP KMS Service

To deploy the CDP KMS Service, follow the steps below:

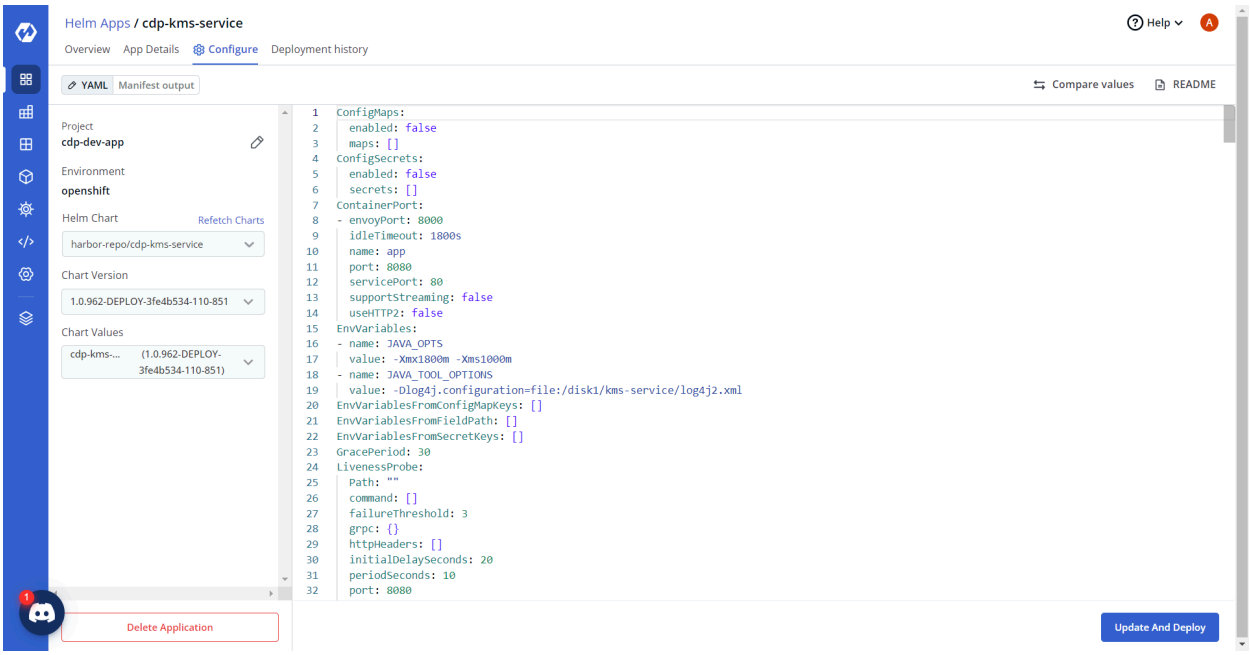
1. Navigate to the Devtron **Chart Store**, and select the cdp-kms-service chart to deploy.



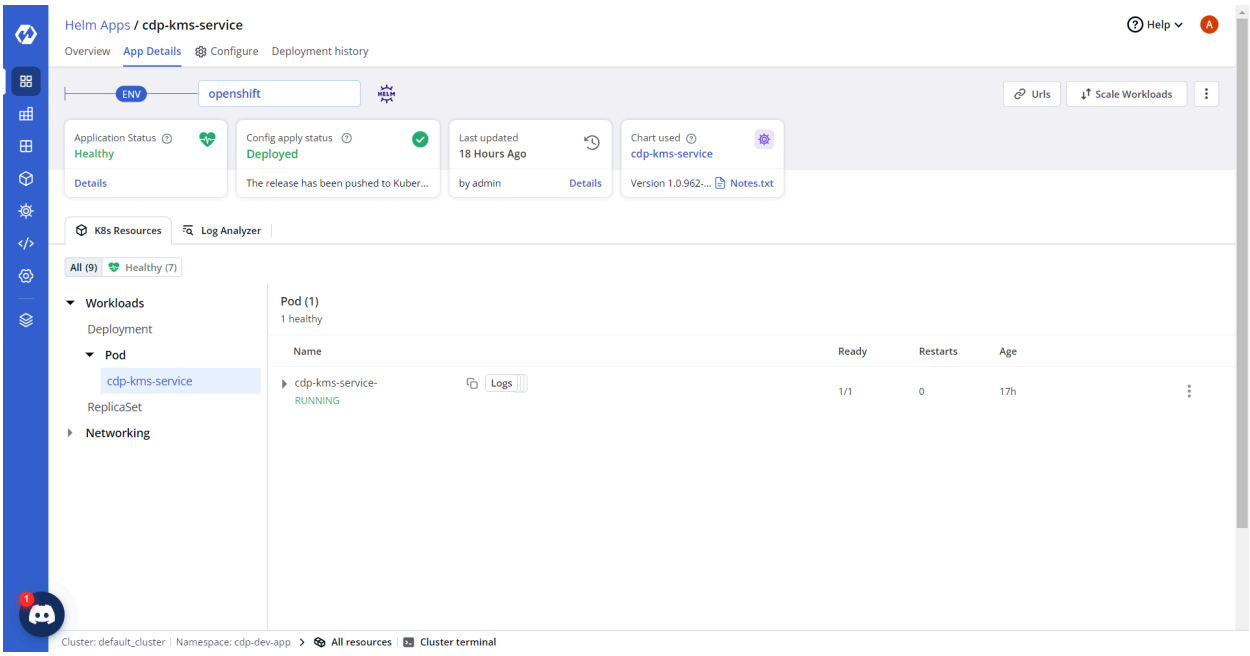
2. Now, configure and deploy the CDP KMS Service.



3. In the **YAML** section, update the ConfigMap using below details, and deploy the charts.



4. On successful deployment, validate the deployment as shown below.

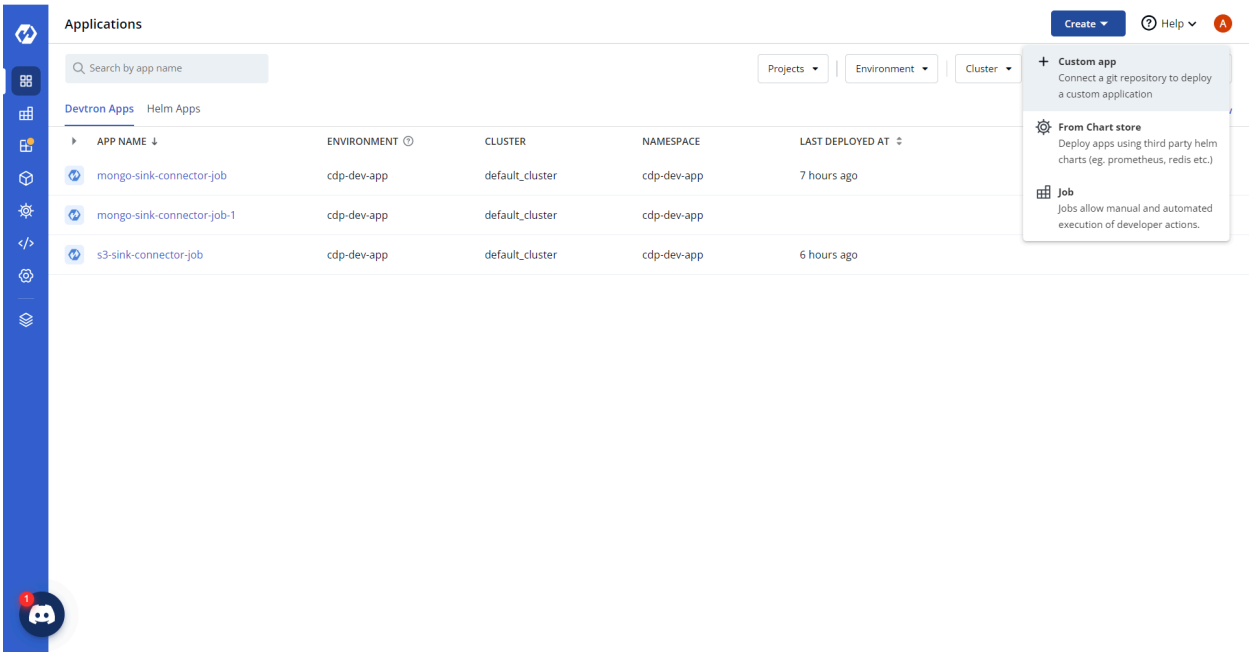


Deploying CDP Mongo Connector Sink Job

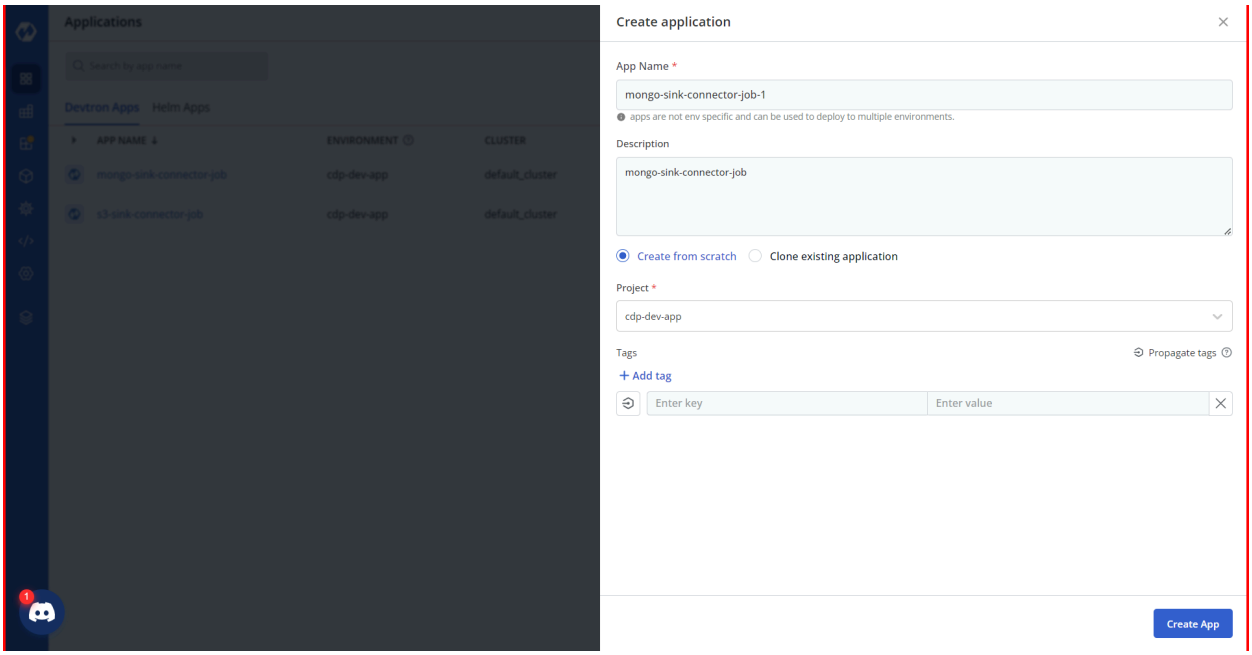
This section provides detailed instructions on how to deploy HCL CDP Mongo Connector Sink Job using the Devtron.

Creating custom app

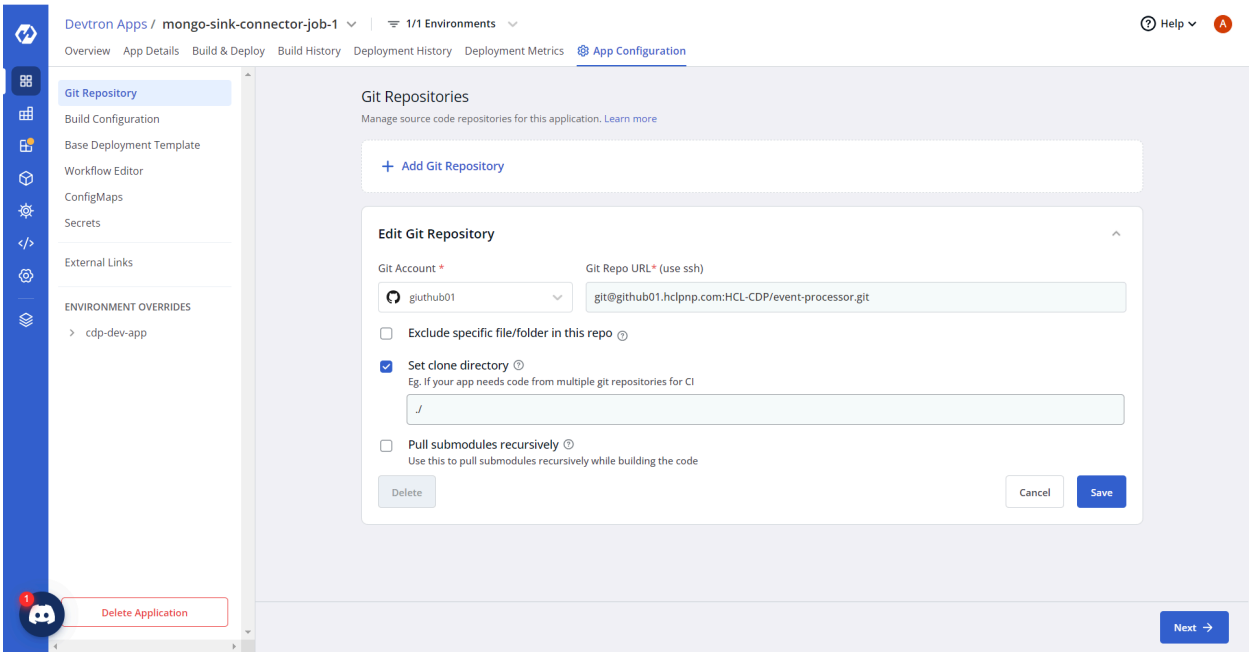
1. In the devtron console, select **Applications**, and click **Create** > **Custom app**.



2. In the **Create Application** section, enter application name as "mongo-sink-connector-job" and select a Project to store the application.



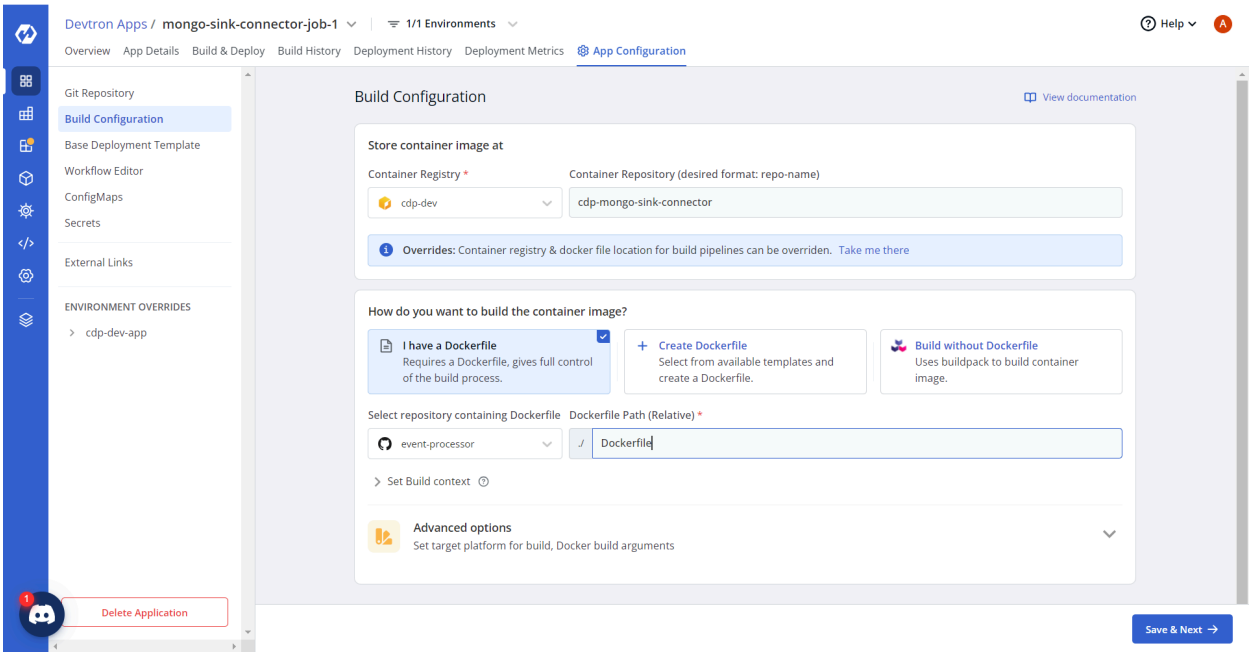
3. Click **Create App** to create the application. Now, click **App Configuration** tab, and update source code repositories for the application.



Building custom app

After successfully creating the custom app, build the custom app with docker image, and configure a workflow for the custom app.

1. In the Devtron Apps page, click **Build Configurations**. In the **App Configuration** tab, select **I have a Dockerfile** option and update the dockerfile path.



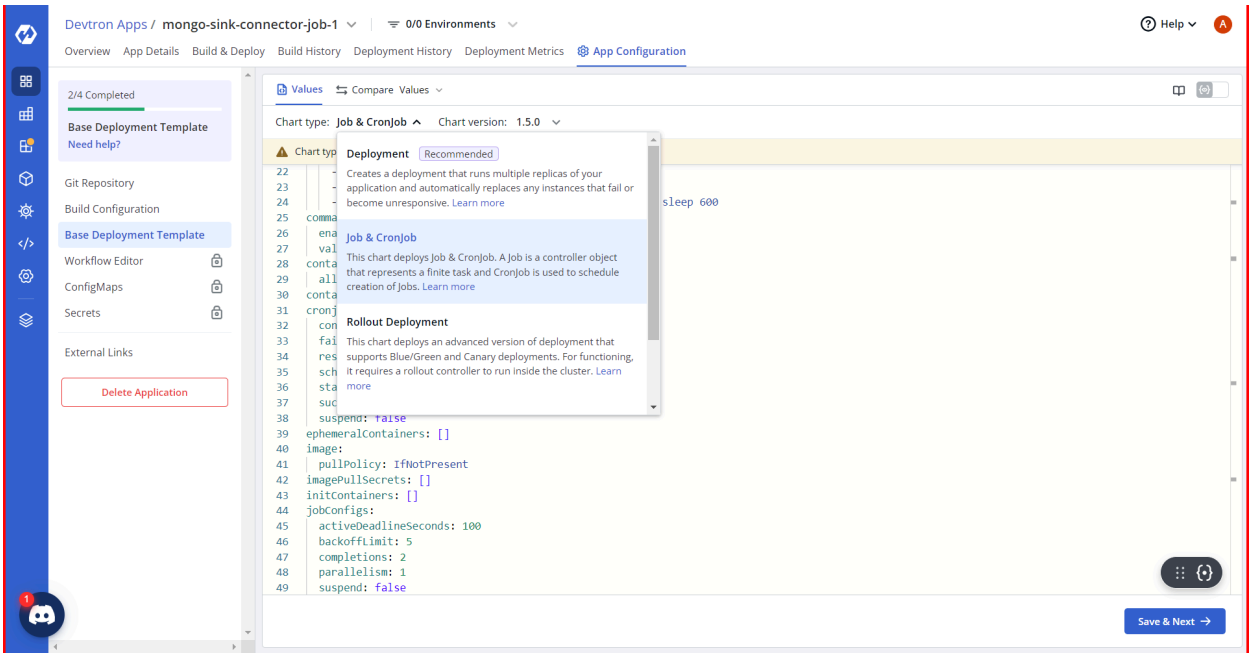
2. Click **Save & Next**, and in the **Base Deployment Template** section, update chart type and Mongo Connector Sink URLs in the base deployment template as shown below.

Deployment Template

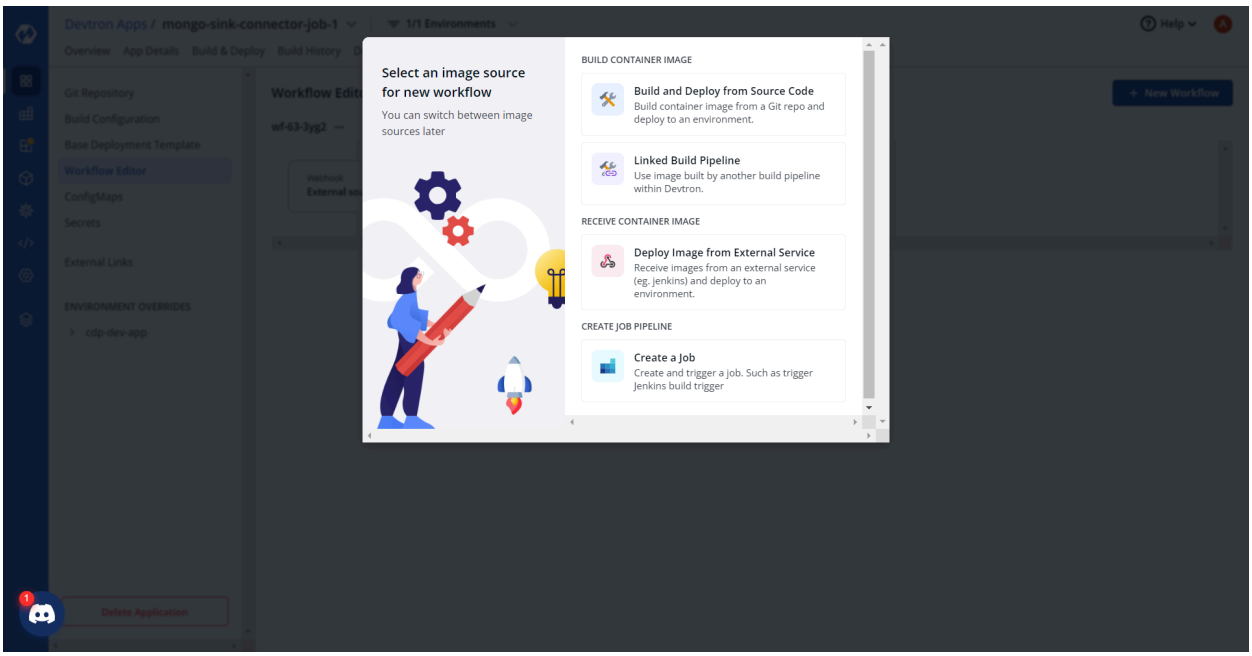
```
ContainerPort:
  - envoyPort: 8799
    idleTimeout: 1800s
    name: app
    port: 8080
    servicePort: 80
    supportStreaming: true
    useHTTP2: true
EnvVariables: []
GracePeriod: 30
MaxSurge: 1
MaxUnavailable: 0
MinReadySeconds: 60
Spec:
  Affinity:
    Key: null
    Values: nodes
    key: ""
args:
  enabled: true
  value:
    - /bin/sh
    - -c
    - >
      until $(curl --output /dev/null --silent --head --fail
      http://cdp-mongo-sink-connector-service.cdp-dev-app.svc.cluster.local:80);
```

```
      do
        echo "Waiting for Kafka Connect API..."
        sleep 5
      done

      echo "Posting Kafka Connect configuration..."

      curl -X POST -H "Content-Type: application/json" --data @/data/${APP_NAME}
 http://cdp-mongo-sink-connector-service.cdp-dev-app.svc.cluster.local:80/connectors
command:
  enabled: false
  value: []
containerSecurityContext:
  allowPrivilegeEscalation: false
containers: []
cronjobConfigs:
  concurrencyPolicy: Allow
  failedJobsHistoryLimit: 1
  restartPolicy: OnFailure
  schedule: "* * * * *"
  startingDeadlineSeconds: 100
  successfulJobsHistoryLimit: 3
  suspend: false
ephemeralContainers: []
image:
  pullPolicy: IfNotPresent
imagePullSecrets: []
initContainers: []
jobConfigs:
  activeDeadlineSeconds: 100
  backoffLimit: 5
  completions: 1
  parallelism: 1
  suspend: false
kedaAutoscaling:
  envSourceContainerName: ""
  failedJobsHistoryLimit: 5
  maxReplicaCount: 2
  minReplicaCount: 1
  pollingInterval: 30
  rolloutStrategy: default
  scalingStrategy:
    customScalingQueueLengthDeduction: 1
    customScalingRunningJobPercentage: "0.5"
    multipleScalersCalculation: max
    pendingPodConditions:
      - Ready
      - PodScheduled
      - AnyOtherCustomPodCondition
    strategy: custom
  successfulJobsHistoryLimit: 5
  triggerAuthentication:
    enabled: false
    name: ""
    spec: {}
  triggers:
    - authenticationRef: {}
      metadata:
```

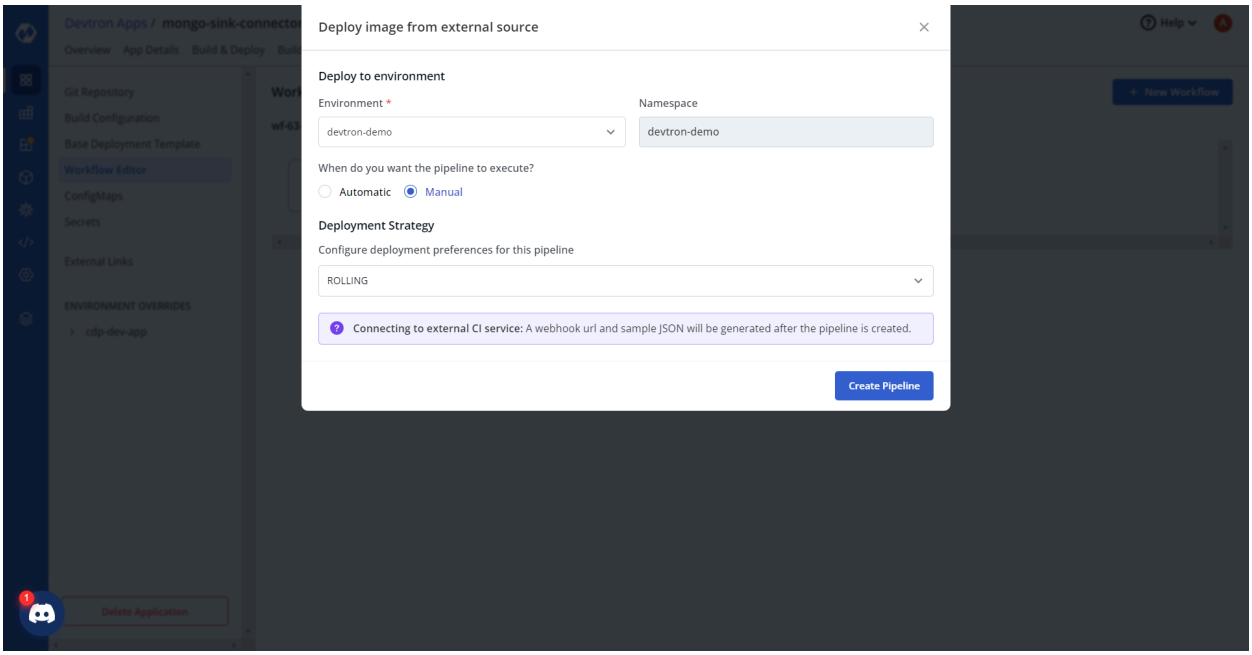
```
        host: RabbitMqHost
        queueLength: "5"
        queueName: hello
      type: rabbitmq
kind: Job
pauseForSecondsBeforeSwitchActive: 30
podAnnotations: {}
podLabels: {}
podSecurityContext: {}
prometheus:
  release: monitoring
rawYaml: []
readinessGates: []
resources:
  limits:
    cpu: "0.05"
    memory: 50Mi
  requests:
    cpu: "0.01"
    memory: 10Mi
secret:
  data: {}
  enabled: false
server:
  deployment:
    image: ""
    image_tag: 1-95af053
service:
  annotations: {}
  enabled: false
  type: ClusterIP
servicemonitor:
  additionalLabels: {}
setHostnameAsFQDN: false
shareProcessNamespace: false
tolerations: []
topologySpreadConstraints: []
volumeMounts: []
volumes: []
waitForSecondsBeforeScalingDown: 30
```



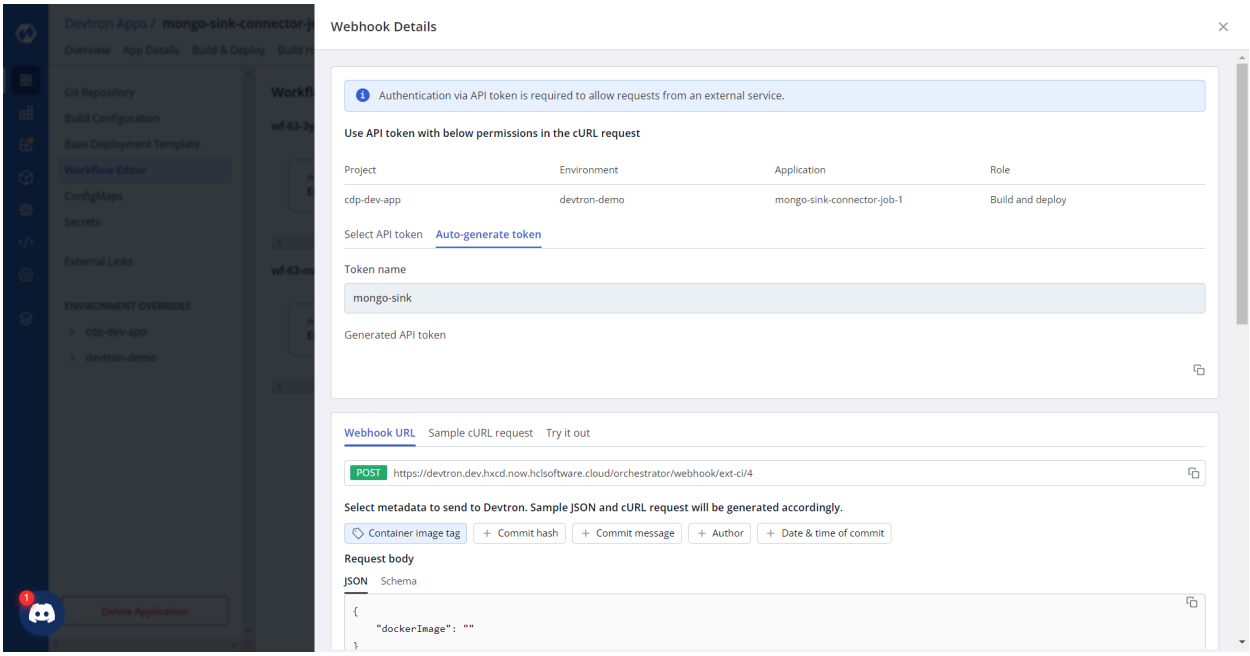
3. Click **Save & Next**. In the Workflow Editor section, add new workflow, and select the **Deploy Image from External Source** option.



4. Configure required parameters, and click **Create pipeline** at the right bottom corner of the Deploy image from external source section.

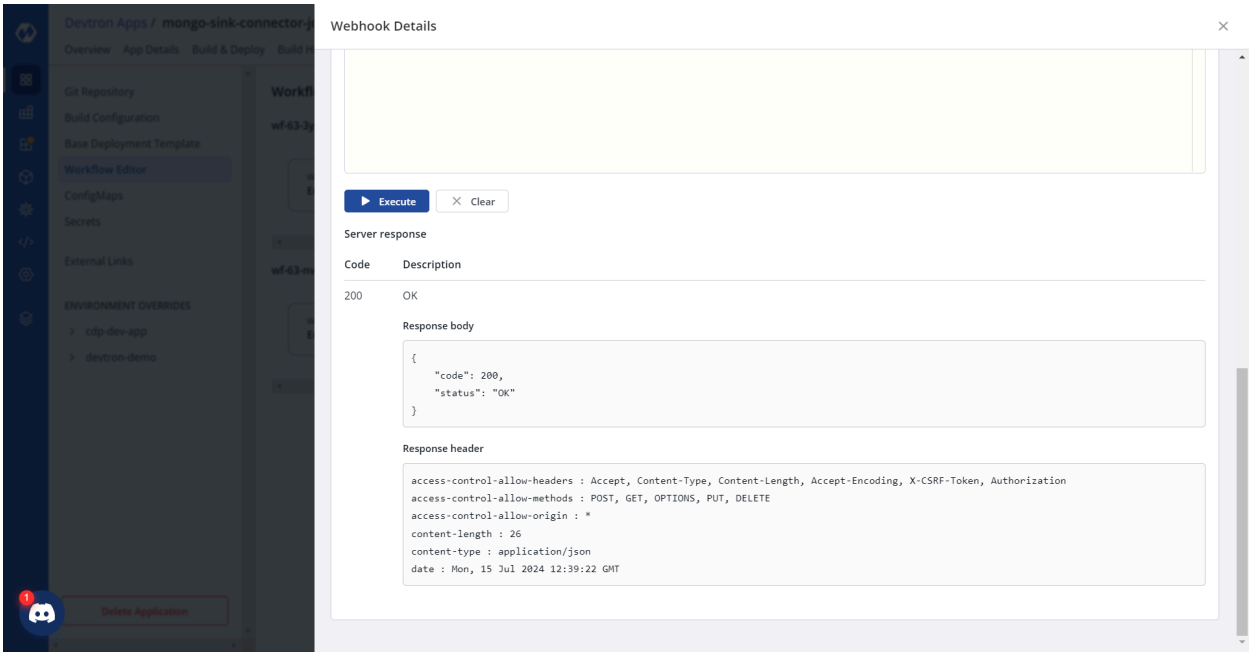


5. In the Workflow Editor, click **Edit Workflow**, and in the **Webhook Details** section, click the **Auto Generate Token** tab, and copy the token details.



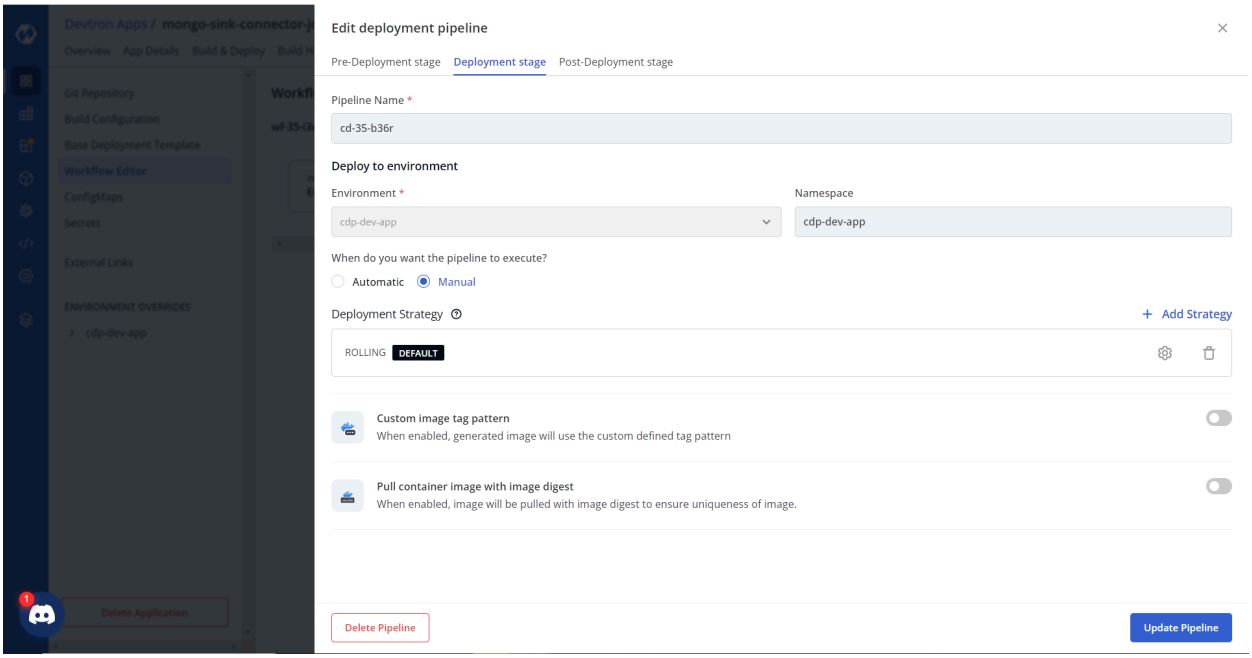
6. Now, click **Try it out** tab below the Auto-generate token section, and update api-token and dockerImage details.

7. Click **Execute**, and on successful execution, check the server response.

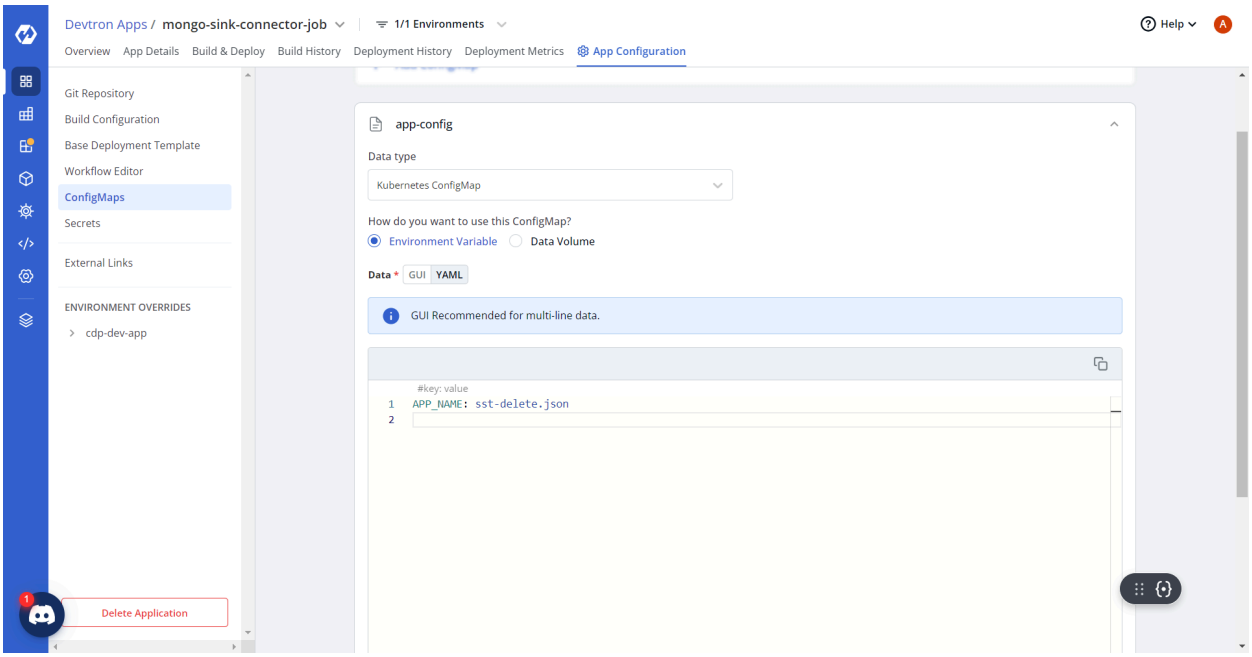


Configuring and deploying custom app

1. Click Edit deployment pipeline, check the deployment stage of the workflow and click **Update Pipeline**.



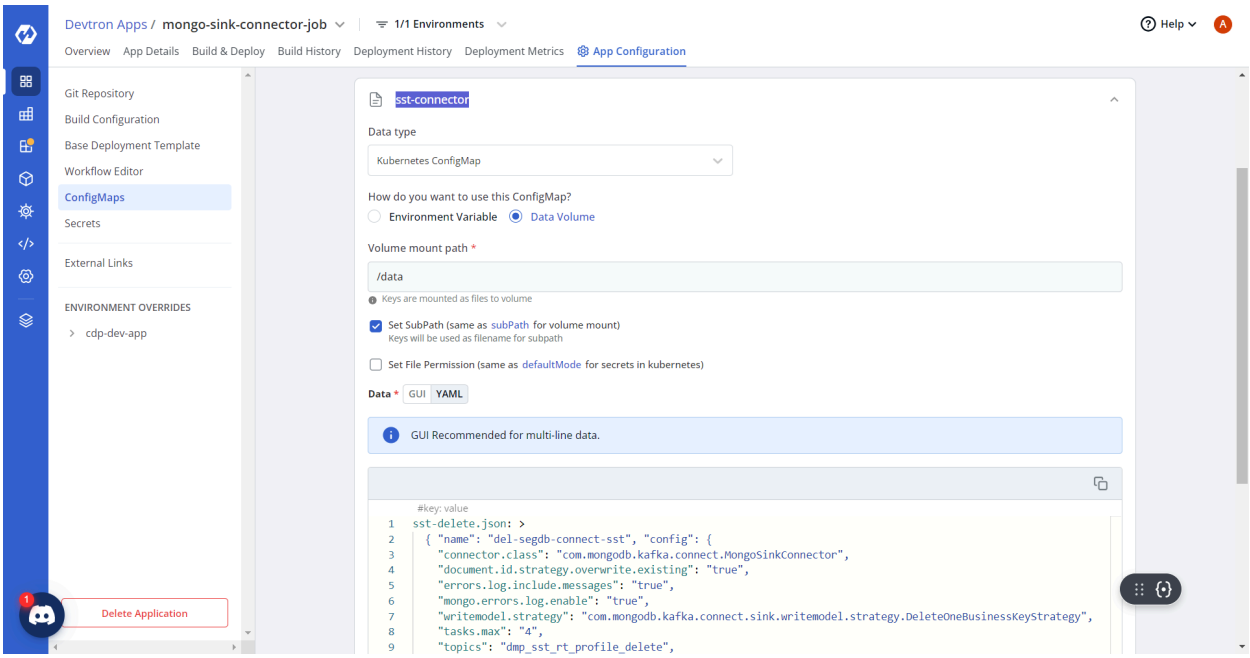
2. In the ConfigMaps section, under the **App Configuration** tab, update the app-config as shown below.



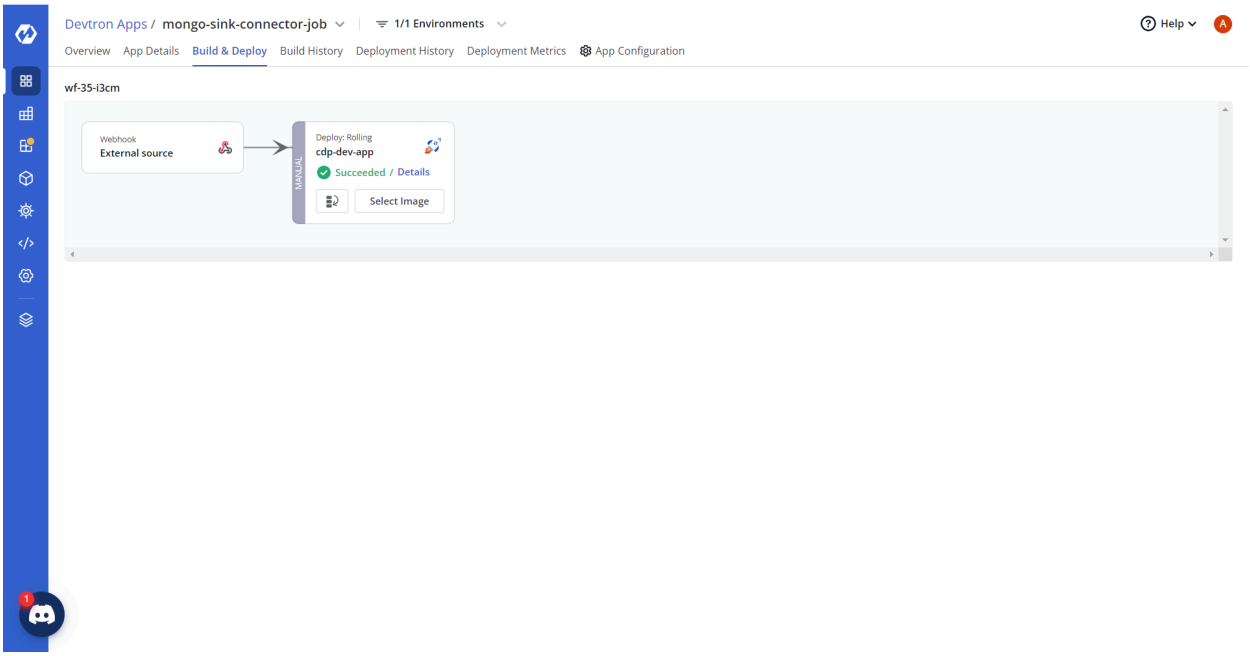
3. Similarly, update ConfigMaps for sst-connector as shown below.

