Cloud Native Unica V12.1.6
Installation and Configuration Guide

Contents

Chapter 1. Pre-installation configurations

Before installing or upgrading to Cloud Native Unica, you should complete some configurations.

The list of pre-installation or pre-upgrade configurations are as follows:

- Configure the resources for containers. For more information, see *Cloud Native Unica Getting Started Guide*.
- Ensure that you have installed Docker Enterprise version 19.xx.x. For more information, see Docker documentation.
- Ensure that you have installed Kubernetes. For more information, see Kubernetes documentation.
- Verify if:
  - you have configured a Kubernetes cluster.
  - the Kubernetes environment has the appropriate image enforcement policy to allow access to the required repositories.
  - the application server is setup. For more information, see see one of the following topics based on the application server that is installed on your system:
    - Setup for Apache Tomcat on page 2
    - Setup for Red Hat JBoss Enterprise Application Platform on page 2
      - Configuring JBoss for Cloud Native Unica on page 3
    - Setup for Oracle WebLogic on page 2
      - Configuring WebLogic for Cloud Native Unica on page 3
  - the database is setup. For more information, see one of the following topics based on the application server that is installed on your system:
    - For Apache Tomcat on page 4
    - For Red Hat JBoss Enterprise Application Platform on page 5
    - For Oracle WebLogic on page 6
- Ensure that you have installed Helm. For more information, see Helm documentation.

Avoiding timeout issues

To avoid timeout issues, perform the following steps.

1. Access the path *<chart-location>*/unica/.
2. Open the file `values-local.yaml`.
3. Add the following lines of code in the annotations section within ingress.

   ```
   nginx.ingress.kubernetes.io/proxy-connect-timeout: "30"
   nginx.ingress.kubernetes.io/proxy-read-timeout: "1800"
   nginx.ingress.kubernetes.io/proxy-send-timeout: "1800"
   nginx.ingress.kubernetes.io/proxy-body-size: 50m
   ingress.kubernetes.io/proxy-body-size: 50m
   ```
4. Save the changes.

Application server setup

Cloud Native Unica supports Apache Tomcat®, Red Hat® JBoss® Enterprise Application Platform (EAP), and Oracle® WebLogic Server application servers.

Setup for Apache Tomcat

You do not have to set up the Apache Tomcat application server as it is embedded in the Cloud Native Unica image.

1. Edit the `setenv.sh` file for the respective product instances script in the `bin` directory under your Tomcat instances directory to add the following Java options:
   ```
   -Dfile.encoding=UTF-8
   -Dclient.encoding.override=UTF-8
   ```
2. If you are deploying on a non-production setup, add:
   ```
   -DENABLE_NON_PROD_MODE=true
   ```
3. If you are deploying on a production setup, remove the Java option `-DENABLE_NON_PROD_MODE=true`, or set it to `false`.
4. After saving the changes, restart the Apache Tomcat server.

Setup for Red Hat JBoss Enterprise Application Platform

To setup the JBoss application server, place the JBoss EAP `ZIP` file on the mount location and configure the path in the Helm chart.

1. For JBoss server, edit the `standalone.conf` script in the `JBoss/bin` directory to add the following Java options to `JAVA_VENDOR`:
   ```
   -Dfile.encoding=UTF-8
   -Dclient.encoding.override=UTF-8
   -Djboss.as.management.blocking.timeout=3600
   ```
2. If you are deploying on a non-production setup, add:
   ```
   -DENABLE_NON_PROD_MODE=true
   ```
3. If you are deploying on a production setup, remove the Java option `-DENABLE_NON_PROD_MODE=true`, or set it to `false`.
4. After saving the changes, restart the JBoss server.

Setup for Oracle WebLogic

Install Oracle WebLogic Server on the shared filesystem.

1. For WebLogic server, edit the `setDomainEnv` script in the `bin` directory, within the WebLogic domain directory, to add the following Java options to `JAVA_VENDOR`:
2. If you are deploying on a non-production setup, add:
3. If you are deploying on a production setup, remove the Java option `-DENABLE_NON_PROD_MODE=true`, or set it to `false`.
4. After saving the changes, restart the WebLogic server.

Configuring application servers for Cloud Native Unica

You must perform additional configurations on JBoss and WebLogic.

Configuring JBoss for Cloud Native Unica

To use JBoss with Cloud Native Unica, complete the following steps:

1. Open the file `common-configMap.yaml`. To locate the file, access the `JBOSSOracle/unica/templates/` location.
2. For the **_JBOSS_ZIP_LOCATION** parameter, provide the folder name, residing within the `HOME` folder, containing the JBoss installation `ZIP` file. For example, `/docker/unica/JBossZip/JBOSS.Zip`.
3. For the **_JBOSS_ZIP_NAME_** parameter, provide the name of the JBoss installer `ZIP` file. For example, `jboss-eap-7.1.0.zip`.
4. For the **_DEST_JBOSS_UNZIP_LOCATION_** parameter, provide the absolute directory location where you want to install JBoss. For example, if you want to install JBoss inside the container, provide the value `/opt`. If you want to install JBoss in a mapped shared folder, provide the value `/docker/unica`.
5. For the **_DEST_UNZIP_FOLDER_** parameter, provide a folder name that contains the unzipped contents of the JBoss installer `ZIP` file. For example, if the `ZIP` file is `jboss-eap-7.1.0.zip` and the folder within the `ZIP` file is `jBoss710`, provide the value `jBoss710`.

Results

Completing the earlier mentioned configuration will automatically install JBoss and the required Unica component.

Configuring WebLogic for Cloud Native Unica

Your system should have WebLogic installed to use it with Cloud Native Unica. Cloud Native Unica uses the utilities from WebLogic to create a domain for the required Unica component.

About this task

To use WebLogic with Cloud Native Unica, complete the following steps:

1. For the **JAVA_HOME_WEBLOGIC** parameter, WebLogic requires Oracle JDK. The value of this setting is the `HOME` location of the JDK used for the existing WebLogic installation. For example, `/docker/unica/jdk18_oracle`.
2. For the **WLS_HOME_DIR** parameter, provide the home directory of the WebLogic server installation. For example, `/docker/unica/oracle_products/middleware/wlse`.
3. For the **WLS_DOMAIN_LOCATION** parameter, provide the fully qualified path of the directory under which the domains for the products will be created. For example, `/docker/unica/wlsdomains`.

   CAUTION: The creation of a new pod creates a directory for the domain of that Pod. If you discard a pod, the directory is not deleted automatically. You should delete such directories as they consume a lot of disk space.

Database setup

You need to set up the database before you begin installation.

You can setup the database in one of the following ways.

- Use your database Docker image
- Connect to an external database system

In case of Managed Kubernetes Clusters on Cloud, the system data and the customer data must reside on Cloud.

If your database resides in an external system, the configuration of the following parameters, in Unica Helm chart, is mandatory.

- Database Users
- Tablespace Users
- Operating System Users

The database can reside within Kubernetes cluster. If the database resides within the Kubernetes cluster, use any available database image, and edit the Unica Helm chart. Ensure that user creation is complete before the Cloud Native solution starts.

For Apache Tomcat

To use Tomcat, within the cluster or with an external database, complete the following steps:

1. Download Cloud Native Unica images and Helm Chart.
2. Create `Databases` and `Users` and enter those details in the Unica Helm Chart.