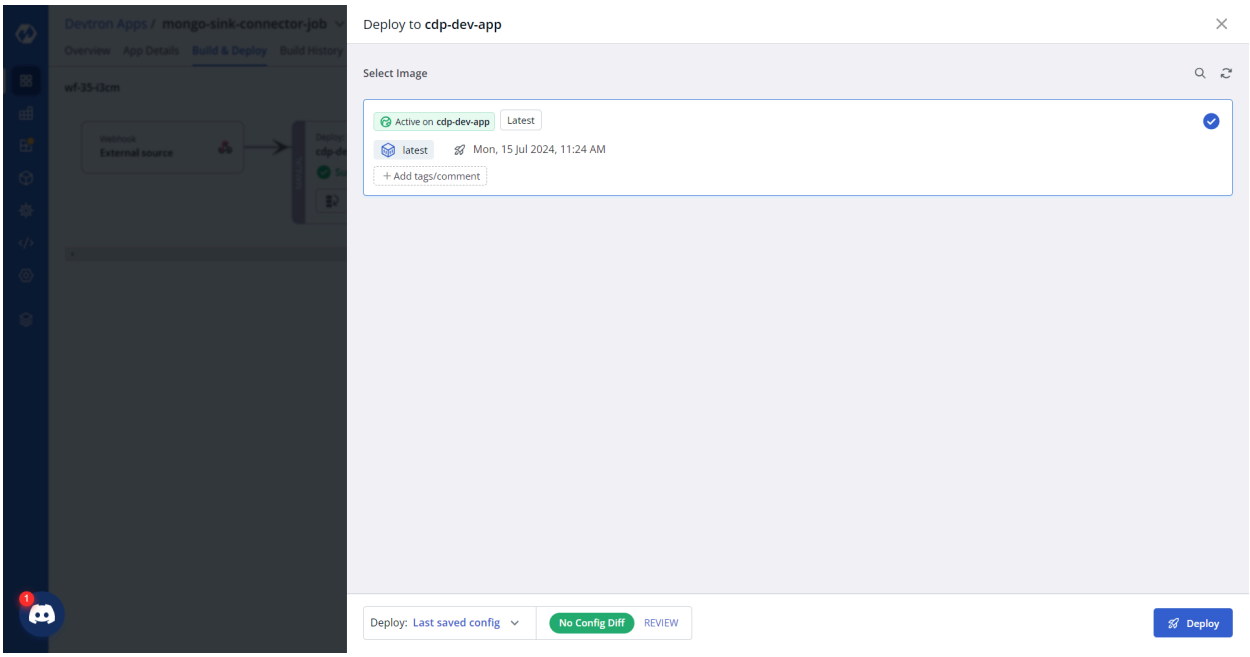
```
sst-delete.json: >
  { "name": "del-segdb-connect-sst", "config": {
    "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
    "document.id.strategy.overwrite.existing": "true",
    "errors.log.include.messages": "true",
    "mongo.errors.log.enable": "true",
    "writemodel.strategy":
 "com.mongodb.kafka.connect.sink.writemodel.strategy.DeleteOneBusinessKeyStrategy",
    "tasks.max": "4",
    "topics": "dmp_sst_rt_profile_delete",
    "namespace.mapper.value.collection.field": "campaignId",
    "collection": "profile",
    "namespace.mapper": "com.mongodb.kafka.connect.sink.namespace.mapping.FieldPathNamespaceMapper",
    "errors.deadletterqueue.context.headers.enable": "true",
    "mongo.errors.tolerance": "all",
    "database": "segmentation",
    "document.id.strategy": "com.mongodb.kafka.connect.sink.processor.id.strategy.PartialValueStrategy",
    "document.id.strategy.partial.value.projection.list": "key",
    "namespace.mapper.error.if.invalid": "false",
    "errors.deadletterqueue.topic.name": "mongo_profile_delete_dlq",
    "connection.uri": "mongodb://segdbconnector:46WNWhr6qXgm2JRmTdlyA==@<mongodb
 hostnames>?replicaSet=cdprsg&authSource=admin",
    "errors.tolerance": "all",
     "name": "del-segdb-connect-sst",
    "value.converter": "org.apache.kafka.connect.storage.StringConverter",
    "document.id.strategy.partial.value.projection.type": "AllowList",
    "post.processor.chain": "com.mongodb.kafka.connect.sink.processor.DocumentIdAdder",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "transforms": "customTransformer",
    "transforms.customTransformer.type":
 "co.lemnisk.kafka.connect.transforms.MongoSinkTransformer$Value"
        }
```

```
    },
sst-update.json: >
  { "name": "kafka-segdb-connect-sst", "config": {
    "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
    "errors.log.include.messages": "true",
    "mongo.errors.log.enable": "true",
    "writemodel.strategy":
 "com.mongodb.kafka.connect.sink.writemodel.strategy.ReplaceOneBusinessKeyStrategy",
    "tasks.max": "12",
    "namespace.mapper.value.collection.field": "campaignId",
    "transforms": "customTransformer,tsConverter",
    "transforms.customTransformer.type":
 "co.lemnisk.kafka.connect.transforms.MongoSinkTransformer$Value",
    "transforms.tsConverter.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value",
    "transforms.tsConverter.field": "expts",
    "transforms.tsConverter.target.type": "Date",
    "transforms.tsConverter.format": "yyyy-MM-dd",
    "errors.deadletterqueue.context.headers.enable": "true",
    "mongo.errors.tolerance": "all",
    "database": "segmentation",
    "document.id.strategy": "com.mongodb.kafka.connect.sink.processor.id.strategy.PartialValueStrategy",
    "namespace.mapper.error.if.invalid": "false",
    "document.id.strategy.partial.value.projection.type": "AllowList",
    "value.converter": "org.apache.kafka.connect.storage.StringConverter",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "document.id.strategy.overwrite.existing": "true",
    "topics": "dmp_sst_profile",
    "collection": "profile",
    "namespace.mapper": "com.mongodb.kafka.connect.sink.namespace.mapping.FieldPathNamespaceMapper",
    "document.id.strategy.partial.value.projection.list": "key_type,key",
    "errors.deadletterqueue.topic.name": "mongo_profile_upsert_dlq",
    "connection.uri": "mongodb://<MongoDB VM IPs>/?replicaSet=cdprsg&authSource=admin",
    "name": "kafka-segdb-connect-sst",
    "value.converter.schemas.enable": "false",
    "errors.tolerance": "all",
    "post.processor.chain": "com.mongodb.kafka.connect.sink.processor.DocumentIdAdder"
        }
    }
```

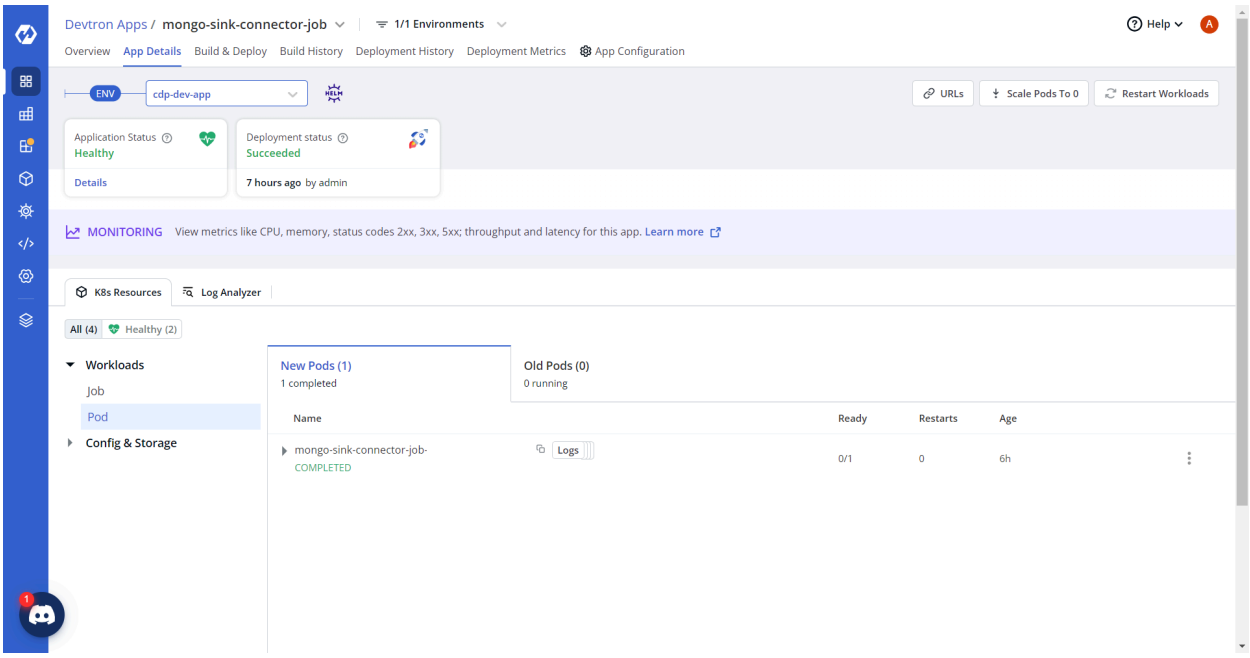


4. Save the configuration, navigate to the **Build & Deploy** tab, and run the execution.





5. On successful deployment, the Pod is in completed status as shown below.

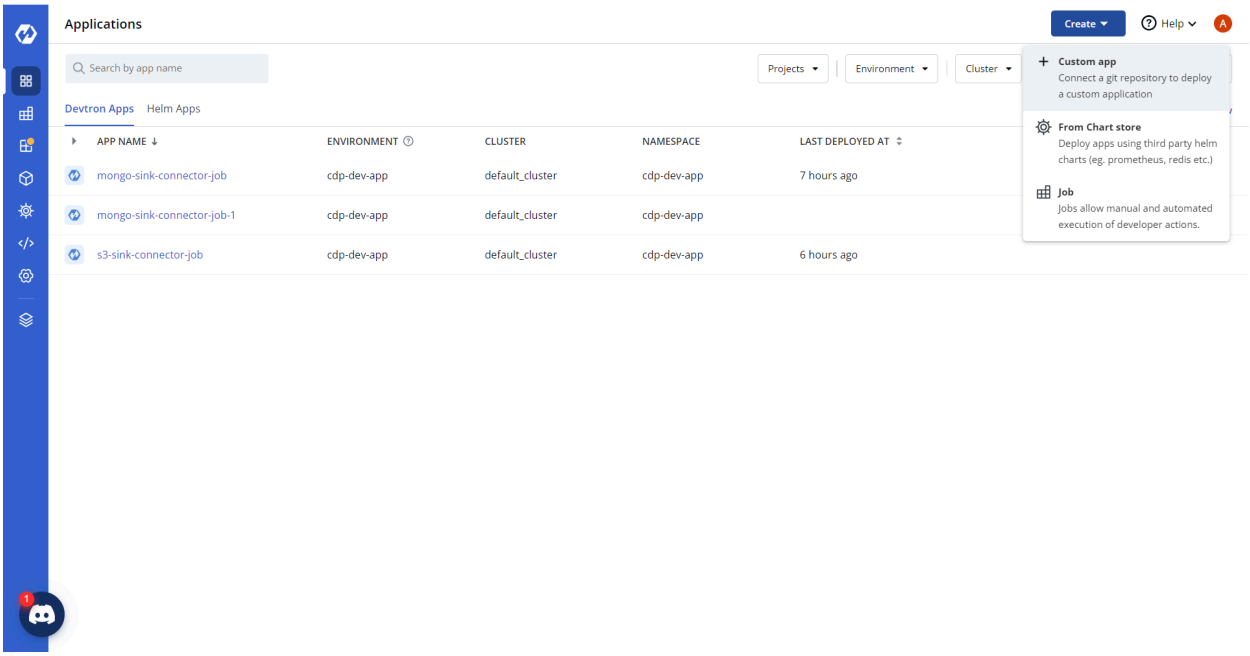


Deploying CDP S3 Connector Sink Job

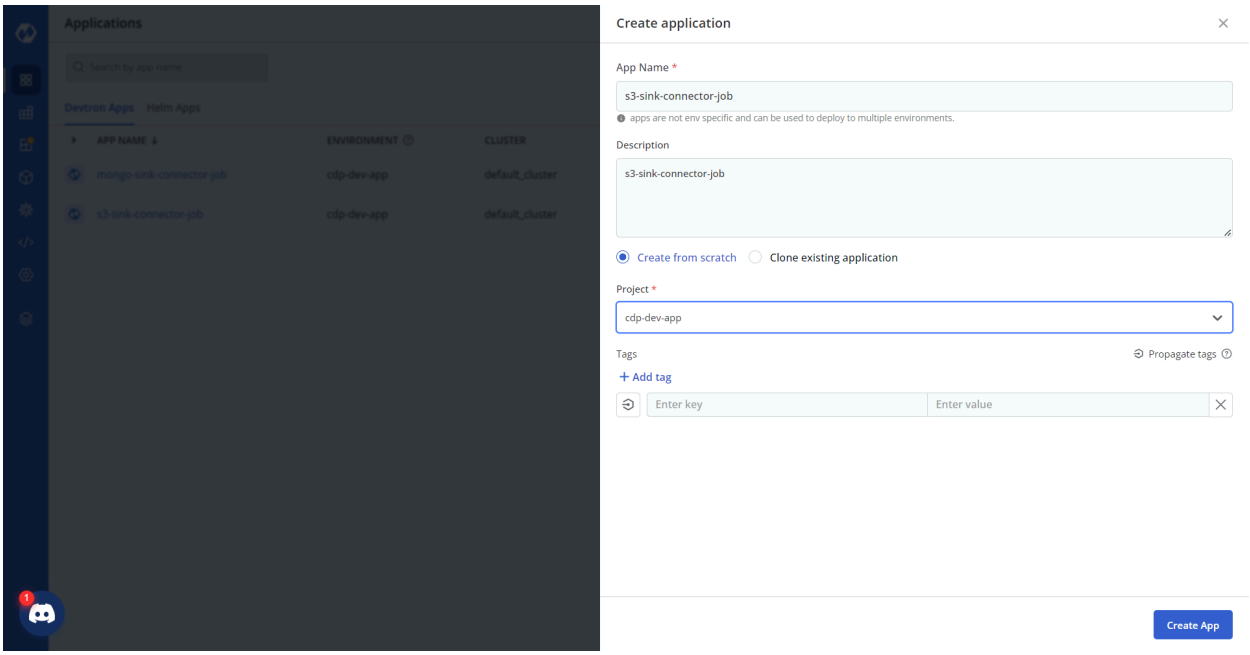
This section provides detailed instructions on how to deploy HCL CDP S3 Connector Sink Job using the Devtron in the AWS

Creating custom app

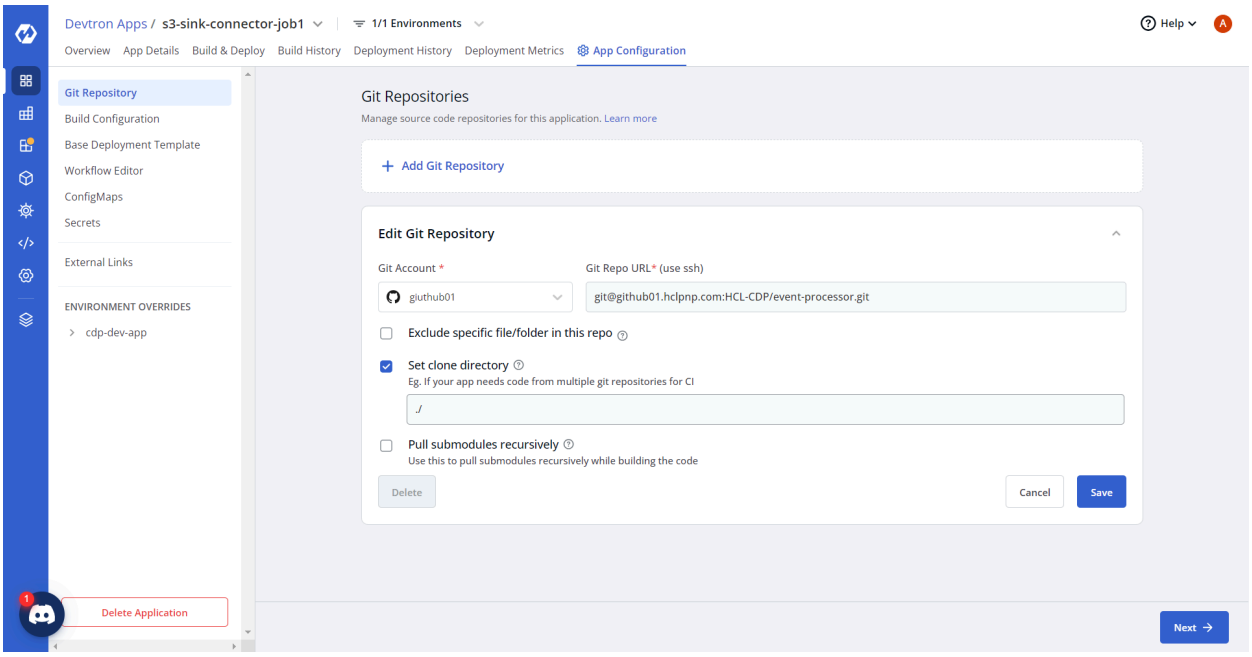
1. In the devtron console, select Applications, and click **Create** > **Custom app**.



2. In the **Create Application** section, enter application name as "s3-sink-connector-job" and select a project to store the application.



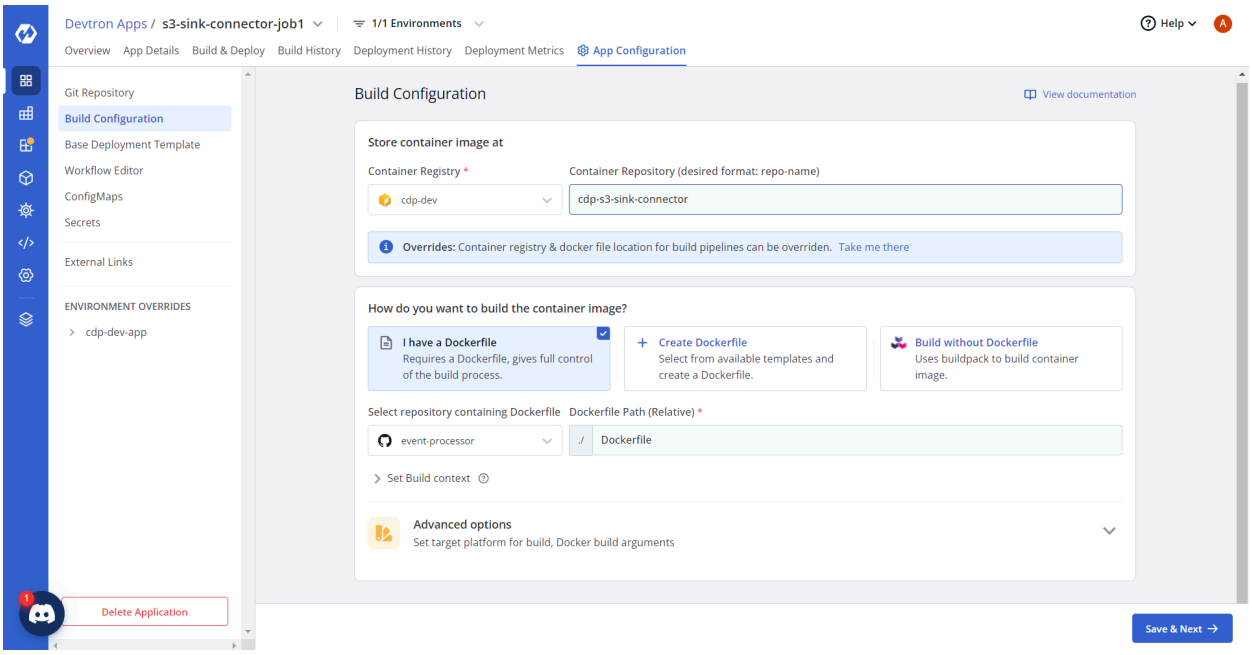
3. Click **Create App** to create the application. Now, click **App Configuration** tab, and update source code repositories for the application.



Building custom app

After successfully creating the custom app, build the custom app with docker image, and configure a workflow for the custom app.

1. In the Devtron Apps page, click **Build Configurations**. In the **App Configuration** tab, select **I have a Dockerfile** option and update the dockerfile path.



2. Click **Save & Next**, and in the **Base Deployment Template** section, update chart type and S3 Connector Sink URLs in the base deployment template as shown below.

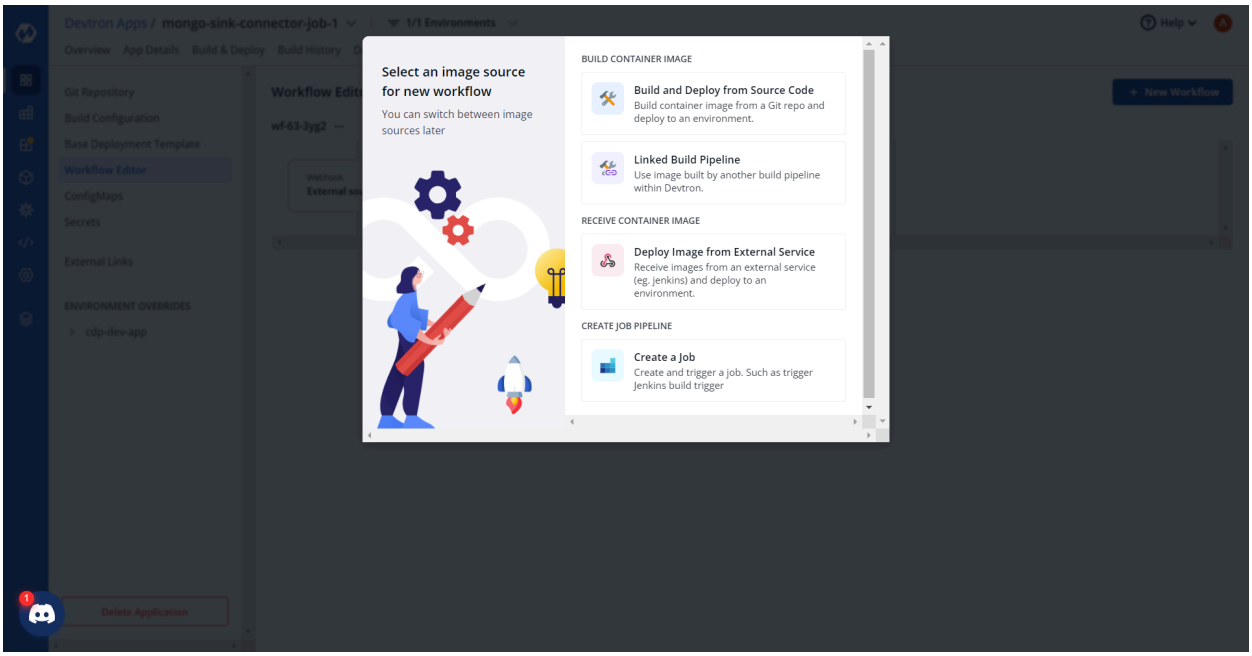
Deployment Template

```
ContainerPort:
  - envoyPort: 8799
    idleTimeout: 1800s
    name: app
    port: 8080
    servicePort: 80
    supportStreaming: true
    useHTTP2: true
EnvVariables: []
GracePeriod: 30
MaxSurge: 1
MaxUnavailable: 0
MinReadySeconds: 60
Spec:
  Affinity:
    Key: null
    Values: nodes
    key: ""
args:
  enabled: true
  value:
    - /bin/sh
    - -c
    - >
      until $(curl --output /dev/null --silent --head --fail
      http://cdp-s3-sink-connector-service.cdp-dev-app.svc.cluster.local:80); do
```

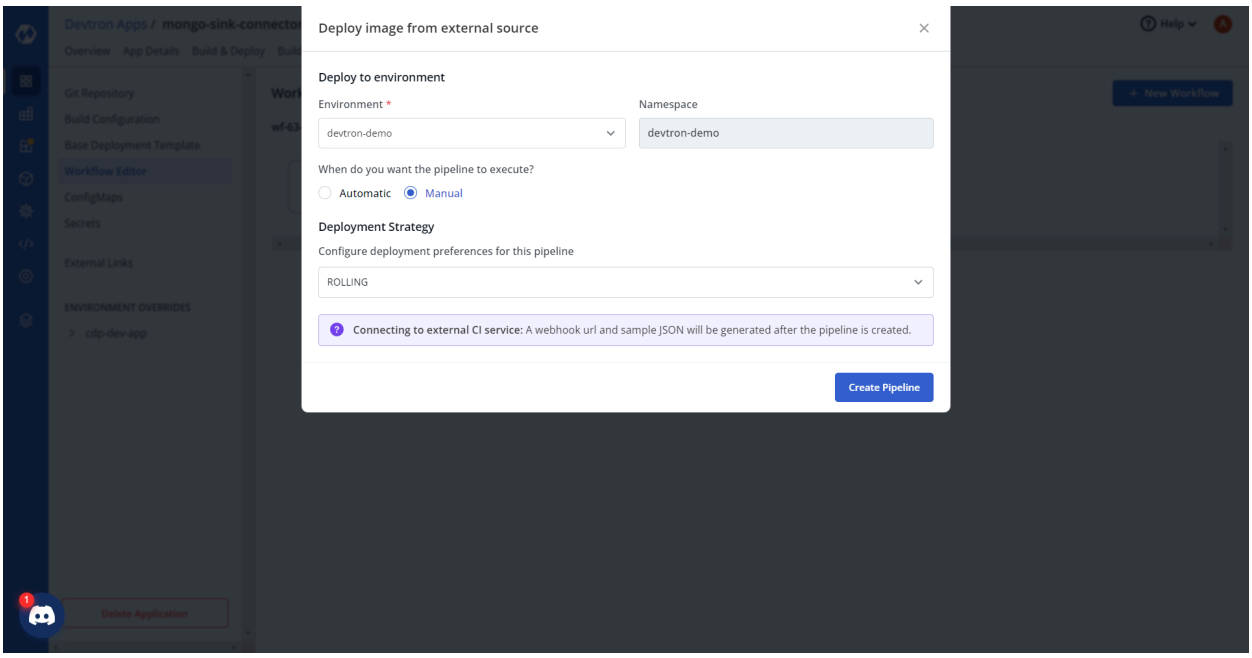
```
        echo "Waiting for Kafka Connect API..."
        sleep 5
      done
      echo "Posting Kafka Connect configuration..."
      curl -X POST -H "Content-Type: application/json" --data @/data/${APP_NAME}
 http://cdp-s3-sink-connector-service.cdp-dev-app.svc.cluster.local:80/connectors
command:
  enabled: false
  value: []
containerSecurityContext:
  allowPrivilegeEscalation: false
containers: []
cronjobConfigs:
  concurrencyPolicy: Allow
  failedJobsHistoryLimit: 1
  restartPolicy: OnFailure
  schedule: "* * * * *"
  startingDeadlineSeconds: 100
  successfulJobsHistoryLimit: 3
  suspend: false
ephemeralContainers: []
image:
  pullPolicy: IfNotPresent
imagePullSecrets: []
initContainers: []
jobConfigs:
  activeDeadlineSeconds: 100
  backoffLimit: 5
  completions: 1
  parallelism: 1
  suspend: false
kedaAutoscaling:
  envSourceContainerName: ""
  failedJobsHistoryLimit: 5
  maxReplicaCount: 2
  minReplicaCount: 1
  pollingInterval: 30
  rolloutStrategy: default
  scalingStrategy:
    customScalingQueueLengthDeduction: 1
    customScalingRunningJobPercentage: "0.5"
    multipleScalersCalculation: max
    pendingPodConditions:
      - Ready
      - PodScheduled
      - AnyOtherCustomPodCondition
    strategy: custom
  successfulJobsHistoryLimit: 5
  triggerAuthentication:
    enabled: false
    name: ""
    spec: {}
  triggers:
    - authenticationRef: {}
      metadata:
        host: RabbitMqHost
        queueLength: "5"
        queueName: hello
```

```
      type: rabbitmq
kind: Job
pauseForSecondsBeforeSwitchActive: 30
podAnnotations: {}
podLabels: {}
podSecurityContext: {}
prometheus:
  release: monitoring
rawYaml: []
readinessGates: []
resources:
  limits:
    cpu: "0.05"
    memory: 50Mi
  requests:
    cpu: "0.01"
    memory: 10Mi
secret:
  data: {}
  enabled: false
server:
  deployment:
    image: ""
    image_tag: 1-95af053
service:
  annotations: {}
  enabled: false
  type: ClusterIP
servicemonitor:
  additionalLabels: {}
setHostnameAsFQDN: false
shareProcessNamespace: false
tolerations: []
topologySpreadConstraints: []
volumeMounts: []
volumes: []
waitForSecondsBeforeScalingDown: 30
```

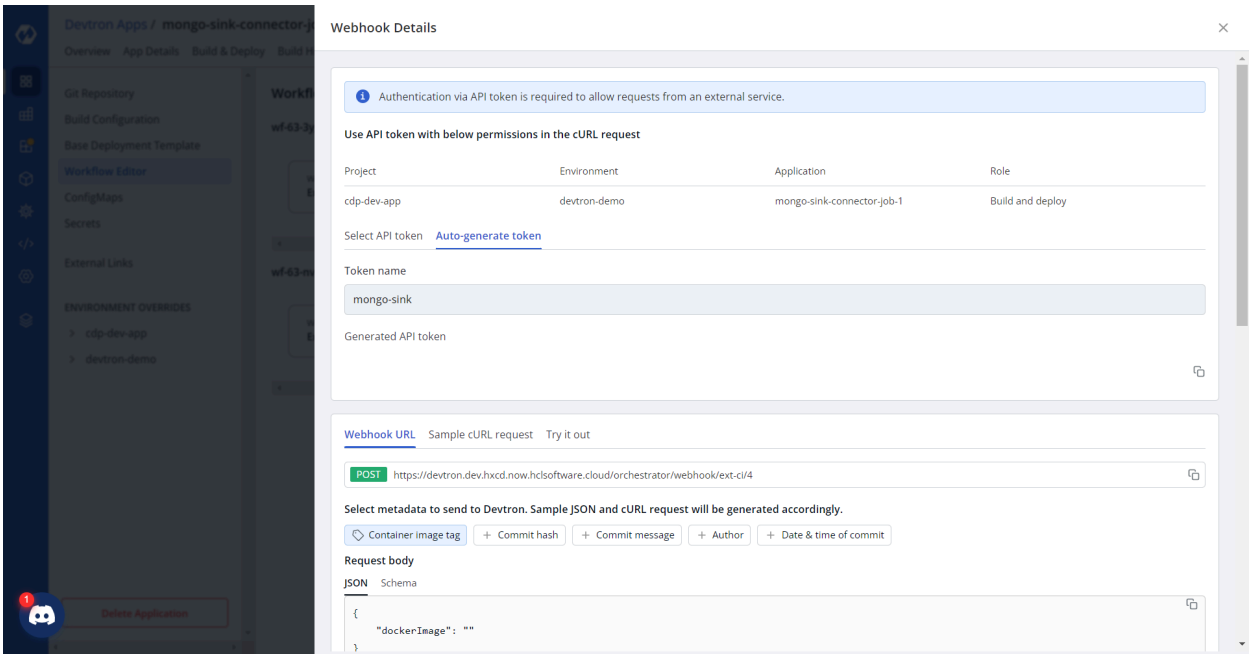
3. Click **Save & Next**. In the Workflow Editor section, add new workflow, and select the **Deploy Image from External Source** option.



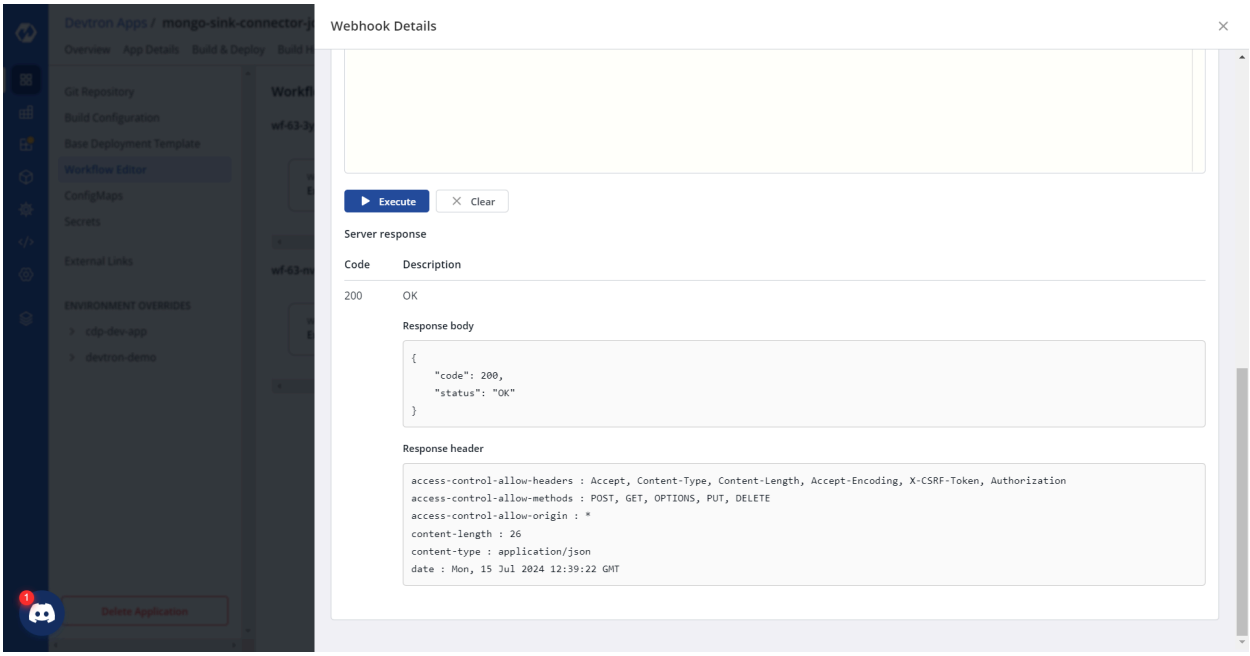
4. Configure required parameters, and click **Create pipeline** at the right bottom corner of the Deploy image from external source section.



5. In the Workflow Editor, click **Edit Workflow**, and in the **Webhook Details** section, click the **Auto Generate Token** tab, and copy the token details.

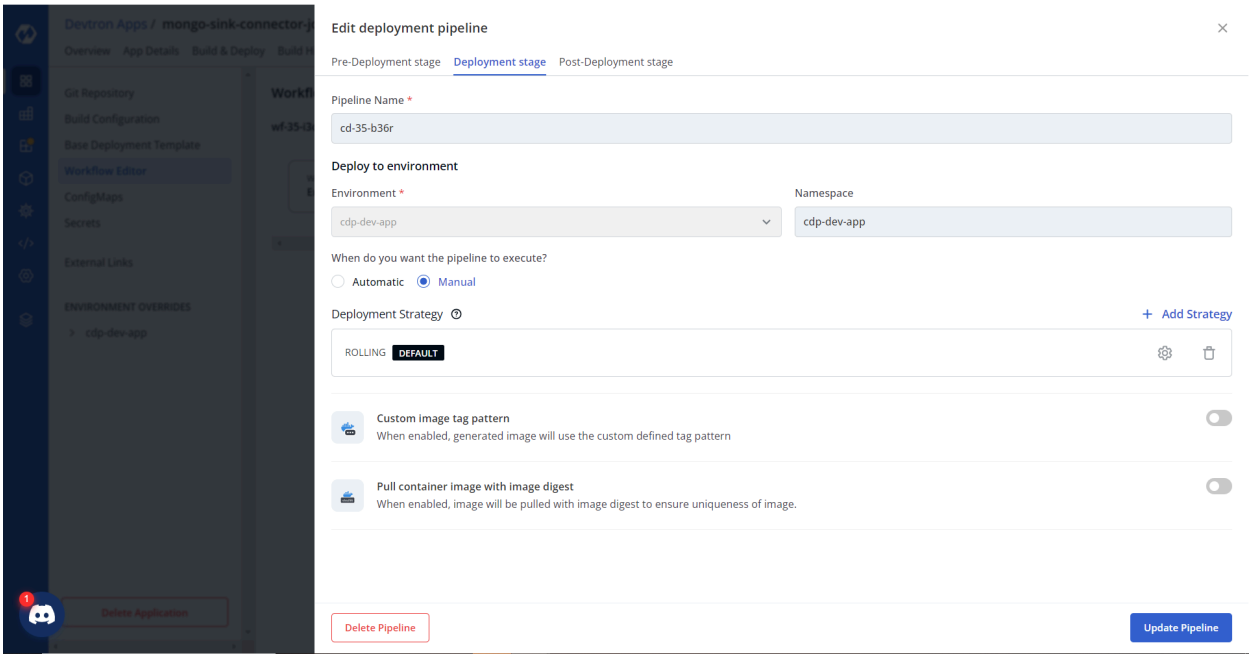


6. Now, click **Try it out** tab below the Auto-generate token section, and update api-token and dockerImage details.
7. Click **Execute**, and on successful execution, check the server response.

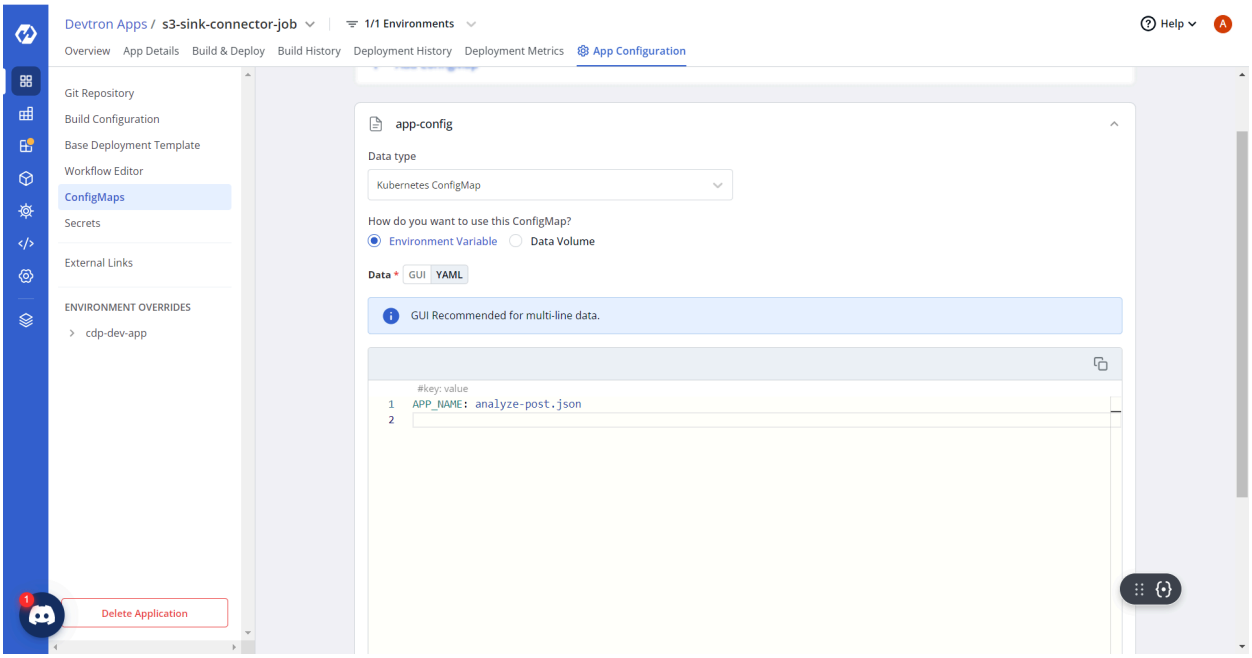


Configuring and deploying custom app

1. Click Edit deployment pipeline, check the deployment stage of the workflow and click **Update Pipeline**.



2. In the **ConfigMaps** section, under the **App Configuration** tab, update the app-config as shown below.



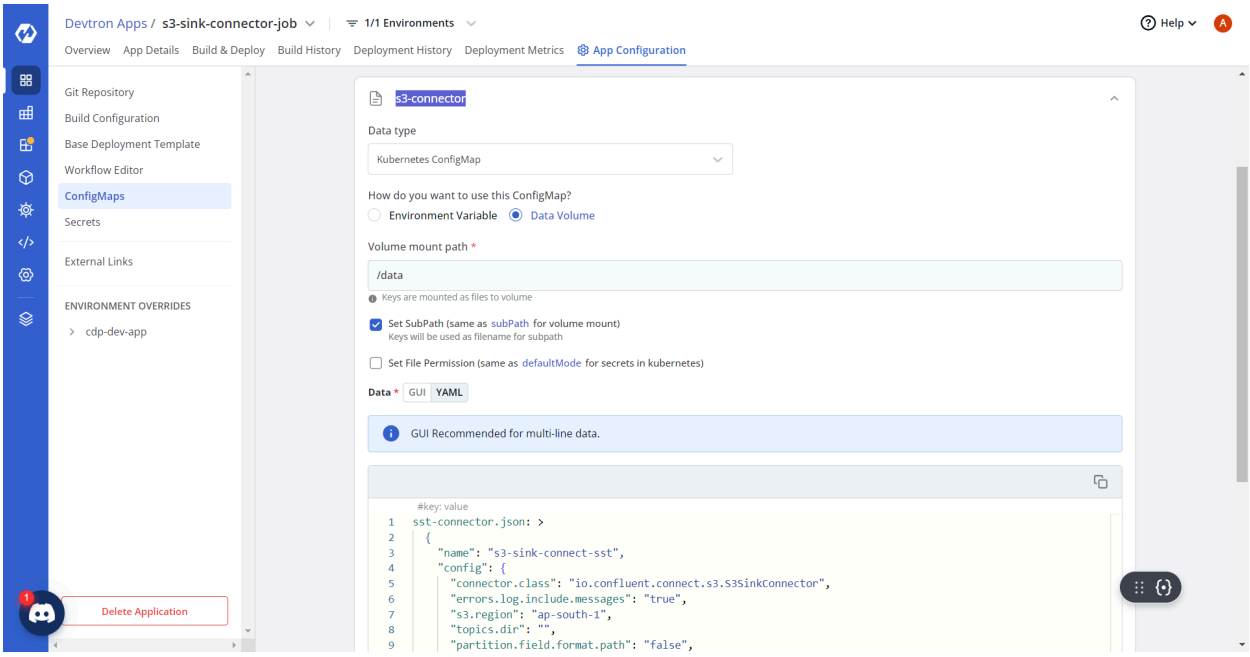
3. Similarly, update the ConfigMaps for sst-connector as shown below.

```
sst-connector.json: >
  {
    "name": "s3-sink-connect-sst",
```

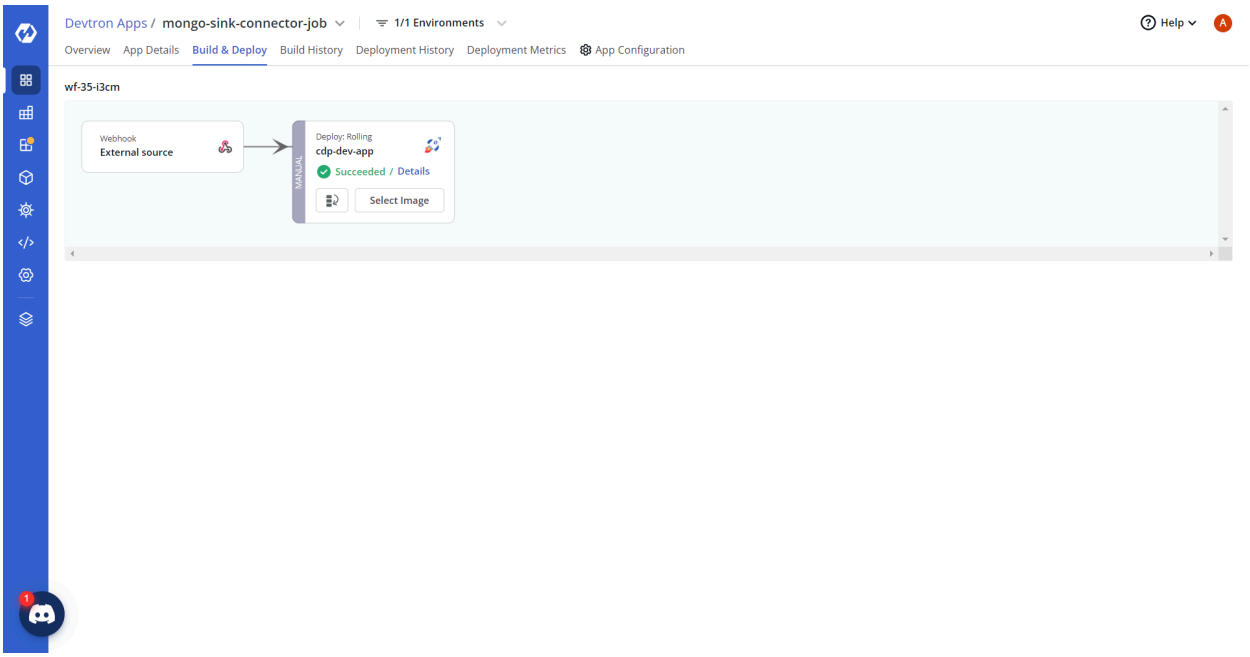
```
    "config": {
      "connector.class": "io.confluent.connect.s3.S3SinkConnector",
      "errors.log.include.messages": "true",
      "s3.region": "ap-south-1",
      "topics.dir": "",
      "partition.field.format.path": "false",
      "flush.size": "10000",
      "tasks.max": "6",
      "s3.part.size": "5242880",
      "timezone": "UTC",
      "transforms": "GenericTransformer",
      "rotate.interval.ms": "-1",
      "locale": "US",
      "errors.deadletterqueue.context.headers.enable": "true",
      "format.class": "io.confluent.connect.s3.format.bytearray.ByteArrayFormat",
      "transforms.GenericTransformer.type":
 "co.lemnisk.kafka.connect.transforms.GenericTransformer$Value",
      "aws.access.key.id": "",
      "errors.deadletterqueue.topic.replication.factor": "2",
      "value.converter": "org.apache.kafka.connect.json.JsonConverter",
      "errors.log.enable": "true",
      "s3.bucket.name": "hclsw-hxcd-hss-prd-s3-cdp-app",
      "key.converter": "org.apache.kafka.connect.storage.StringConverter",
      "partition.duration.ms": "300000",

 "topics":"sst_blacklistProfile,sst_mergeProfile,sst_outboundEvents,sst_inboundEvents,sst_failedEvents,s
st_blacklist,sst_deletedOid,sst_profile","directory.delim":"/","aws.secret.access.key":"","transforms.Ge
nericTransformer.meta":"[{\"topic\":\"sst_blacklistProfile\",\"tsCol\":\"ts\",\"dataFormat\":
\"json\"},{\"topic\":\"sst_mergeProfile\",\"tsCol\":\"ts\",\"dataFormat\":\"json\"},{\"topic\":
\"sst_outboundEvents\",\"tsCol\":\"ts\",\"dataFormat\":\"json\"},{\"topic\":\"sst_inboundEvents\",
\"tsCol\":\"ts\",\"dataFormat\":\"json\"},{\"topic\":\"sst_failedEvents\",\"tsCol\":\"ts\",
\"dataFormat\":\"json\"},{\"topic\":\"sst_blacklist\",\"tsCol\":\"4\",\"dataFormat\":
\"tsv\"},{\"topic\":\"sst_deletedOid\",\"tsCol\":\"5\",\"dataFormat\":\"tsv\"},{\"topic\":
\"sst_profile\",\"tsCol\":\"ts\",\"dataFormat\":\"json\"}]",
      "partition.field.name": "p1,p2",
      "s3.compression.type": "gzip",
      "errors.deadletterqueue.topic.name": "dlq_sst",
      "partitioner.class": "co.lemnisk.kafka.connect.partitioner.GenericPartitioner",
      "value.converter.schemas.enable": "false",
      "name": "s3-sink-connect-sst",
      "errors.tolerance": "all",
      "storage.class": "io.confluent.connect.s3.storage.S3Storage",
      "rotate.schedule.interval.ms": "900000",
      "path.format": "YYYY/MM/dd",
      "timestamp.extractor": "Record"
    }
    },
diapi-connector.json: >
  {
    "name": "s3-sink-connect-diapi",
    "config": {
      "connector.class": "io.confluent.connect.s3.S3SinkConnector",
      "errors.log.include.messages": "true",
      "s3.region": "ap-south-1",
      "topics.dir": "",
      "partition.field.format.path": "false",
      "flush.size": "10000",
      "tasks.max": "6",
```

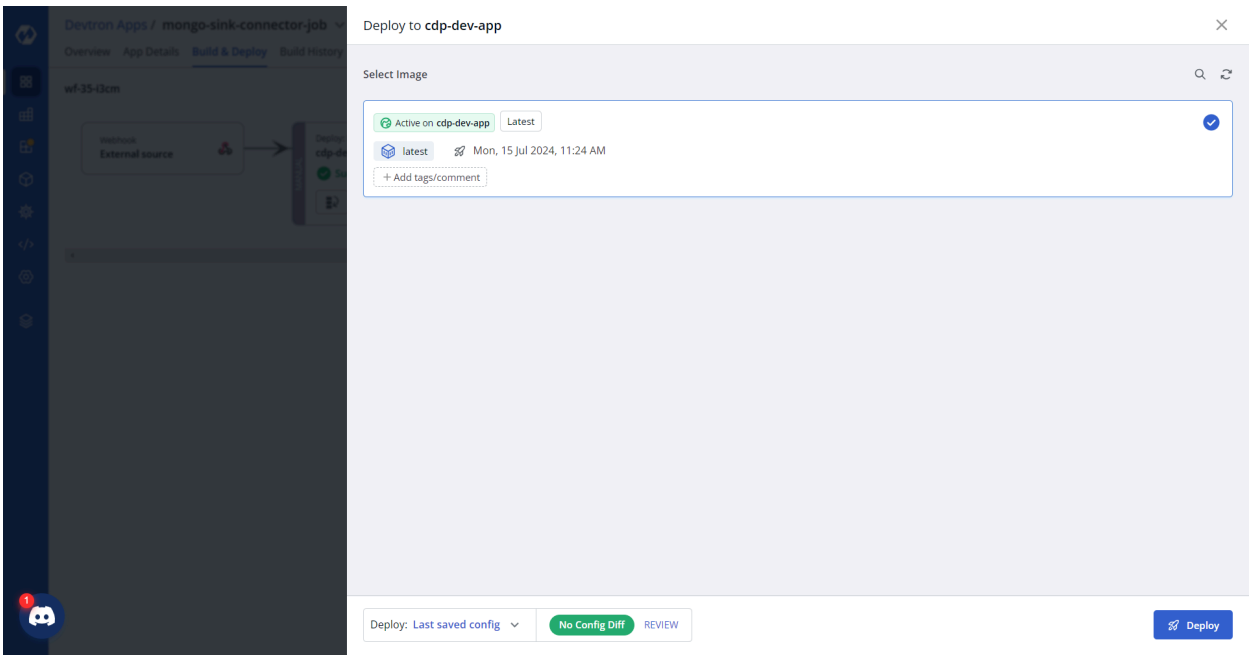
```
      "s3.part.size": "5242880",
      "timezone": "UTC",
      "transforms": "GenericTransformer",
      "rotate.interval.ms": "-1",
      "locale": "US",
      "errors.deadletterqueue.context.headers.enable": "true",
      "format.class": "io.confluent.connect.s3.format.bytearray.ByteArrayFormat",
      "transforms.GenericTransformer.type":
 "co.lemnisk.kafka.connect.transforms.GenericTransformer$Value",
      "aws.access.key.id": "",
      "errors.deadletterqueue.topic.replication.factor": "2",
      "value.converter": "org.apache.kafka.connect.json.JsonConverter",
      "errors.log.enable": "true",
      "s3.bucket.name": "hclsw-hxcd-hss-prd-s3-cdp-app",
      "key.converter": "org.apache.kafka.connect.storage.StringConverter",
      "partition.duration.ms": "300000",
      "topics": "diapi_outbound",
      "directory.delim": "/",
      "aws.secret.access.key": "",
      "transforms.GenericTransformer.meta":
 "[{\"topic\":\"diapi_outbound\",\"tsFormat\":\"yyyy-MM-dd'T'HH:mm:ss\",\"tsCol\":\"ts\",\"dataFormat\":
\"json\",\"inputTz\":\"UTC\",\"outputTz\":\"Asia/Kolkata\"}]",
      "partition.field.name": "eventType,campaignId",
      "s3.compression.type": "gzip",
      "errors.deadletterqueue.topic.name": "dlq_diapi",
      "partitioner.class": "co.lemnisk.kafka.connect.partitioner.GenericPartitioner",
      "value.converter.schemas.enable": "false",
      "name": "s3-sink-connect-diapi",
      "errors.tolerance": "all",
      "storage.class": "io.confluent.connect.s3.storage.S3Storage",
      "rotate.schedule.interval.ms": "900000",
      "path.format": "YYYY/MM/dd",
      "timestamp.extractor": "Record"
    }
    },
analyze-post.json: >
  { "name": "s3-sink-connect-analyze_post", "config": {
      "connector.class": "io.confluent.connect.s3.S3SinkConnector",
      "errors.log.include.messages": "true",
      "s3.region": "ap-south-1",
      "topics.dir": "",
      "partition.field.format.path": "false",
      "flush.size": "10000",
      "tasks.max": "6",
      "s3.part.size": "5242880",
      "timezone": "UTC",
      "transforms": "GenericTransformer",
      "rotate.interval.ms": "-1",
      "locale": "US",
      "errors.deadletterqueue.context.headers.enable": "true",
      "format.class": "io.confluent.connect.s3.format.bytearray.ByteArrayFormat",
      "transforms.GenericTransformer.type":
 "co.lemnisk.kafka.connect.transforms.GenericTransformer$Value",
      "aws.access.key.id": "",
      "errors.deadletterqueue.topic.replication.factor": "2",
      "value.converter": "org.apache.kafka.connect.storage.StringConverter",
      "errors.log.enable": "true",
      "s3.bucket.name": "hclsw-hxcd-hss-prd-s3-cdp-app",
```

```
      "key.converter": "org.apache.kafka.connect.storage.StringConverter",
      "partition.duration.ms": "300000",
      "topics": "analyze_post",
      "directory.delim": "/",
      "aws.secret.access.key": "",
      "transforms.GenericTransformer.meta":
 "[{\"topic\":\"analyze_post\",\"tsFormat\":\"yyyy-MM-dd'T'HH:mm:ss\",\"dataFormat\":\"json\",
\"inputFormat\":\"tsv\",\"dataCol\":3,\"partitionerKeys\":\"AnalyzePost,{{campaignId}}\",\"tsCol\":
\"receivedAt\",\"inputTz\":\"UTC\",\"outputTz\":\"Asia/Kolkata\"}]",
      "s3.compression.type": "gzip",
      "errors.deadletterqueue.topic.name": "dlq_analyze_post",
      "partitioner.class": "co.lemnisk.kafka.connect.partitioner.GenericPartitioner",
      "value.converter.schemas.enable": "false",
      "name": "s3-sink-connect-analyze_post",
      "errors.tolerance": "all",
      "storage.class": "io.confluent.connect.s3.storage.S3Storage",
      "rotate.schedule.interval.ms": "900000",
      "path.format": "YYYY/MM/dd",
      "timestamp.extractor": "Record"
  } }
```

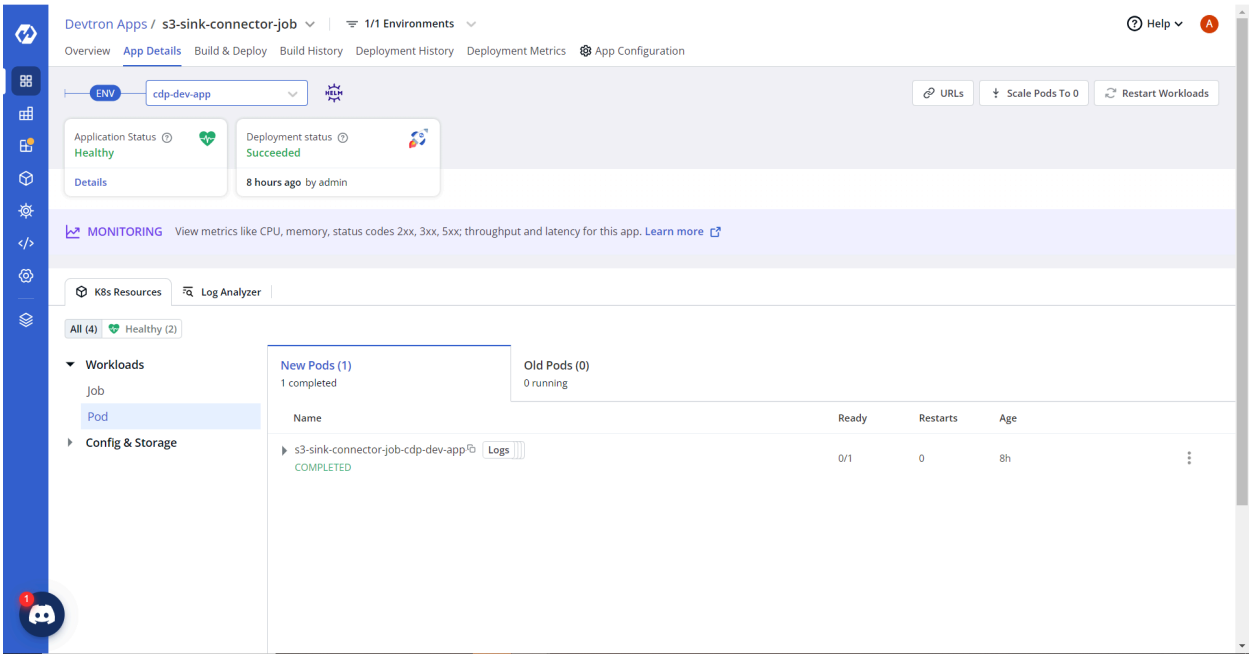


4. Save the configuration, navigate to the **Build & Deploy** tab, and run the execution.


Devtron Apps / mongo-sink-connector-job
1/1 Environments
Help
Overview
App Details
Build & Deploy
Build History
Deployment History
Deployment Metrics
App Configuration
wf-35-i3cm
Webhook
External source
Deploy: Rolling
cdp-dev-app
Succeeded / Details
Select Image


Deploy to cdp-dev-app
Select Image
Active on cdp-dev-app
Latest
latest
Mon, 15 Jul 2024, 11:24 AM
+ Add tags/comment
Deploy: Last saved config
No Config Diff
REVIEW
Deploy

5. On successful deployment, the Pod is in completed status as shown below.



Chapter 3. Deploying HCL CDP Marketing Automation Component on AWS

This section details the deploying Marketing Automation components required for HCL CDP on the AWS platform.

Introduction

This document provides a comprehensive guide to installing and configuring the necessary services and components for deploying HCL CDP Marketing Automation on the AWS platform using Devtron.

Prerequisites

To deploy the micro-services based HCL CDP Marketing Automation components on a containerized platform using the Devtron CI/CD tool, ensure the following prerequisites are met:

The following tools must be available to facilitate the installation process:

- **Kubernetes CLI:** Verify the availability of the `kubectl` command.
- **Helm CLI:** Ensure the `helm` command-line tool is installed.
- **Devtron Deployment Tool:** Confirm that Devtron is set up for managing deployments.

List of components for Marketing Automation modules

The following components must be deployed and configured on the cluster to support the deployment of HCL CDP Marketing Automation modules.

These components provide the essential functionalities required for end-to-end operations of the CDP:

Application/Services	Deployment Type
CDP Trigger	Helm chart
CDP Core	Helm chart
CDP Flip	Helm chart
CDP SMSEmailSender	Helm chart
CDP ExternalAPI	Helm chart
CDP Tryitout	Helm chart
CDP Celerycallbackconsumer	Helm chart
CDP Celerykafkaetaconsumer	Helm chart
CDP Audienceuploader	Helm chart
CDP Clicktrackers	Helm chart

CDP Etaevents	Helm chart
CDP Webpush	Helm chart
CDP Webhooks	Helm chart
TC-Redis	VM Based Deployment
Rabbitmq-Celery	VM Based Deployment
Rabbitmq UI	VM Based Deployment
Scheduler	VM Based Deployment

Deployment Process

This section provides step-by-step instructions for deploying each service, component, and module required for HCL CDP Marketing Automation.

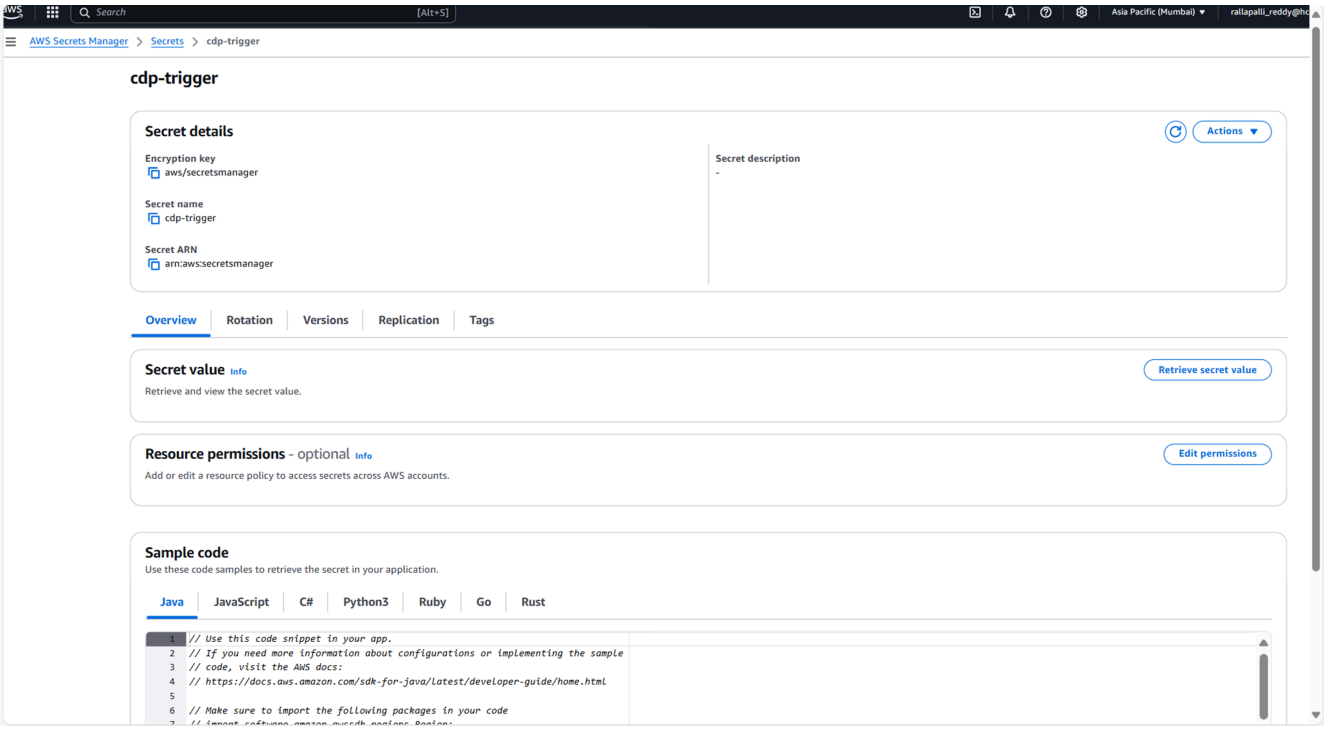
Follow the guidelines in each subsection to ensure the successful deployment and configuration of the respective elements.

Deploying CDP Trigger

This section provides detailed instructions on how to deploy HCL CDP Trigger using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-trigger secret in the AWS Secret Manager before deploying CDP Trigger.



To create CDP Trigger secret in the AWS secret manager, follow the steps below:

1. Create a cdp-trigger secret sample key and value, and update the actual values as shown below.

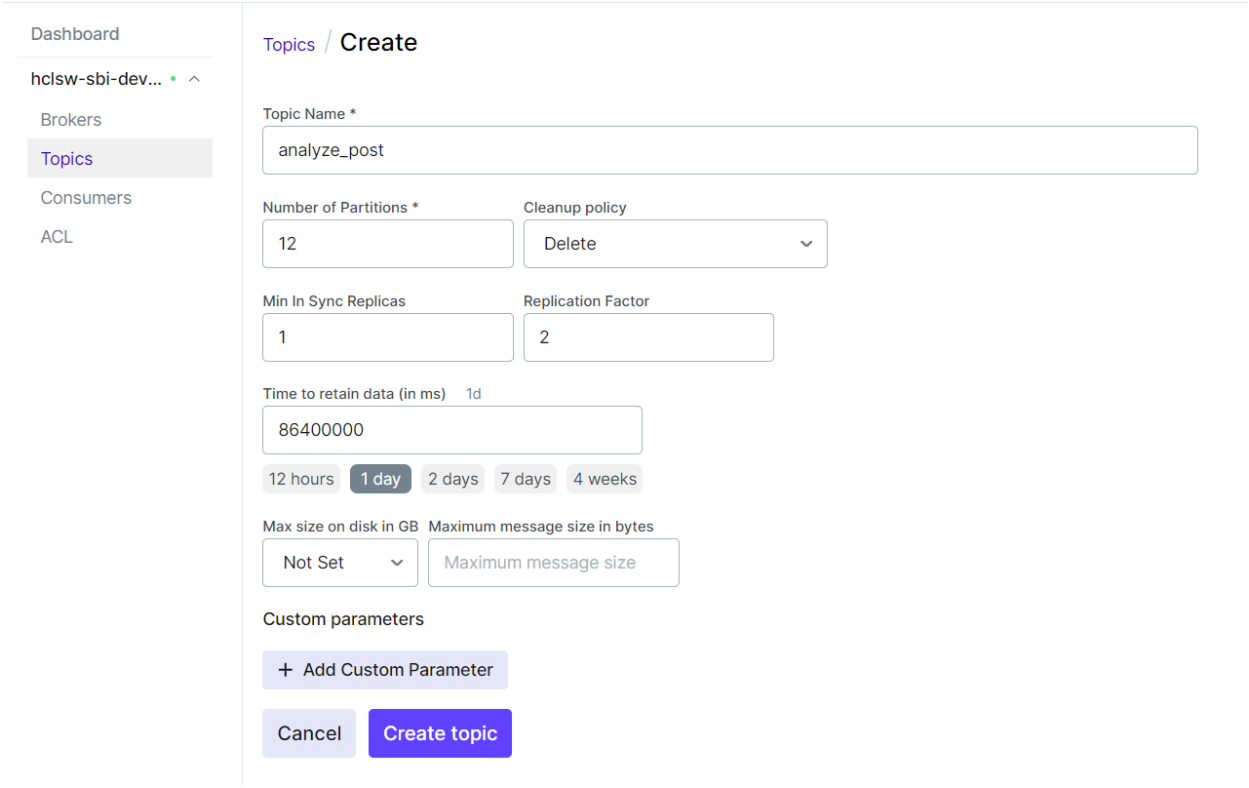
```
{
"AerospikeCluster":"",
"AerospikeTimeout":"10000",
"MY_SQL_HOST":"",
"MysqlDatabase":"vrm",
"MysqlUsername":"",
"MysqlPassword":"",
"MysqlPort":"",
"allowTrafficForSpecificCampaign":"false",
"KafkaConsumerConfig":"",
"KafkaProducerConfig":"",
"KafkaProducerConfigaps1":""
}
```

2. Create a Kubernetes Secret with Vault credentials to fetch the secrets from the vault.

```
apiVersion: v1
kind: Secret
metadata:
  name: vault-creds
  namespace: cdp-dev-app
type: Opaque
data:
  username: <vault username>
  password: <vault password>
```

3. Create separate service account cdp-dev-app-sa with appropriate roles and permissions, and update ConfigMaps data with actual values.

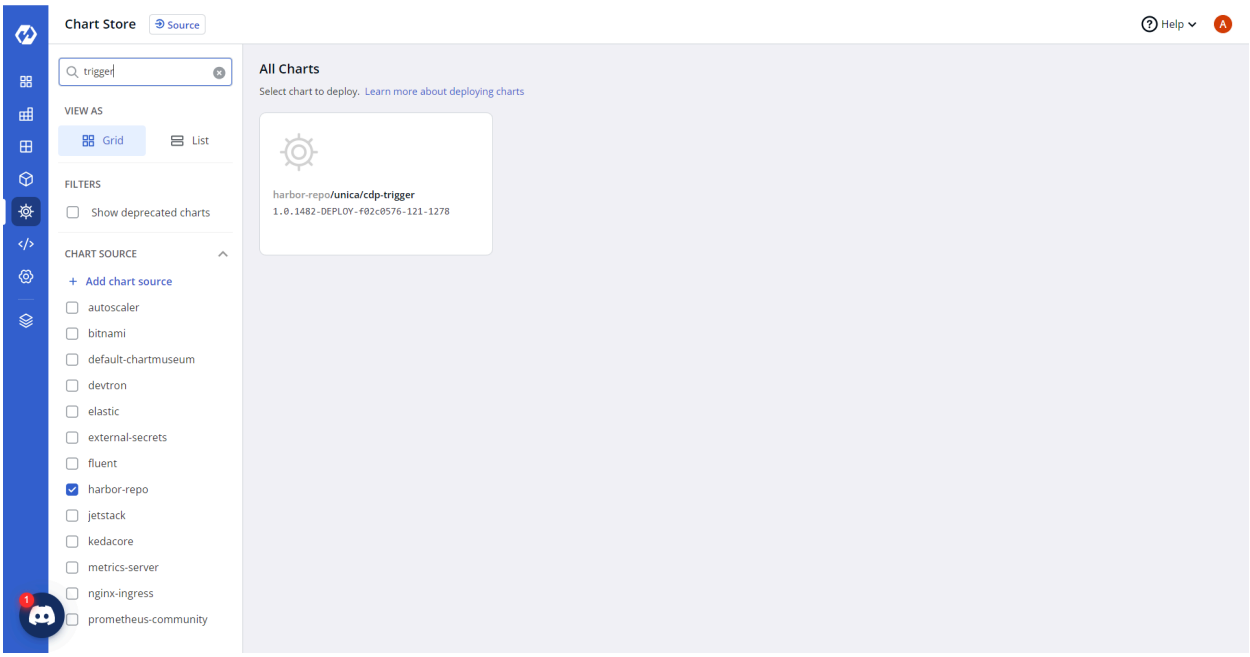
4. Create required Kafka topics before deployment. A sample Kafka topic configuration is shown below.



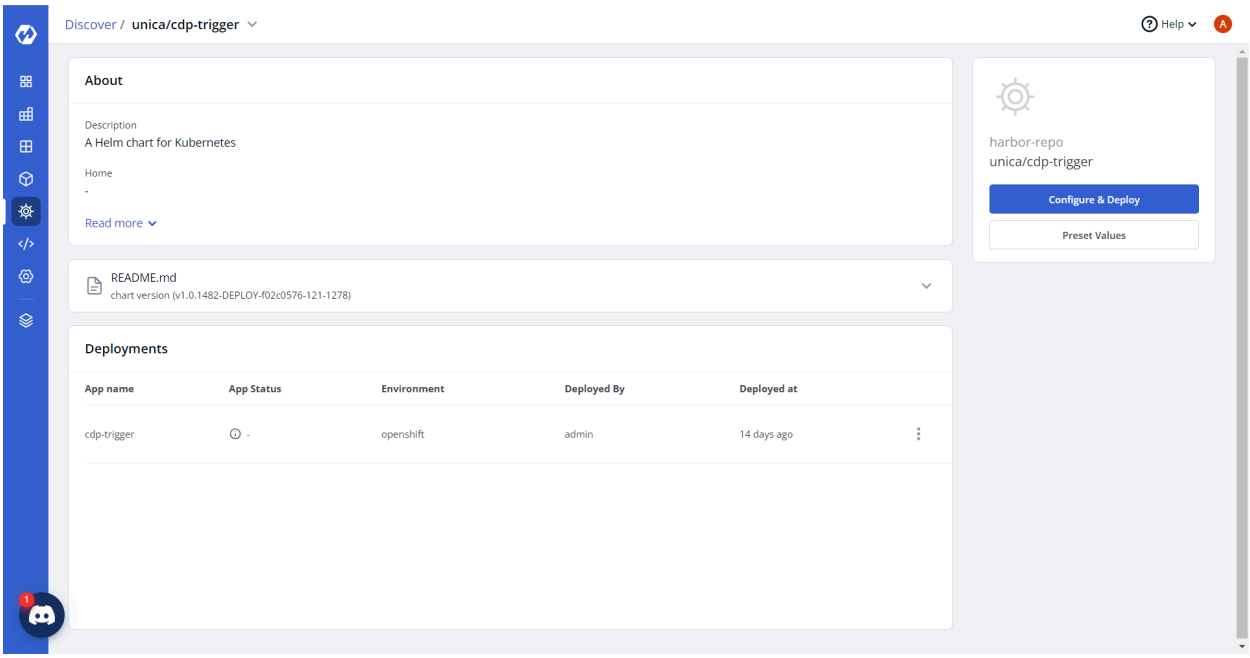
Deploy CDP Trigger

To deploy CDP Trigger using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select cdp-trigger chart to deploy.



2. Now, configure and deploy the cdp-trigger charts.

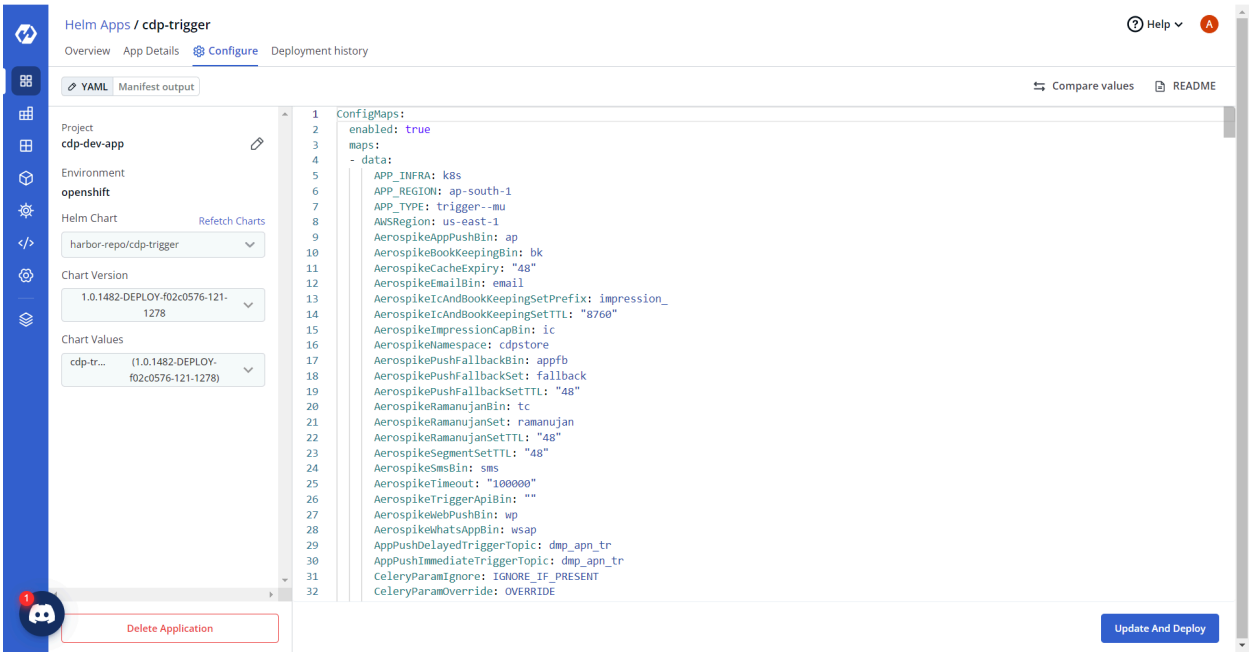


3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

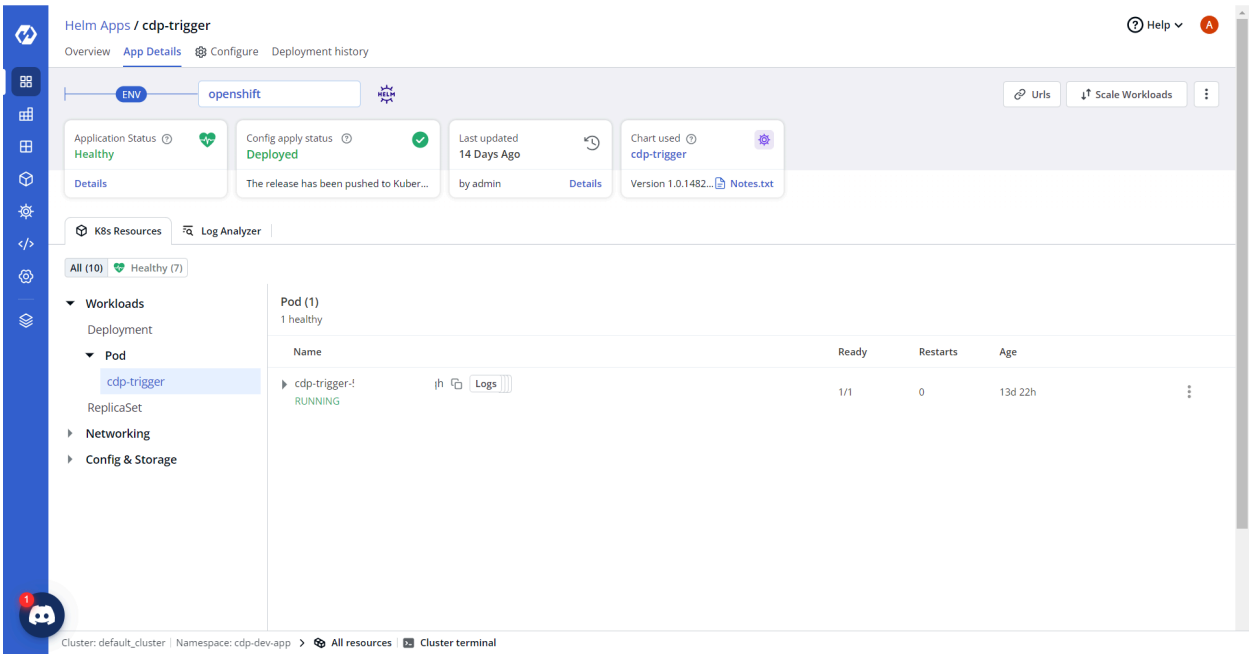
```
APP_INFRA: k8s
APP_REGION: ap-south-1
APP_TYPE: trigger--mu
AerospikeAppPushBin: ap
AerospikeBookKeepingBin: bk
```

```
AerospikeCacheExpiry: "48"
AerospikeEmailBin: email
AerospikeIcAndBookKeepingSetPrefix: impression_
AerospikeIcAndBookKeepingSetTTL: "8760"
AerospikeImpressionCapBin: ic
AerospikeNamespace: cdpstore
AerospikePushFallbackBin: appfb
AerospikePushFallbackSet: fallback
AerospikePushFallbackSetTTL: "48"
AerospikeRamanujanBin: tc
AerospikeRamanujanSet: ramanujan
AerospikeRamanujanSetTTL: "48"
AerospikeSegmentSetTTL: "48"
AerospikeSmsBin: sms
AerospikeTimeout: "100000"
AerospikeTriggerApiBin: ""
AerospikeWebPushBin: wp
AerospikeWhatsAppBin: wsap
AppPushDelayedTriggerTopic: dmp_apn_tr
AppPushImmediateTriggerTopic: dmp_apn_tr
CeleryParamIgnore: IGNORE_IF_PRESENT
CeleryParamOverride: OVERRIDE
CeleryParamTrigger: TRIGGER
CeleryTimeThresholdMinutes: "30"
CeleryTriggerTopic: CeleryETALem
EXECUTOR_SERVICE_MAX_THREADS: "4"
EXECUTOR_SERVICE_MIN_THREADS: "2"
EXECUTOR_SERVICE_QUEUE_SIZE: "100"
EXECUTOR_SERVICE_THREAD_TTL: "5"
EmailDelayedTriggerTopic: dmp_email_tr
EmailImmediateTriggerTopic: dmp_email_tr
InboundKafkaTopic: trigger_inbound
KAFKA_CONSUMER_GROUP_ID: -trigger
KAFKA_POLL_INTERVAL: "200"
KAFKA_PRODUCER_EVENT_TYPE: "5656"
KAFKA_PRODUCER_META: "true"
KAFKA_PRODUCER_POOL_SIZE: "10"
KAFKA_TOPIC_CONSUMER_BATCH_SIZE: 10,10,10
KAFKA_TOPIC_CONSUMER_DEFAULT_BATCH_TIMEOUT_MS: 1000,1000,1000
KAFKA_TOPIC_CONSUMER_DEFAULT_BATCH_TIMEOUT_TRIGGER: 5000,5000,5000
KAFKA_TOPIC_CONSUMERS: 6,6,6
KAFKA_TOPIC_LIST: dmp_rt_segments,ETANotify,dmp_journey_trigger
KafkaCeleryNotify: ETANotify
KafkaJourneyTopic: dmp_journey_trigger
KafkaRTSegment: dmp_rt_segments
KafkaRTSegmentOffline: dmp_rt_segments_offline
LOG_LEVEL_APP: debug
LOG_LEVEL_METRICS: info
LOG_LEVEL_ROOT: info
LOG_LEVEL_UTILS: info
MAX_BATCH_EXECUTE_RETRIES: "3"
MAX_EVENT_BATCH_LAG: "8"
MysqlMapRefreshIntervalMinute: "1"
OPEN_TSDB_HOST: hxcb-cdp-opentsdb-service.cdp-dev-app.svc.cluster.local
OPEN_TSDB_HOST_TO_BE_ADDED: "true"
OPEN_TSDB_PORT: "80"
ReportingIntervalInMin: "55"
RtsSegOfflineRefreshDsName: rtsegofflinerefresh
```

```
RtsSegOnlineRefreshDsName: rtsegonlinerefresh
RtsSegRefreshDsName: rtsegrefresh
SmsDelayedTriggerTopic: dmp_sms_tr
SmsImmediateTriggerTopic: dmp_sms_tr
TriggerApiDelayedTriggerTopic: dmp_trapi_tr
TriggerApiImmediateTriggerTopic: dmp_trapi_tr
UserFailureKafkaTopic: trigger_failure
WebPushDelayedTriggerTopic: dmp_wpn_tr
WebPushImmediateTriggerTopic: dmp_wpn_tr
WhatsAppDelayedTriggerTopic: dmp_wsap_tr
WhatsAppImmediateTriggerTopic: dmp_wsap_tr
fromEmailId: ""
alertToEmailId: ""
hostname: trigger-prod-mu
emailType: SES
```



4. On successful deployment, validate the deployment as shown below.

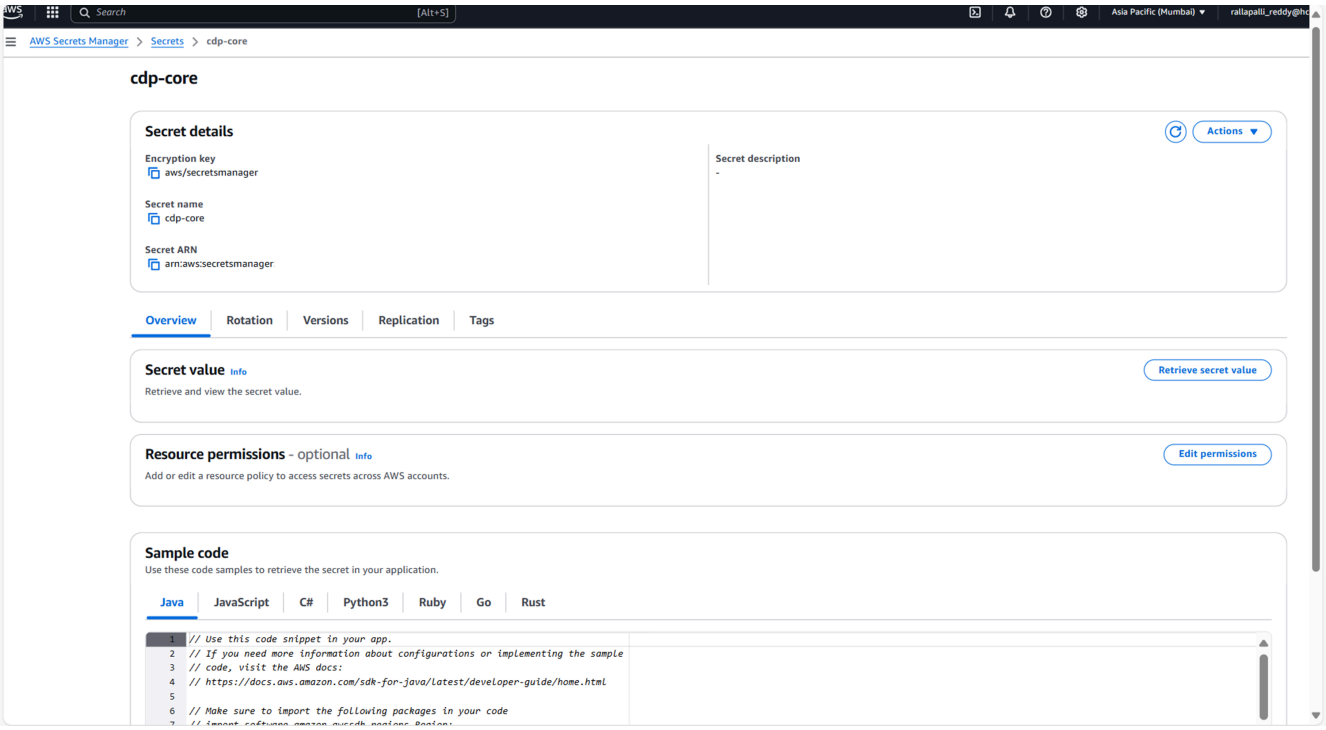


Deploying HCL CDP Core

This section provides detailed instructions on how to deploy HCL CDP Core using the Devtron in the AWS.