If you set the Database as a sub-chart in Unica chart, you can completely automate data import using Shell scripts. For import, data should be available on the Database container mount point. You can also place the data after the container starts. Ensure that Database configuration and user creation activity is completed before running the Unica chart.

For auto-installation of database client on listener pod or container, complete the following steps:

Note: The commands and filenames are mentioned specific to Oracle database. Provide appropriate values based on the the database you use.

1. Place the Oracle client installer, named `linuxamd64_12102_client.zip`, inside the `/tmp` folder.
2. To extract the installer file, run the unzip command.

   A new folder, named `client` is created in the location `/tmp`.
3. Run the following command:

   cp /tmp/client/response/client_install.rsp /tmp/oracle_client.rsp
4. Access the `oracle_client.rsp` file and make the following changes in the file:

   ```
   UNIX_GROUP_NAME=oinstall
   INVENTORY_LOCATION=/home/oracle/oraInventory
   ORACLE_HOME=/home/oracle/app/oracle/product/12.1.0/client_1
   ORACLE_BASE=/home/oracle/app/oracle
   oracle.install.client.installType=Administrator
   ```
5. Run the following commands:

```
cd /tmp
```

```
mkdir linuxamd64_12102_client
```

```
mv client linuxamd64_12102_client
```

```
tar -cvf Oracle_client.tar linuxamd64_12102_client oracle_client.rsp
```

```
gzip Oracle_client.tar
```

```
mv Oracle_client.tar.gz oracle_client.rsp /docker/unica
```

6. In the `/docker/unica/` location, create a file named `oracle.sh` and add the following content in the file:

   ```
   yum install -y libaio
   /tmp/Oracle_client_install/linuxamd64_12102_client/client/runInstaller
   -silent -ignoreSysPrereqs -responseFile /tmp/Oracle_client_install/oracle_client.rsp
   ```

For Red Hat JBoss Enterprise Application Platform

To use JBoss, within the cluster or within an external database, complete the following steps.

1. Download Cloud Native Unica images and Helm Chart.
2. Add the installable `JBoss` and `JDBC Drivers` to the mount location.
3. Create `Databases` and `Users` and enter those details in the Unica Helm Chart.

If you set the Database as a sub-chart in Unica chart, you can completely automate data import using Shell scripts. For import, data should be available on the Database container mount point. You can also place the data after the container starts. Ensure that Database configuration and user creation activity is completed before running the Unica chart.

For auto-installation of database client on listener pod or container, complete the following steps:

Note: The commands and filenames are mentioned specific to Oracle database. Provide appropriate values based on the the database you use.

1. Place the Oracle client installer, named `linuxamd64_12102_client.zip`, inside the `/tmp` folder.
2. To extract the installer file, run the unzip command.

   A new folder, named `client` is created in the location `/tmp`.
3. Run the following command:

   cp /tmp/client/response/client_install.rsp /tmp/oracle_client.rsp
4. Access the `oracle_client.rsp` file and make the following changes in the file:

   ```
   UNIX_GROUP_NAME=oinstall
   INVENTORY_LOCATION=/home/oracle/oraInventory
   ORACLE_HOME=/home/oracle/app/oracle/product/12.1.0/client_1
   ORACLE_BASE=/home/oracle/app/oracle
   oracle.install.client.installType=Administrator
   ```
5. Run the following commands:

   ```
   cd /tmp
   ```

   ```
   mkdir linuxamd64_12102_client
   ```

   ```
   mv client linuxamd64_12102_client
   ```

```
tar -cvf Oracle_client.tar linuxamd64_12102_client oracle_client.rsp
```

```
gzip Oracle_client.tar
```

```
mv Oracle_client.tar.gz oracle_client.rsp /docker/unica
```

6. In the `/docker/unica/` location, create a file named `oracle.sh` and add the following content in the file:

```
yum install -y libaio
/tmp/Oracle_client_install/linuxamd64_12102_client/client/runInstaller
-silent -ignoreSysPrereqs -responseFile /tmp/Oracle_client_install/oracle_client.rsp
```

For Oracle WebLogic

To use WebLogic, within the cluster or within an external database, complete the following steps:

1. Download Cloud Native Unica images and Helm Chart.
2. Create `Databases` and `Users` and enter those details in the Unica Helm Chart.

If you set the Database as a sub-chart in Unica chart, you can completely automate data import using Shell scripts. For import, data should be available on the Database container mount point. You can also place the data after the container starts. Ensure that Database configuration and user creation activity is completed before running the Unica chart.

For auto-installation of database client on listener pod or container, complete the following steps:

Note: The commands and filenames are mentioned specific to Oracle database. Provide appropriate values based on the the database you use.

1. Place the Oracle client installer, named `linuxamd64_12102_client.zip`, inside the `/tmp` folder.
2. To extract the installer file, run the unzip command.

   A new folder, named `client` is created in the location `/tmp`.
3. Run the following command:

   cp /tmp/client/response/client_install.rsp /tmp/oracle_client.rsp
4. Access the `oracle_client.rsp` file and make the following changes in the file:

```
UNIX_GROUP_NAME=oinstall
INVENTORY_LOCATION=/home/oracle/oraInventory
ORACLE_HOME=/home/oracle/app/oracle/product/12.1.0/client_1
ORACLE_BASE=/home/oracle/app/oracle
oracle.install.client.installType=Administrator
```

5. Run the following commands:

```
cd /tmp
```

```
mkdir linuxamd64_12102_client
```

```
mv client linuxamd64_12102_client
```

```
tar -cvf Oracle_client.tar linuxamd64_12102_client oracle_client.rsp
```

```
gzip Oracle_client.tar
```

```
mv Oracle_client.tar.gz oracle_client.rsp /docker/unica
```

6. In the `/docker/unica/` location, create a file named `oracle.sh` and add the following content in the file:

```
yum install -y libaio
/tmp/Oracle_client_install/linuxamd64_12102_client/client/runInstaller
-silent -ignoreSysPrereqs -responseFile /tmp/Oracle_client_install/oracle_client.rsp
```

Listener Database client setup

To establish an ODBC connection to the database, the Campaign listener requires a database client.

About this task

If you do not have a database client, you must install it. For a seamless installation of the database client, perform the following steps:

1. Place the database client installer at the mount locaction (NFS).
2. Configure the location of the database client installer in the `campaign-configMap.yaml` file. For more information, see *Cloud Native Unica Implementation Guide* for the respective Application Server,

Setting up the Cloud Native environment

You must set up the Cloud Native environment before implementing Cloud Native Unica. The chart that you download uses Helm as a package manager for Kubernetes. The chart is a preconfigured application resource and it deploys Unica suite on a specified Kubernetes cluster. Extract the chart ZIP file to a location in the cloud VM, where you plan to deploy Unica. For reference purposes, this chart contains a placeholder for the database. Unica does not own the database and is not responsible for database management. If required, set a containerized database (the charts and subcharts folders are for reference) as a subchart to the Cloud Native Unica chart. You can use scripts to automate the restoration of database on a container.

Before you begin

The prerequisites for running a Helm chart are as follows:

- Download the required Docker images from Flex Net Operations (FNO).
- To import the downloaded Docker images for all the products, run the following command:

  ```
  docker load -i product_image_name.tar
  ```
- To verify if all products images are loaded and available for use, run the following command:

  ```
  docker images
  ```
- To tag the images appropriately, run the following command:

  ```
  docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
  ```
- To push the images to the docker registry, run the following command:

  ```
  docker push TARGET_IMAGE[:TAG]
  ```
- Open the `values.yaml` file, which is placed inside the `Unica` folder, and edit:
  - the Docker images name in the `repository` section
  - the tag numbers in the `tag` section

  See the following code snippet for reference:

```
image:
 repository:
  init: TARGET_IMAGE
  platform: TARGET_IMAGE
 tag:
  init: TAG
  platform: TAG
```

- Configure the database in one of the following ways:
    - **Database within Kubenetes cluster** - Set the database as a subchart to Unica helm chart. Unica will not own or manage the database chart.
    - **Pointing to an external database** - Configure the database to reside on the same subnet as the worker nodes to ensure good performance.

About this task

To set up Cloud Native Unica environment, complete the following steps:

1. **Update chart configurations**:

    a. Update or customize database and application server details in the `configMap` files for each products. For more information on `configMap` files, see the *Cloud Native Unica Implemetation Guide* for your respective application server. An example for updating or customizing the `campaign-configMap.yaml` is as follows:

    ```
    CAMPAIGN_DATABASE_HOST: "{{ .Release.Name }}-unica-suite-database"
    CAMPAIGN_DATABASE_PORT: "1521"
    CAMPAIGN_DATABASE_NAME: "xe"
    CAMPAIGN_DATABASE_USERNAME: "campuser"
    CAMPAIGN_DATABASE_PASSWORD: "unica"
    CAMPAIGN_DS_INITIAL_SIZE: "1"
    CAMPAIGN_DS_MIN_IDLE: "1"
    CAMPAIGN_DS_MAX_IDLE: "15"
    CAMPAIGN_DS_MAX_TOTAL: "80"
    CAMPAIGN_DS_STATEMENT_CACHE_SIZE: "300"
    ```

    For more information on configurations related to `values.yaml` file, see see the *Cloud Native Unica Implemetation Guide* for your respective application server.
2. **Update persistence volume**:

    a. Based on the persistent volume of your choice, update the following files:

    ```
    - unica/extra-configs/local-pv.yaml
       - unica/templates/pvc.yaml
    ```
3. **Perform an upgrade**:

    a. You can use one of the following methods to upgrade:
        - Upgrade from On-premises to Cloud Native (for example, Unica version 9.1.2 to Cloud Native version 12.1.0)
        - Upgrade from earlier Cloud Native version to a new version (for example, Cloud Native version 12.0 to Cloud Native version 12.1)

b. Before the upgrade, ensure that you have backed up the file system and the Database.

c. Place the file system on the mount point and configure the **BASE_FOLDER** parameter in the `common-configMap.yaml` file to point to the file system location.

d. Also, update the database details in the `common-configMap.yaml` file. For example, refer the following code snippet:

```
DATABASE_EXPORT_DIR: "/DBBACKUP/"
BASE_FOLDER: "OLDINSTALL/IBMUnica_86"
SOURCE_SCHEMA: "camp86"
TARGET_SCHEMA: "camp86"
SOURCE_SCHEMA_RT: "camp86"
TARGET_SCHEMA_RT: "camp86"
SOURCE_SCHEMA_PROD: "intpr86"
TARGET_SCHEMA_PROD: "intpr86"
SOURCE_SCHEMA_LRN: "intlr86"
TARGET_SCHEMA_LRN: "intlr86"
SOURCE_SCHEMA_RUN: "intrt86"
TARGET_SCHEMA_RUN: "intrt86"
```

e. In case of managed Kubernetes clusters, change the value of the **storageClassNames** parameter in the `values.yaml` file.

Note: Active MQ Image or Chart, provided by Unica, is for reference purposes only. Unica does not own or is not responsible for Active MQ Deployments.

Cloud Native Unica setup on SSL

You can configure SSL on Cloud Native Unica setup at the ingress level.

A provision exists to create a secret with a CERT file. For additional details, see *nginx-ingress documentation for TLS configuration*.

Chapter 2. Installation and verifying the installation

The following topics provide information related to installation and verification of installation.

Installation

You can install Unica using Helm charts. Override the following Helm chart values using `--set name=value`.

Before you begin

- Ensure that `configMaps` in the helm chart are correctly configured.
- Verify all the configurations and ensure that the mount location does not have any Unica-related installation files.

1. `kubectl apply -f ./omnix-unica/extra-configs/local-pv.yaml`
2. `helm install --name nginx stable/nginx-ingress -f ./omnix-unica/extra-configs/nginx-conf.yaml`
3. `helm install --name unica -f ./omnix-unica/values-local.yaml omnix-unica --set service.hostname=kubernetes.nonprod.hclpnp.com --set service.applicationDomain='nonprod.hclpnp.com' --set ingress.enabled=true`

What to do next

After installation, add the installation related parameters in the `commom-configMap.yaml` file for version 12.1.4.

Verifying the chart

About this task

Follow the instructions after the completion of Helm installation for chart verification. The chart generates an output for all the resources it creates.

1. To confirm if a chart has generated output for all the resources, run the following command:

   ```
   helm ls
   ```
2. To view the installed helm release, run the following command:

   ```
   helm status unica
   ```
3. To view the Unica Kubernetes pods, run the following command:

   ```
   kubectl get pods
   ```

Log files

Confirm if the required containers are up and running. Upon confirmation, check the logs for all the running services.

- **Installation log files**:

  The installation log files are placed in the logs folder at the mount point. For example, `$HOME_DIR/logs`.
- **Product log files**:

  Log files are persisted out of the containers at the mount location. The log files for the products are placed in their respective install location folders. For example, if the product is Campaign and the mount location is `/docker/unica`, the Campaign log files will be available within the `/docker/unica/Campaign/logs/` location.

Campaign Log Files

To enable the `ETL`, `Engage`, and `UBX` logs within the Campaign logs folder, provide the absolute path in the `$CAMPAIGN_HOME/conf/campaign_log4j.xml` file.

Example

```
log4j.appender.ETL.File=/docker/unica/Campaign/logs/ETL.log
log4j.appender.ENGAGE_ETL.File=/docker/unica/Campaign/logs/EngageETL.log
log4j.appender.UBX.File=/docker/unica/Campaign/logs/UBX.log
```

Chapter 3. Post installation configurations

The following topics contain details about post installation configurations related to the products of Unica.

- Configurations for Campaign on page 12
- Configurations for Director on page 13
- Configurations for Interact on page 13
- Configurations for Platform on page 14

Configurations for Campaign

To add user database in Campaign, complete the following steps:

1. Connect to the Listener pod.
2. Add the user database.
3. In the application, navigate to **Campaign > Configuration**.
4. Add an entry for Datasources.

Configuring multi-partitions for Campaign

For Unica Campaign, you can configure the application within the partitions where you have configured an instance of Campaign.

Application users, within each partition, can access the Campaign functions, data, and customer tables that are configured for Campaign in the same partition.

Multiple partitions are useful for setting up a strong security between groups of users, because each partition has its own set of Campaign system tables.

You must not create multiple partitions if groups of users have to share data with each other.

Each partition has its own set of configuration settings. You can customize Campaign for each group of users. However, all partitions share the same installation binaries.

With the same binaries for all partitions, you can minimize the installation and upgrade efforts for multiple partitions.