Prerequisites

- Make sure to create cdp-core secret in the AWS Secret Manager before deploying HCL CDP CDP Core.



To create CDP Core secret in the AWS secret manager, follow the steps below:

1. Create a cdp-core secret sample key and value, and update the actual values as shown below.

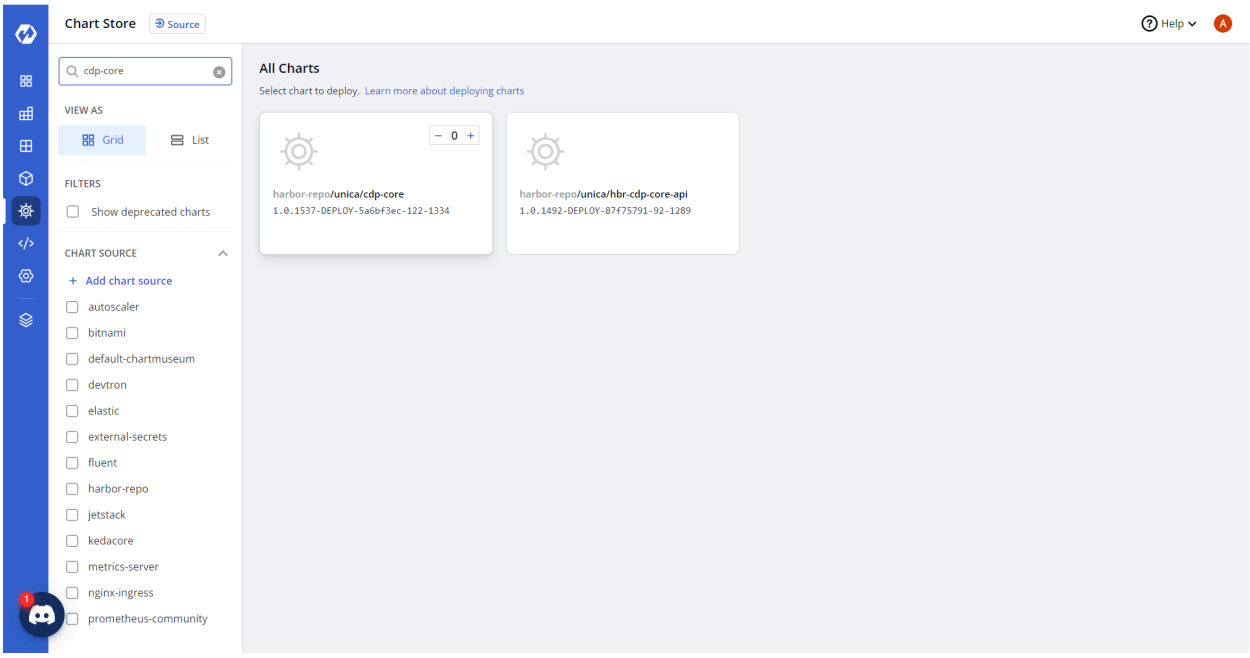
```
{
  "database": {
    "IC_AEROSPIKE_SERVER_ADDRESS": "",
    "NBA_AEROSPIKE_SERVER_ADDRESS": "",
    "VRM_DB_USERNAME": "",
    "VRM_DB_PASSWORD": "",
    "MY_SQL_HOST": "",
    "MY_SQL_PORT": "",
    "MY_SQL_DB": "vrm?autoReconnect=true",
    "TSDB_HOST": "",
    "TSDB_PORT": "80"
  },
  "kafka": {
    "KafkaBootstrapServerUSE1": "",
    "KafkaBootstrapServerAPS1": "",
    "KafkaProducerHostName": "trigger-core-us",
    "KafkaProducerRegion": "",
    "KafkaConsumerConfig": "",
    "KafkaProducerConfig": "",
    "KafkaProducerConfigaps1": "",
    "KAFKA_CONSUMER_GROUP": "k8s-trigger-core-prod",
    "KAFKA_TOPIC_LIST":
 "dmp_sms_tr,dmp_email_tr,dmp_wsap_tr,dmp_wpn_tr,dmp_apn_tr,dmp_email_sch,dmp_trapi_tr",
    "KAFKA_TOPIC_CONSUMERS": "6,6,6,6,6,6,6",
    "KAFKA_BATCH_SIZE": "10,10,10,10,10,10,10",
    "BATCH_TIMEOUT_MS": "1000,1000,1000,1000,1000,1000,1000",
    "BATCH_TIMEOUT_TRIGGER": "5000,5000,5000,5000,5000,5000,5000",
    "MAX_EVENT_BATCH_LAG": "32"
```

```
  },
  "lambda_functions": {
    "APN_LAMBDA_FUNCTIONNAME": "apppush-sender-prod",
    "WPN_LAMBDA_FUNCTIONNAME": "web-push-sender"
  },
  "features": {
    "USE_RTS_MS": "false",
    "allowTrafficForSpecificCampaign": "false",
    "ApnKafkaEnabled": "false",
    "SesEnabled": "true",
    "WpnKafkaEnabled": "true"
  },
  "topics": {
    "ApnKafkaTopic": "app_push_core",
    "WpnKafkaTopic": "web_push_core"
  }
}
```

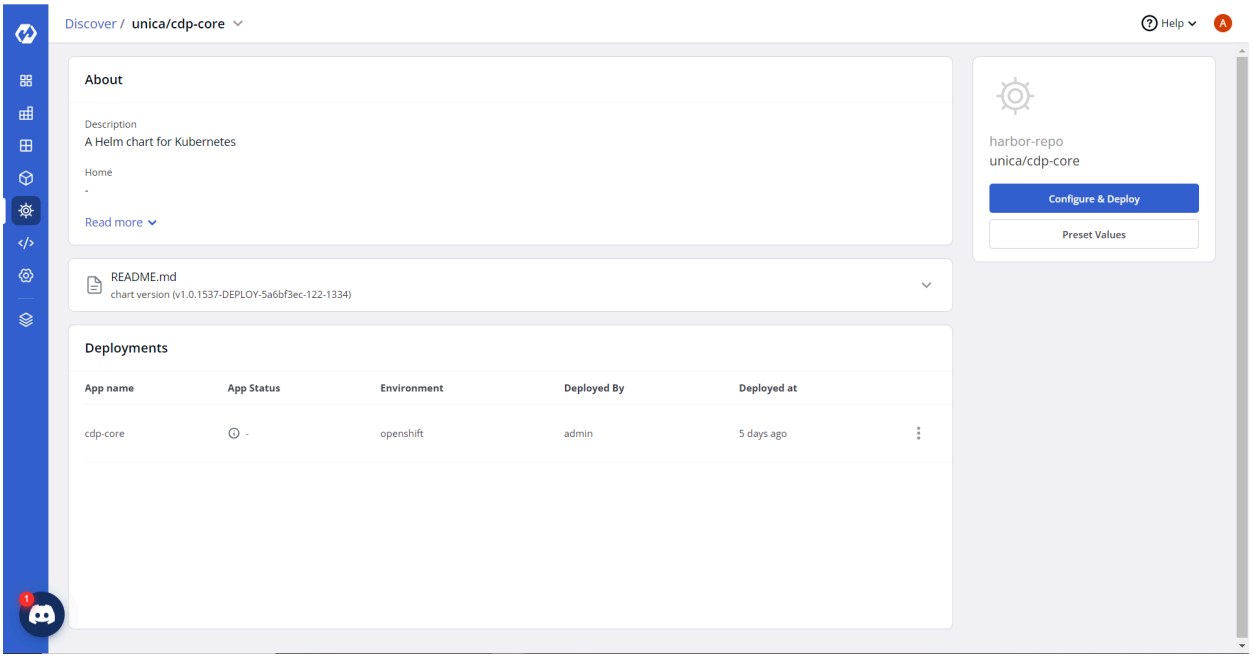
2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

Deploy CDP Core

1. Navigate to the Devtron **Chart Store**, and select cdp-core chart to deploy.



2. Now, configure and deploy the cdp-core charts.

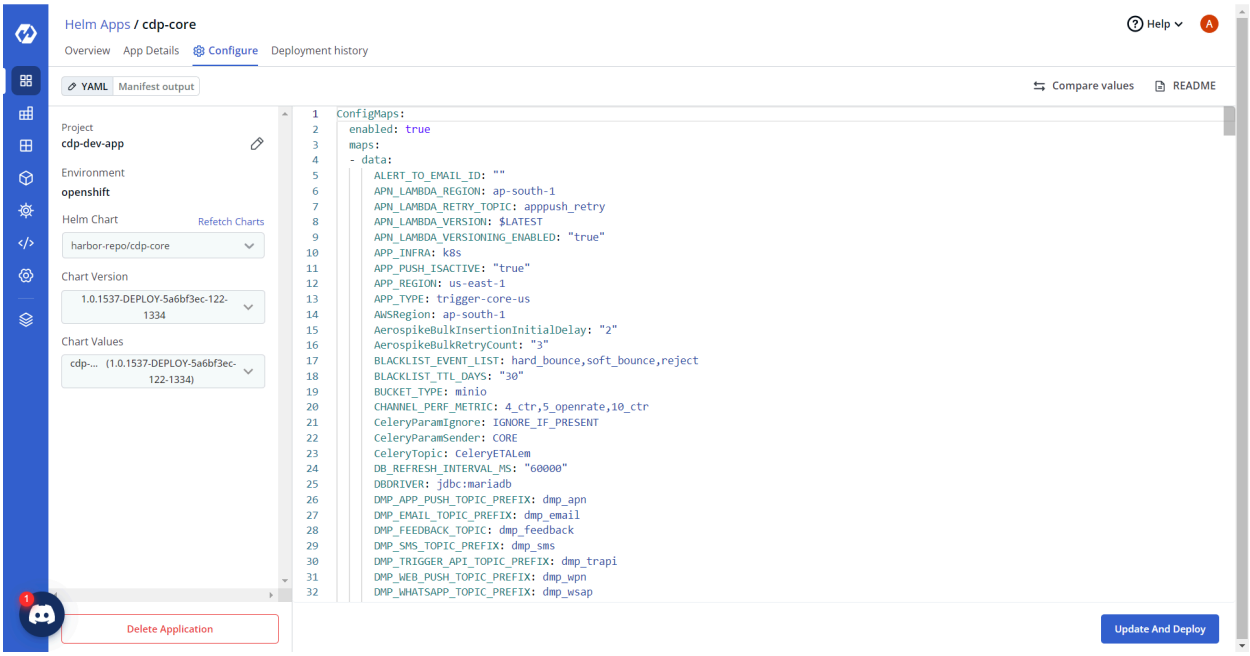


3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

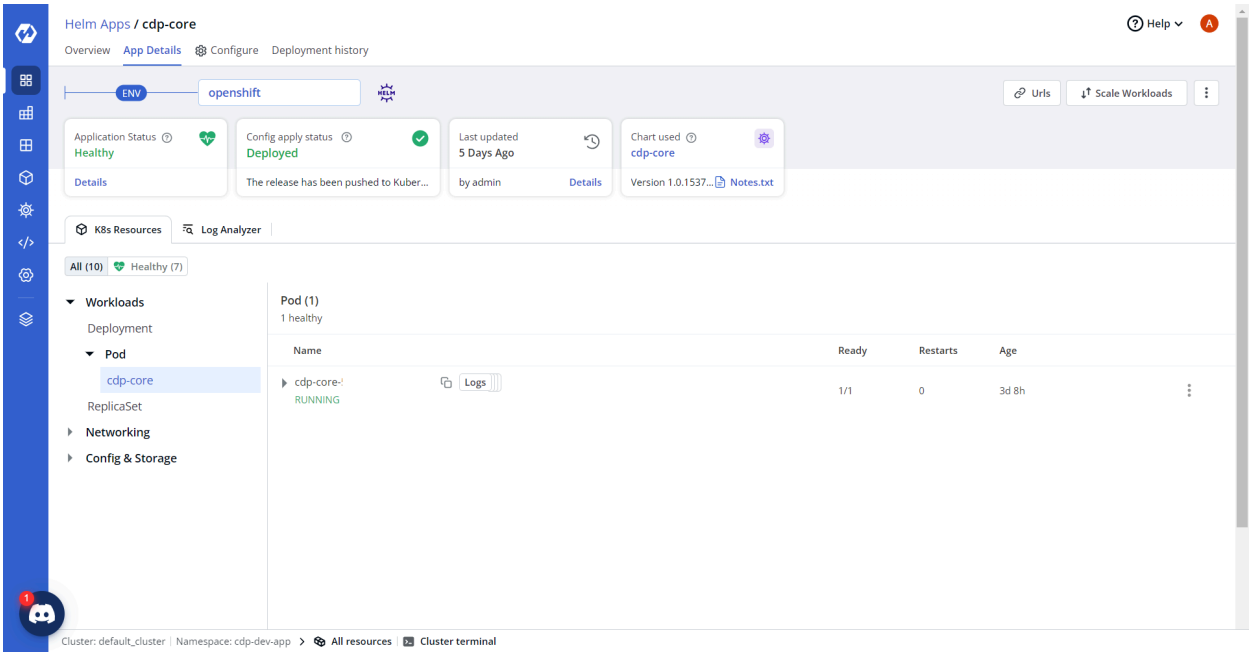
```
ALERT_TO_EMAIL_ID: ""
APN_LAMBDA_REGION: ap-south-1
APN_LAMBDA_RETRY_TOPIC: apppush_retry
APN_LAMBDA_VERSION: $LATEST
APN_LAMBDA_VERSIONING_ENABLED: "true"
APP_INFRA: k8s
APP_PUSH_ISACTIVE: "true"
APP_REGION: us-east-1
APP_TYPE: trigger-core-us
BLACKLIST_EVENT_LIST: hard_bounce,soft_bounce,reject
BLACKLIST_TTL_DAYS: "30"
CHANNEL_PERF_METRIC: 4_ctr,5_openrate,10_ctr
DB_REFRESH_INTERVAL_MS: "60000"
DMP_APP_PUSH_TOPIC_PREFIX: dmp_apn
DMP_EMAIL_TOPIC_PREFIX: dmp_email
DMP_FEEDBACK_TOPIC: dmp_feedback
DMP_SMS_TOPIC_PREFIX: dmp_sms
DMP_TRIGGER_API_TOPIC_PREFIX: dmp_trapi
DMP_WEB_PUSH_TOPIC_PREFIX: dmp_wpn
DMP_WHATSAPP_TOPIC_PREFIX: dmp_wsap
EMAIL_ISACTIVE: "true"
EMAIL_SENDER_KAFKA_TOPIC: dmp_email_core
ENCRYPTION_ENABLED: "false"
emailType: SES
FROM_EMAIL_ID: ""
HTTPS_CONNECTION_COUNT: "8"
IC_AEROSPIKE_NAMESPACE: cdpstore
IC_AEROSPIKE_PROFILE_BIN: ic
IC_AEROSPIKE_SERVER_PORT: "13000"
IC_AEROSPIKE_TIMEOUT: "100000"
IC_AEROSPIKE_TTL: "31536000"
INBOUND_KAFKA_TOPIC: core_inbound
```

```
KAFKA_CONSUMER_PREFIX: core
KAFKA_EXECUTOR_SERVICE_MAX_THREADS: "64"
KAFKA_EXECUTOR_SERVICE_MIN_THREADS: "32"
KAFKA_EXECUTOR_SERVICE_QUEUSE_SIZE: "1000"
KAFKA_EXECUTOR_SERVICE_THREAD_TTL: "5"
KAFKA_PRODUCER_META: "true"
KAFKA_PUSH_SENT_TOPIC: webpush_sent_metrics
KAFKA_SENDER: tctrcore
KEEP_ALIVE_SECONDS: "600"
KMS_DECRYPT_TIMEOUT: "100"
LAMBDA_MAX_PAYLOAD_SIZE: "128000"
LAMBDA_RETRY_COUNT: "3"
LAMBDA_RETRY_INTERVAL: "1000"
LOG_LEVEL_APP: debug
LOG_LEVEL_METRICS: off
LOG_LEVEL_ROOT: debug
LOG_LEVEL_UTILS: off
MAPPING_BIN: mp
MAXIMUM_IDLE_THREADS: "6"
MINIMUM_IDLE_THREADS: "3"
NBA_AEROPIKE_TIMEOUT: "100000"
NBA_AEROSPIKE_BIN: nba
NBA_AEROSPIKE_NAMESPACE: cdpstore
NBA_AEROSPIKE_SERVER_PORT: "13000"
OPEN_TRACKER: https://ck.dev.hxcb.now.hclsoftware.cloud/emailOpen?
OUTBOUND_KAFKA_TOPIC: core_outbound
PROCESSING_RETRY_COUNT: "3"
PRODUCER_EVENT_TYPE: "5656"
PRODUCER_POOL_SIZE: "16"
PUSH_ISACTIVE: "true"
PUSH_SERVER_ENDPOINT: ""
REPORT_SCHEDULER_ISACTIVE: "true"
REPORTING_INTERVAL_IN_MIN: "55"
RTS_MS_ENDPOINT: http://rts-ms.dev.hxcb.now.hclsoftware.cloud/usermacros
S3_ENDPOINT: s3.amazonaws.com
S3_MAX_CONNECTIONS: "10"
S3_MAX_ERROR_RETRY: "3"
S3_MAX_RW_RETRY: "3"
S3_PROTOCOL: HTTPS
S3_REGION: ap-south-1
S3_TIMEOUT_CLIENT_EXECUTION: "50000"
S3_TIMEOUT_CONNECTION: "50000"
S3_TIMEOUT_REQUEST: "10000"
S3_TIMEOUT_SOCKET: "10000"
SHORTEN_OPEN_TRACKER: https://ck.dev.hxcb.now.hclsoftware.cloud/eos
SMS_ISACTIVE: "true"
SMS_SENDER_KAFKA_TOPIC: dmp_sms_core
TAXIMAIL_EMAIL_SENDER_KAFKA_TOPIC: dmp_taximail_email_core
TRIGGER_API_ISACTIVE: "true"
TRIGGER_API_SENDER_KAFKA_TOPIC: dmp_trapi_core
URL_SHORTENER_MULTIPLY_FACTOR: "10000"
URL_SHORTNER_AVOID_CONFLICT: "true"
URL_SHORTNER_CONFLICT_MAX_RETRIES: "3"
URL_SHORTNER_S3_BUCKET: lem-email-url-shortener
USER_FAILURE_KAFKA_TOPIC: core_failure
VRM_DB_DRIVER: org.mariadb.jdbc.Driver
DBDRIVER: jdbc:mariadb
WEB_PUSH_ISACTIVE: "true"
```

```
WHATSAPP_ISACTIVE: "true"
WHATSAPP_SENDER_KAFKA_TOPIC: dmp_wsap_core
WPN_LAMBDA_REGION: ap-south-1
WPN_LAMBDA_RETRY_TOPIC: webpush_retry
WPN_LAMBDA_VERSION: $LATEST
WPN_LAMBDA_VERSIONING_ENABLED: "true"
WPN_CUSTOM_LAMBDA_CAMPAIGNLIST: "0"
defaultBusinessRegion: aps1
ProducerEventType: "5650"
CeleryParamSender: CORE
CeleryParamIgnore: IGNORE_IF_PRESENT
CeleryTopic: CeleryETALem
AerospikeBulkRetryCount: "3"
AerospikeBulkInsertionInitialDelay: "2"
SubscriptionEventInsertionTimeout: "3000"
SubscriptionEventInsertionRetry: "3"
SubscriptionEventTTL: "-1"
```



4. On successful deployment, validate the deployment as shown below.

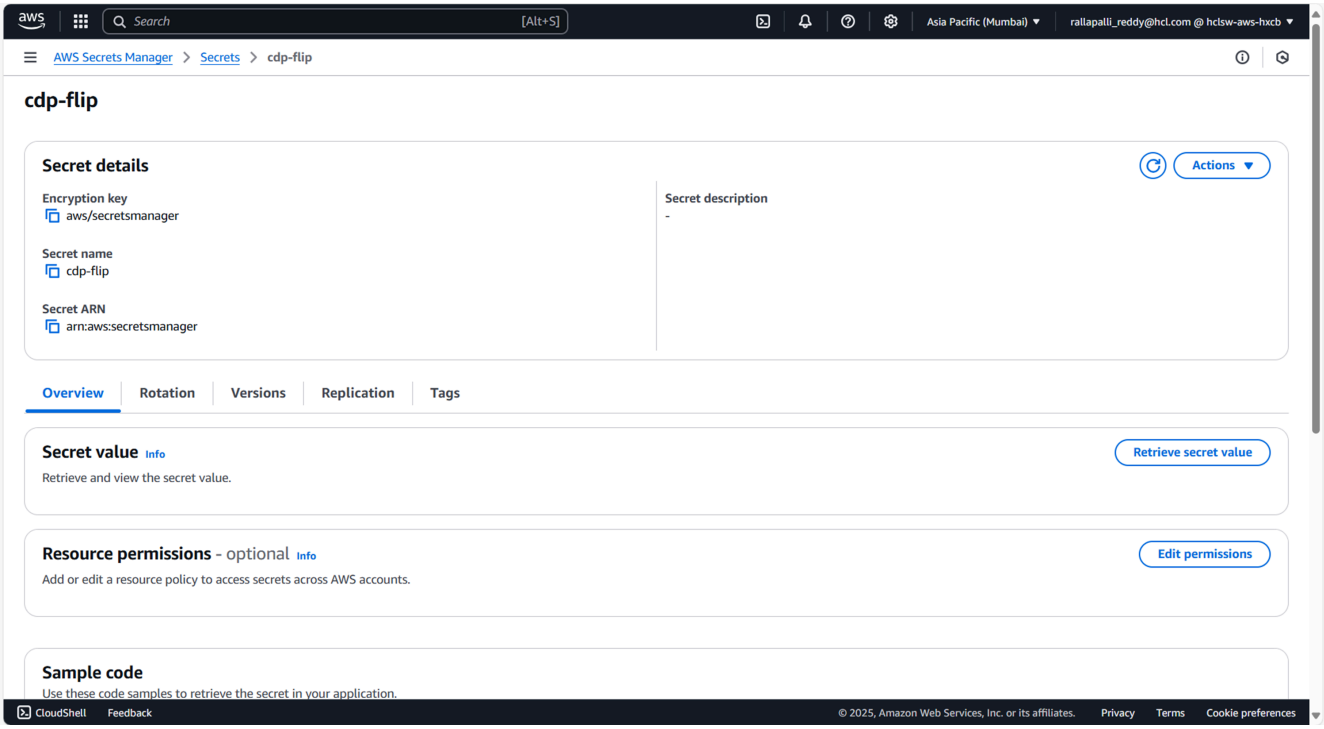


Deploying CDP Flip

This section provides detailed instructions on how to deploy CDP Flip using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-flip secret in the AWS Secret Manager before deploying CDP Flip.



To create CDP Flip secret in the AWS secret Manager, follow the steps below:

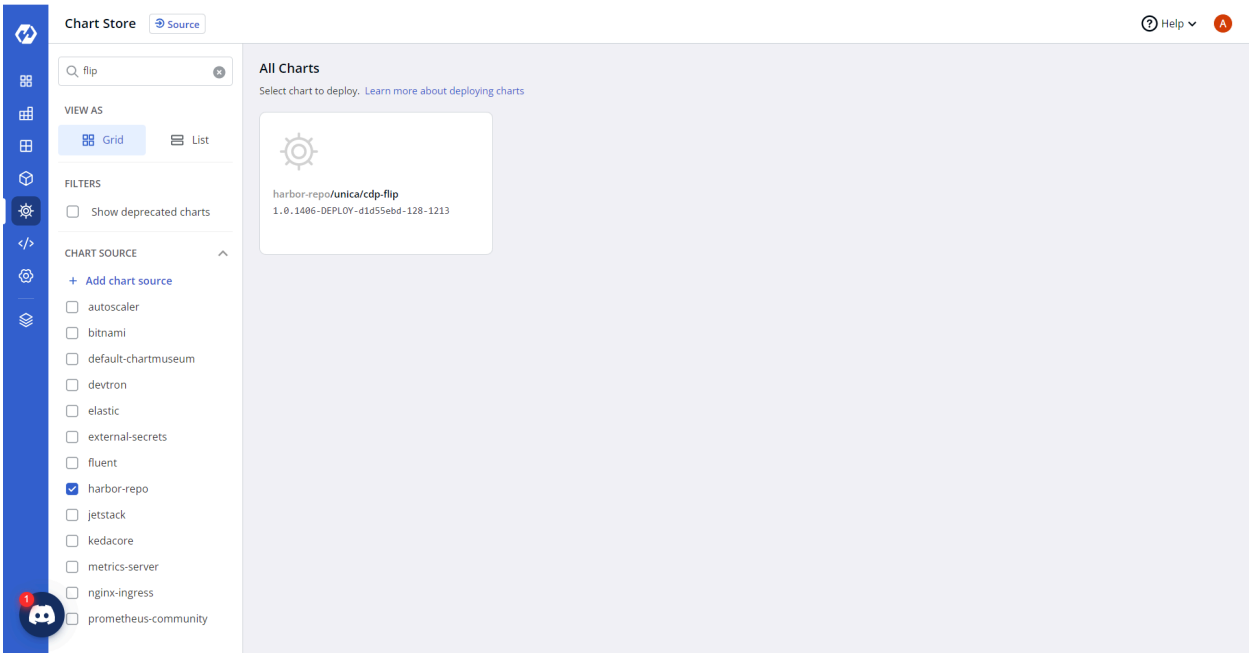
1. Create a cdp-flip secret sample key and value, and update the actual values as shown below.

```
{
  "CUSTOMIZATION_AERO_CLUSTER": "",
  "DB_USERNAME": "",
  "DB_PASSWORD": "",
  "DB_URL": "",
  "KafkaConsumerConfig": "",
  "KAFKA_CONSUMER_PREFIX": "flip",
  "KAFKA_TOPIC_LIST": "dmp_rt_segments"
}
```

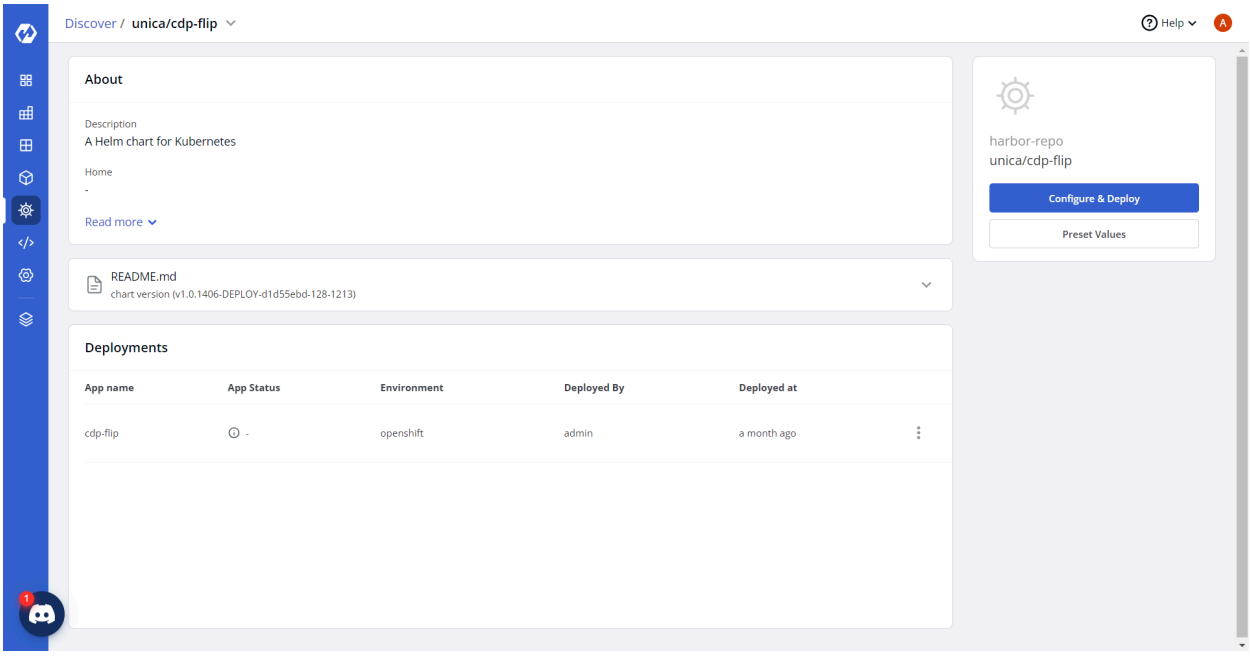
2. Create separate service account cdp-dev-app-sa with appropriate roles and permissions, and update ConfigMaps data with actual values.
3. Create required Kafka topics before deployment.

Deploy CDP Flip

1. Navigate to the Devtron **Chart Store**, and select the cdp-flip chart to deploy.



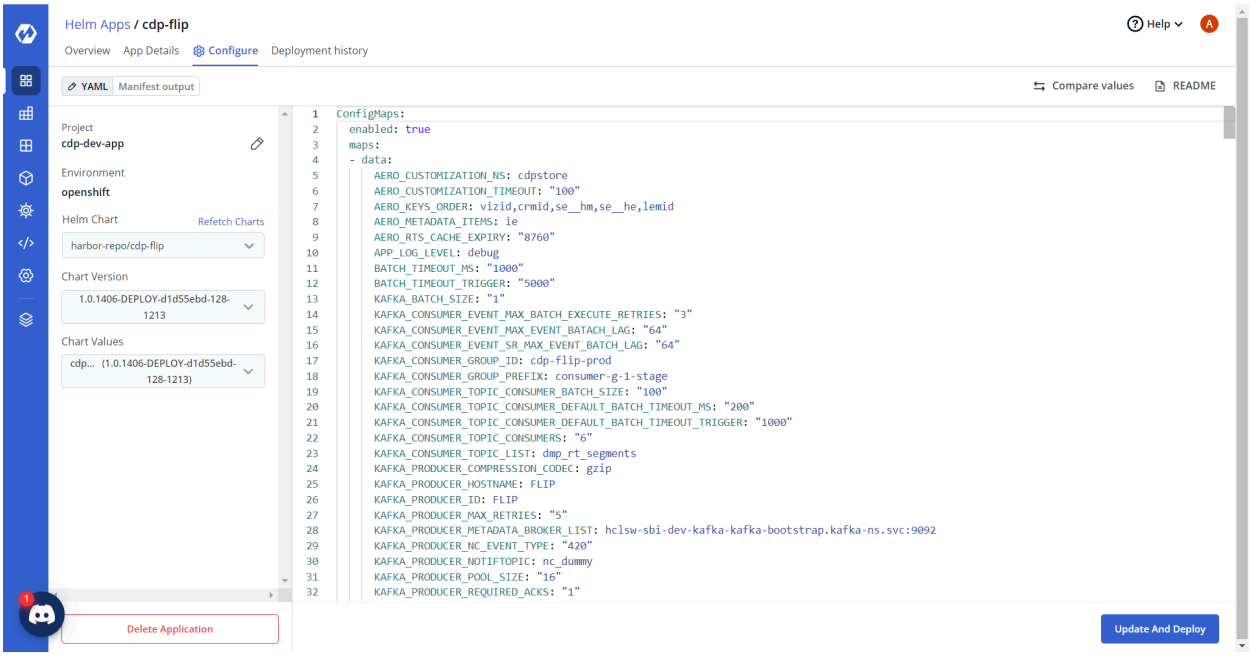
2. Now, configure and deploy the cdp-flip charts.



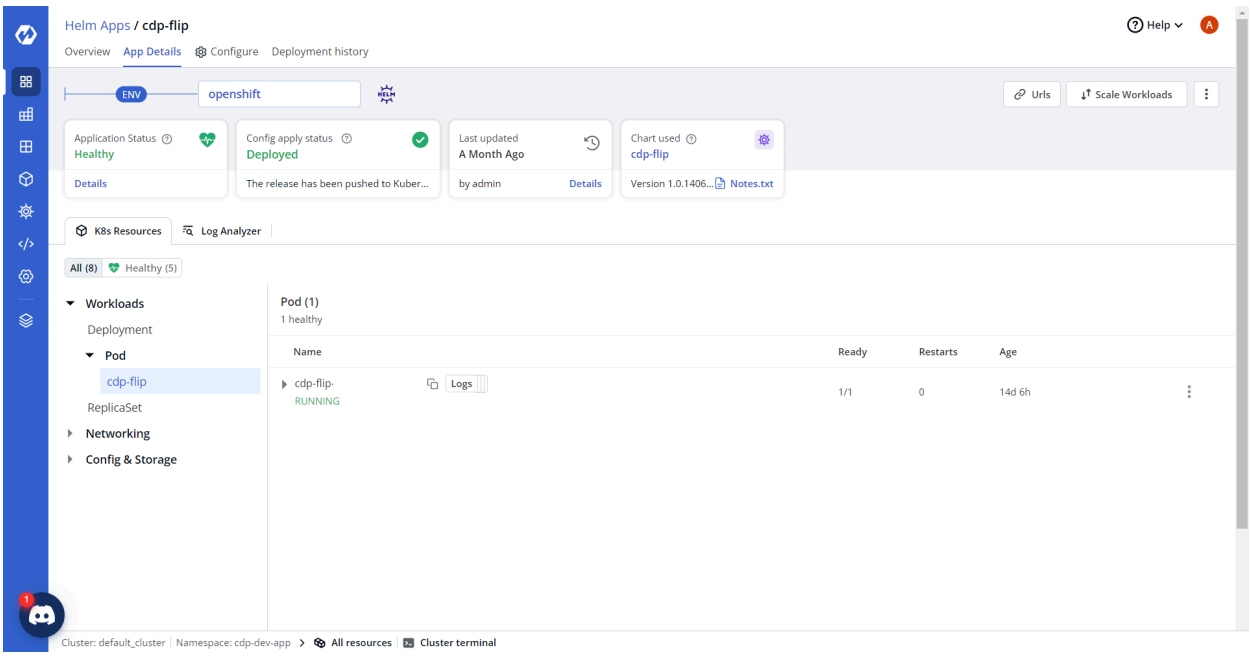
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
AERO_CUSTOMIZATION_NS: cdpstore
AERO_CUSTOMIZATION_TIMEOUT: "100"
AERO_RTS_CACHE_EXPIRY: "8760"
```

```
MYSQL_DBDRIVER: org.mariadb.jdbc.Driver
MYSQL_REFRESH_DBREFRESHTIMEMIN: "30"
MT_PROPERTIES_FLIP_EXECUTOR_SERVICE_MIN_THREADS: "64"
MT_PROPERTIES_FLIP_EXECUTOR_SERVICE_MAX_THREADS: "64"
MT_PROPERTIES_FLIP_EXECUTOR_SERVICE_THREAD_TTL: "5"
MT_PROPERTIES_FLIP_EXECUTOR_SERVICE_QUEUSE_SIZE: "10000000"
KAFKA_CONSUMER_GROUP_ID: cdp-flip-prod
KAFKA_CONSUMER_GROUP_PREFIX: consumer-g-1-stage
KAFKA_REBALANCE_MAX_RETRIES: "60"
KAFKA_CONSUMER_TOPIC_LIST: dmp_rt_segments
KAFKA_CONSUMER_TOPIC_CONSUMERS: "6"
KAFKA_CONSUMER_TOPIC_CONSUMER_BATCH_SIZE: "100"
KAFKA_CONSUMER_TOPIC_CONSUMER_DEFAULT_BATCH_TIMEOUT_MS: "200"
KAFKA_CONSUMER_TOPIC_CONSUMER_DEFAULT_BATCH_TIMEOUT_TRIGGER: "1000"
KAFKA_CONSUMER_EVENT_MAX_EVENT_BATACH_LAG: "64"
KAFKA_CONSUMER_EVENT_SR_MAX_EVENT_BATCH_LAG: "64"
KAFKA_CONSUMER_EVENT_MAX_BATCH_EXECUTE_RETRIES: "3"
KAFKA_PRODUCER_METADATA_BROKER_LIST: <kafka borkers>
KAFKA_PRODUCER_SERIALIZER_CLASS: kafka.serializer.StringEncoder
KAFKA_PRODUCER_ID: FLIP
KAFKA_PRODUCER_REQUIRED_ACKS: "1"
KAFKA_PRODUCER_REQUIRED_TIMEOUT: "10000"
KAFKA_PRODUCER_TYPE: async
KAFKA_PRODUCER_COMPRESSION_CODEC: gzip
KAFKA_PRODUCER_MAX_RETRIES: "5"
KAFKA_PRODUCER_RETRY_BACKOFF: "1000"
KAFKA_PRODUCER_NC_EVENT_TYPE: "420"
KAFKA_PRODUCER_HOSTNAME: FLIP
KAFKA_PRODUCER_POOL_SIZE: "16"
KAFKA_PRODUCER_NOTIFTOPIC: nc_dummy
LOG_TSDB_METRIC_LOGGING_FREQUENCY_MS: "60000"
LOG_TSDB_MAX_WAIT_FOR_CONNECTION_REESTABLISH: "5"
LOG_TSDB_SERVER_ADDRESS: hxcb-cdp-opentsdb-service.cdp-app.svc.cluster.local
LOG_TSDB_SERVER_PORT: "80"
LOG_TSDB_HOSTNAME: hxcb-cdp-opentsdb-service.cdp-app.svc.cluster.local
LOG_GANGLIA_GMOND_COLLECTORS: ""
LOG_GANGLIA_GMOND_PORT: "8650"
LOG_GANGLIA_HOSTNAME_OVERRIDE: cdpstore
LOG_GANGLIA_METRIC_PREFIX: dmpflip
AERO_KEYS_ORDER: vizid,crmid,se__hm,se__he,lemid
AERO_METADATA_ITEMS: ie
APP_LOG_LEVEL: debug
KAFKA_BATCH_SIZE: "1"
MAX_EVENT_BATCH_LAG: "32"
BATCH_TIMEOUT_MS: "1000"
BATCH_TIMEOUT_TRIGGER: "5000"
KAFKA_TOPIC_CONSUMERS: "6"
```



4. On successful deployment, validate the deployment as shown below.

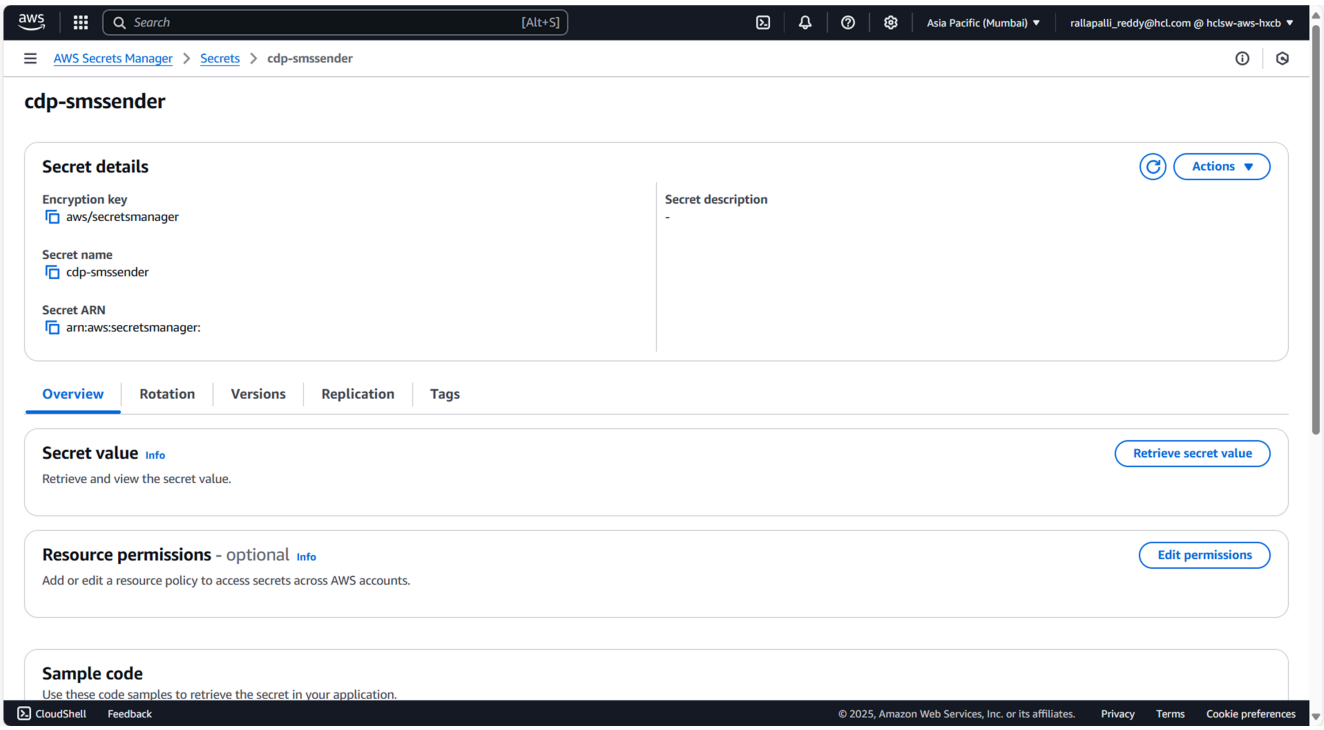


Deploying HCL CDP SMSEmailSender

This section provides detailed instructions on how to deploy HCL CDP SMS Email Sender using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-smsemailsender secret in the AWS secret manager before deploying CDP SMS Email Sender.



To create CDP SMS Email Sender secret in the AWS secret manager, follow the steps below:

1. Create a cdp-smsemailsender secret sample key and value, and update the actual values as shown below.

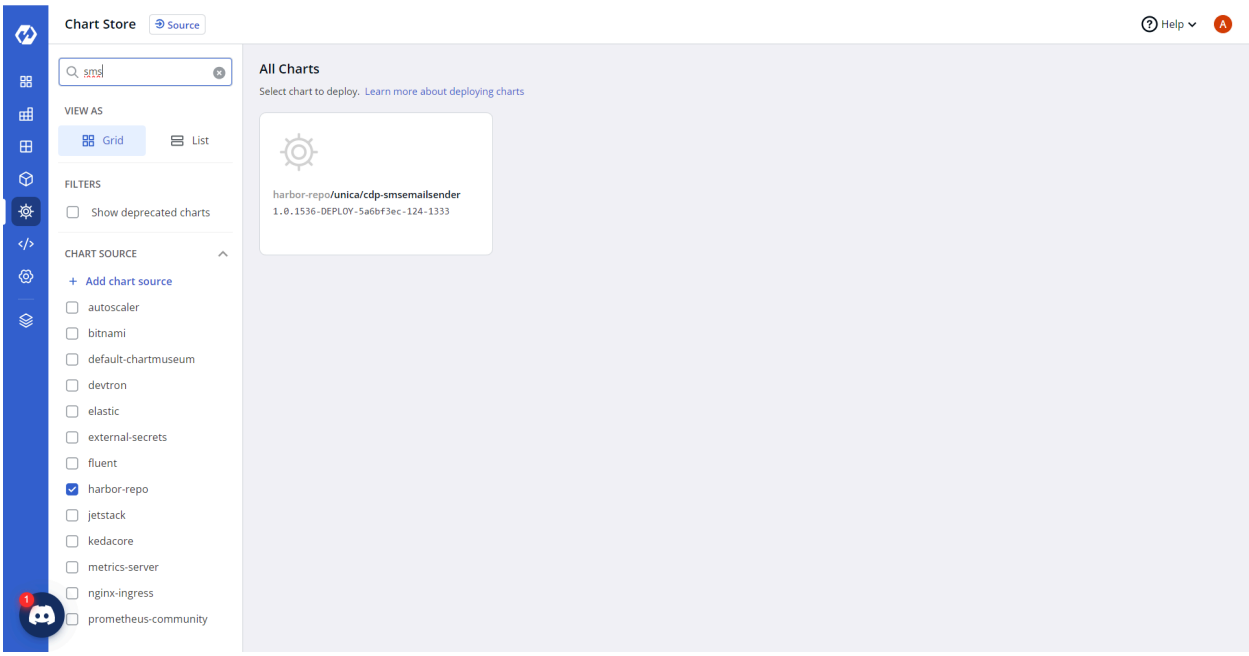
```
{
  "KafkaBootstrapServer": "",
  "ADCB_DEVMAG_CERTIFICATE_PASSWORD": "",
  "ADCB_PRODMAG_CERTIFICATE_PASSWORD": "",
  "s3.accessKeyId": "",
  "s3.secretKey": "",
  "s3.region": "ap-south-1",
  "RedisIP": "",
  "RedisPort": "6379",
  "MysqlHost": "",
  "MysqlPort": "3307",
  "MysqlDb": "vrm?autoReconnect=true ",
  "5979_ACL_USER_ID": "test",
  "5979_ACL_PASS": "test",
  "5979_ACL_APP_ID": "test",
  "5979_ACL_SUB_APP_ID": "test",
  "BLOW_FISH_KEY": "test",
  "MGAGE_XML_API": "test",
  "KENSCIO_CLIENT_END_POINT_URL": "",
  "KENSCIO_USERNAME": "",
  "KENSCIO_PASSWORD": "",
  "KENSCIO_SYSTEM_ID_KEY": "",
  "KENSCIO_SYSTEM_ID": "",
  "KENSCIO_MESSAGE_ID_KEY": "",
  "KENSCIO_MESSAGE_ID": "",
  "AerospikeCluster": "",
  "AerospikePort": "13000",
  "Namespace": "cdpstore",
```