The utility to create multi-partition is available in the `$HOME_DIR/Platform/tools/bin` location.

Provide values for the following parameters in the Campaign chart:

- **PARTITIONS** - Name of the partition you want to configure. In case of multiple partitions specify partition name separated by a semi-colon. For example `partition2;partition3`.
- **SOURCE_PARTITION** - The name of the source partition to be replicated.
- **DEST_PARTITION** - The name of the destination partition to be created.
- **PARTITION_USER** - Specifies the user name of the admin user for the replicated partition. The name must be unique within the instance of Unica Platform.
- **PARTITION_GROUP** - Specifies the name of the Platform admin group that the utility creates. The name must be unique within the instance of Unica Platform.

- **CAMPAIGN_PARTITION2_DATABASE_HOST** - Host system details of the system hosting the Campaign Partition2 database.
- **CAMPAIGN_PARTITION2_DATABASE_PORT** - Port number of the Campaign Partition2 database.
- **CAMPAIGN_PARTITION2_DATABASE_NAME** - Name of the Campaign Partition2 database.
- **CAMPAIGN_PARTITION2_DATABASE_USERNAME** - Username to access the Campaign Partition2 database.
- **CAMPAIGN_PARTITION2_DATABASE_PASSWORD** - Password to access the Campaign Partition2 database.
- **CAMPAIGN_PARTITION2_DS_INITIAL_SIZE** - The initial size of the Campaign Partition2 datasource connection pool.
- **CAMPAIGN_PARTITION2_DS_MIN_IDLE** - The minimum number of idle connections (not connected to a database) in the Campaign Partition2 datasource connection pool.
- **CAMPAIGN_PARTITION2_DS_MAX_IDLE** - The maximum number of idle connections (not connected to a database) in the Campaign Partition2 datasource connection pool.
- **CAMPAIGN_PARTITION2_DS_MAX_TOTAL** - The maximum number of connections that the Campaign Partition2 datasource can hold. If the number of connection requests exceed the configured value, the connection will be refused.
- **CAMPAIGN_PARTITION2_DS_STATEMENT_CACHE_SIZE** - Maximum number of statements that can be cached in the Campaign Partition2 datasource. Statement caching improves performance by caching executable statements that are used repeatedly.
- **CAMPAIGN_PARTITION2_JNDI_NAME** - JNDI name for Campaign Partition2.
- **CAMPAIGN_PARTITION2_POOL_NAME** - Pool name for Campaign Partition2.

The syntax to generate a partition is:

```
./multiPartition.sh >> output.out
```

After running the utitilty, restart the Platform and Campaign pod. After restarting the pods, login with `platform_admin`.

You can login with **PARTITION_USER** and the partition name you specify is used as the password for the `admin` user

Configurations for Director

`ActiveMQ` image is for reference or for tests. Unica does not own `ActiveMQ`. You can plug in your own `ActiveMQ` image in the helm chart.

About this task

To configure Director, complete the following step:

Update the **_DIR_HOME_** in the `Campaign/bin/setenv.sh` location with the actual path.

Configurations for Interact

About this task

For Gateway configurations to work, perform the following step.

1. Add the required `JAR` files and the configuration files to the mount location.
2. On JMX console, use the CentOS desktop and the VNC viewer to view the individual pod consoles. Enable port forwarding on different ports.

Configurations for Platform

For Director and Campaign History tab, you should configure the Platform settings.

About this task

To configure Platform settings, complete the following steps:

1. Log in to Unica Platform.
2. Select **Settings > Configuration**.
3. On the left pane, select **Unica Platform > Security > API management > Unica Platform**.
4. On the left pane, select **Authentication** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/authentication/login`
Block API access	Disabled
Secure API access over HTTPS	Enabled
Require authentication for API access	Disabled

5. On the left pane, select **User** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/usr/partitions/*`
Block API access	Disabled
Secure API access over HTTPS	Disabled
Require authentication for API access	Enabled

6. On the left pane, select **Policy** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/policy/partitions/*`
Block API access	Disabled
Secure API access over HTTPS	Disabled
Require authentication for API access	Enabled

7. On the left pane, select **Configurations** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/datasource/config`
Block API access	Disabled
Secure API access over HTTPS	Disabled
Require authentication for API access	Enabled

8. On the left pane, select **Datasource** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/datasource`
Block API access	Disabled
Secure API access over HTTPS	Disabled
Require authentication for API access	Enabled

9. On the left pane, select **Login** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/authentication/v1/login`
Block API access	Disabled
Secure API access over HTTPS	Disabled
Require authentication for API access	Disabled

10. On the left pane, select **Unica Campaign > Campaign REST API Filter** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/rest/v1/*`
Block API access	Disabled
Secure API access over HTTPS	Disabled
Require authentication for API access	Enabled

11. On the left pane, select **Unica Campaign > Campaign REST API V2 Filter** and in the right pane click **Edit settings**. The value for the fields should be:

Field name	Value
API URI	`/rest/v2/*`
Block API access	Disabled

Field name	Value
Secure API access over HTTPS	Disabled
Require authentication for API access	Enabled

Chapter 4. Migration of on-premises applications to Cloud Native Unica

You can migrate an on-premise version of Unica to the Cloud Native version. The Cloud Native version will be deployed on the application server.

Migration prerequisites

The prerequisites for the migration are as follows:

- Take a backup of your existing database.
- Copy the file system of the previous version to the mount location.
- Provide appropriate values the database parameters of all Unica components.
- For Interact, the schema name in the target setup should be the same as the one in the base setup.
- Manually map the tables and restart the Campaign Pod.

`common-configMap` configurations

In the `common-configMap.yaml` file, provide values for the following fields:

Table 1. Configurable Parameters to perform an Upgrade

Parameter Name	Example Value
BASE_FOLDER	`"OLDINSTALL/HCLUnica_86"`
FROM	`"8.6.0"`
TO	`"12.0.0"`
SOURCE_SCHEMA	`"CAMP86"`
TARGET_SCHEMA	`"DBO"`
DB_DRIVER_CLASS	`com.microsoft.sqlserver.jdbc.SQLServerDriver`
AC_VERSION	`"12.1.x"`
ACI_UNICODE	`"No"`
CONFIGURE_ON_ERROR_PROMPT	`"Yes"`
LOCALE	`"en_US"`
TYPE	`UPGRADE`
DATABASE_EXPORT_DIR	`/DBBACKUP/`
ISEXTERNALDB	`false`
DB_IMPORT_WAIT_TIME	`1050`

Table 1. Configurable Parameters to perform an Upgrade (continued)

Parameter Name	Example Value
DB_PRE_IMPORT_WAIT_TIME	`1050`
IS_UNICODE	`false`
UPGRADE_FROM_TO	`11.1+To12.1`
LISTENER_HOST_NAME	`{{ .Release.Name }}-omnix-unica-listener`
SOURCE_SCHEMA_RT	`camp86`
TARGET_SCHEMA_RT	`camp86`
DB_DRIVER_CLASS_RT	`com.ibm.db2.jcc.DB2Driver`
SOURCE_SCHEMA_PROD	`intpr86`
TARGET_SCHEMA_PROD	`intpr86`
DB_DRIVER_CLASS_PROD	`com.ibm.db2.jcc.DB2Driver`
SOURCE_SCHEMA_LRN	`intlr86`
TARGET_SCHEMA_LRN	`intlr86`
DB_DRIVER_CLASS_LRN	`com.ibm.db2.jcc.DB2Driver`
SOURCE_SCHEMA_RUN	`intrt86`
TARGET_SCHEMA_RUN	`intrt86`
DB_DRIVER_CLASS_RUN	`com.ibm.db2.jcc.DB2Driver`

`JVM` option configurations

Add the `JVM` option `-DFAST_UPGRADE_VERSION=<BASE_VERSION>`. For example: `JAVA_OPTIONS="${JAVA_OPTIONS} -DFAST_UPGRADE_VERSION=8.6.x`.