```
  "Expiry": "-1",
  "Timeout": "3000",
  "USAerospikeCluster": "",
  "USAerospikePort": "13000",
  "USNamespace": "",
  "MUAerospikeCluster": "10.214.109.162,10.214.111.148",
  "MUAerospikePort": "13000",
  "MUNamespace": "cdpstore",
  "SupportedRegion": "aps1",
  "ExcludedCampaigns": "",
  "KAFKA_TOPIC_LIST": "dmp_sms_core_aps1,dmp_email_core_aps1,dmp_wsap_core_aps1,dmp_trapi_core_aps1",
  "KAFKA_TOPIC_CONSUMERS": "6,6,6,6",
  "KAFKA_TOPIC_CONSUMER_BATCHSIZE": "10,10,10,10",
  "KAFKA_TOPIC_CONSUMER_DEFAULT_BATCH_TIMEOUT_MS": "1000,1000,1000,1000",
  "KAFKA_TOPIC_CONSUMER_DEFAULT_BATCHTIMEOUT_TRIGGER": "5000,5000,5000,5000",
  "TSDB_HOST": "hxcb-cdp-opentsdb-service.cdp-dev-app.svc.cluster.local",
  "allowTrafficForSpecificCampaign": "false",
  "IncludedCampaigns": "4658,6035,6238,6037,6184,6349,6450,6455,6525",
  "URL_SHORTENER_S3_BUCKET": "lem-url-shortener",
  "URL_SHORTENER_S3_BUCKET_EMAIL": "lem-email-url-shortener",
  "PII_MIN_IDLE_THREADS": "2",
  "PII_MAX_IDLE_THREADS": "4",
  "KafkaBootstrapServerUS": "",
  "KafkaUsernameUS": "",
  "KafkaPasswordUS": "",
  "SesEnabled": "true",
  "PIILookUpEnabled": "true",
  "KafkaConsumerConfig": "",
  "KafkaProducerConfig": "",
  "KafkaProducerRegion": "aps1",
  "KafkaProducerConfigaps1": ""
}
```

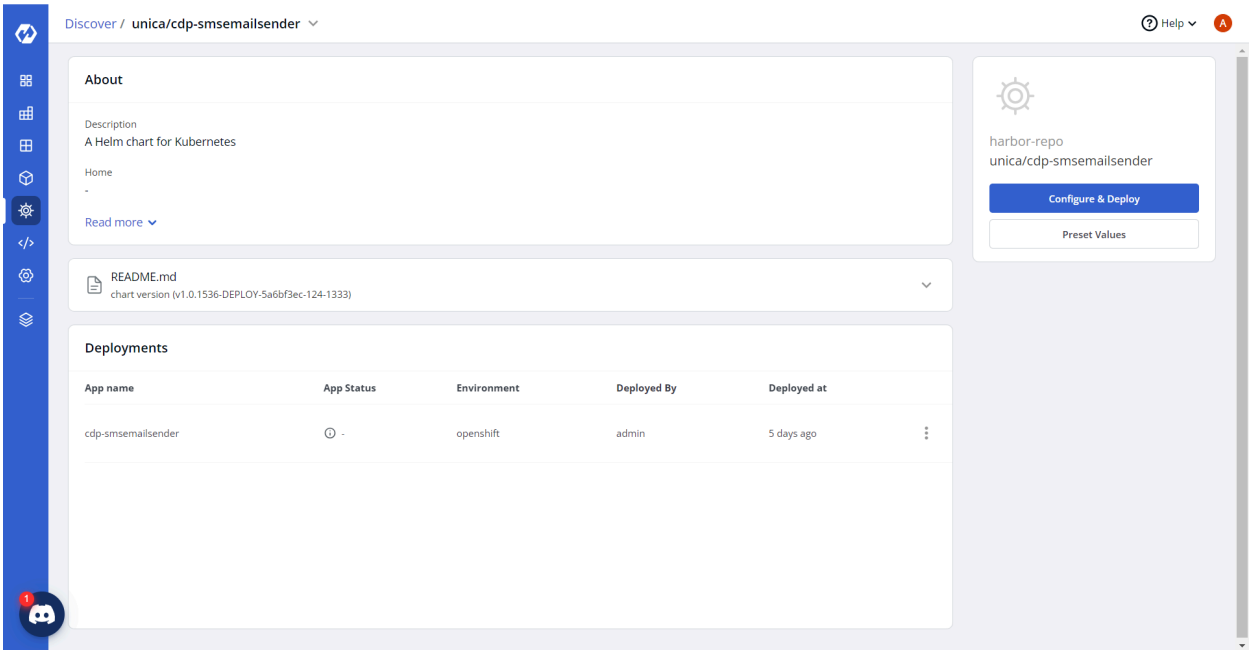
2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

Deploy CDP SMS Email Sender

1. Navigate to the Devtron **Chart Store**, and select the cdp-smsemailsender chart to deploy.



2. Now, configure and deploy the cdp-smsemailsender charts.



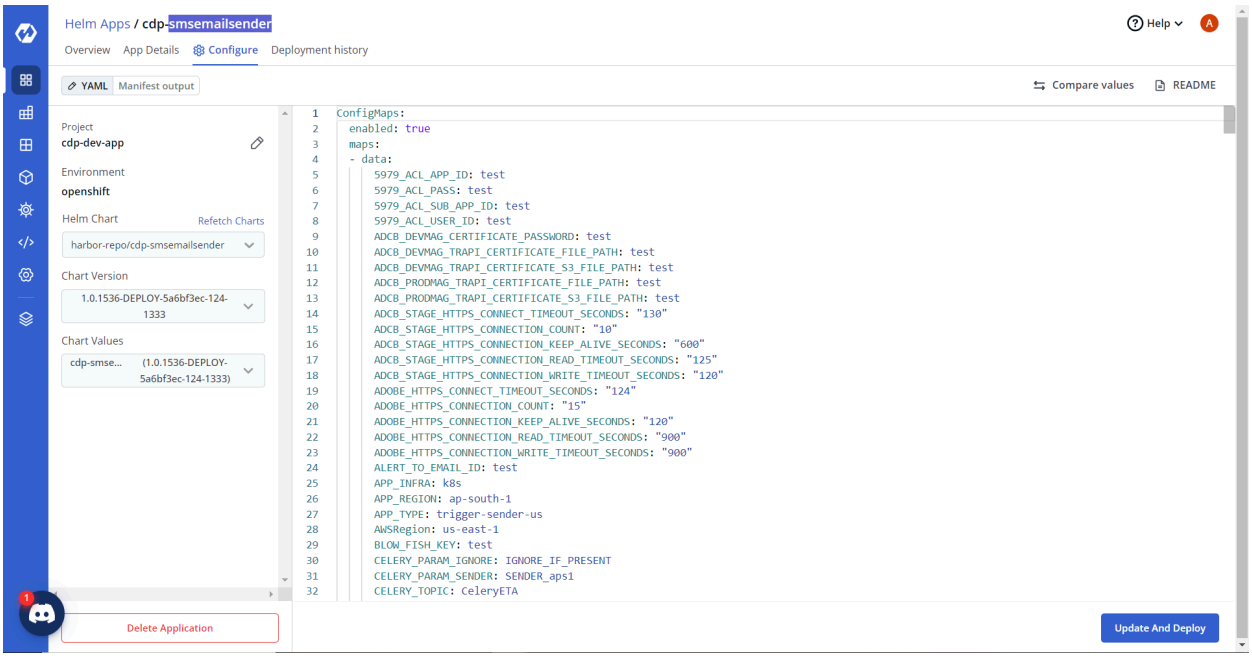
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
KAFKA_EXECUTOR_SERVICE_MIN_THREADS: "32"
KAFKA_EXECUTOR_SERVICE_MAX_THREADS: "64"
KAFKA_EXECUTOR_SERVICE_THREAD_TTL: "5"
```

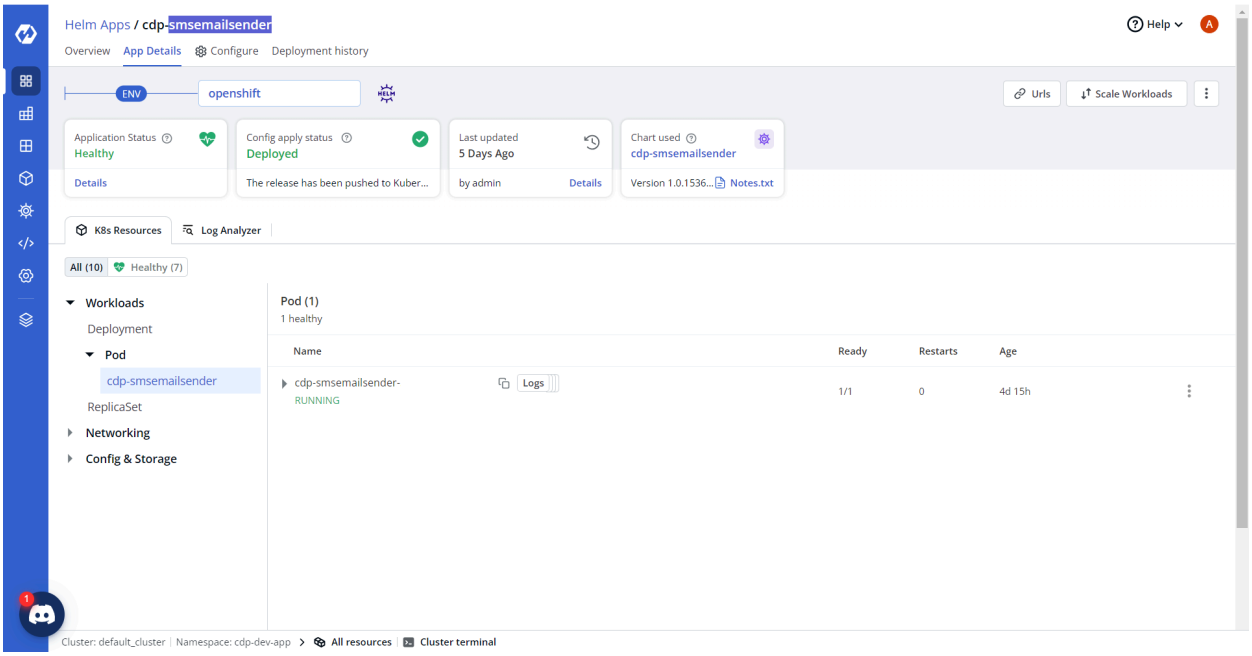
```
KAFKA_EXECUTOR_SERVICE_QUEUES_SIZE: "1000"
TRAPI_KAFKA_EXECUTOR_SERVICE_MIN_THREADS: "32"
TRAPI_KAFKA_EXECUTOR_SERVICE_MAX_THREADS: "64"
TRAPI_KAFKA_EXECUTOR_SERVICE_THREAD_TTL: "5"
TRAPI_KAFKA_EXECUTOR_SERVICE_QUEUES_SIZE: "1000"
MAX_EVENT_BATCH_LAG: "16"
KAFKA_CONSUMER_GROUP: k8s-trigger-sender
EMAIL_HTTPS_CONNECTION_COUNT: "10"
EMAIL_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "6000"
MANDRILL_CLIENT_END_POINT_URL: test
kMS_DECRYPT_TIMEOUT: "3"
FROM_EMAIL_ID: test
ALERT_TO_EMAIL_ID: test
SMS_HTTPS_CONNECTION_COUNT: "10"
SMS_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "6000"
REDIS_MAX_CONNECTIONS: "20"
REDIS_KEY_TTL_IN_SEC: "7776000"
TRAPI_HTTPS_CONNECTION_COUNT: "45"
TRAPI_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "4000"
TRAPI_HTTPS_CONNECTION_READ_TIMEOUT_SECONDS: "122"
TRAPI_HTTPS_CONNECTION_WRITE_TIMEOUT_SECONDS: "123"
TRAPI_HTTPS_CONNECT_TIMEOUT_SECONDS: "124"
CLICK_TRACKER: https://<url>.cloud/emailClick?
PASS_THROUGH_SUPPORTED_VENDOR: Phonon
DB_REFRESH_INTERVAL_MS: "120000"
TRAPI_CONFIG_REFRESH_INTERVAL: "600000"
MINIMUM_IDLE_THREADS: "3"
MAXIMUM_IDLE_THREADS: "6"
CLICK_TRACKER_SMS: https://<url>.cloud/smsClick?
OPEN_TRACKER: https://<url>.cloud/emailOpen?
TRAPI_TOPIC_PREFIX: dmp_trapi_core
S3_ENDPOINT: s3.amazonaws.com
S3_PROTOCOL: HTTPS
S3_MAX_ERROR_RETRY: "3"
S3_MAX_CONNECTIONS: "10"
S3_MAX_RW_RETRY: "3"
S3_TIMEOUT_SOCKET: "5000"
S3_TIMEOUT_CONNECTION: "10000"
S3_TIMEOUT_CLIENTEXECUTION: "50000"
S3_TIMEOUT_REQUEST: "5000"
S3_REQUEST_METADATA_FOLDER: trapi_req_metadata
URL_SHORTENER_DOMAIN: https://s.cdp.bz
URL_SHORTENER_DOMAIN_EMAIL: https://<url>/ck/ecs
URL_SHORTENER_AVOID_CONFLICT: "true"
URL_SHORTENER_CONFLICT_MAX_RETRIES: "3"
URL_SHORTENER_MULTIPLY_FACTOR: "10000"
SHORTEN_CLICK_TRACKER_EMAIL: https://<url>.cloud/ecs
KAFKA_PRODUCER_POOL_SIZE: "10"
KAFKA_PRODUCER_META: "true"
PRODUCER_EVENT_TYPE: "1234"
CELERY_PARAM_SENDER: SENDER_aps1
CELERY_PARAM_IGNORE: IGNORE_IF_PRESENT
CELERY_TOPIC: CeleryETA
KAFKA_PRODUCER_CHANNELS: email,sms,trapi,wasp
EMAIL_IMPRESSION_TOPIC: sms_email_events
SMS_IMPRESSION_TOPIC: sms_email_events
TRAPI_IMPRESSION_TOPIC: trapi_events
TRAPI_FEEDBACK_IMPRESSION_TOPIC: sms_email_events
```

```
WASP_IMPRESSION_TOPIC: whatsapp_events
USER_FAILURE_KAFKA_TOPIC: sender_failure
ADCB_DEVMAG_TRAPI_CERTIFICATE_FILE_PATH: test
ADCB_DEVMAG_TRAPI_CERTIFICATE_S3_FILE_PATH: test
ADCB_PRODMAG_TRAPI_CERTIFICATE_S3_FILE_PATH: test
ADCB_PRODMAG_TRAPI_CERTIFICATE_FILE_PATH: test
TRAPI_CERTIFICATE_ENDPOINT: test
TRAPI_CERTIFICATE_CAMPAIGN_SUFFIX: test
COMPONENT_NAME: trigger_sender_us
REPORTING_INTERVAL_IN_MIN: "55"
WHATSAPP_HTTPS_CONNECTION_COUNT: "120"
WHATSAPP_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "10"
METRIC_FILE_PATH: /disk1/config/sms-email-sender/metrics.ini
METRIC_DUMPER_FILE_PATH: /disk1/config/sms-email-sender/metricDumper.ini
HTTPS_CONNECTION_COUNT: "10"
HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "60"
SERVER_PORT: "8085"
COMMON_HTTPS_CONNECTION_COUNT: "100"
COMMON_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "60"
ADCB_STAGE_HTTPS_CONNECTION_COUNT: "10"
ADCB_STAGE_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "600"
ADCB_STAGE_HTTPS_CONNECTION_READ_TIMEOUT_SECONDS: "125"
ADCB_STAGE_HTTPS_CONNECTION_WRITE_TIMEOUT_SECONDS: "120"
ADCB_STAGE_HTTPS_CONNECT_TIMEOUT_SECONDS: "130"
ADOBE_HTTPS_CONNECTION_COUNT: "15"
ADOBE_HTTPS_CONNECTION_KEEP_ALIVE_SECONDS: "120"
ADOBE_HTTPS_CONNECTION_READ_TIMEOUT_SECONDS: "900"
ADOBE_HTTPS_CONNECTION_WRITE_TIMEOUT_SECONDS: "900"
ADOBE_HTTPS_CONNECT_TIMEOUT_SECONDS: "124"
SENDER_FEEDBACK_TOPIC: dmp_sst_nba_di_api
LIVSPACE_IVR_LPEU_APID: "99"
LIVSPACE_IVR_BPEU_APID: "89"
APP_TYPE: trigger-sender-us
APP_REGION: ap-south-1
APP_INFRA: k8s
LOG_LEVEL_APP: debug
LOG_LEVEL_METRICS: info
LOG_LEVEL_UTILS: debug
LOG_LEVEL_PII_STORAGE: debug
LOG_LEVEL_KMS: debug
LOG_LEVEL_ROOT: info
S3_ACCESSKEYID: ""
S3_SECRETKEY: ""
S3_REGION: ap-south-1
defaultBusinessRegion: aps1
inboundKafkaTopic: sender_inbound
ergoLeadApi: ""
VRM_DRIVER: org.mariadb.jdbc.Driver
DBDRIVER: jdbc:mariadb
TSDB_PORT: "80"
KafkaProducerConfiguse1: test
KafkaProducerConfigtaw1: test
KENSCIO_USERNAME: test
KENSCIO_PASSWORD: test
KENSCIO_SYSTEM_ID_KEY: test
KENSCIO_SYSTEM_ID: test
KENSCIO_MESSAGE_ID_KEY: test
KENSCIO_MESSAGE_ID: test
```

```
ADCB_DEVMAG_CERTIFICATE_PASSWORD: test
5979_ACL_USER_ID: test
5979_ACL_PASS: test
5979_ACL_APP_ID: test
5979_ACL_SUB_APP_ID: test
BLOW_FISH_KEY: test
MGAGE_XML_API: test
emailType: SES
KMS_IMPLEMENTATION: AWS
```



4. On successful deployment, validate the deployment as shown below.

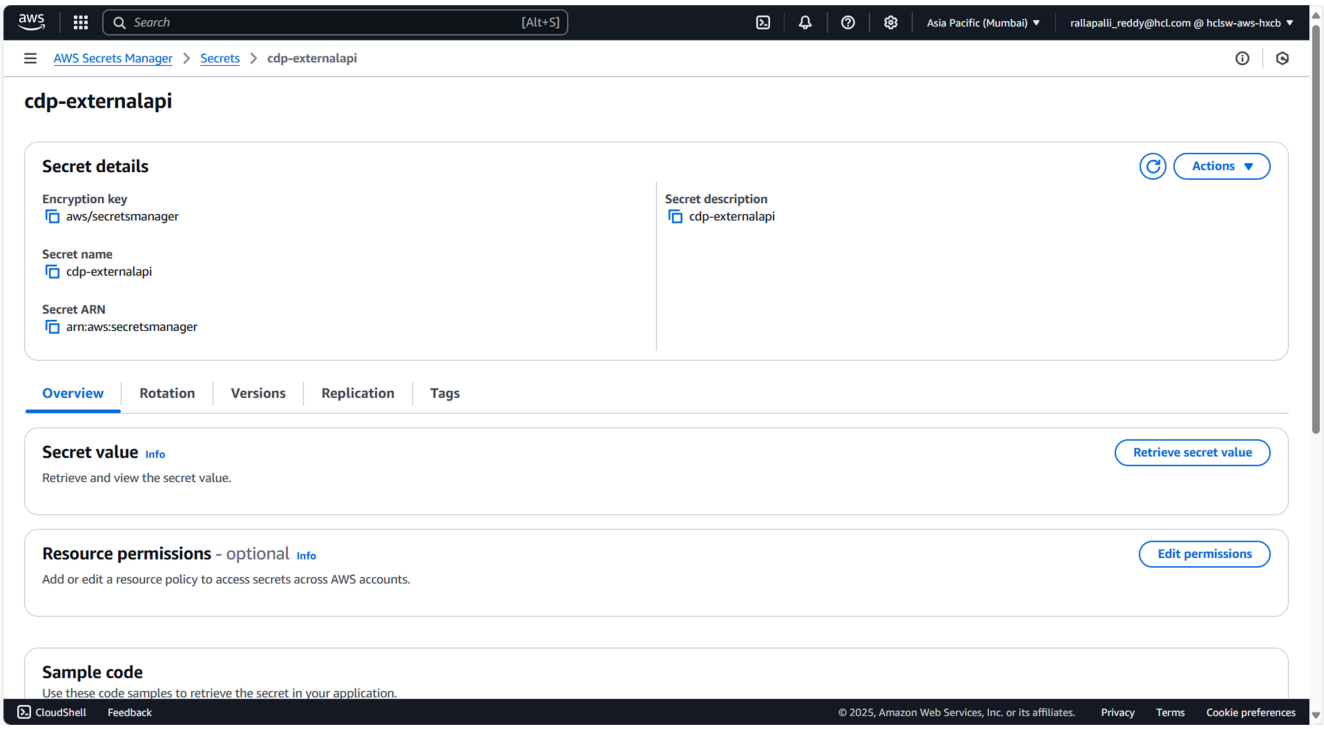


Deploying HCL CDP ExternalAPI

This section provides detailed instructions on how to deploy HCL CDP External API using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-externalapi secret in the AWS secret manager before deploying CDP External API.



To create cdp-externalapi secret in the AWS secret manager, follow the steps below:

1. Create a cdp-externalapi secret sample key and value, and update the actual values as shown below.

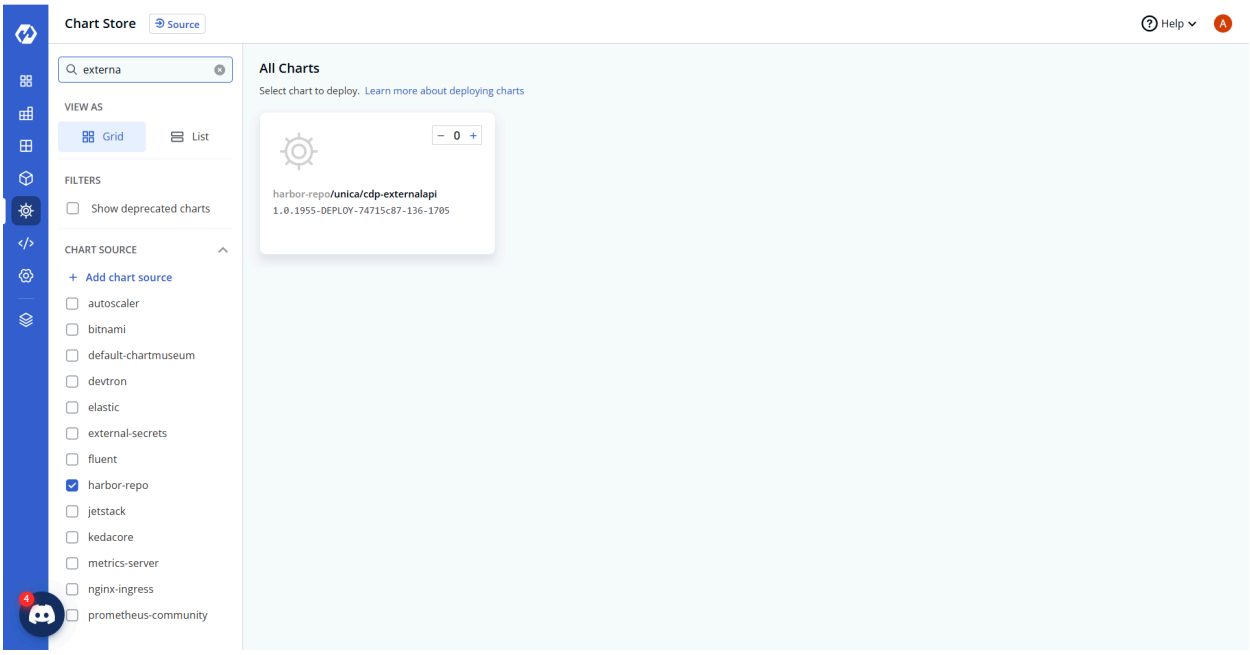
```
{
"DbUrl":"",
"DbUsername":"",
"DbUserPassword":""
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

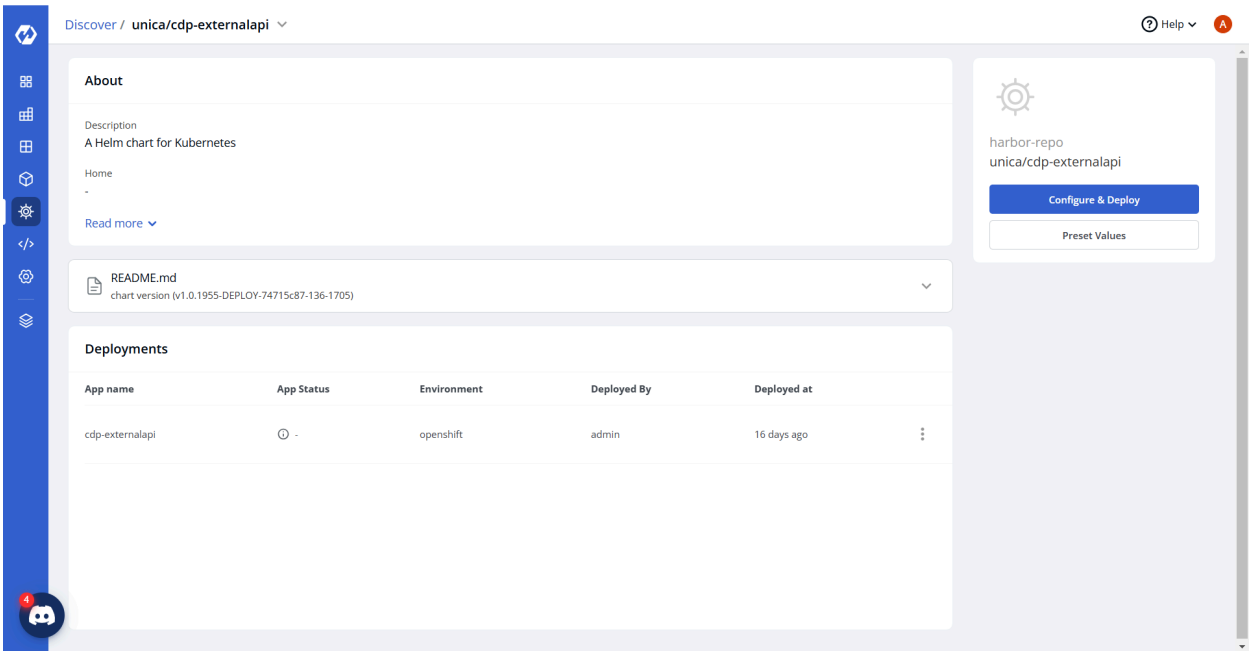
Deploy HCL CDP External API

To deploy HCL CDP External API using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-externalapi chart to deploy.



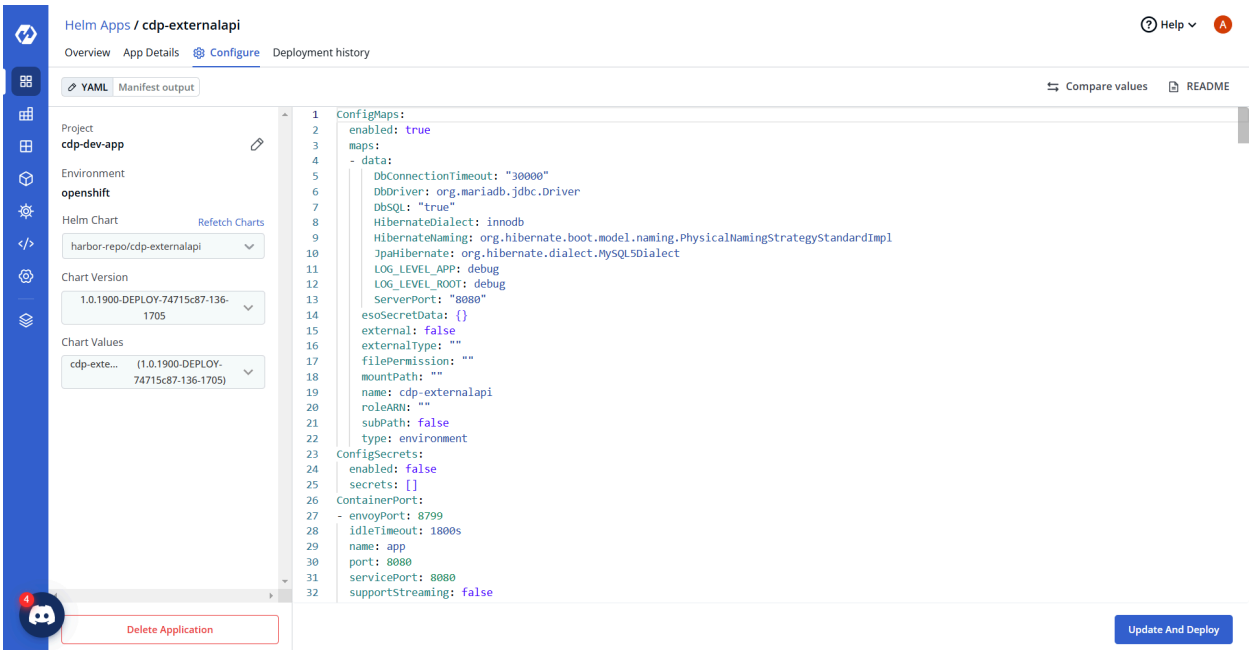
2. Now, configure and deploy the cdp-externalapi charts.



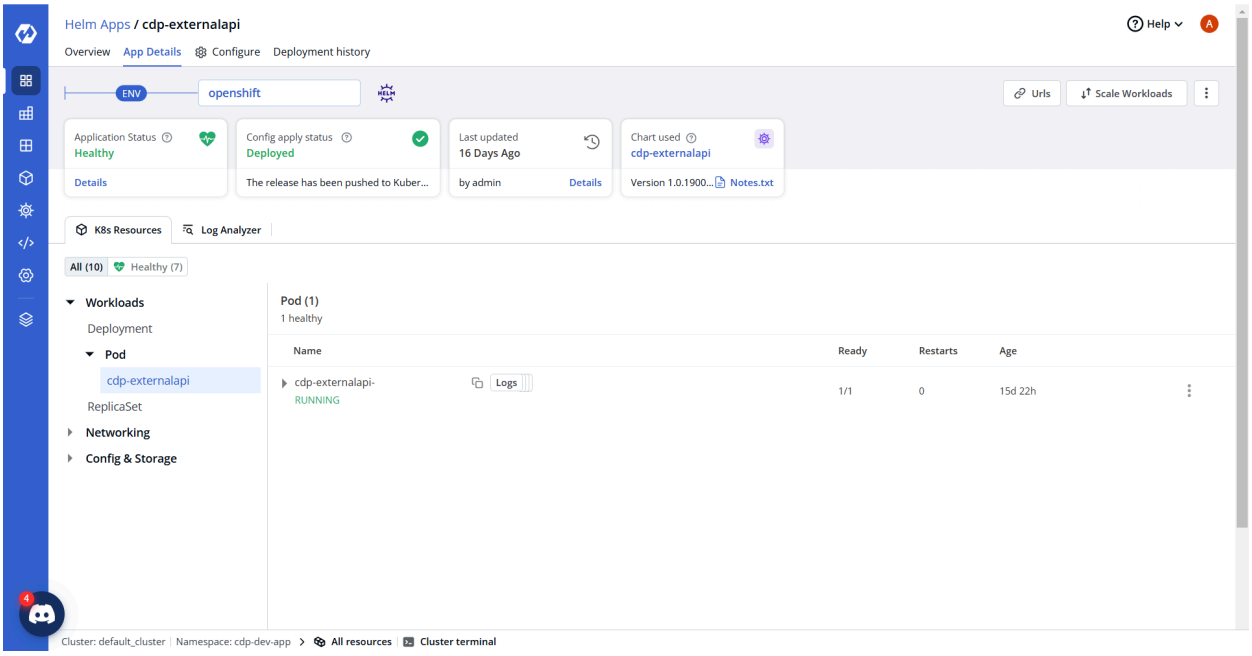
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
ServerPort: "8080"
HibernateNaming: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
DbDriver: org.mariadb.jdbc.Driver
DbSQL: "true"
JpaHibernate: org.hibernate.dialect.MySQL5Dialect
DbConnectionTimeout: "30000"
```

```
HibernateDialect: innodb
LOG_LEVEL_APP: debug
LOG_LEVEL_ROOT: debug
```



4. On successful deployment, validate the deployment as shown below.

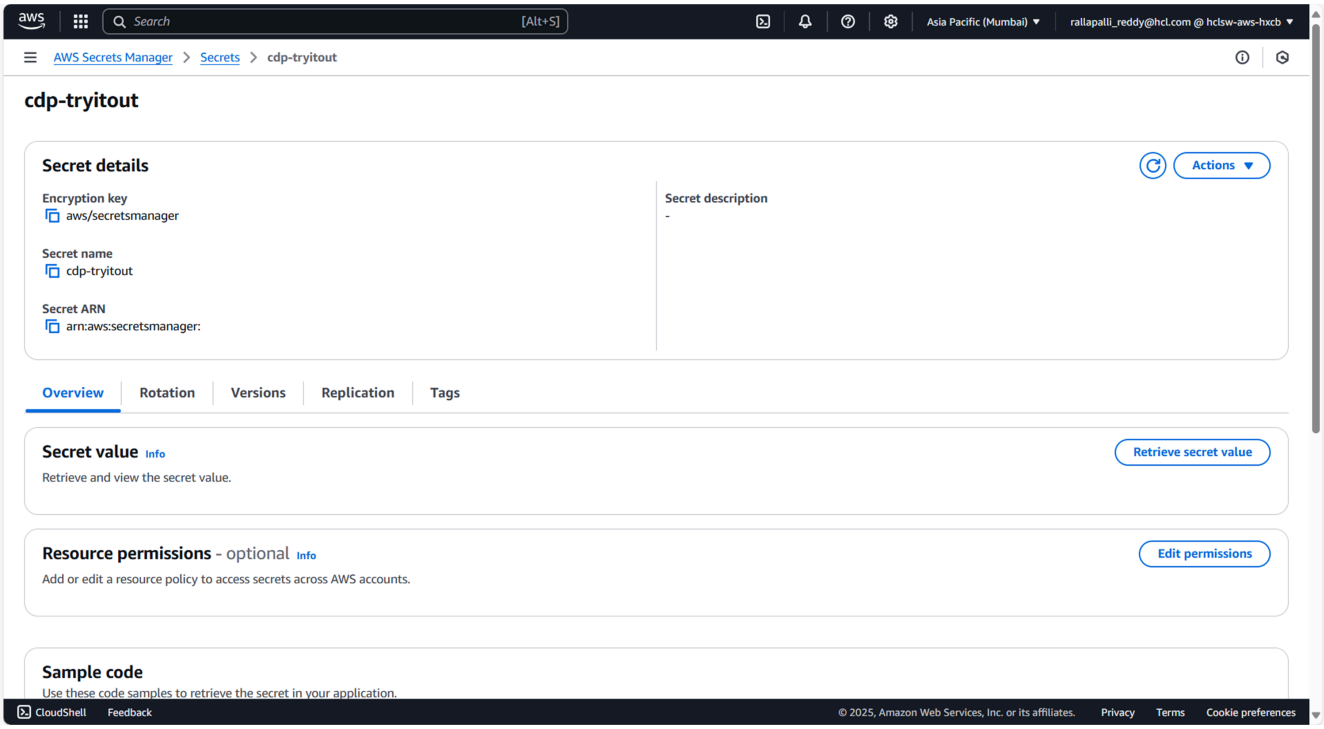


Deploying HCL CDP Tryitout

This section provides detailed instructions on how to deploy HCL CDP Tryitout using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-tryitout secret in the AWS secret manager before deploying CDP Tryitout.



To create CDP Tryitout secret in the AWS secret manager, follow the steps below:

1. Create a cdp-tryitout secret sample key and value, and update the actual values as shown below.

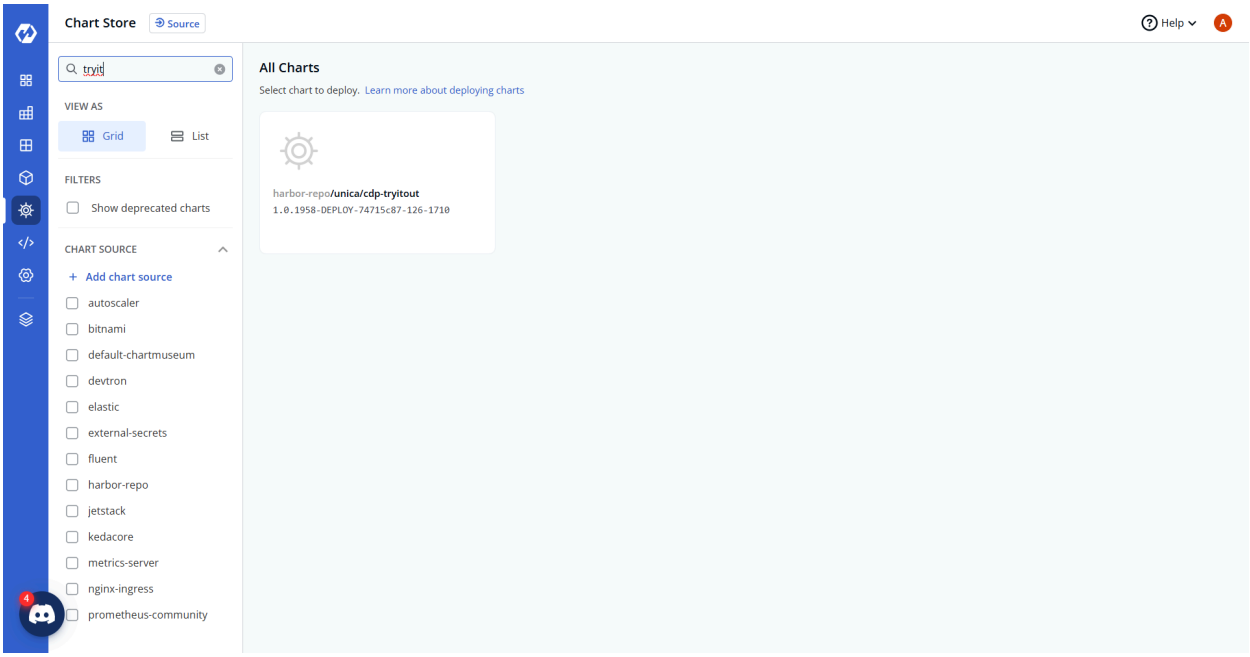
```
{
  "DbUrl": "jdbc:mariadb://ip:port/dbName",
  "DbUserPassword": "dbPwd",
  "DbUsername": "dbUser"
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

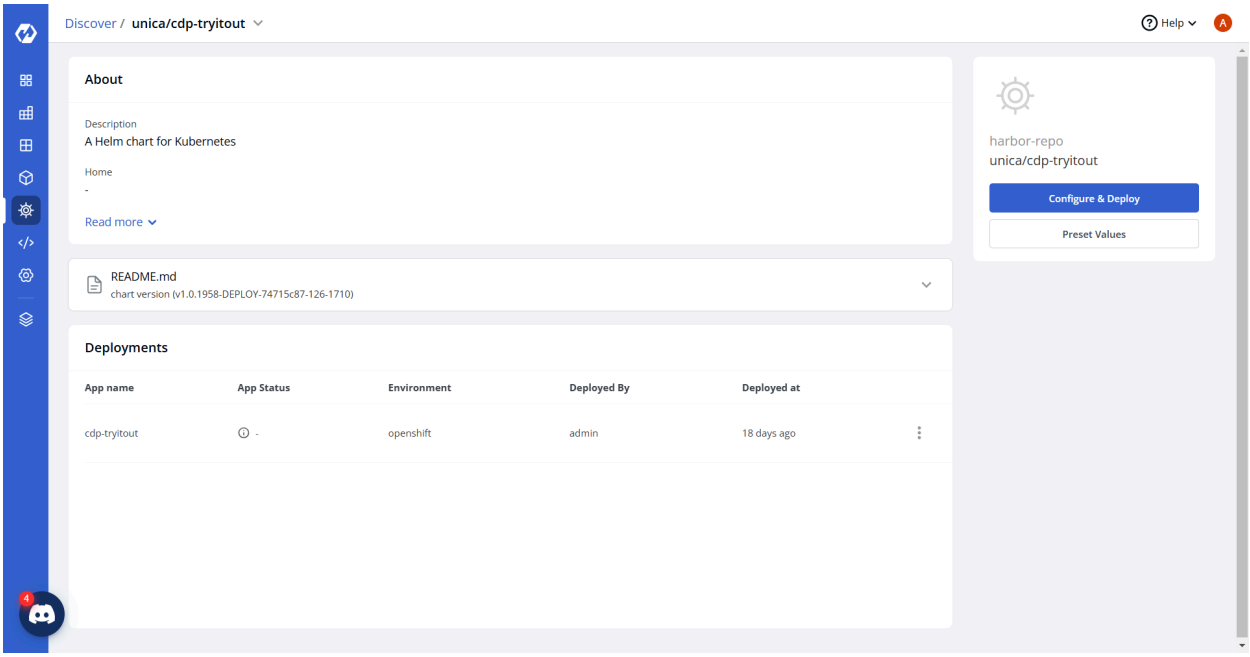
Deploy CDP TryItOut

To deploy CDP TryItOut using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select cdp-tryitout chart to deploy.



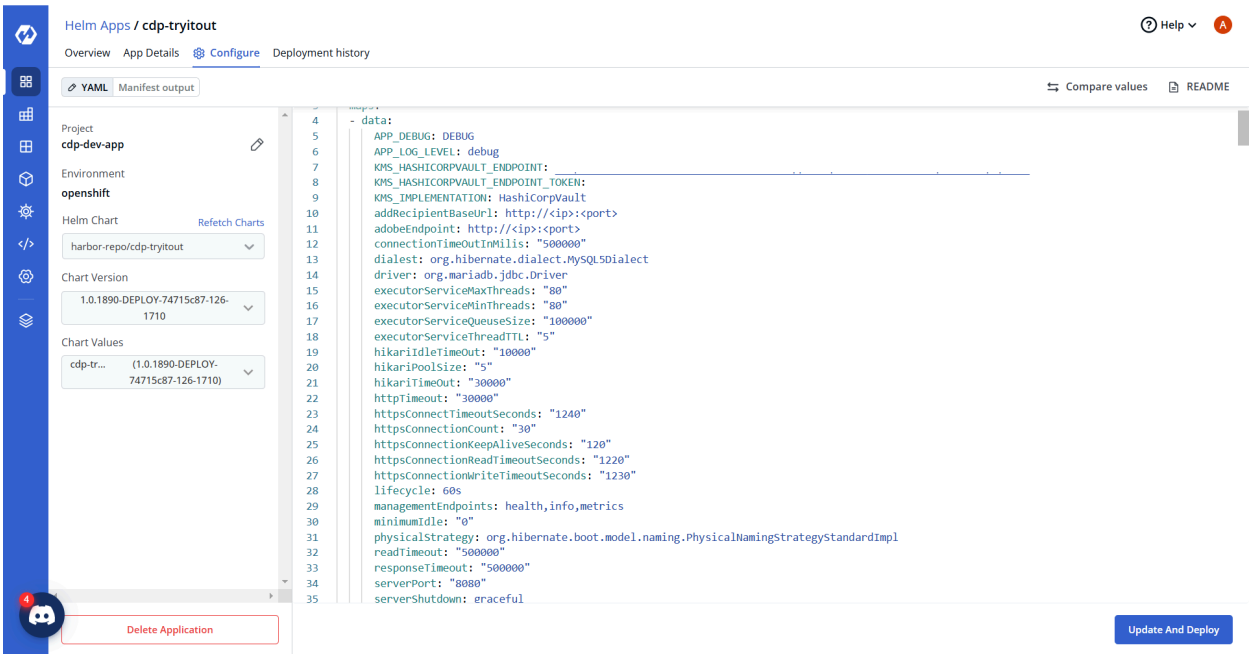
2. Now, configure and deploy the cdp-tryitout charts.



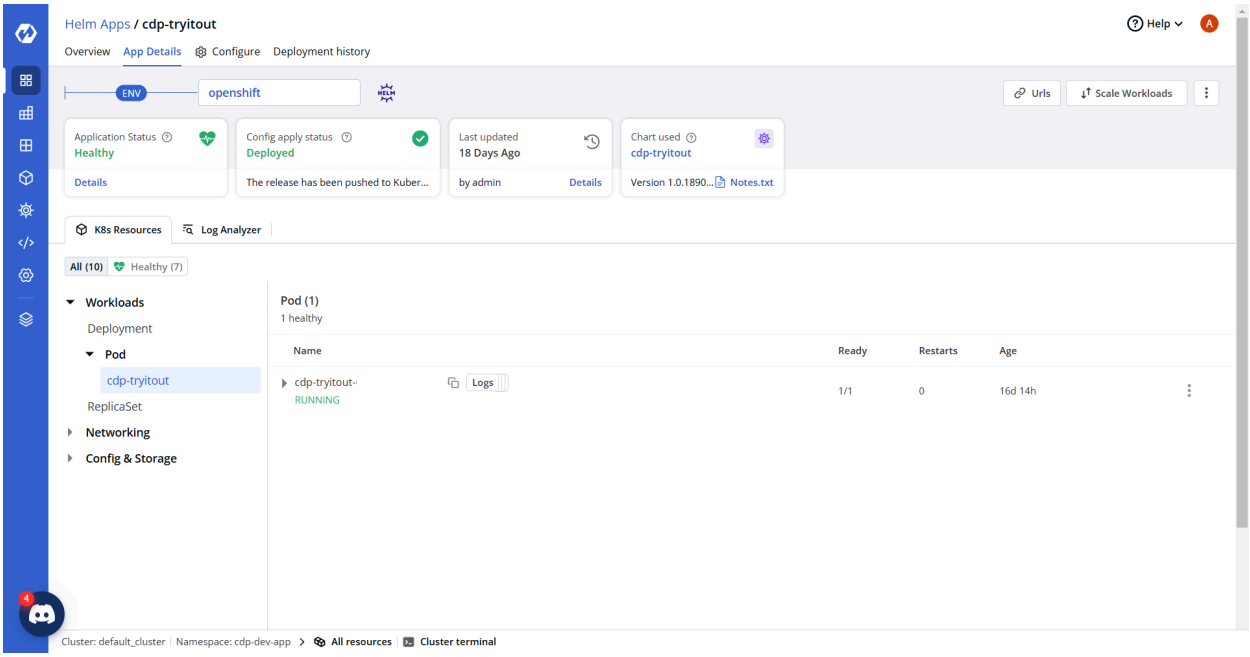
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
serverPort: "8080"
serverShutdown: graceful
lifecycle: 60s
physicalStrategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
driver: org.mariadb.jdbc.Driver
sqlStatement: "true"
```

```
dialest: org.hibernate.dialect.MySQL5Dialect
httpTimeout: "30000"
storageEngine: innodb
APP_DEBUG: DEBUG
connectionTimeOutInMilis: "500000"
responseTimeout: "500000"
readTimeout: "500000"
writeTimeOut: "50000"
executorServiceMinThreads: "80"
executorServiceMaxThreads: "80"
executorServiceThreadTTL: "5"
executorServiceQueuseSize: "100000"
smsEmailSenderBaseUrl: https://sender.hclsoftware.cloud
managementEndpoints: health,info,metrics
httpsConnectionCount: "30"
httpsConnectionKeepAliveSeconds: "120"
httpsConnectionReadTimeoutSeconds: "1220"
httpsConnectionWriteTimeoutSeconds: "1230"
httpsConnectTimeoutSeconds: "1240"
addRecipientBaseUrl: http://<ip>:<port>
hikariPoolSize: "5"
hikariTimeOut: "30000"
hikariIdleTimeOut: "10000"
minimumIdle: "0"
adobeEndpoint: http://<ip>:<port>
APP_LOG_LEVEL: debug
KMS_IMPLEMENTATION: AWS
```



4. On successful deployment, validate the deployment as shown below.

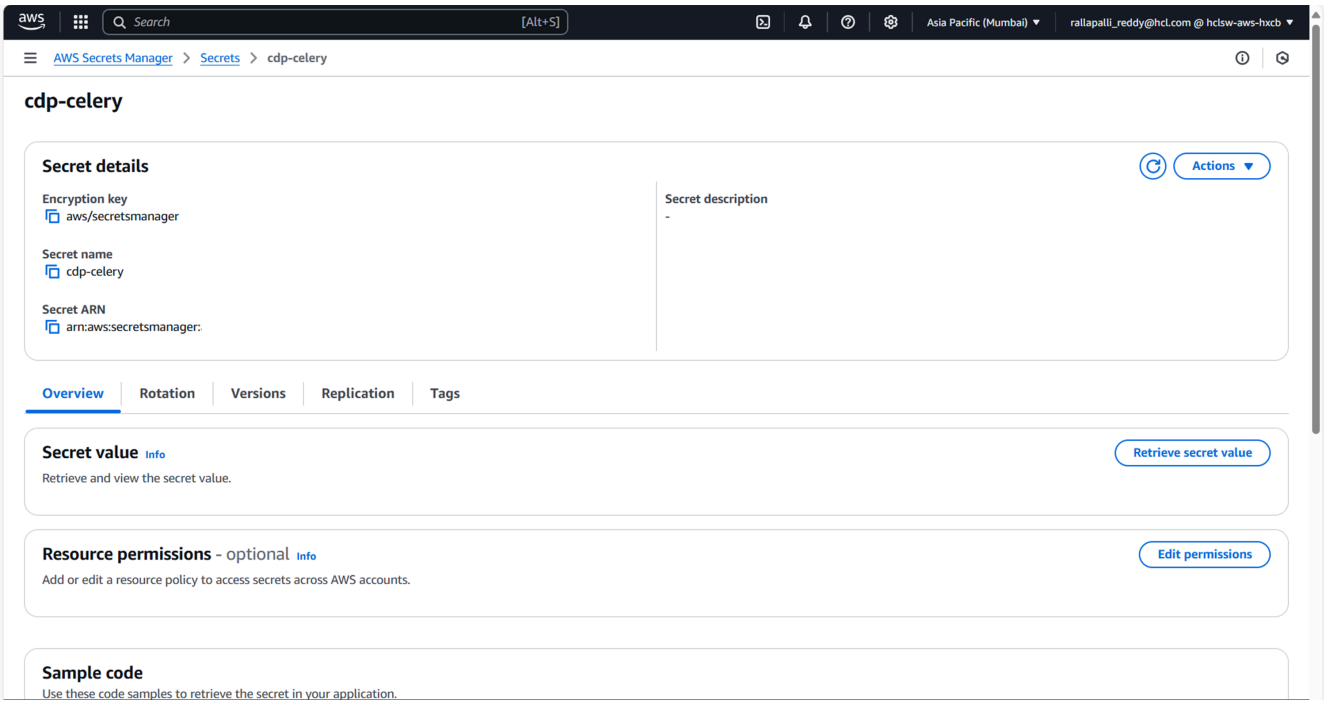


Deploying HCL CDP Celery callback consumer

This section provides detailed instructions on how to deploy HCL CDP Celerykafkacallbackprocessor using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-Celerykafkacallbackprocessor secret in the AWS secret manager before deploying CDP Celery callback consumer.



To create CDP Celerykafkacallbackconsumer secret in the AWS secret manager , follow the steps below:

1. Create a cdp-Celerykafkacallbackprocessor secret sample key and value, and update the actual values as shown below.

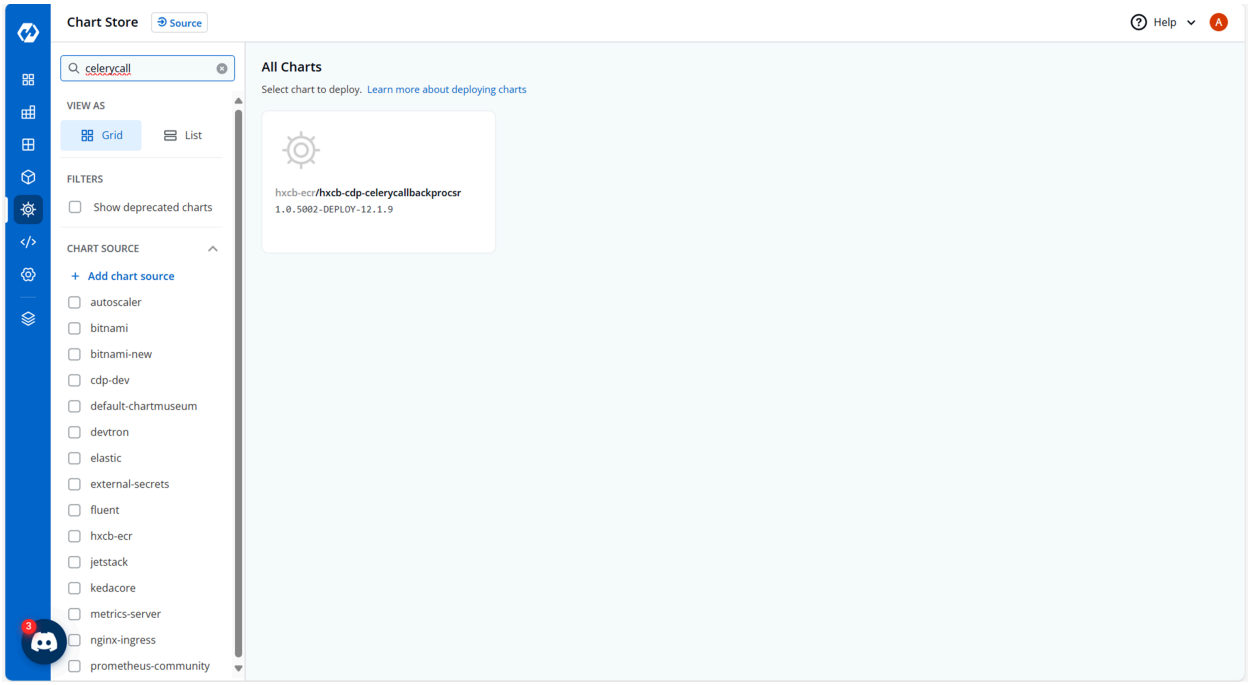
```
{
"bootstrap_servers":"",
"RABBITMQ_USERNAME":"",
"RABBITMQ_HOST":"",
"RABBITMQ_PORT":"5672"
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

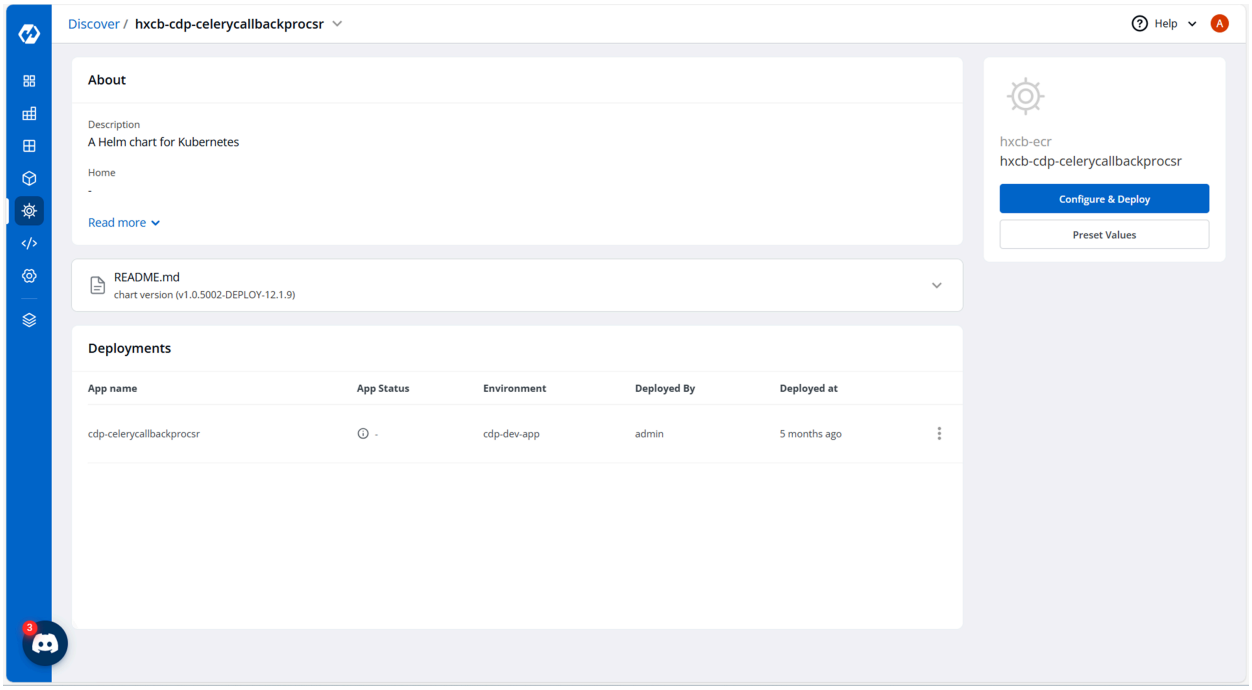
Deploy CDP Celerykafkacallbackconsumer

To deploy CDP Celerykafkacallbackconsumer using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-Celerykafkacallbackprocessor chart to deploy.



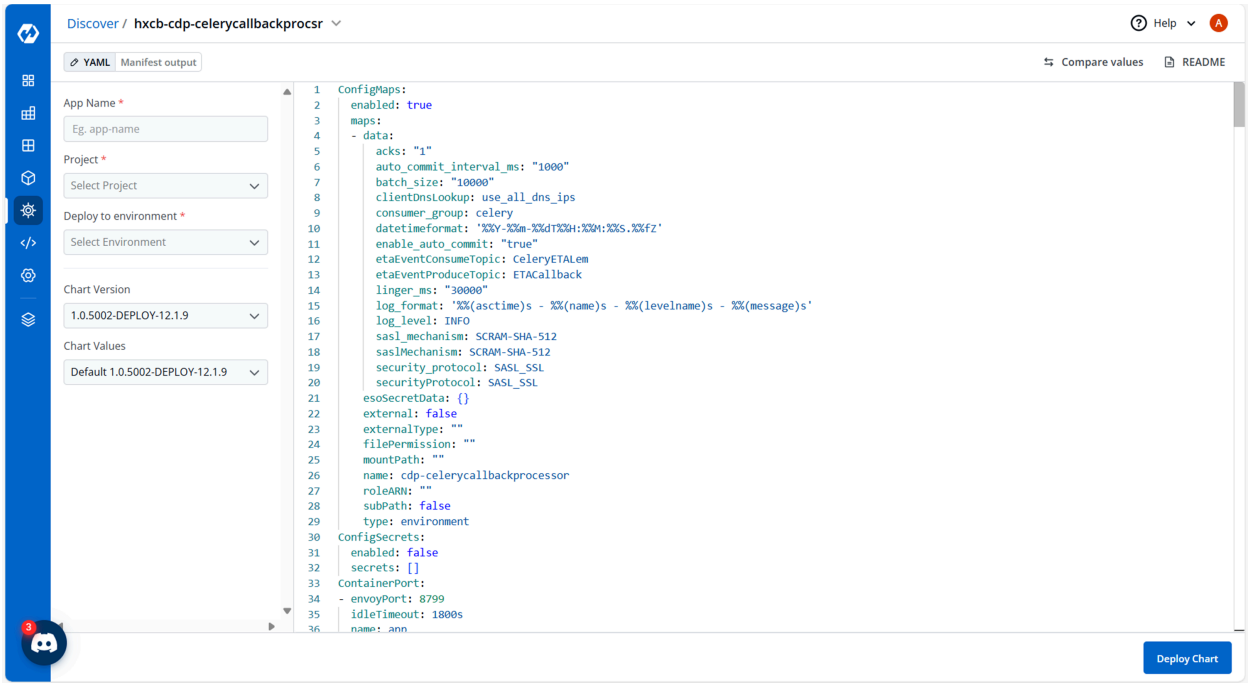
2. Now, configure and deploy the cdp-Celerykafkacallbackprocessor charts.



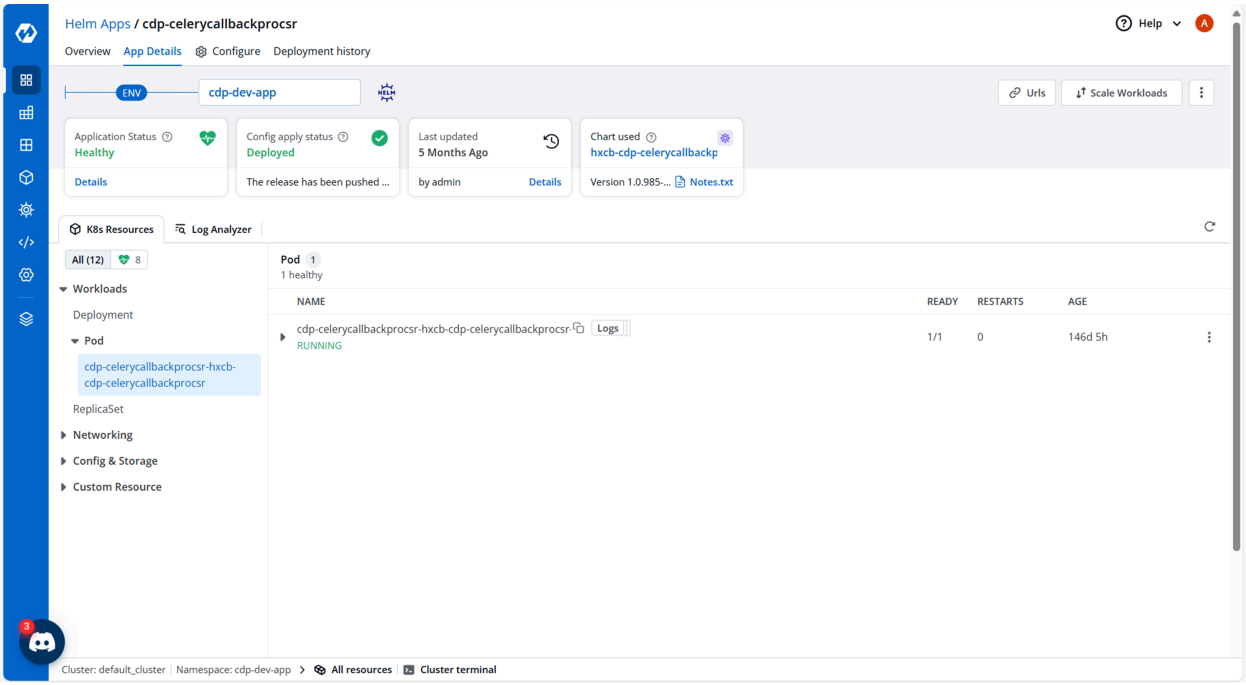
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
batch_size: "10000"
linger_ms: "30000"
etaEventProduceTopic: ETACallback
```

```
securityProtocol: SASL_SSL
saslMechanism: SCRAM-SHA-512
acks: "1"
consumer_group: celery
enable_auto_commit: "true"
auto_commit_interval_ms: "1000"
etaEventConsumeTopic: CeleryETALem
security_protocol: SASL_SSL
sasl_mechanism: SCRAM-SHA-512
clientDnsLookup: use_all_dns_ips
datetimeformat: "%%Y-%%m-%%dT%%H:%%M:%%S.%%fZ"
log_level: INFO
log_format: "%%(asctime)s - %%(name)s - %%(levelname)s - %%(message)s"
```



4. On successful deployment, validate the deployment as shown below.

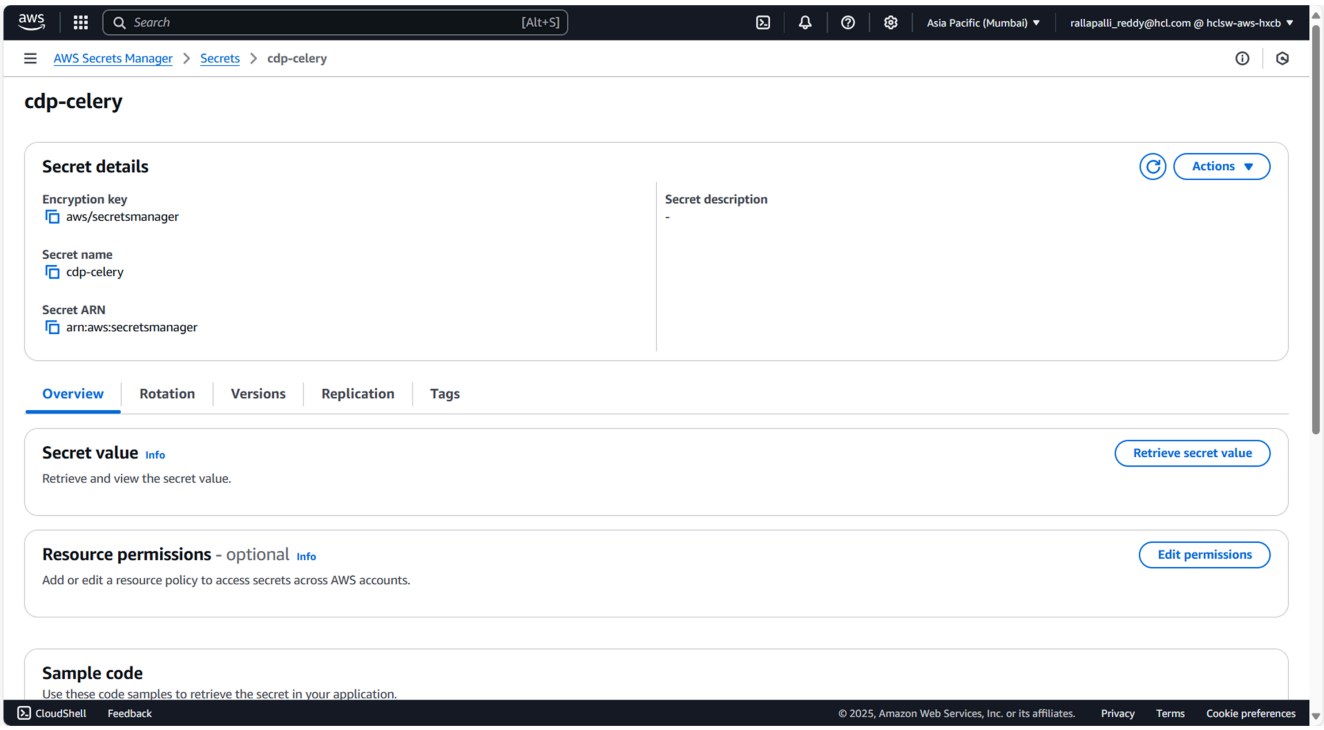


Deploying HCL CDP Celery Kafka eta Consumer

This section provides detailed instructions on how to deploy HCL CDP Celerykafkaetaconsumer using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-Celerykafkaetaconsumer secret in the AWS secret manager before deploying CDP Celery Kafka eta Consumer.



To create CDP Celerykafkaetaconsumer secret in the AWS secret manager , follow the steps below:

1. Create a cdp-Celerykafkaetaconsumer secret sample key and value, and update the actual values as shown below.

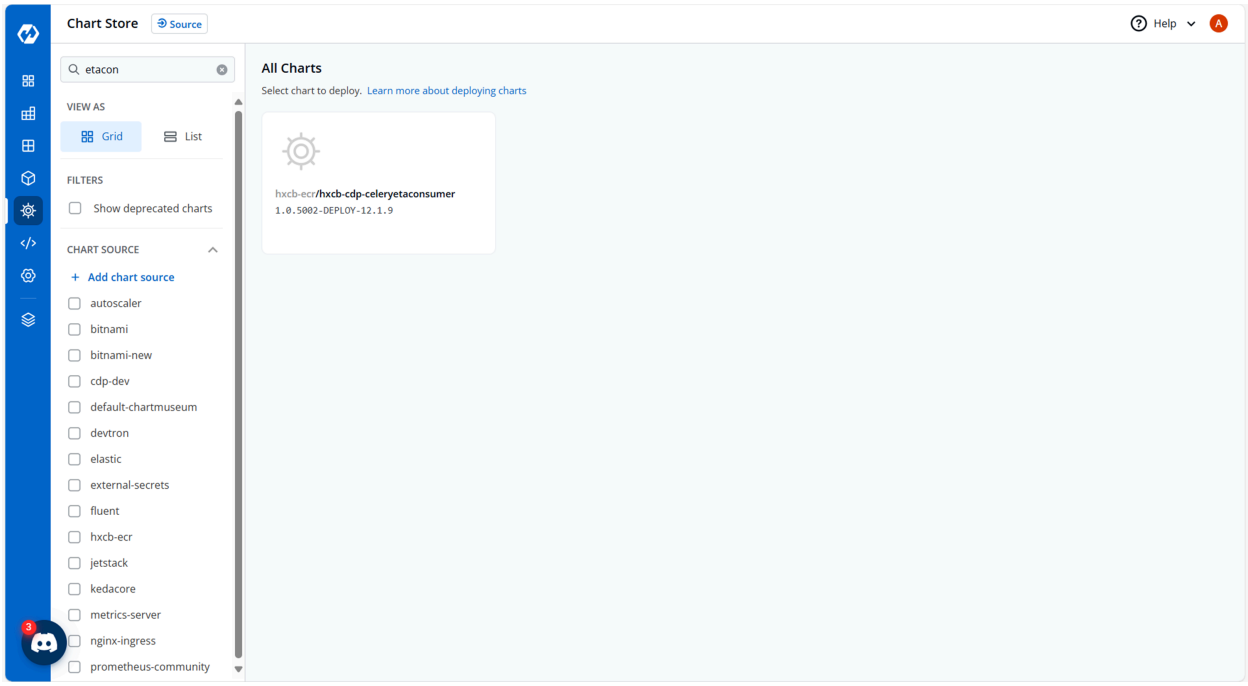
```
{
"bootstrap_servers":"",
"RABBITMQ_USERNAME":"",
"RABBITMQ_HOST":"",
"RABBITMQ_PORT":"5672"
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

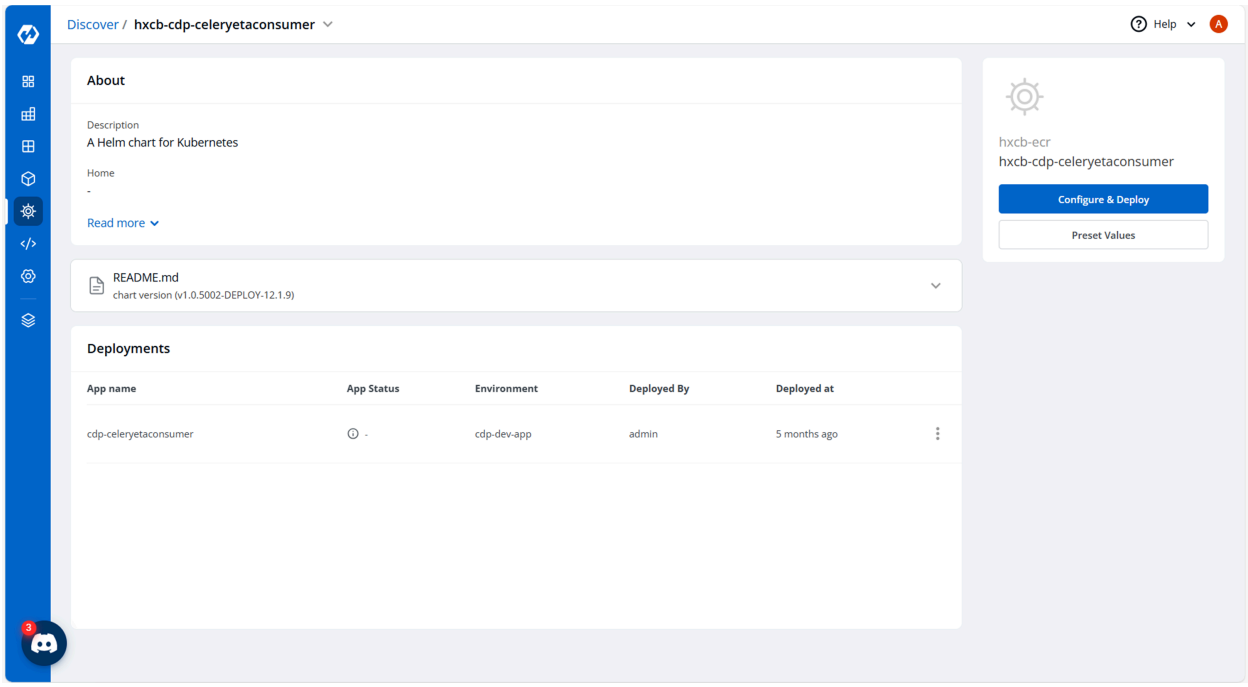
Deploy CDP Celerykafkaetaconsumer

To deploy CDP cdp-Celerykafkaetaconsumer using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-Celerykafkaetaconsumer chart to deploy.



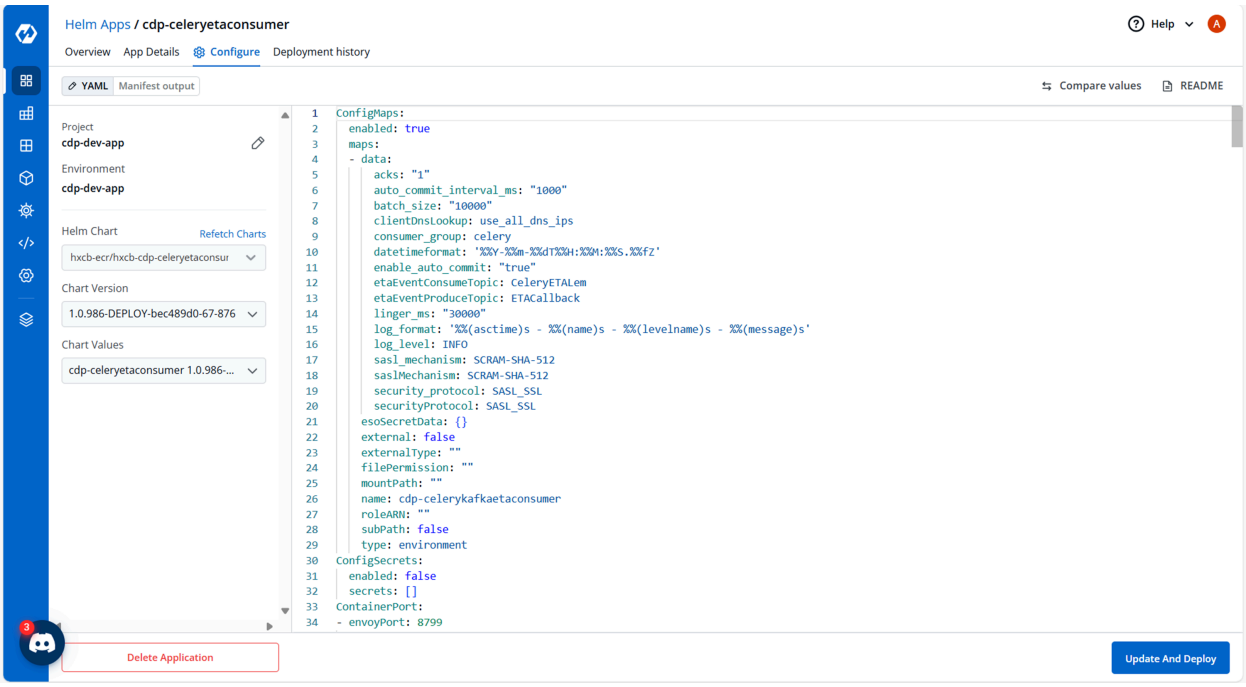
2. Now, configure and deploy the cdp-Celerykafkaetaconsumer charts.



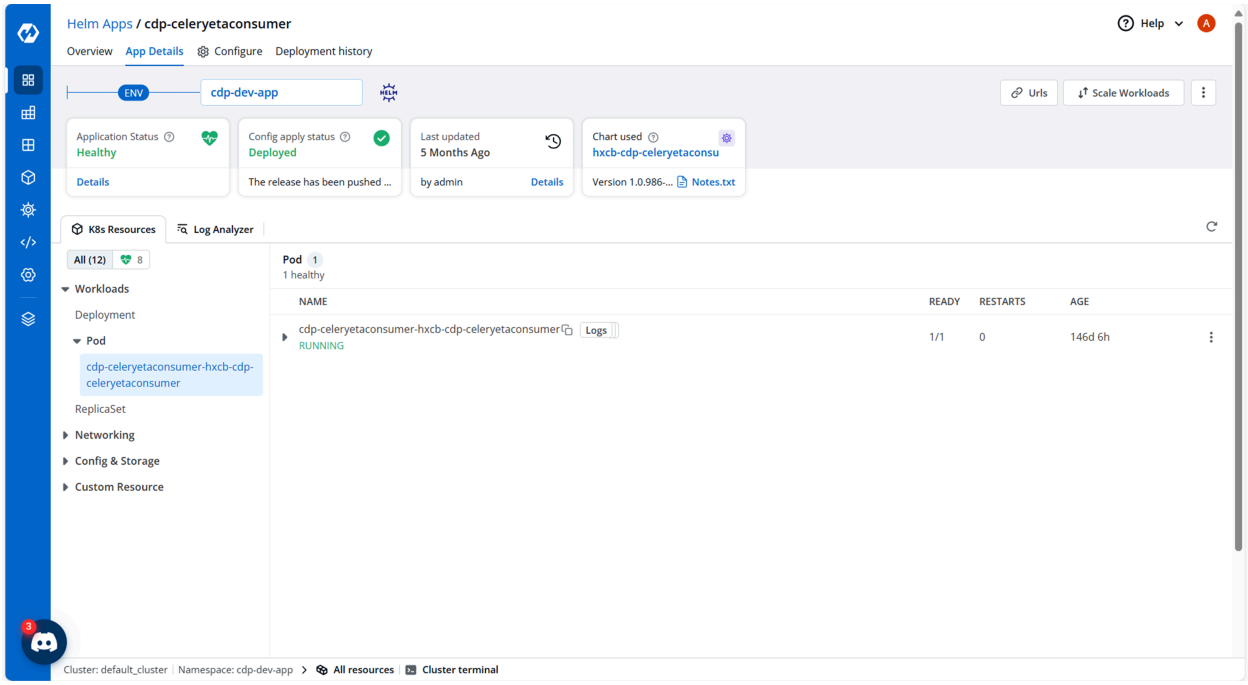
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
batch_size: "10000"
linger_ms: "30000"
etaEventProduceTopic: ETACallback
securityProtocol: SASL_SSL
```

```
saslMechanism: SCRAM-SHA-512
acks: "1"
consumer_group: celery
enable_auto_commit: "true"
auto_commit_interval_ms: "1000"
etaEventConsumeTopic: CeleryETALem
security_protocol: SASL_SSL
sasl_mechanism: SCRAM-SHA-512
clientDnsLookup: use_all_dns_ips
datetimeformat: "%%Y-%%m-%%dT%%H:%%M:%%S.%%fZ"
log_level: INFO
log_format: "%%(asctime)s - %%(name)s - %%(levelname)s - %%(message)s"
```



4. On successful deployment, validate the deployment as shown below.

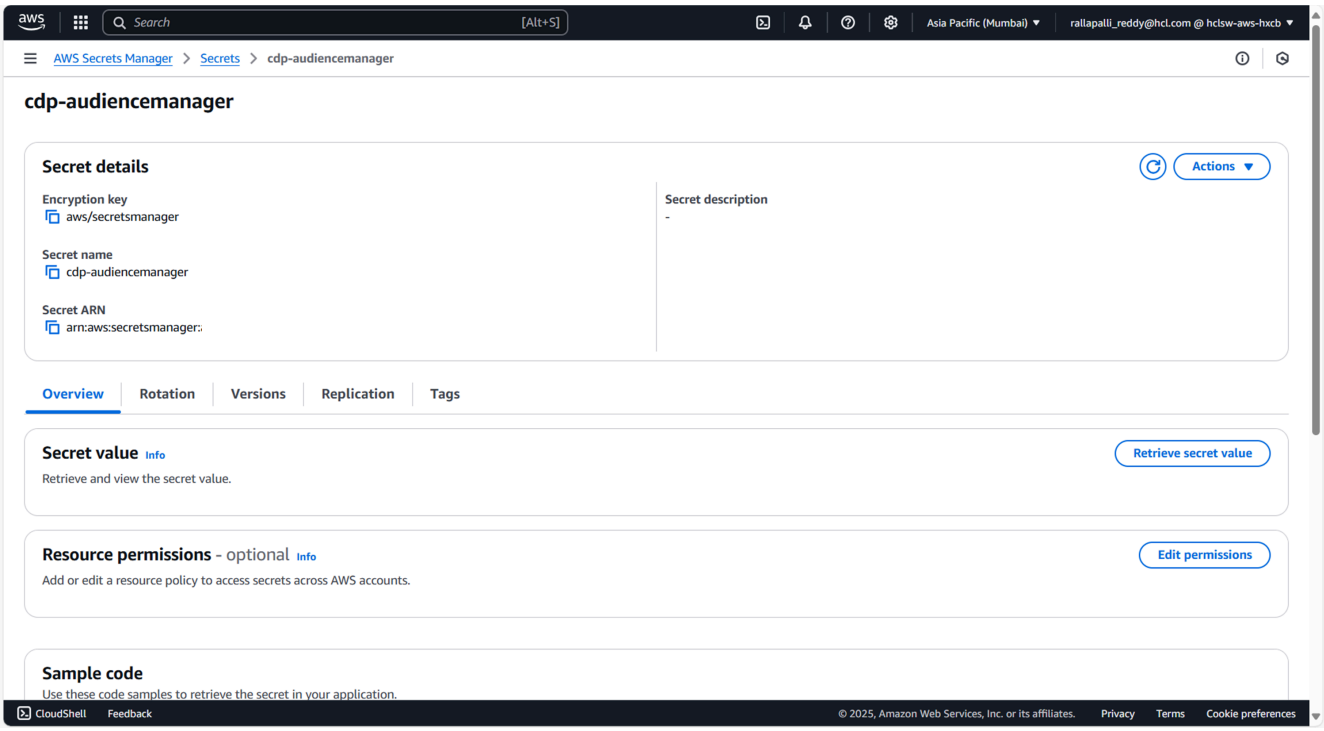


Deploying HCL CDP Audience Uploader

This section provides detailed instructions on how to deploy HCL CDP Audienceuploader using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-Audienceuploader secret in the AWS secret manager before deploying CDP Audience Uploader.