Performing the migration

The mount location should contain the old version of the Unica file system. Cloud Native containers will manage the database upgrade and the file system updates.

1. To perform the migration, run the following command.
   helm install --name unica omnix-unica --set service.hostname=<kubernetes.nonprod.hclpnp.com --set service.applicationDomain='nonprod.hclpnp.com' --set ingress.enabled=true
2. Access the migration logs from the mount location.

Configuring Unica Campaign post migration

To configure Unica Campaign post migration, complete the following steps:

Update the parameter **internalServerURL** to point to your Campaign pod.
For example, `http://hcl-unica-campaign:9125/Campaign`.

Configuring Unica Interact post migration

To configure Unica Interact post migration, complete the following steps:

1. Back up the current configurations.
2. Navigate to **Affinium > Campaign > partitions > partition1 > Interact > serverGroups**.
3. In Unica configuration, delete the old `serverGroup` and retain only the Interact `serverGroup`.
4. Define Interact as the `serverGroup` for the following configurations:
    - **flowchart** configuration within **Affinium > Campaign > partitions > partition1 > Interact**
    - **simulator** configuration within **Affinium > Campaign > partitions > partition1 > Interact**
5. Update the Interact design schema by replacing the old `serverGroup` name with a new name. Execute the following commands:
    - `update uaci_deployment set servergroupname='interact';`
    - `update uaci_ICTOSVRGROUP set servergroupname='interact';`
    - `update uaci_OfferMappingSG set servergroupname='interact';`

Configuring Unica Platform post migration

To configure Unica Platform post migration, complete the following steps:

1. The Unica Platform application URL will point to the old base environment. Change the navigation URL using the SQL script from the Platform system database.
2. Manually change the URL of the start page, which appears when you log in to Unica Platform, from the `USM_PERSONALIZATION` table.
3. Copy the following properties files from the source environment to the destination environment. Ensure that all the URLs mentioned in the files are also updated to the destination environment.
    - `Platform_Admin_URL.properties`
    - `Platform_Admin_View_Priv.properties`
    - `Platform_Admin_URL.properties`
    - `Platform_Admin_Scheduler_Scripts.properties`
    - `Platform_Admin_Scheduler_API.properties`

Chapter 5. Cloud Native Unica upgrade

To upgrade an earlier version of Cloud Native Unica to a newer version, complete the following steps:

1. Unica support team will roll out the Helm Charts after you specify the offering related details and requirements. Please contact Unica support team to get a Helm chart.
2. Download the required version image and push it to the Docker registry.
3. Update the image URLs in the helm charts.
4. Back up the Database and the file system before you start the upgrade.
5. Run the following helm upgrade command:

   ```
   helm upgrade hcl unica -f ./unica/values-local.yaml --set service.hostname=kubernetes.nonprod.hclpnp.com
    --set service.applicationDomain='nonprod.hclpnp.com' --set ingress.enabled=true
   ```
6. Add upgrade related parameters in the `common-configMap.yaml` file when upgrading to version 12.1.6.
7. Edit the helm chart `platform-deployment.yaml`. In the file, replace `args: ["chmod 755 /docker/unica && ./entrypoint.sh"]` with the following entry:

   `args: ["chmod 755 /docker/unica && echo 'find /opt/generate_datasource_snippet.sh -type f -print0 | xargs -0 sed -i \"s/export DB_URL=\\$/#export DB_URL=/g\"' > /docker/unica/centos_patch.sh && chmod 777 /docker/unica/centos_patch.sh && ./entrypoint.sh"]`

Custom listener scripts and Cloud Native Unica container OS upgrade

Unica container OS upgrade from CentOS 8 to RHEL Universal Base Image (UBI) v8.5 may cause listener pod custom scripts to fail (applies to Cloud Native Unica versions 12.1.2 / 12.1.3).

1. Modify custom scripts as per RHEL UBI OS. Example: Database Client Installation Script on listener pod.
2. Centos8 OS was updated to RHEL UBI 8.5 because of CentOS 8 end of life.
3. Also, RHEL UBI containers are less vulnerable to security threats because of the frequent fixing and release cycle.

Chapter 6. Scaling Unica containers

The following topics provide information on scaling the containers of Unica:

- For details related to scaling Listener containers, see Scaling Listener containers on page 21.
- For details related to scaling Interact containers, see Scaling Interact containers on page 23.
- For details related to scaling Journey engine containers, see Scaling Journey engine containers on page 24.

Horizontal scaling of Unica containers

Scaling a deployment ensures creation and scheduling of new Pods. Scaling increases the number of Pods to the new required state. Kubernetes also supports autoscaling of Pods.

For Multicast, perform the configurations on Kubernetes host to support it. For example, weave supports multicast and can be configured for multicast support.

Note: Autoscaling of Unica containers is not supported.

Scaling Listener containers

Listeners are defined as StatefulSets in Kubernetes. Each Pod in a StatefulSet derives its hostname from the name of the StatefulSet and the ordinal of the Pod.

The Pod domain is managed by the service and it takes the following form:

```
$(service name).$(namespace).svc.cluster.local.
```

For example, the listener pod entry is registered as follows:

```
listener-0.listener.default.svc.cluster.local
```

These can be configured in the Helm chart in the `campaign-configMap.yaml` file.

Like a Deployment, a StatefulSet manages the Pods that are based on identical container specifications. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods.

The location of Campaign shared home is `$HOME_DIR/Campaign`.

For the scaled instances of StatefulSet, `listener-0`, `listener-1`, `listener-2`,..`listener-n`, each instance has a file system mapped on the mount location. For example, `$HOME_DIR/listener/listener-0`.

Ordered scale up and scale down

1. Ordered and graceful deployment and scaling.

   If you want to scale up the Listener pod, run the following command:

   ```
   kubectl scale StatefulSets listener --replicas=2
   ```

2. First instance gets deleted in the end.

   If you want to scale down the Listener pod, run the following command:

   ```
   kubectl scale StatefulSets listener --replicas=1
   ```

Listener-Optimize merge

1. Single scalable deployment in Kubernetes.
2. Configuration and license driven `config.xml`.
3. Listener integration on page 22

Cluster mode

1. To enable scaling, by default, cluster mode must be `TRUE`.

Also perform the following listener-related scaling activities:

- Load balancing on page 22
- Listener integration on page 22

Load balancing

For load balancing, there is a single listener that executes commands related to Campaign flowchart and Optimize sessions. In comparison to Campaign flowchart, an Optimize session requires a significantly better hardware configuration, which exceeds the minimum recommendation, for successful execution.