To create CDP Audienceuploader secret in the AWS secret manager , follow the steps below:

1. Create a cdp-Audienceuploader secret sample key and value, and update the actual values as shown below.

```
{
  "GANGLIA_COLLECTORS": "",
  "GANGLIA_PORT": "8650",
  "CONSUMER_TOPIC_LIST": "dmp_rt_segments",
  "KAFKA_TOPIC_LIST": "dmp_rt_segments",
  "KAFKA_BATCH_SIZE": "100",
  "KAFKA_TOPIC_CONSUMERS": "6",
  "KAFKA_DEFAULT_TIMEOUT": "1000",
  "KAFKA_DEFAULT_TRIGGER": "5000",
  "CONSUMER_GROUP": "",
  "ZOOKEEPER_LIST": "",
  "KAFKA_CONSUMER_CONFIG": "",
  "DB_HOST": "",
  "DB_PORT": "3307",
  "DB": "vrm",
  "DB_USERNAME": "cdpapp_ro",
  "DM_PASSWORD": "",
  "SQS_QUEUE_URL": "",
  "AEROSPIKE_HOST": "",
  "AEROSPIKE_PORT": "13000",
  "APS1_HOST": "",
  "KafkaProducerConfig": "",
  "KafkaProducerPoolSize": "10",
  "KafkaProducerMeta": "true",
  "KafkaProducerHostname": "cdp-audience-manager",
  "KafkaRegion": "aps1",
  "6525_ApiCtx": "",
  "CLIENT_ID": "",
```

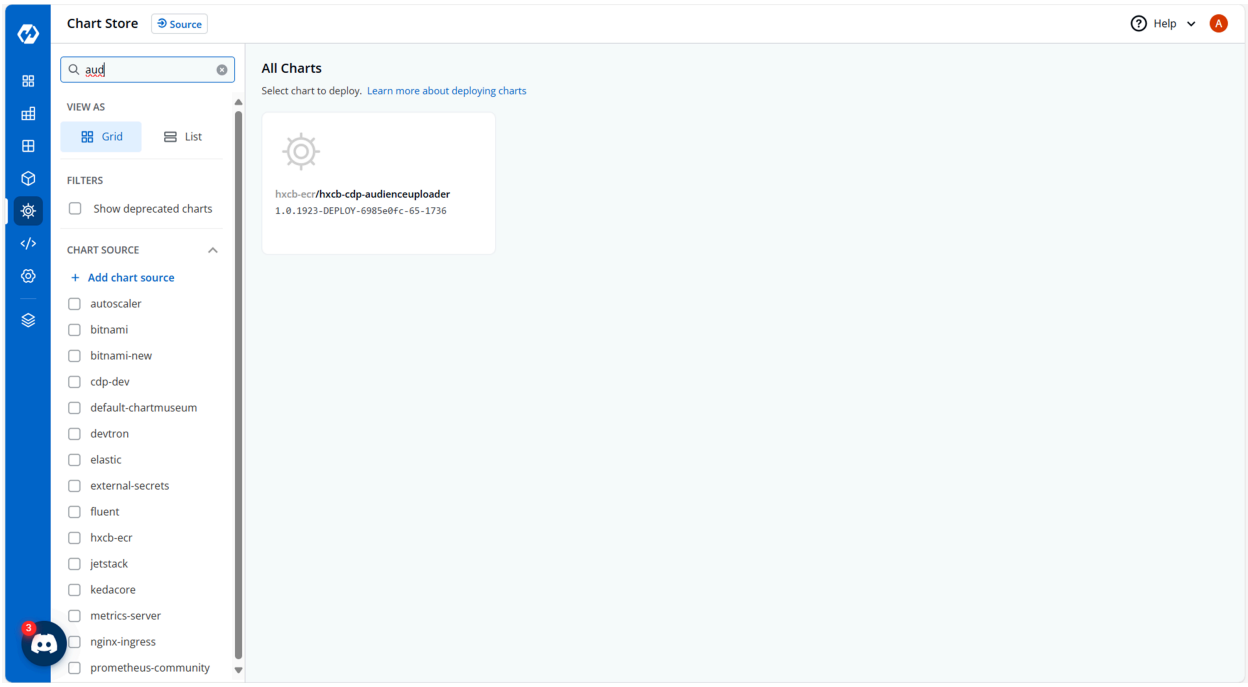
```
  "CLIENT_SECRET": ""
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

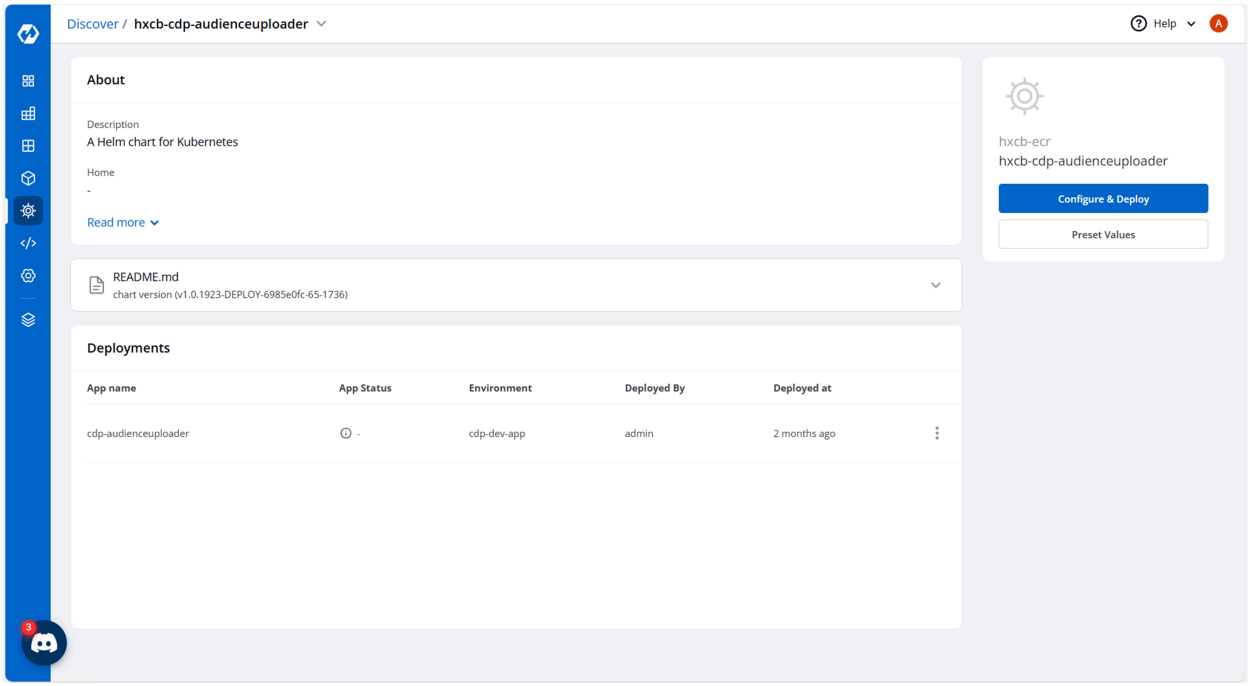
Deploy CDP Audienceuploader

To deploy cdp-Audienceuploader using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-Audienceuploader chart to deploy.



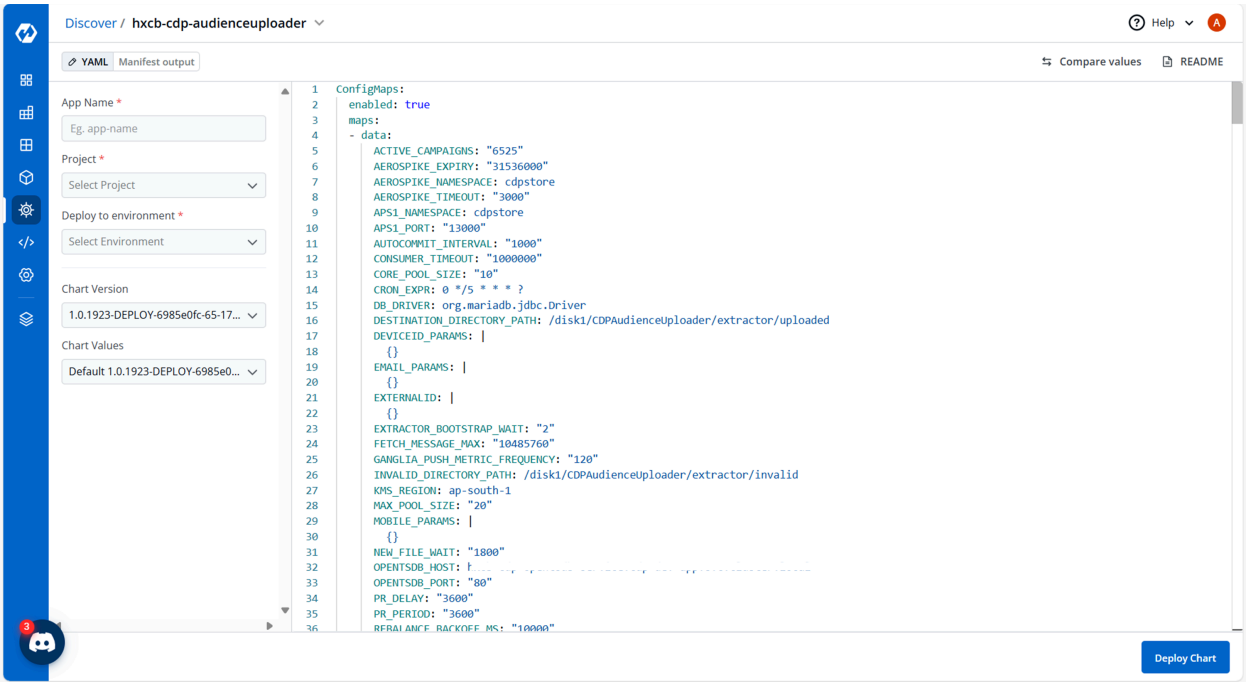
2. Now, configure and deploy the cdp-Audienceuploader charts.



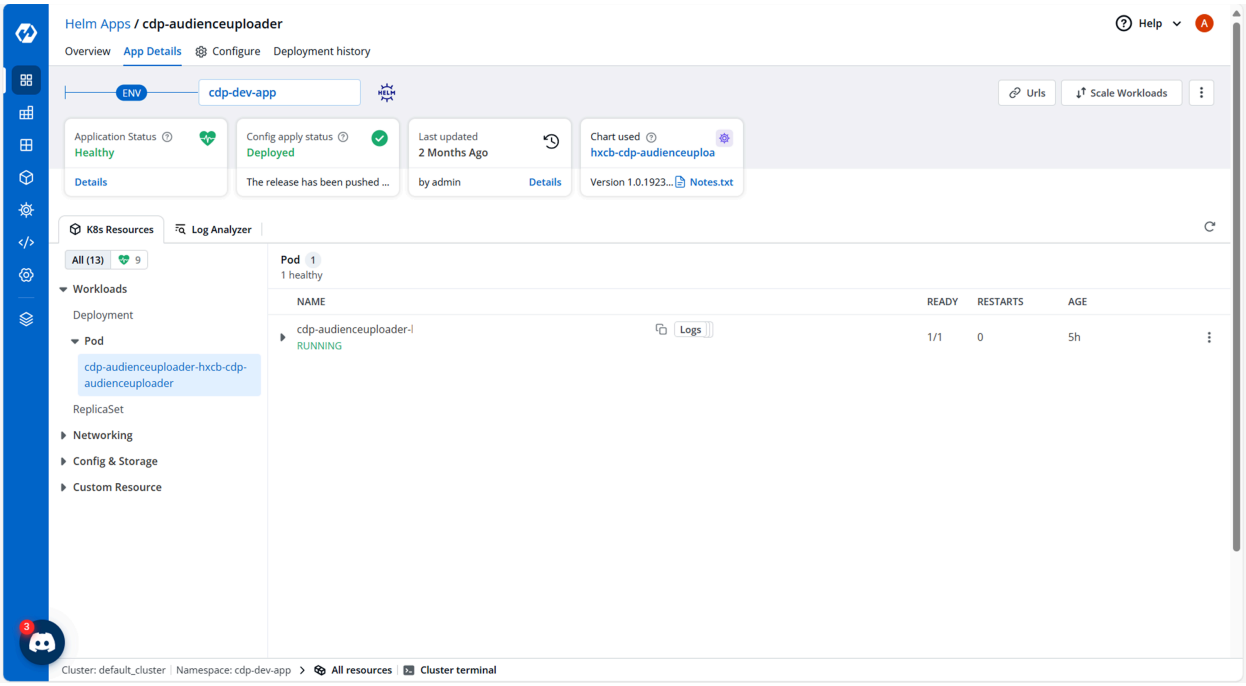
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
OPENTSDB_HOST: hxcb-cdp-opentsdb-service.cdp-dev-app.svc.cluster.local
OPENTSDB_PORT: "80"
GANGLIA_PUSH_METRIC_FREQUENCY: "120"
PR_DELAY: "3600"
PR_PERIOD: "3600"
ZOOKEEPER_TIMEOUT: "60000"
ZOOKEEPER_SYNC_TIME: "2000"
AUTOCOMMIT_INTERVAL: "1000"
ZOOKEEPER_CONNECTION_TIMEOUT: "60000"
CONSUMER_TIMEOUT: "1000000"
SOCKET_RECEIVE_BUFFER: "2097152"
FETCH_MESSAGE_MAX: "10485760"
REBALANCE_BACKOFF_MS: "10000"
SOURCE_DIRECTORY_PATH: /disk1/CDPAudienceUploader/extractor/processed
DESTINATION_DIRECTORY_PATH: /disk1/CDPAudienceUploader/extractor/uploaded
INVALID_DIRECTORY_PATH: /disk1/CDPAudienceUploader/extractor/invalid
EXTRACTOR_BOOTSTRAP_WAIT: "2"
NEW_FILE_WAIT: "1800"
MAX_POOL_SIZE: "20"
CORE_POOL_SIZE: "10"
EMAIL_PARAMS: |
  {}
MOBILE_PARAMS: |
  {}
DEVICEID_PARAMS: |
  {}
EXTERNALID: |
  {}
ACTIVE_CAMPAIGNS: "6525"
DB_DRIVER: org.mariadb.jdbc.Driver
CRON_EXPR: 0 */5 * * * ?
```

```
AEROSPIKE_NAMESPACE: cdpstore
AEROSPIKE_EXPIRY: "31536000"
AEROSPIKE_TIMEOUT: "3000"
SUPPORTED_REGIONS: aps1
APS1_PORT: "13000"
APS1_NAMESPACE: cdpstore
KMS_REGION: ap-south-1
REGION: ap-south-1
batchsize: "1"
KMS_IMPLEMENTATION: AWS
```



4. On successful deployment, validate the deployment as shown below.

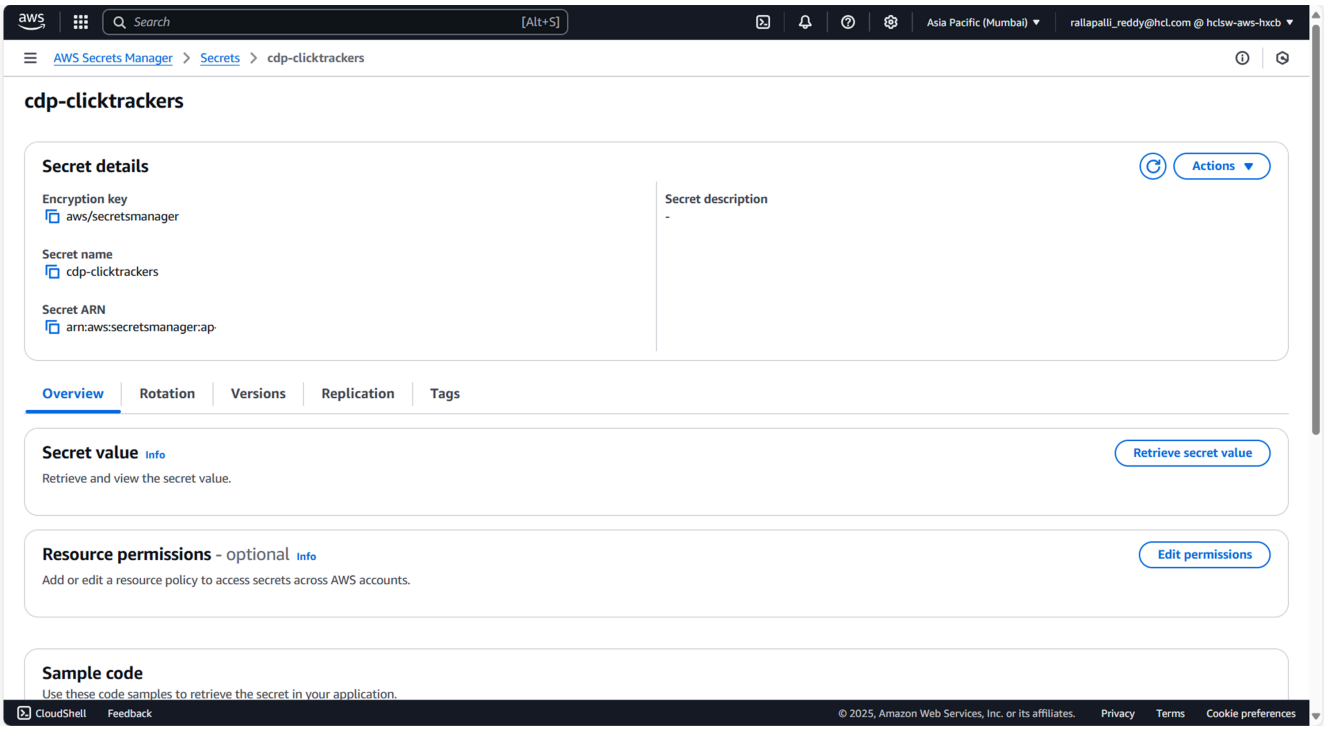


Deploying HCL CDP Click Trackers

This section provides detailed instructions on how to deploy HCL CDP Clicktrackers using the Devtron in the AWS.

Prerequisites

Make sure to create cdp-clicktrackers secret in the AWS secret manager before deploying CDP Click Trackers.



To create a CDP Clicktrackers secret in the AWS secret manager , follow the steps below:

1. Create a cdp-clicktrackers secret sample key and value, and update the actual values as shown below.

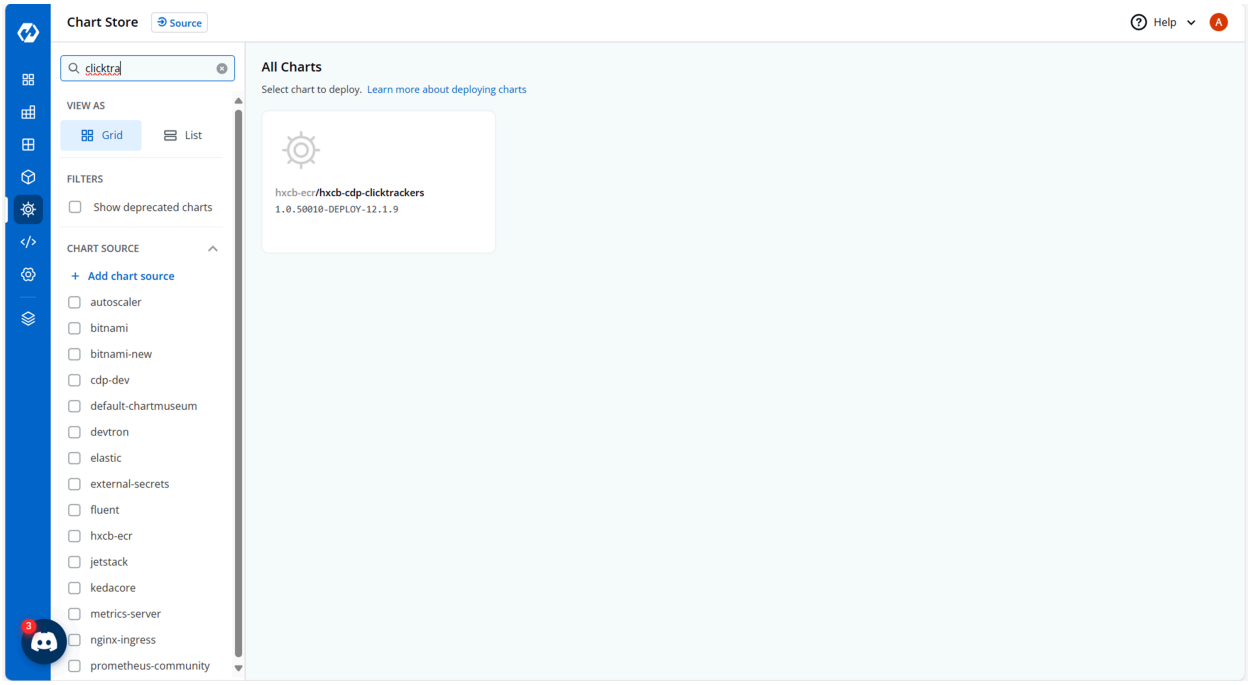
```
{
  "DbUsername": "",
  "DbPassword": "",
  "DbUrl": "jdbc:mariadb://:3307/vrm",
  "DbDriver": "org.mariadb.jdbc.Driver",
  "DbRefreshTimeMin": "5",
  "KafkaProducerConfigaps1": "",
  "KafkaProducerConfig": "",
  "KafkaSMSClickTopic": "clk_sms_click",
  "KafkaEmailClickTopic": "clk_email_click"
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

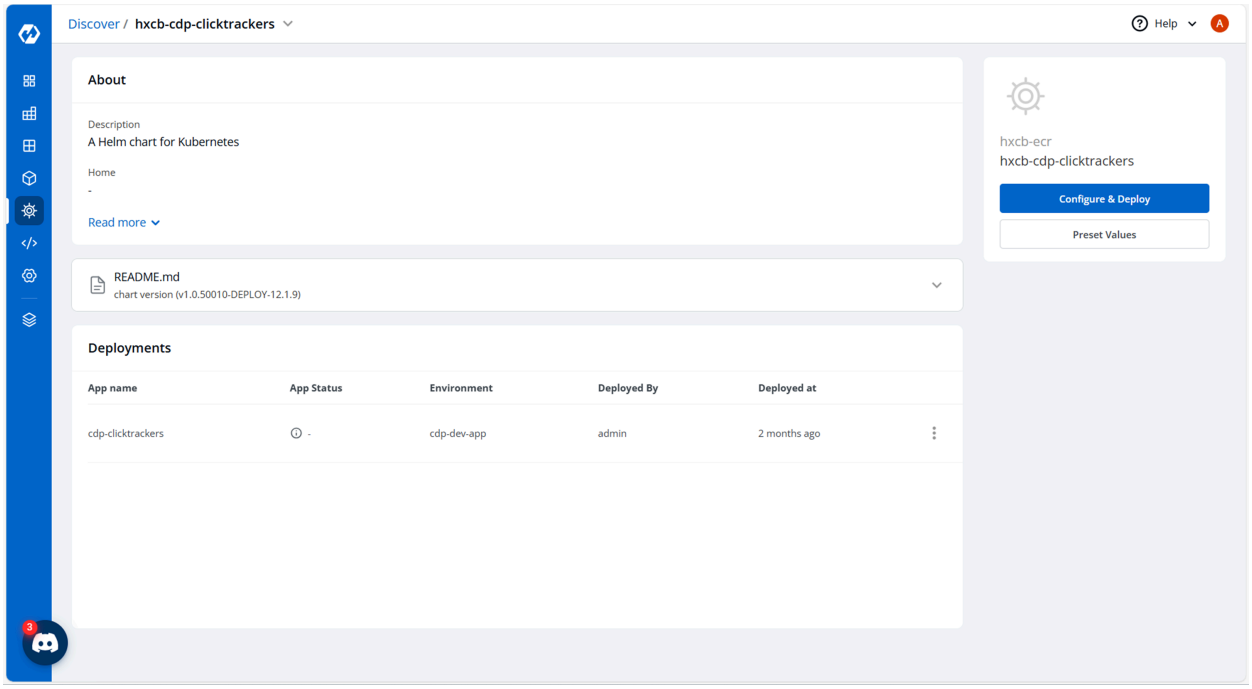
Deploy CDP Clicktrackers

To deploy the cdp-clicktrackers using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-clicktrackers chart to deploy.



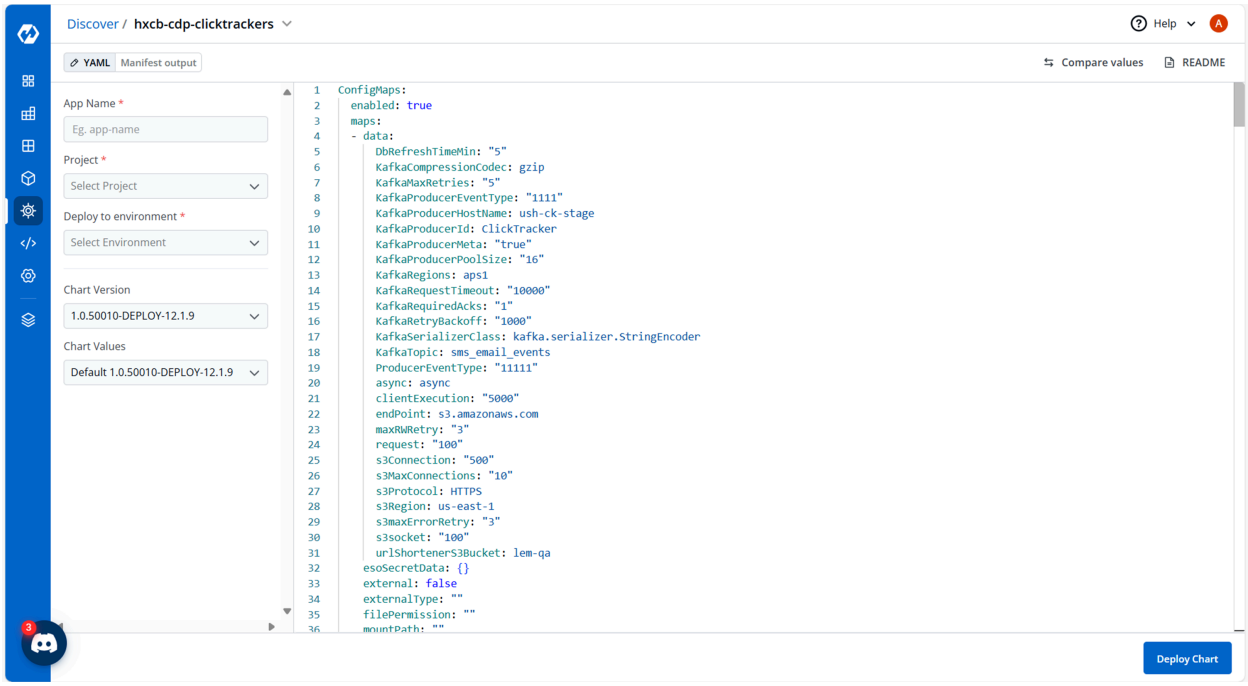
2. Now, configure and deploy the cdp-clicktrackers charts.



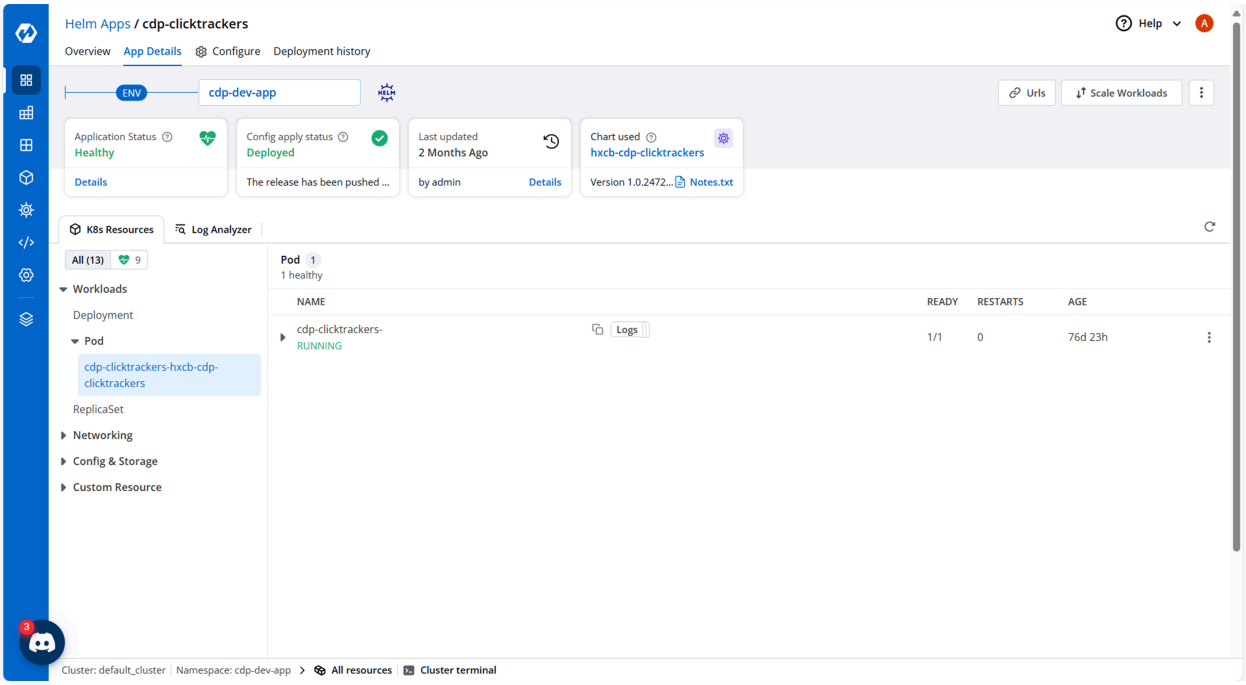
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
DbRefreshTimeMin: "5"
urlShortenerS3Bucket: lem-qa
s3Region: us-east-1
```

```
s3Protocol: HTTPS
s3maxErrorRetry: "3"
s3MaxConnections: "10"
maxRWRetry: "3"
s3socket: "100"
s3Connection: "500"
clientExecution: "5000"
request: "100"
endPoint: s3.amazonaws.com
KafkaSerializerClass: kafka.serializer.StringEncoder
KafkaProducerId: ClickTracker
LOG_LEVEL_APP: debug
LOG_LEVEL_ROOT: info
KafkaRequiredAcks: "1"
KafkaRequestTimeout: "10000"
async: async
KafkaCompressionCodec: gzip
KafkaMaxRetries: "5"
KafkaRetryBackoff: "1000"
KafkaProducerPoolSize: "16"
ProducerEventType: "11111"
KafkaProducerMeta: "true"
KafkaProducerEventType: "1111"
KafkaTopic: sms_email_events
KafkaRegions: aps1
KafkaProducerHostName: ush-ck-stage
```



4. On successful deployment, validate the deployment as shown below.

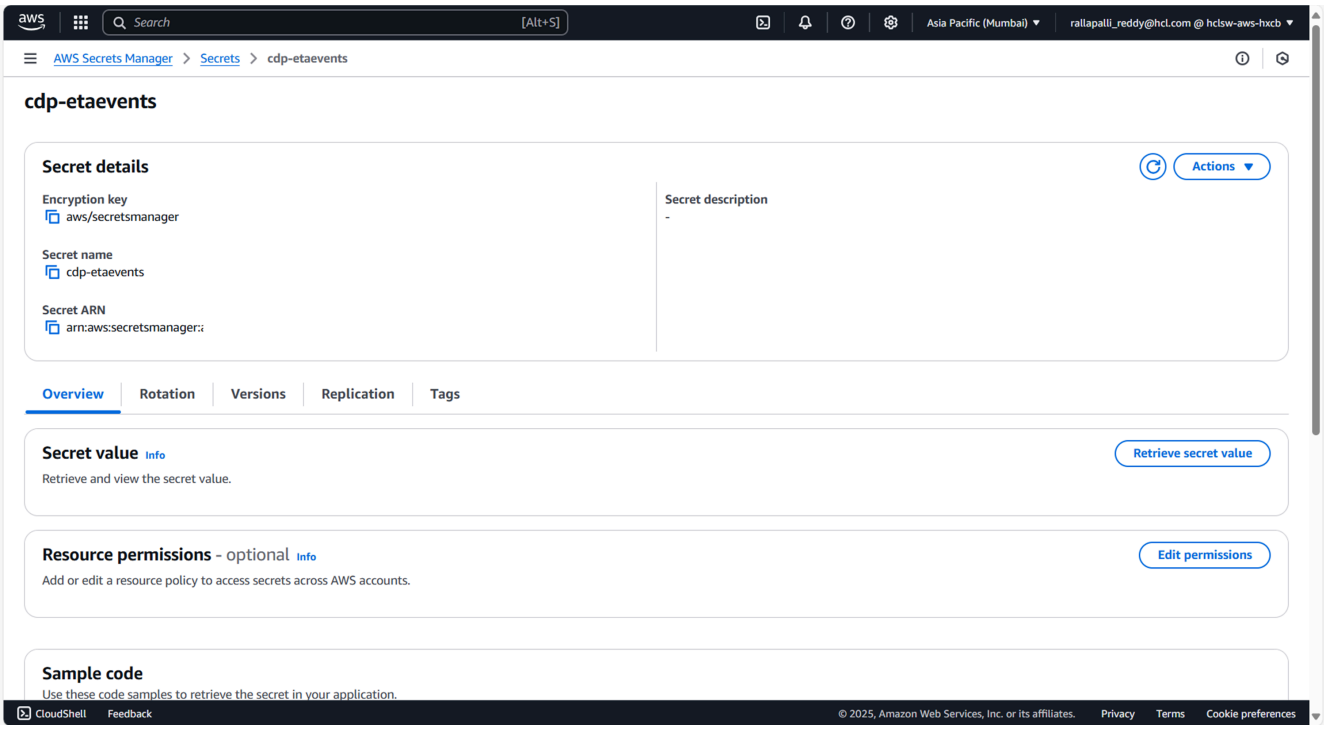


Deploying HCL CDP ETA Events

This section provides detailed instructions on how to deploy HCL CDP Etaevents using the Devtron in the AWS.

Prerequisites

Make sure to create a cdp-etaevents secret in the AWS secret manager before deploying CDP ETA Events.



To create a CDP Etaevents secret in the AWS secret manager , follow the steps below:

1. Create a cdp-etaevents secret sample key and value, and update the actual values as shown below.

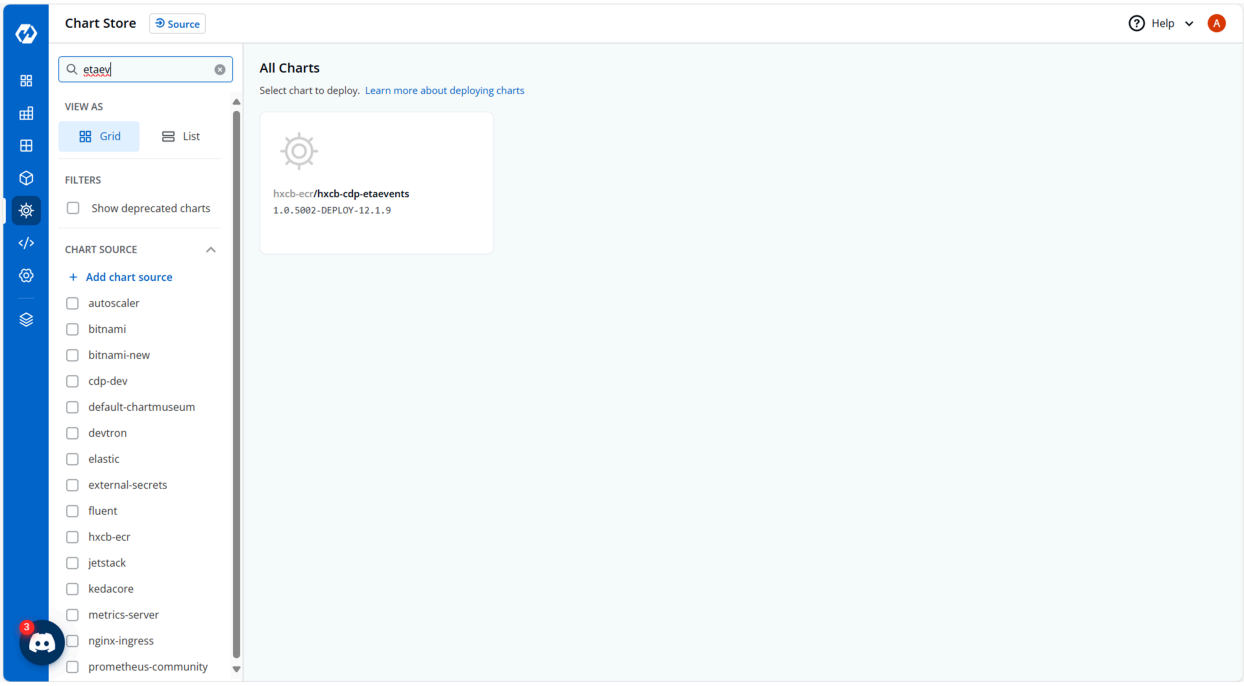
```
{
  "KafkaConsumerConfig": "",
  "KafkaProducerConfig": "",
  "REDIS_SERVER_IP": "",
  "REDIS_SERVER_PORT": "6379",
  "KAFKA_PRODUCER_HOSTNAME": "etaeventshclhost",
  "TSDB_HOST": "",
  "TSDB_PORT": "80",
  "KAFKA_TOPIC_CONSUMER_DEFAULT_BATCH_TIMEOUT_MS": "1000,1000,1000",
  "KAFKA_TOPIC_CONSUMER_DEFAULT_BATCHTIMEOUT_TRIGGER": "5000,5000,5000",
  "MIN_KAFKA_MESSAGE_SEGMENTS": "CeleryETALem::4|ETACallback::2",
  "KAFKA_MESSAGE_TIMESTAMP_INDEX": "CeleryETALem::1|ETACallback::0",
  "KafkaProducerConfigaps1": ""
}
```

2. Update ConfigMaps data with actual values, and create required Kafka topics before deployment.

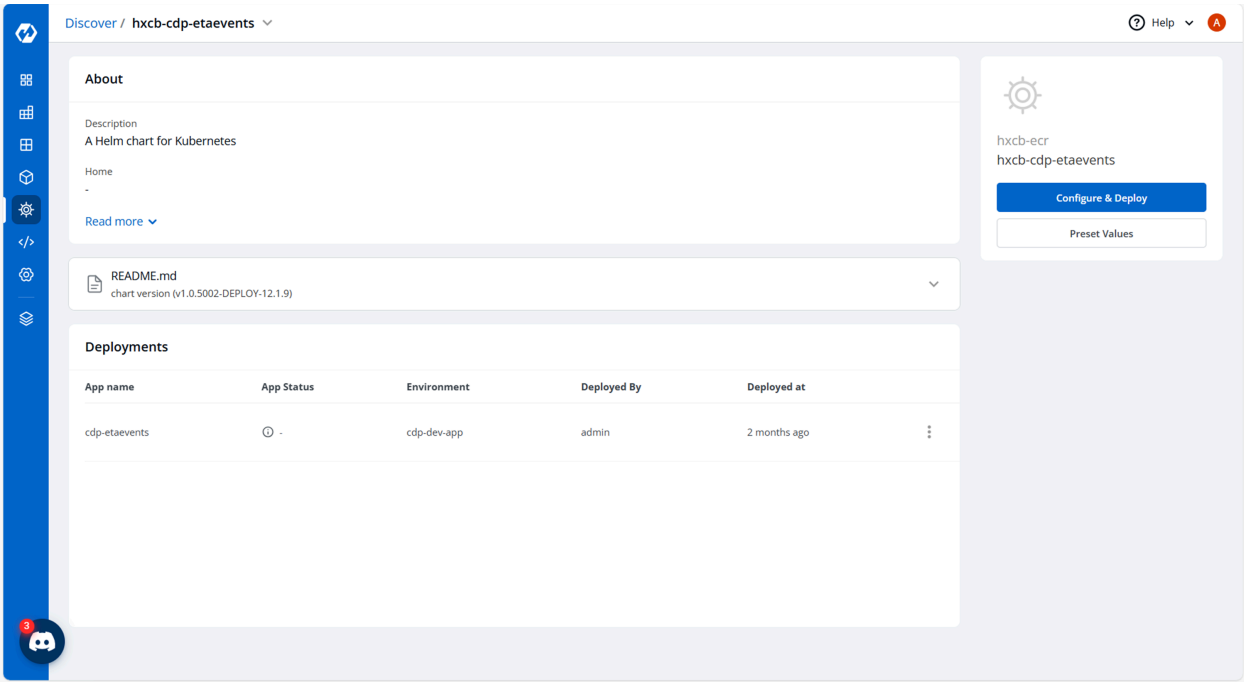
Deploy CDP Etaevents

To deploy the cdp-etaevents using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-etaevents chart to deploy.



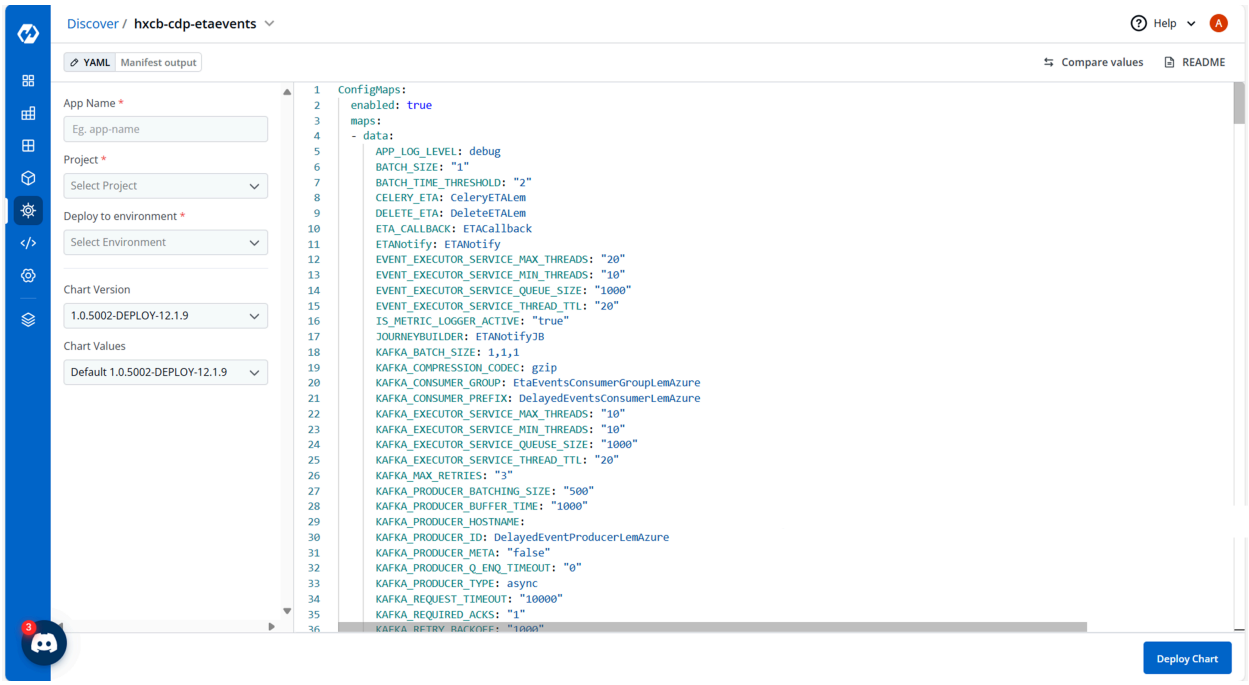
2. Now, configure and deploy the cdp-etaevents charts.



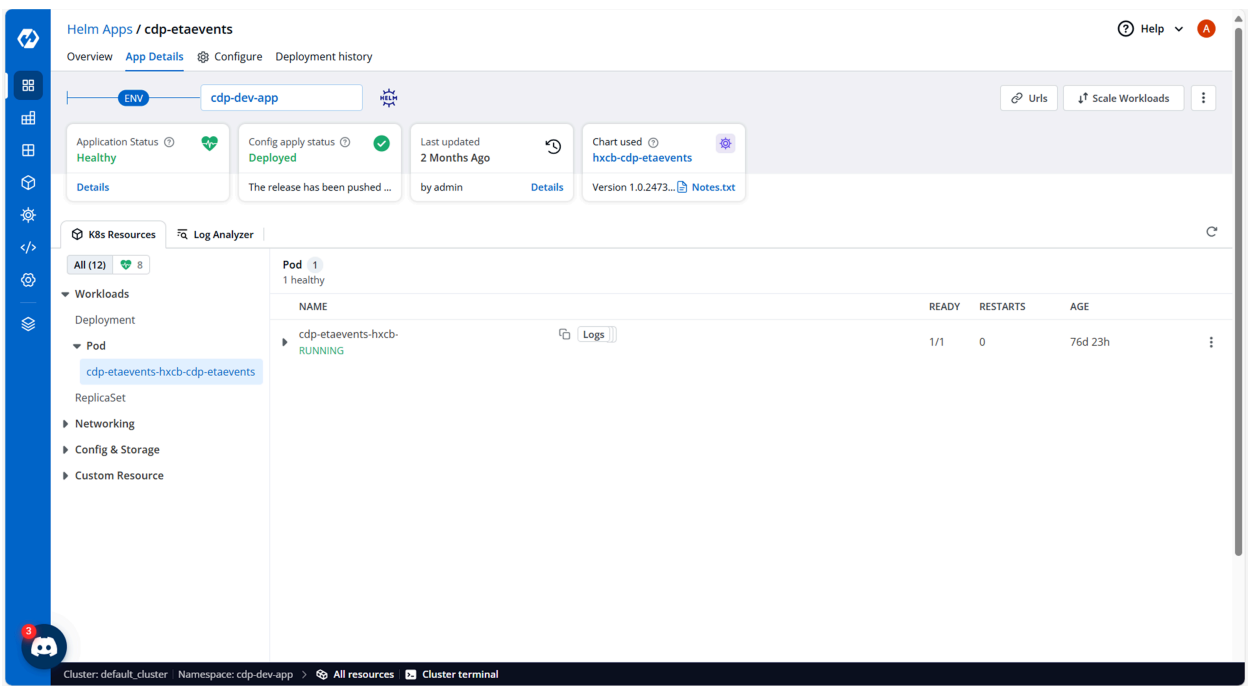
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
APP_LOG_LEVEL: debug
ROOT_LOG_LEVEL: WARN
LOG_LEVEL_UTILS: debug
KAFKA_CONSUMER_GROUP: EtaEventsConsumerGroupLemAzure
```

```
KAFKA_CONSUMER_PREFIX: DelayedEventsConsumerLemAzure
KAFKA_TOPIC_LIST: CeleryETALem,DeleteETALem,ETACallback
KAFKA_TOPIC_CONSUMERS: 3,3,3
KAFKA_BATCH_SIZE: 1,1,1
ETANotify: ETANotify
TRIGGER: ETANotify
JOURNEYBUILDER: ETANotifyJB
TsdbEventsSent: TsdbEventsSent
KAFKA_PRODUCER_META: "false"
TsdbEventsRcvdMeanLatency: TsdbEventsRcvdMeanLatency
TsdbEventsRcvdLatency: TsdbEventsRcvdLatency
TsdbEventsRcvd: TsdbEventsRcvd
KAFKA_EXECUTOR_SERVICE_MIN_THREADS: "10"
KAFKA_EXECUTOR_SERVICE_MAX_THREADS: "10"
KAFKA_EXECUTOR_SERVICE_THREAD_TTL: "20"
KAFKA_EXECUTOR_SERVICE_QUEUSE_SIZE: "1000"
BATCH_SIZE: "1"
BATCH_TIME_THRESHOLD: "2"
CELERY_ETA: CeleryETALem
DELETE_ETA: DeleteETALem
ETA_CALLBACK: ETACallback
KAFKA_PRODUCER_ID: DelayedEventProducerLemAzure
KAFKA_SERIALIZER_CLASS: kafka.serializer.StringEncoder
KAFKA_REQUIRED_ACKS: "1"
KAFKA_REQUEST_TIMEOUT: "10000"
KAFKA_PRODUCER_TYPE: async
KAFKA_COMPRESSION_CODEC: gzip
KAFKA_MAX_RETRIES: "3"
KAFKA_RETRY_BACKOFF: "1000"
KAFKA_PRODUCER_BATCHING_SIZE: "500"
KAFKA_PRODUCER_BUFFER_TIME: "1000"
KAFKA_PRODUCER_Q_ENQ_TIMEOUT: "0"
KAFKA_PRODUCER_HOSTNAME:
 b-1.hclswhsshxcbdevmsk.us1d78.c4.kafka.ap-south-1.amazonaws.com:9096,b-2.hclswhsshxcbdevmsk.us1d78.c4.k
afka.ap-south-1.amazonaws.com:9096
PRODUCER_BATCH_SIZE: "200"
PRODUCER_BATCH_TIME_THRESHOLD: "2"
PRODUCER_POOL_SIZE: "10"
EVENT_EXECUTOR_SERVICE_MIN_THREADS: "10"
EVENT_EXECUTOR_SERVICE_MAX_THREADS: "20"
EVENT_EXECUTOR_SERVICE_THREAD_TTL: "20"
EVENT_EXECUTOR_SERVICE_QUEUE_SIZE: "1000"
REDIS_CONNECTION_TIMEOUT: "2000"
TTL_MULTIPLIER: "1.2"
IS_METRIC_LOGGER_ACTIVE: "true”
```



4. On successful deployment, validate the deployment as shown below.

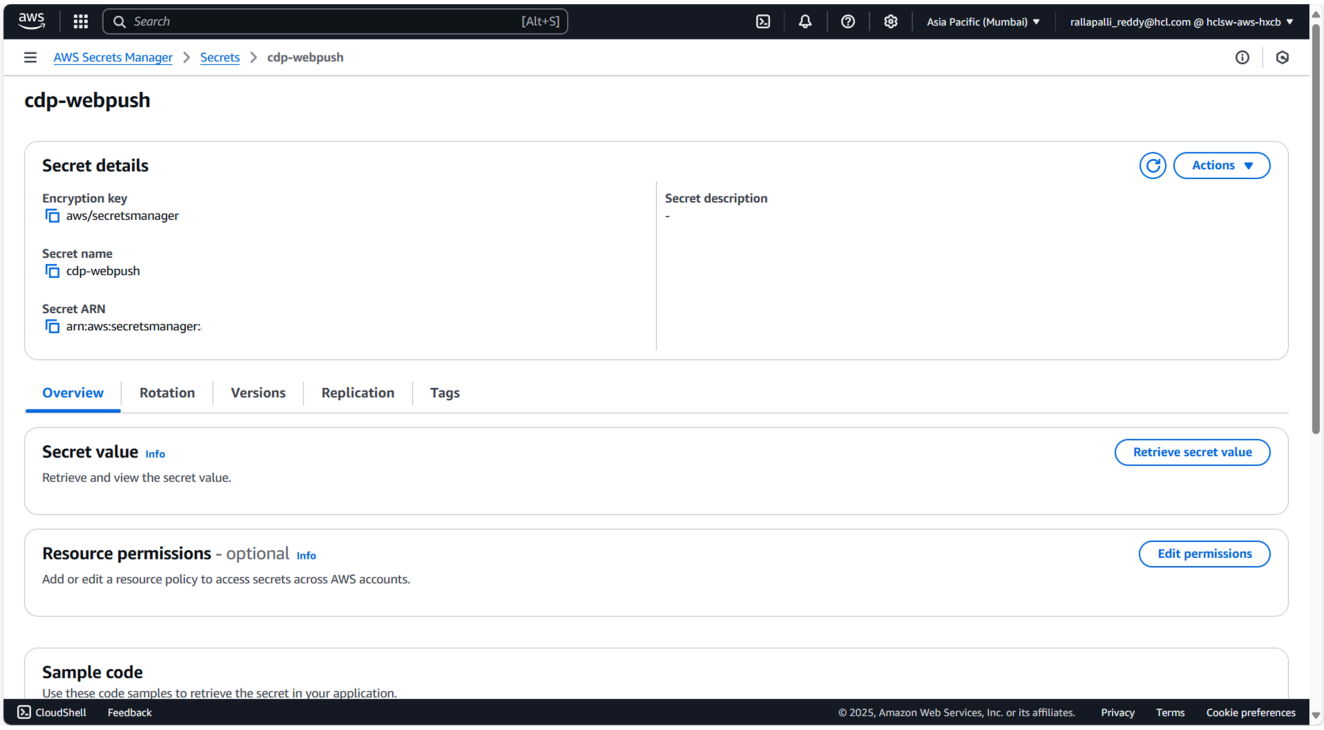


Deploying HCL CDP Webpush

This section provides detailed instructions on how to deploy HCL CDP Webpush using the Devtron in the AWS.

Prerequisites

Make sure to create a cdp-webpush secret in the AWS secret manager before deploying the CDP Webpush.



To create a CDP Webpush secret in the AWS secret manager , follow the steps below:

1. Create a cdp-webpush secret sample key and value, and update the actual values as shown below.

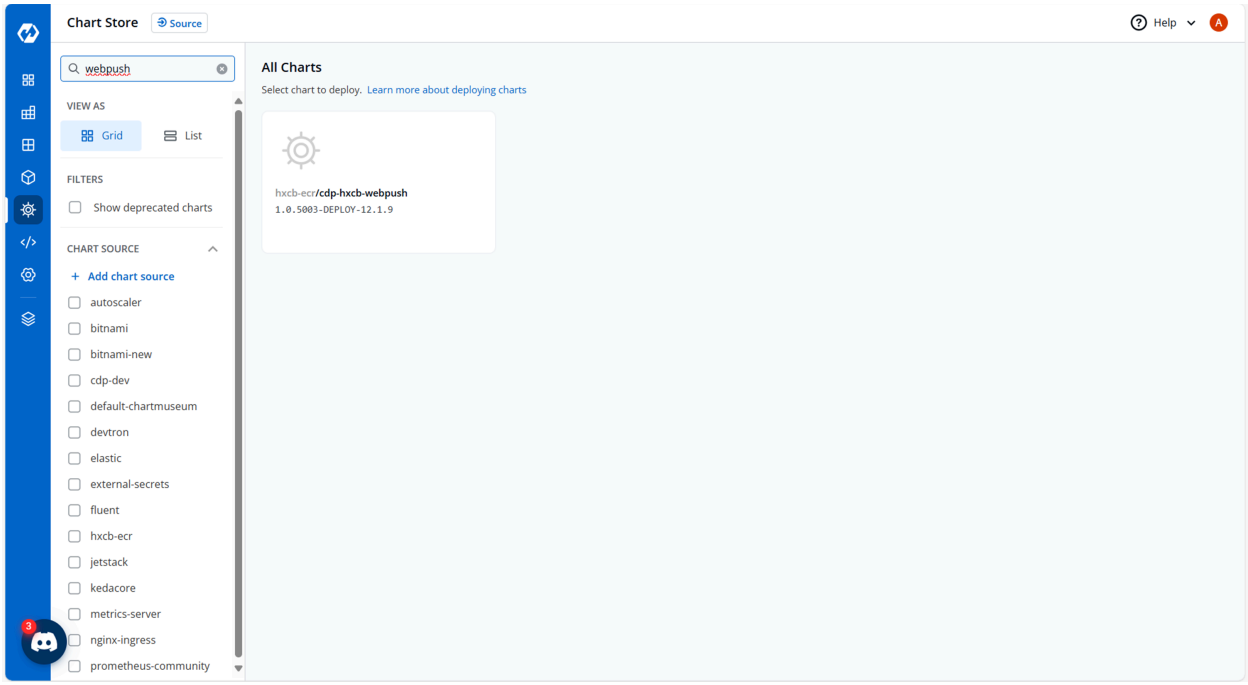
```
{
  "6525": "",
  "6525_vapidPrivateKey": "X",
  "6525_vapidPublicKey": "",
  "kafkaConsumerGroupId": "dev-web-push-us",
  "KafkaClientId": "web-push-us",
  "KafkaBrokerList": "['']",
  "TsdbhostString": "",
  "saslMechanism": "scram-sha-512",
  "KafkaUsername": "",
  "kafkaConsumerTopic": "web_push_core",
  "KafkaPassword": "",
  "6526_vapidPublicKey": ""
}
```

2. Update the ConfigMaps data with actual values, and create required Kafka topics before deployment.

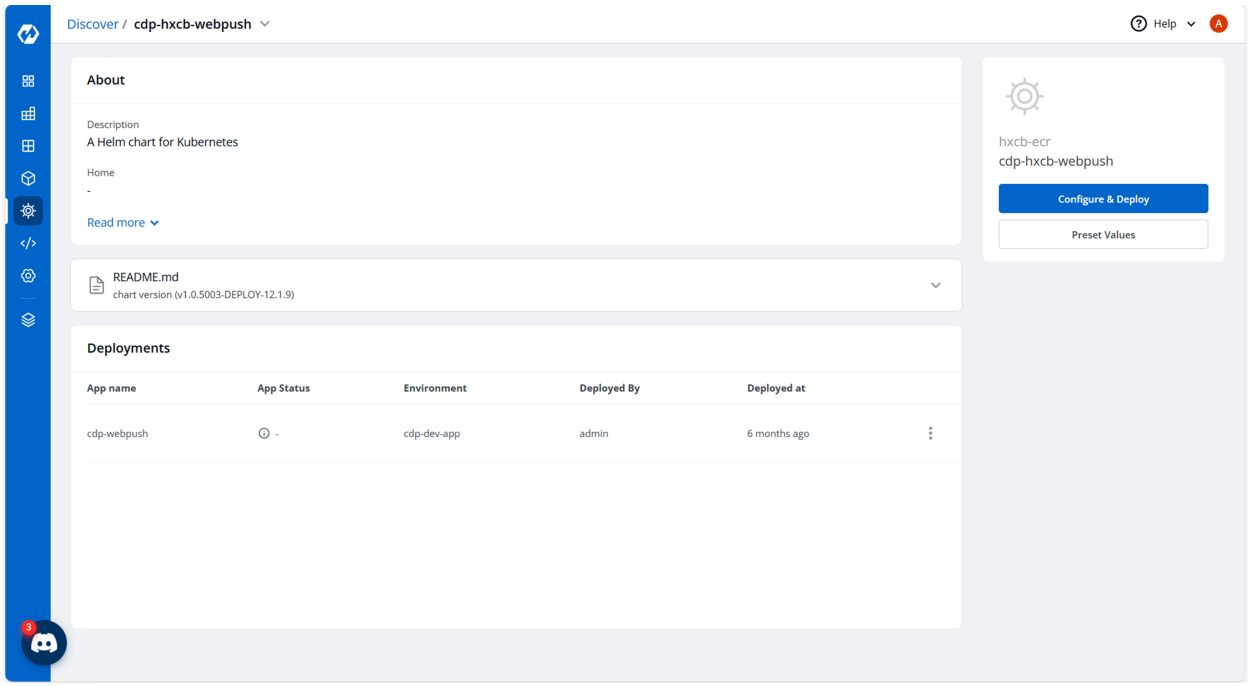
Deploy CDP Webpush

To deploy the cdp-webpush using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-webpush chart to deploy.



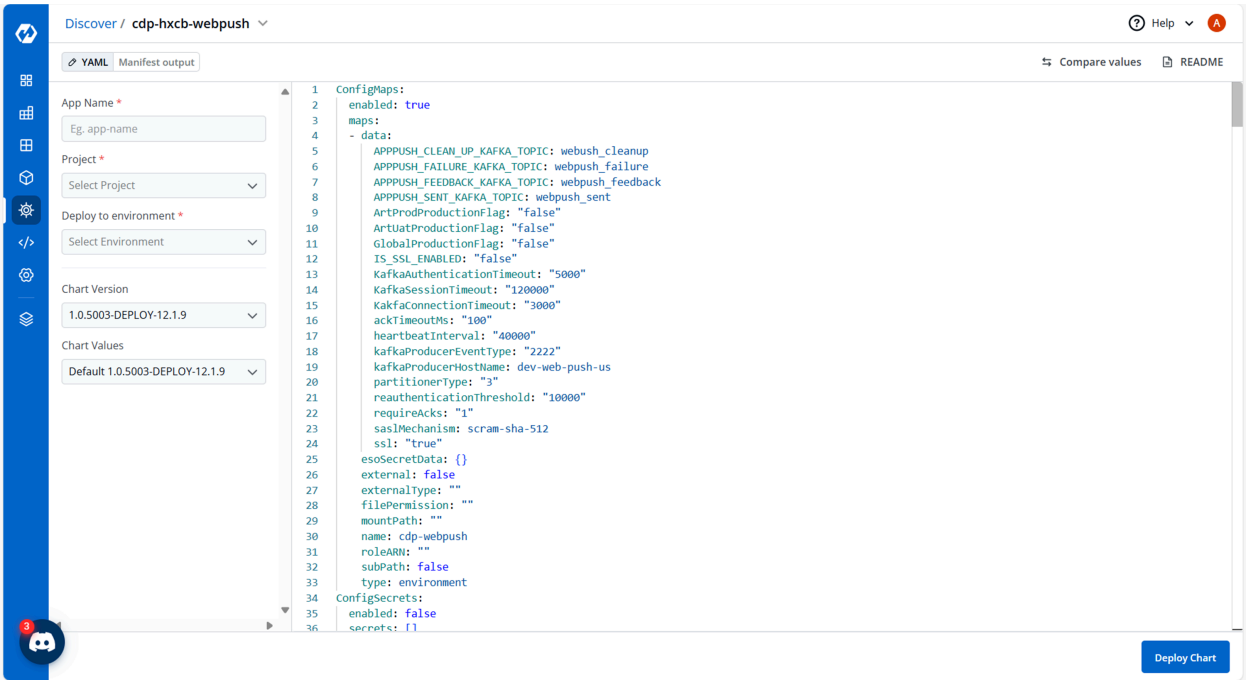
2. Now, configure and deploy the cdp-webpush charts.



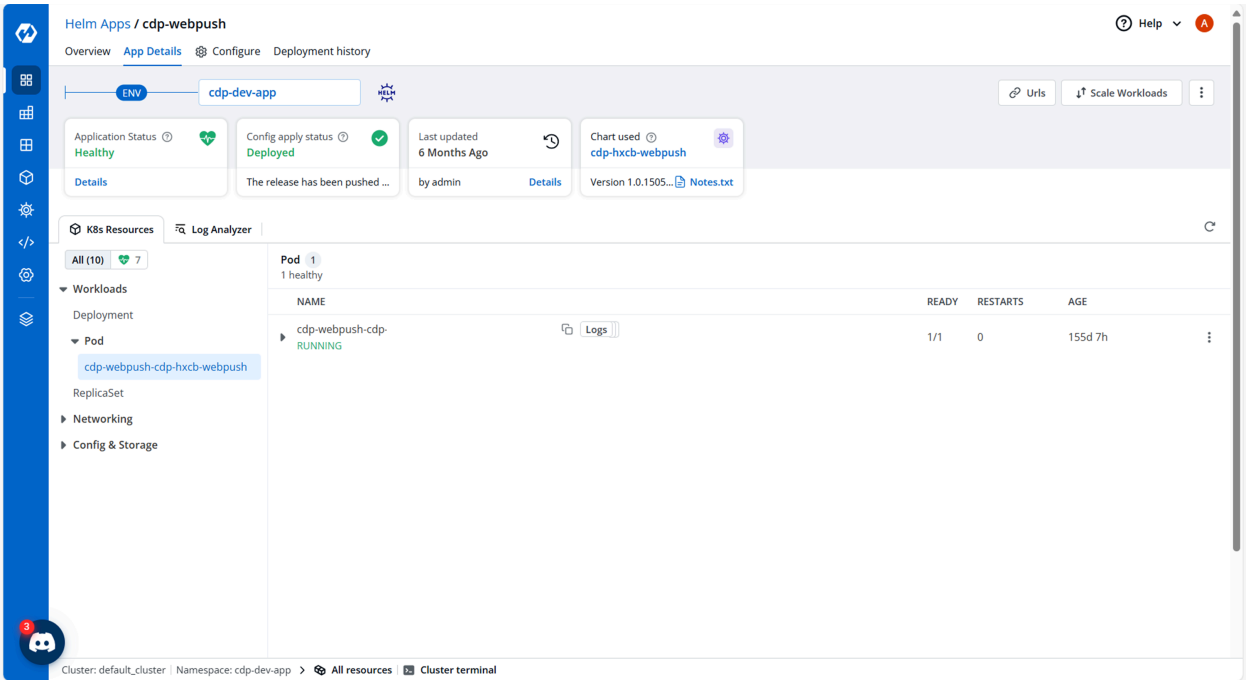
3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
saslMechanism: scram-sha-512
kafkaProducerHostName: dev-web-push-us
KakfaConnectionTimeout: "3000"
KafkaAuthenticationTimeout: "5000"
```

```
KafkaSessionTimeout: "120000"
heartbeatInterval: "40000"
reauthenticationThreshold: "10000"
requireAcks: "1"
ackTimeoutMs: "100"
partitionerType: "3"
kafkaProducerEventType: "2222"
GlobalProductionFlag: "false"
ArtUatProductionFlag: "false"
ArtProdProductionFlag: "false"
APPPUSH_SENT_KAFKA_TOPIC: webpush_sent
APPPUSH_FAILURE_KAFKA_TOPIC: webpush_failure
APPPUSH_FEEDBACK_KAFKA_TOPIC: webpush_feedback
APPPUSH_CLEAN_UP_KAFKA_TOPIC: webush_cleanup
IS_SSL_ENABLED: "false"
ssl: "true"
```



4. On successful deployment, validate the deployment as shown below.

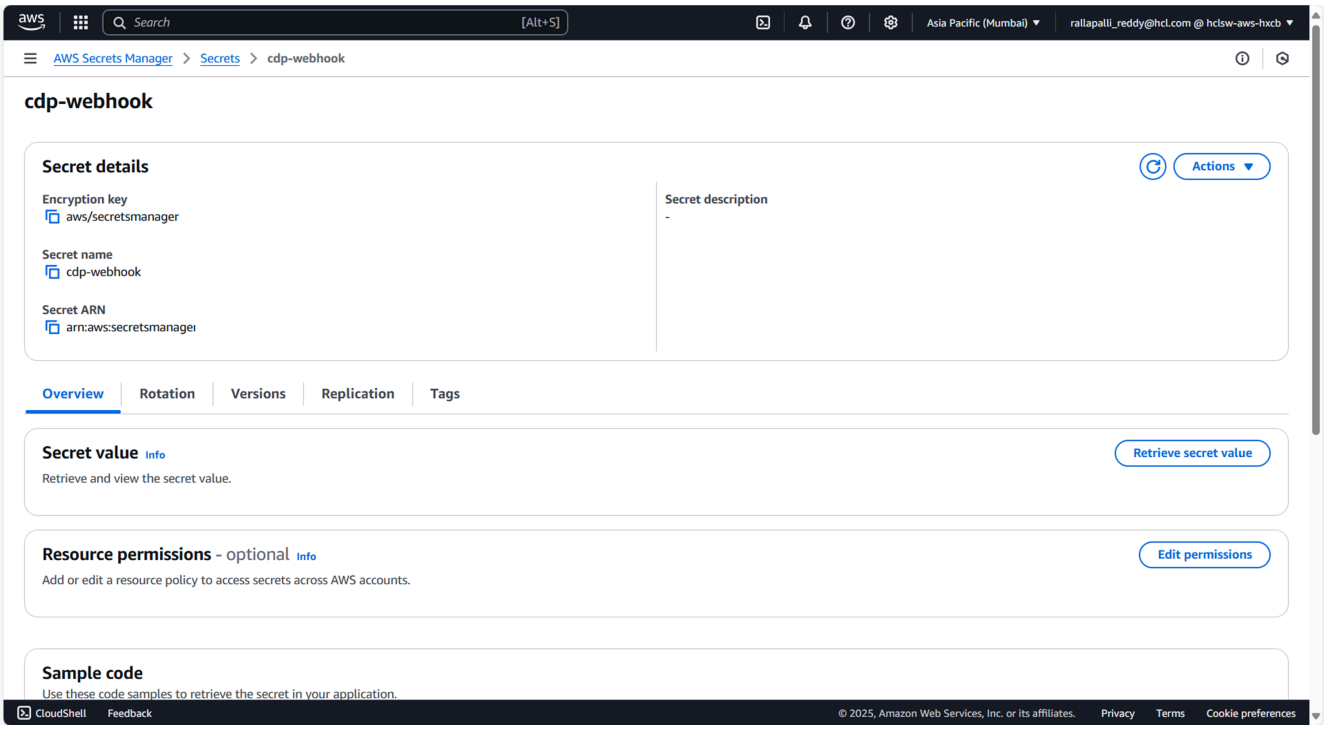


Deploying HCL CDP Webhooks

This section provides detailed instructions on how to deploy HCL CDP Webhooks using the Devtron in the AWS.

Prerequisites

Make sure to create a cdp-webhooks secret in the AWS secret manager before deploying CDP Webhooks.



To create a CDP Webhooks secret in the AWS secret manager , follow the steps below:

1. Create a cdp-webhooks secret sample key and value, and update the actual values as shown below.

```
{
  "RedisIp": "",
  "s3.accessKeyId": "",
  "s3.secretKey": "",
  "s3.region": "",
  "bucket": "",
  "VrmDbUsername": "cdpapp_ro",
  "VrmDbPassword": "",
  "VrmDbUrl": "jdbc:mariadb://:3307/vrm",
  "KafkaConsumerConfig": "",
  "KafkaUsername": "hclsw-hxcb-hss-cdp",
  "KafkaConsumerGroup": "",
  "KafkaTopicList": "",
  "KafkaPassword": "Yd8qyhZcnDJWQJL386tfxQ",
  "KafkaTopicConsumers": "",
  "KafkaTopicConsumerBatchSize": "",
  "KafkaTopicConsumerDefaultBatchTimeoutMS": "",
  "KafkaTopicConsumerDefaultBatchTimeoutTrigger": "",
  "s3objectPrefix": "trapi_req_metadata/",
  "smsClickTrackerEndpoint": "ck.dev.hxcb.now.hclsoftware.cloud",
  "emailClickTrackerEndpoint": "",
  "openTrackerEndpoint": "",
  "VrmDbDriver": "org.mariadb.jdbc.Driver",
  "from": "",
  "to": "",
  "KafkaProducerPoolSize": "16",
  "ProducerEventType": "8888",
  "KafkaProducerHostName": "",
```

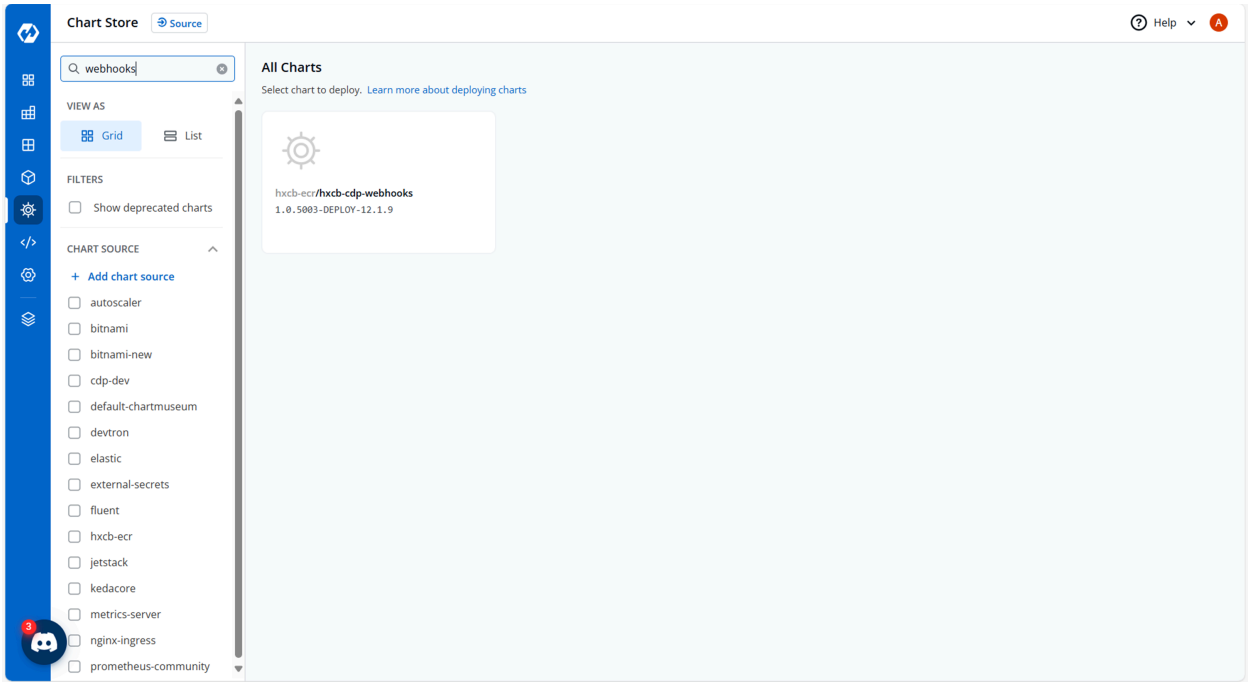
```
  "indexKeyMapping":
 "0:openCount,1:mid,3:sentTime,4:deliveryTime,5:SOFTBOUNCE,6:HARDBOUNCE,7:noOfMsgsSent,8:noOfMsgsProcess
ed,9:statusCode",
  "TSDBHost": "",
  "TSDBPort": "80",
  "KafkaProducerConfig": "",
  "KafkaProducerEnabled": "true",
  "KafkaConsumerEnabled": "false",
  "impressionTopic": "impression",
  "KafkaProducerMeta": "false",
  "KafkaProducerEventType": "impression"
}
```

2. Update the ConfigMaps data with actual values, and create required Kafka topics before deployment.

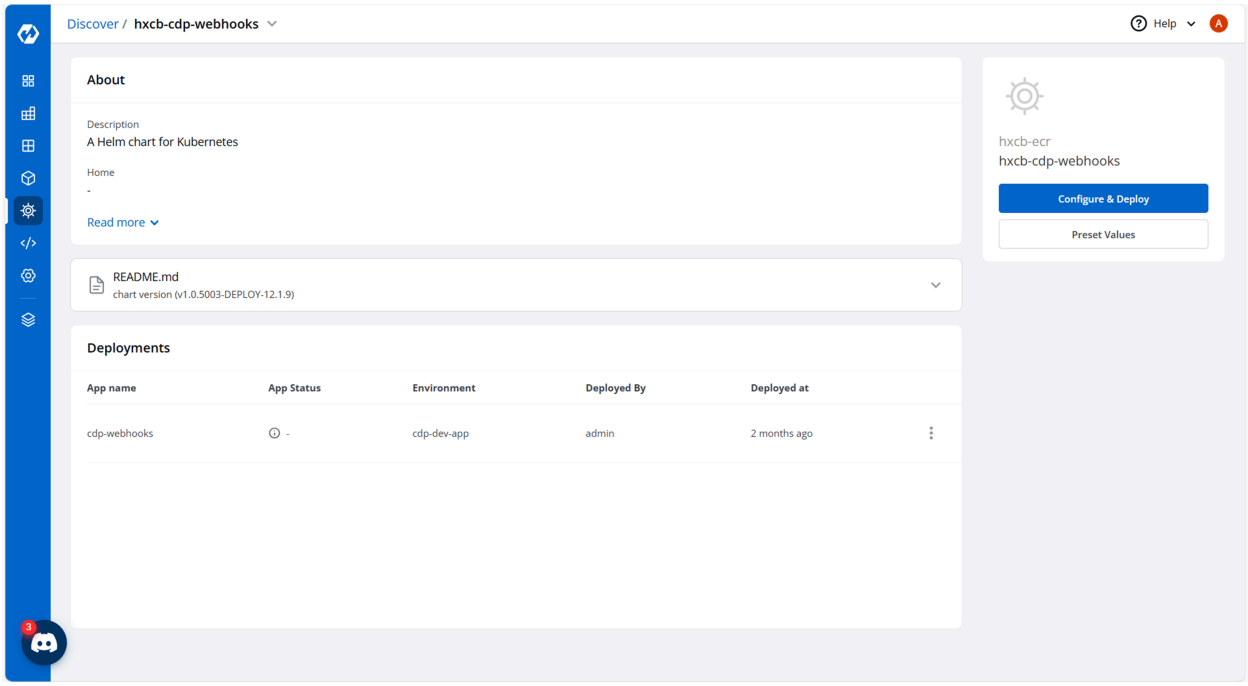
Deploy CDP Etaevents

To deploy the CDP cdp-webhooks using Devtron in the AWS platform, follow the steps below:

1. Navigate to the Devtron **Chart Store**, and select the cdp-webhooks chart to deploy.



2. Now, configure and deploy the cdp-webhooks charts.

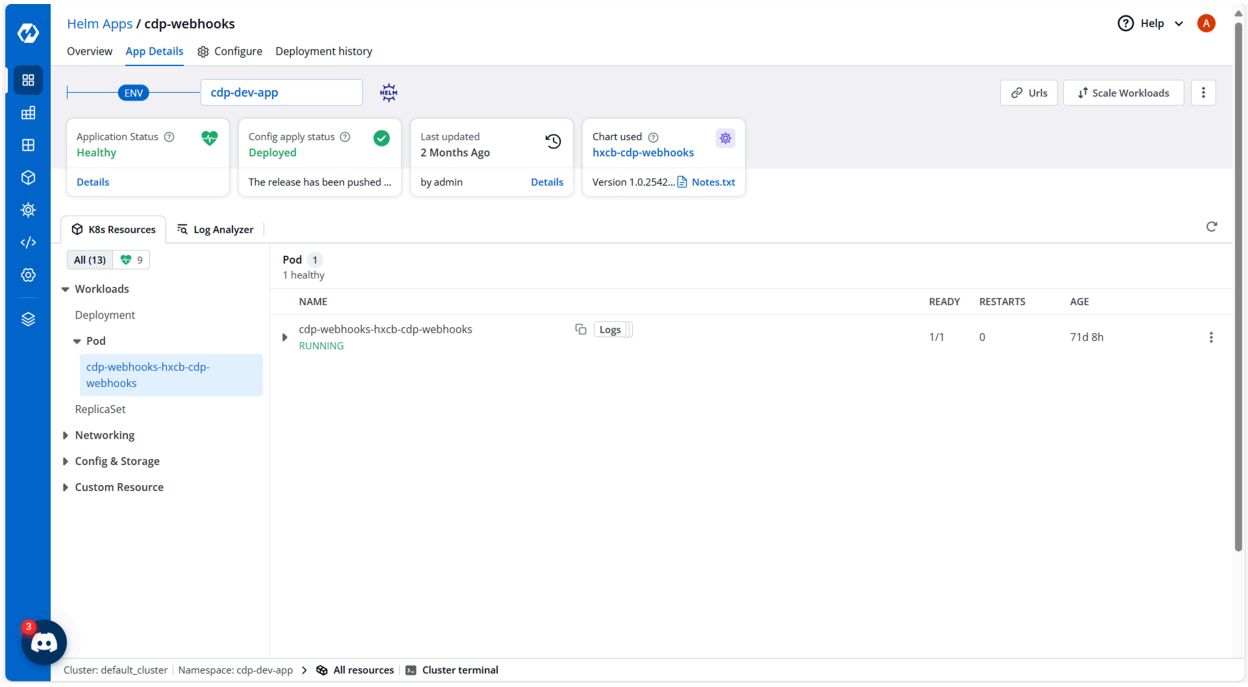


3. In the YAML section, update the ConfigMap using the below details and deploy the chart.

```
APP_TYPE: webhooks
APP_REGION: ap-south-1
APP_INFRA: k8s
DBRefreshIntervalMS: "60000"
RedisPort: "6379"
RedisMaxConnections: "20"
RedisKeyTTLInSec: "7776000"
ThreadExecutorMinThreads: "16"
ThreadExecutorMaxThreads: "32"
ThreadExecutorThreadTTL: "5"
ThreadExecutorQueueSize: "1000"
connectionKeepAliveSeconds: "600"
connectionCount: "5"
readTimeoutSeconds: "1"
writeTimeoutSeconds: "10"
connectTimeoutSeconds: "5"
s3.protocol: HTTPS
s3.maxErrorRetry: "3"
s3.maxConnections: "10"
s3.maxRWRetry: "3"
s3.timeout.socket: "100"
s3.timeout.connection: "500"
s3.timeout.clientExecution: "5000"
s3.timeout.request: "100"
KafkaExecutorServiceMinThreads: "16"
KafkaExecutorServiceMaxThreads: "32"
KafkaExecutorServiceThreadTTL: "5"
KafkaExecutorServiceQueuseSize: "1000"
reportingIntervalInMin: "55"
LOG_LEVEL_APP: debug
LOG_LEVEL_METRICS: error
```

```
LOG_LEVEL_ROOT: error
LOG_LEVEL_UTILS: error
emailType: SES
```

4. On successful deployment, validate the deployment as shown below.



Deploying VM

This section provides detailed instructions on how to deploy VM in the AWS.

TC-Redis

To install TC-Redis, follow the steps below:

1. **Update and install Redis**:

```
dnf update
dnf install epel-release
dnf install redis
```

2. **Start Redis**:

```
systemctl status redis
systemctl start redis
```

3. **Enable remote access**: Edit `/etc/redis.conf`, comment.

```
cd /etc
vi redis.conf
comment #bind 127.0.0.1
add bind 0.0.0.0
```

Rabbitmq-Celery

To install the Rabbitmq-Celery, follow the steps below:

1. **Update and install Redis**:

```
dnf update
dnf install epel-release
dnf install redis
```

2. **Start Redis**:

```
systemctl status redis
systemctl start redis
```

3. **Enable remote Redis access**: Edit `/etc/redis.conf`, comment.

```
cd /etc
vi redis.conf
comment #bind 127.0.0.1
add bind 0.0.0.0
```

4. **Install dependencies**: `dnf install socat logrotate -y`.
5. **Install Erlang**:

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

6. **Add RabbitMQ repo and install**:

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

7. **Enable and start RabbitMQ**:

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

8. **Enable management plugin**: `rabbitmq-plugins enable rabbitmq_management`.
9. **Configure user/vhost/permissions**: Create `/etc/rabbitmq/definitions.json` with user `celery`.

```
vi definitions.json
{
    "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": []
}
```

10. **Point config to definitions file**: Add `load_definitions = /etc/rabbitmq/definitions.json` in `rabbitmq.conf`.

Rabbitmq-UI

To install the Rabbitmq-UI, follow the steps below:

1. **Update and install Redis**:

```
dnf update
dnf install epel-release
dnf install redis
```

2. **Start Redis**:

```
systemctl status redis
systemctl start redis
```

3. **Enable remote access**: Edit `/etc/redis.conf`, comment.

```
cd /etc
vi redis.conf
comment #bind 127.0.0.1
add bind 0.0.0.0
```

4. **Install dependencies**: `dnf install socat logrotate -y`.
5. **Install Erlang & RabbitMQ**:

```
curl -LO -C -
 https://github.com/rabbitmq/erlang-rpm/releases/download/v26.2.5/erlang-26.2.5-1.el8.x86_64.rpm
sudo dnf install ./erlang-26.2.5-1.el8.x86_64.rpm
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.rpm.sh | sudo bash
dnf install -y rabbitmq-server
```

6. **Enable and start RabbitMQ**:

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

7. **Enable management plugin**: `rabbitmq-plugins enable rabbitmq_management`.
8. **Create definitions file**: `/etc/rabbitmq/definitions.json` with user `celery`.

```
vi definitions.json
{
    "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": []
}
```

9. **Link config file**: `load_definitions = /etc/rabbitmq/definitions.json` in `rabbitmq.conf`.

Scheduler

To install the Scheduler, follow the steps below:

1. **Create data dir and add jar**:

```
mkdir /data
          cd /data
          cp scheduler.jar
```

2. **Create config structure**:

```
cd /data
  mkdir /config/scheduler
```

```
  create application.properties
  create kill-switch.properties
  create log4j2.xml
  create 'Report Error.html'
```

3. **Create TSDB config files**:

```
mkdir tsdb
   inside tsdb -> create metricDumper.ini
                  create metric.ini
```

4. **Set permissions**:

```
chown cdpservice:cdpservice /data
        chmod 777 /data/config
```

5. **Create symbolic link**:

```
mkdir -p /disk1
        ln -s /data/config /disk1
```

Chapter 4. HCL CDP Hardware Sizing Details For AWS

This section outlines the minimum hardware requirements to guide AWS infrastructure sizing for the deployment of HCL CDP in support of application deployment.

The objective of this guide is to provide an estimated baseline for the AWS infrastructure components—CPU, memory, storage, and network resources—necessary to achieve expected performance levels and ensure a reliable, efficient operational environment.

The hardware sizing details include:

- Number of servers per component
- Per-server specifications (vCPU, RAM, and Storage)
- Total resource footprint

These recommendations serve as a starting point for capacity planning and should be adapted based on workload characteristics, usage patterns, and scaling expectations.

Component-wise Hardware Summary

The following table provides a detailed breakdown of AWS infrastructure requirements for each system component. For every listed component, it outlines the number of servers allocated, the per-server hardware specifications (vCPU, RAM, and HDD), and the resulting total resource consumption. This granular view helps assess the compute, memory, and storage needs across the architecture and supports capacity planning, resource optimization, and scaling decisions.

Component	No. of Se rvers	vCPU (per server)	RAM (GiB)	HDD (GB)	Total vCPU	Total RAM	Total HDD
Pixel Listeners	1	2	4	100	2	4	100
SST HS - Zookeeper	1	2	4	100	2	4	100
SST HS - Aerospike	1	2	8	100	2	8	100
SST HS - Processor	1	2	4	100	2	4	100
SST HS - Offline Processor	1	2	4	100	2	4	100
Mongo	2	2	8	100	4	16	200
Mongo Connector	1	2	4	50	2	4	50

Component	No. of Servers	vCPU (per server)	RAM (GiB)	HDD (GB)	Total vCPU	Total RAM	Total HDD
RTS	1	1	4	100	1	4	100
RTS - Offline	1	1	4	100	1	4	100
Athena	-	-	-	-	-	-	-
Druid	1	8	64	200	8	64	200
UI/Core API	1	1	4	50	1	4	50
SFTP	1	1	4	200	1	4	200
DMP-Producer	1	1	4	100	1	4	100
PII Cache	1	2	8	100	2	8	100
MySQL Master	1	2	8	100	2	8	100
MySQL Slave	1	4	16	100	4	16	100
Trigger	1	1	4	100	1	4	100
Scheduler	1	4	16	100	4	16	100
Scheduler Core	1	2	4	100	2	4	100
Trigger Core	1	1	4	100	1	4	100
Scheduler Sender	1	1	4	100	1	4	100
Trigger Sender	1	1	4	100	1	4	100
Webhooks	1	2	4	100	2	4	100
TC Redis	1	2	8	100	2	8	100
Flip	1	1	4	100	1	4	100
Flip AS	1	1	8	100	1	8	100
Celery Worker	1	2	8	100	2	8	100
Celery Redis	1	2	8	100	2	8	100
Celery RMQ	1	2	8	100	2	8	100

Component	No. of Servers	vCPU (per server)	RAM (GiB)	HDD (GB)	Total vCPU	Total RAM	Total HDD
DI API	1	2	4	100	2	4	100
Prometheus	1	2	4	200	2	4	200
Grafana	1	1	4	50	1	4	50
Kafka Connectors	1	4	8	100	4	8	100
Kafka Zookeeper	1	4	8	100	4	8	100
Kafka Brokers	1	4	8	200	4	8	200
Kafka ControlCenter	1	2	4	100	2	4	100
Kafka SchemaRegistry	1	2	4	100	2	4	100
Neo4j Graph	1	4	8	100	4	8	100
Offline Segmentation	1	2	4	100	2	4	100
Audience Uploader	1	2	4	100	2	4	100
ES - Master Nodes	1	2	8	100	2	8	100
ES - Data Nodes	1	2	16	100	2	16	100
ES - Ingest Nodes	1	2	4	100	2	4	100
ES - Coordinating Nodes	1	2	8	100	2	8	100
ES - Kibana	1	2	8	100	2	8	100
FluentBit	2	2	4	100	4	8	200

Component	No. of Se rvers	vCPU (per server)	RAM (GiB)	HDD (GB)	Total vCPU	Total RAM	Total HDD
S3Log Uploader	2	4	8	200	8	16	400

Disclaimer - This guide illustrates the minimal hardware sizing requirement, based on various assumptions. This guide does not replace the need for specific Infrastructure details before the deployment. The mentioned consumptions are based on a specific scenario and hardware. Proper estimation should be done before buying the hardware. On top of the suggested hardware, appropriate buffer should be added for any expected peak/future load.

Category-wise Hardware Summary

Category	Components	Total vCPU	Total RAM (GiB)	Total HDD (GB)
Data Layer	Mongo, MySQL Master, MySQL Slave, Aerospike, TC Redis, Celery Redis, Kafka (all), Neo4j	30	80	1000
Processing Layer	Processor, Offline Processor, Celery Worker, Celery RMQ, Trigger, Scheduler, Core/Sender	27	68	900
Monitoring	Prometheus, Grafana	3	8	250
UI/API/Access	UI/Core API, SFTP, DI API, Athena (not specified), Flip, Flip AS	9	28	650
Logging & Search	Logging - ES (all), FluentBit	18	52	800
Uploaders	S3Log Uploader, Audience Uploader	10	20	500
Other	Pixel Listeners, PII Cache, RTS, RTS Offline, Webhooks, DMP Producer, Zookeeper, Druid	23	76	1150

Total Infrastructure Summary

Resource	Total
Total Servers	54
Total vCPUs	110
Total RAM (GiB)	372
Total HDD (GB)	5450