About this task

This newly introduced single listener helps the master listener to decide the node on which it should send the execution of the flowcharts or sessions, considering the `loadBalanceWeight`. We recommend that you avoid executing Optimize sessions on a node, configured to execute Campaign flowcharts. Similarly, we recommend that you avoid setting up a node with a significantly higher configuration of hardware for executing flowcharts. Using the new flag, the master listener can utilize the available resources in an appropriate way.

Choose an appropriate `listenerType` during installation based on the hardware, or configuration, or your requirements.

Listener integration

Prior to Unica 12.0 release, Campaign and Optimize were separate products. Users having both Campaign and Optimize had to run separate listeners. The Campaign listener `unica_aclsnr` to run flowcharts and Optimize listener `unica_acolsnr` to run the Optimize session.

Campaign-Optimize merged scenario

With text-based license for v12, the listener image expects a license file at mount point.

If both listener host name txt (`listener-0.txt` …) and `opt.instance` file exist, it will create only the Optimize listener. If listener host name `TXT` contains the first listener, it creates the listener as `LISTENER_TYPE 3`, which means it is for both Campaign and Optimize, otherwise it creates the listener as `LISTENER_TYPE 2` indicating that it is only for Optimize.

If the listener host name txt, `listener-0.txt` and so on, exists and the `opt.instance` file does not exist, it creates the listener as `LISTENER_TYPE 3`, which indicates that it is for both Campaign and Optimize.

Listener types

- **CAMPAIGN_ONLY (TYPE 1)** - This listener can handle commands for Campaign or flowchart only.
- **OPTIMIZE_ONLY (TYPE 2)** - This listener can handle commands for Optimize session only.
- **ALL ((TYPE 3)**- This listener can handle commands for Campaign or Flowchart or Optimize session.

The Type option is available in the following locations:

- **Settings > Configuration > Campaign > unicaACListener**
- **Settings > Configuration > Campaign > unicaACOListener**

Scaling Interact containers

Each existing Interact machine runs a Kubernetes Interact deployment. If you have set the **hostNetwork** to TRUE, the existing network, which already supports multicast, can be used as it is without changing any settings. You can also use the existing load balancers over the Kubernetes Interact deployments.

To scale Interact pods for multiple server groups, refactor the helm chart to add services and deployments per server group. Each Server Group should point to a different Platform Instance. For example, if there are three RT server groups, there will be three Platform instances (three services and three deployments for Platform and Interact).

The **CONTEXT_ROOTS** variable, in the `interact_configMap.yaml` file drives:

- the context roots for Interact and Platform.
- PLT and RT database details per server group.

If you want to scale pods for a server group, run the following command:

```
kubectl scale deployment hcl-unica-interact --replicas=2
```

If the Interact POD crashes, or if you manually delete the pod, manually delete an entry from the configuration using the following command:

```
./configTool.sh -d -p 'Affinium|Campaign|partitions|partition1|Interact
|serverGroups|interactatm|instanceURLs|$1' -o "
```

In the earlier command $1 refers to the Interact POD name that crashed or was manually deleted.

Monitoring the scaled instances

Note: Ensure that VNC viewer exists on the host machine to monitor instances.

You can perform JMX monitoring for each of the scaled instances using port forwarding.

For POD1, run the following command:

```
kubectl port-forward --address 0.0.0.0 pod/unica-omnix-unica-interact-84d7b47f59-d2rsl 9998:9998 &
```

For POD2, run the following command:

```
kubectl port-forward --address 0.0.0.0 pod/unica-omnix-unica-interact-84d7b47f59-d2rsl 9999:9998 &
```

Additionally, if your application server is WebLogic, the DB hostname should be a fully qualified domain name or else the Kubernetes service name will not work.

Scaling Journey engine containers

In Kubernetes, Journey engine are defined as StatefulSets.

Each Pod in a StatefulSet derives its hostname from:

- the name of the StatefulSet, and
- the ordinal of the Pod.

The service manages the Pod domain and the format is as follows:

```
$(service name).$(namespace).svc.cluster.local
```

Example:

The Journey engine pod is regitered in the following format:

```
journey-0.listener.default.svc.cluster.local
```

In the Helm chart, you can configure these pods in the `journey-configMap.yaml` file.

Like a Deployment, a StatefulSet manages the Pods that are based on identical container specifications.

Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods.

The location of Journey Engine shared home is `$HOME_DIR/Journey/Engine`.

For the scaled instances of StatefulSet, `journey-0`, `journey-1`, `journey-2`,..`journey-n`, each instance has a file system mapped on the mount location.

Example:

```
$HOME_DIR/Journey/journey-0
```

In a Journey engine, by default, clustering is enabled. As soon as the engine starts, it creates `journey-0`, which is a copy of the engine folder. As you keep scaling the Journey engine, it creates folders named `journey-0`, `journey-1`, `journey-2`,..`journey-n`.

The logs for each pod will also be generated as `journey-0`, `journey-1`, `journey-2`,..`journey-n`.

Chapter 7. Using Red Hat OpenShift

You can use OpenShift to develop and runcontainerized applications. OpenShift allows applications, and the data centers that support them, to expand from just a few machines and applications to thousands of machines that serve millions of clients.

About this task

For detailed information related to Red Hat OpenShift Container Platform, see OpenShift Container Platform documentation.

The benefits of using OpenShift Container Platform are as follows:

- Does not require separate charts as the OpenShift charts are customized, or updated, charts when compared to Kubernetes charts.
- Easy to manage and monitor using the OpenShift console.

To configure the changes required for Unica, complete the following steps:

1. Place the following items on a location that is accessible from the listener pod:
    - `unixodbc`
    - `libltdl.so.7`
    - `libltdl.so.7.30`
    - `mariadb driver` (must be installed and then copied to the required location)

   Update the same in `campaign-configmap.yaml` file:

   ```
   export ODBCINI=<driver-path>/etc/odbc.ini
   export ODBCINST=<driver-path>/etc/odbcinst.ini
   export ODBCSYSINI=<driver-path>/odbc1/etc
   ```

   Note: *<driver-path>* is the path where you have copied the driver. For example, `/docker/unica/odbc1`.
2. In the `campaign-configmap.yaml` file, update the namespace for listener domain name.
3. Based on your setup, you can:

   Choose from:
    - update the `PVC.yaml` file before using it.
    - avoid the `PVC.yaml` file.

Security Context Constraints for Unica on Red Hat OpenShift

For any Security Context Constraint (SCC), perform the following steps:

1. If `AllowPrivilegedContainer` is enabled (set to `TRUE`) or not enabled, set it to `FALSE`.
2. Do not assign root access to the users specified in the `deployment.yaml` file.
3. For pods that do not have a `gid` (group ID), perform the following configuration:

   ```
   securityContext:
             runAsUser: 1000610000
   ```

The configuration ensures that the start user of the pods is `1000610000`. The `1000610000` user cannot switch to the `root` user or change the `root` user password.

4. For the Oracle client, in the listener pod, create a user for a valid group and perform the following configurations:

```
securityContext as :
          securityContext:
              runAsUser: 1000
              runAsGroup: 1001

oracle:x:1000:1000::/home/oracle:/bin/bash
dba:x:1001:oracle
1000=oracle and 1001 = dba group
```

The configuration ensures that the Oracle user also cannot switch to the `root` user or change the `root` user password.

5. For the SCC (`anyuid`), configure the following values:

```
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: anyuid provides all features of the restricted SCC
      but allows users to run with any UID and any GID.
    release.openshift.io/create-only: "true"
  creationTimestamp: "2020-08-24T17:55:03Z"
  generation: 6
  name: anyuid
  resourceVersion: "23505934"
  selfLink: /apis/security.openshift.io/v1/securitycontextconstraints/anyuid
  uid: 43877aab-c522-4ca9-9575-e8b212749e29
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:unica:default
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

6. For the listerner pod, remove all `chmod` or `su`.
7. In the listener `rc.unica_ac`, remove the root user `check` and change it to `oracle`.
8. In the Journey configmap, update the namespace from `default` to `unica`.

Chapter 8. Deployment monitoring

The Kubernetes Dashboard is a web-based user interface to monitor deployments.

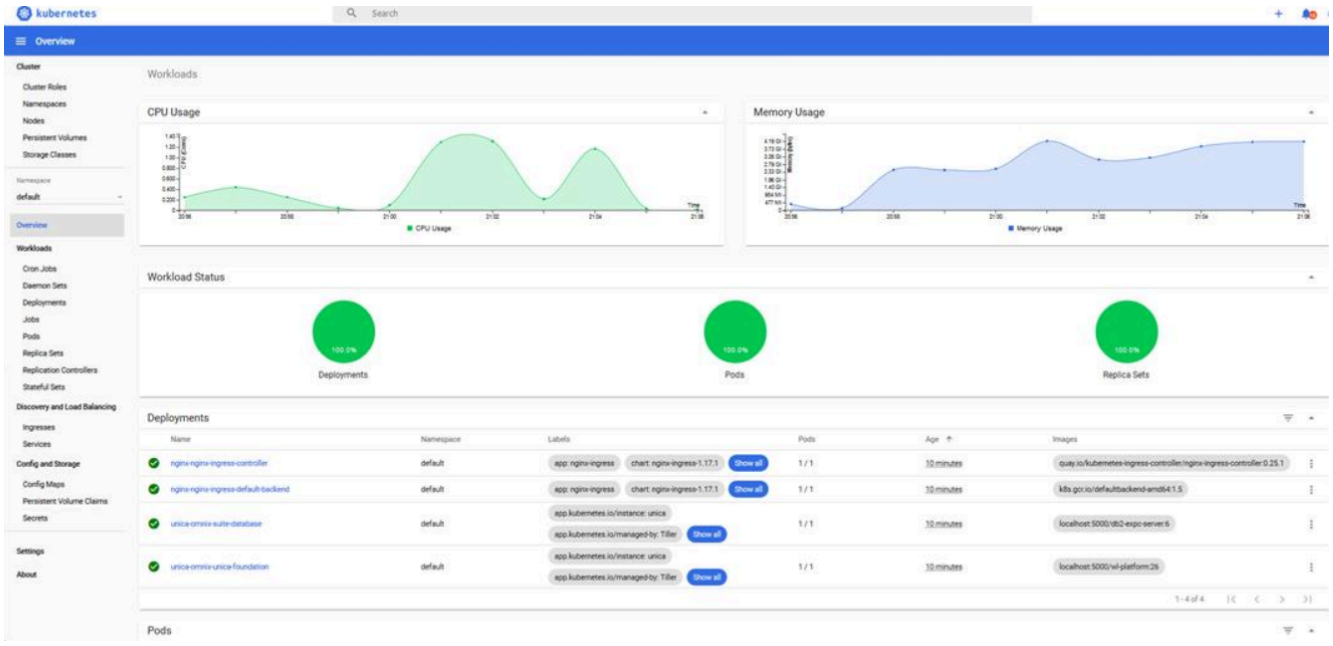
Use the Kubernetes Dashboard to:

- deploy containerized applications to a Kubernetes cluster
- troubleshoot your containerized applications
- managing cluster resources

You can also the use Dashboard to get an overview of the applications running on your cluster, as well as for creating or modifying individual Kubernetes resources.

The Dashboard also provides information on the state of Kubernetes resources in your cluster and on any errors that may have occurred.

Figure 1. Kubernetes dashboard



Deployments

Name	Namespace	Labels	Pods	Age	Images
nginx-nginx-ingress-controller	default	app: nginx-ingress, chart: nginx-ingress-1.17.1	1 / 1	10 minutes	quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.25.1
nginx-nginx-ingress-default-backend	default	app: nginx-ingress, chart: nginx-ingress-1.17.1	1 / 1	10 minutes	k8s.gcr.io/defaultbackend-amd64:1.5
unica-omnix-suite-database	default	app.kubernetes.io/instance: unica, app.kubernetes.io/managed-by: Tiller	1 / 1	10 minutes	localhost:5000/db2-expc-server:6
unica-omnix-unica-foundation	default	app.kubernetes.io/instance: unica, app.kubernetes.io/managed-by: Tiller	1 / 1	10 minutes	localhost:5000/wl-platform:26

1 - 4 of 4

Deploying the dashboard user interface

The Dashboard user interface is not deployed by default.

To deploy the Dashboard user interface, run the following command.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta4/aio/deploy/recommended.yaml
```

Chapter 9. Product utilities

You can execute all the utilities of the Unica products in their assigned pods.

The following table lists the Unica products and their assigned pods for running the product-specific utilities.

Table 2. Unica products and their assigned pods for running the utilities

Unica Product Name	**Pod Name**
Unica Campaign	`Listener`
Unica Platform	`Platform`
Unica Plan	`Plan`

Chapter 10. Using secret to avoid passwords in plain text

To use a secret to avoid using passwords in plain text, complete the following steps:

1. On a Linux virtual machine, run the following command:
   echo -n 'unica*03' | base64
   Result
   You will see the following output: `"dW5pY2EqMDM="`
2. Create a `YAML` file (example `unicadbSecret.yaml`) and in the `YAML` file add the following parameters:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   name: unica-db-token
   type: Opaque
   data:
   PLATFORM_DATABASE_PASSWORD: "dW5pY2EqMDM="
   ```
3. To use the password in Unica Platform, update the Platform deployment, and wherever envFrom exists, add the the following code:

   ```
   envFrom:
   - secretRef:
   name: unica-db-token
   - configMapRef:
   ```
4. Either comment or delete the parameter **PLATFORM_DATABASE_PASSWORD**: `unica*03` from the `platform-configMap.yaml` file.

What to do next

Note:

- The same `unicadbSecret.yaml` can be used for multiple Unica product database passwords. Repeat *Step 3* and *Step 4* for each products deployment and their respectiv `configmap.yaml` file. For example, in case of Unica Plan, with **PLAN_DATABASE_PASSWORD**: `unica*03`, add the following lines of code

  ```
  apiVersion: v1
  kind: Secret
  metadata:
  name: unica-db-token
  type: Opaque
  data:
  PLATFORM_DATABASE_PASSWORD: "dW5pY2EqMDM="
  PLAN_DATABASE_PASSWORD: "dW5pY2EqMDM="
  ```

  In this case, update the Plan deployment and `configmap.yaml` file.
- Limit the secret size to `1 MB`. If the secret size is more than `1 MB`, split it into multiple tokens.

Chapter 11. Using AWS Secrets and Configuration Provider with Kubernetes Secret Store CSI Driver

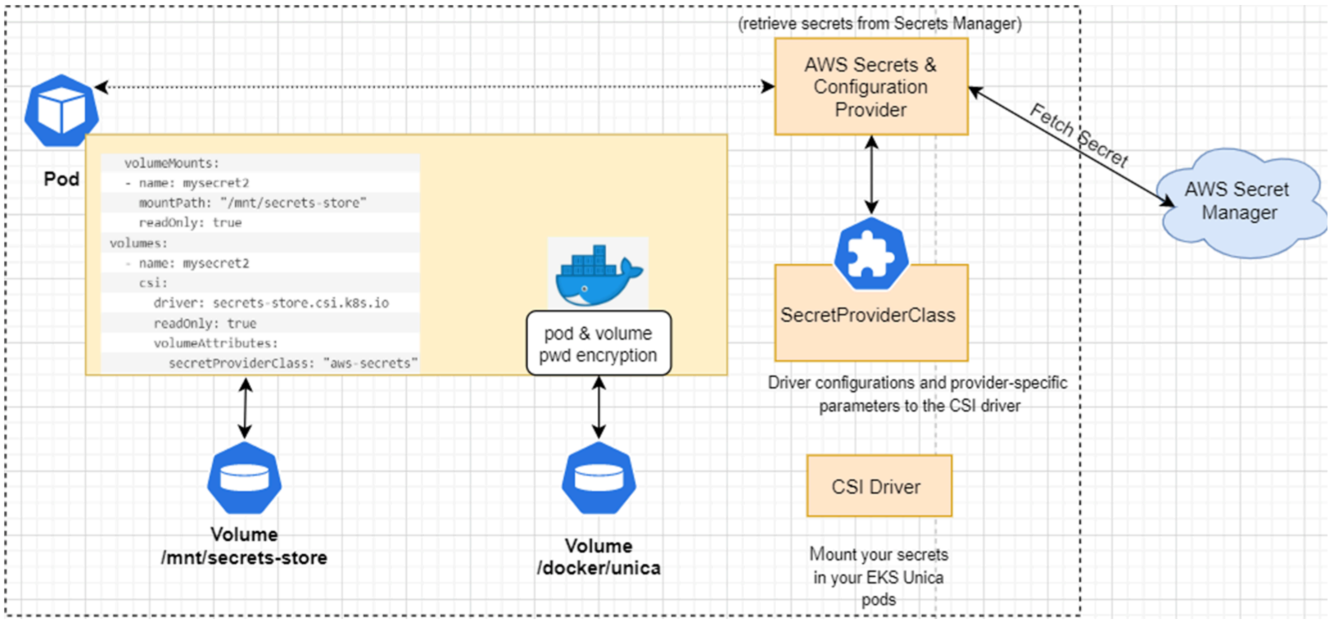
AWS Secrets Manager securely retrieves secrets from the AWS Secrets Manager for Amazon Elastic Kubernetes Service (Amazon EKS) Kubernetes pods.

AWS Secrets and Config Provider (ASCP) contains an an easy-to-use plugin that provides secrets to applications that operate on Amazon EKS. The plugin supports industry-standard Kubernetes Secrets Store and Container Storage Interface (CSI) driver.

The benefits of ASCP are as follows:

- Provides compatibility for legacy Kubernetes workloads that fetched secrets through the file system or `etcd`.
- Securely store and manage your secrets in Secrets Manager.
- Retrieve secrets, using applications that run on Kubernetes, without writing a custom code.
- Use AWS Identity and Access Management (IAM) and resource policies on your secret to limit and restrict access to specific Kubernetes pods inside a cluster to tightly control secrets accessible by the pods.

AWS Secrets Manager Working Concept with Unica


(retrieve secrets from Secrets Manager)
AWS Secrets & Configuration Provider
Fetch Secret
AWS Secret Manager
Pod
pod & volume pwd encryption
SecretProviderClass
Driver configurations and provider-specific parameters to the CSI driver
CSI Driver
Mount your secrets in your EKS Unica pods
Volume /mnt/secrets-store
Volume /docker/unica

```
  volumeMounts:
  - name: mysecret2
    mountPath: "/mnt/secrets-store"
    readOnly: true
volumes:
  - name: mysecret2
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: "aws-secrets"
```

AWS Secret Manager Implementation

To implement AWS Secret Manager, ensure that the prerequisites are met and the configurations are executed.

For more details, see the following topics:

- Prerequisite Software for AWS Secret Manager on page 32
- Prerequisite Configurations for AWS Secret Manager on page 32
- Implementing AWS Secret Manager on page 33

Prerequisite Software for AWS Secret Manager

The prerequisite software requirement for AWS Secret Manager are as follows:

- An AWS account
- AWS Command Line Interface installed
- `kubectl` installed
- Helm installed
- `eksctl` installed
- An existing EKS cluster

Prerequisite Configurations for AWS Secret Manager

Before implementing AWS Secret Manager, make the following configurations:

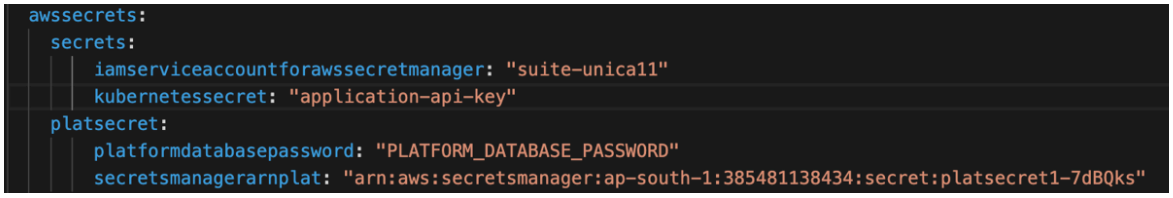
- An IAM policy, with permissions to retrieve secrets from Secret Manager.
- Your secret stored in Secrets Manager, for example `platsecret`, `campsecret`, and `plansecret` with keys `PLATFORM_DATABASE_PASSWORD`, `CAMPAIGN_DATABASE_PASSWORD`, and `PLAN_DATABASE_PASSWORD`:
  - keys should match the `configMap` entries
  - encryption key value must be `aws/secretsmanager`
- A `user` or `iamserviceaccount` that can modify your Kubernetes cluster.
- To the Docker registry, push the new set of images.
- To use the new image tags, update the `values.yaml` file.
- In the Helm chart, comment out the following `_PASSWORD` parameters from the `configMap.yaml` files:
  - `CAMPAIGN_DATABASE_PASSWORD`
  - `PLAN_DATABASE_PASSWORD`
  - `PLATFORM_DATABASE_PASSWORD`
- In the `common-configMap.yaml` file, add the following parameter:

```
TOMCAT_FACTORY: "com.unica.manager.tomcat.utils.TomcatDSFactory"
```

- Create secrets for the following Unica products with the corresponding names:

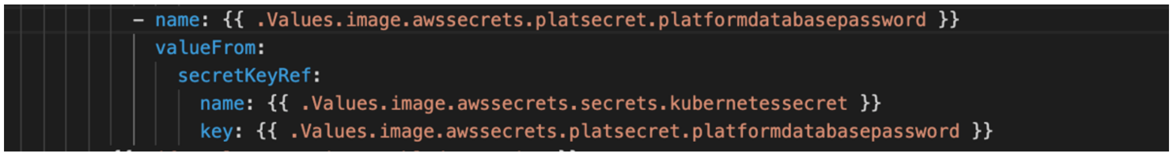
Unica Porduct	**Secret name**
Unica Campaign	`campsecret`
Unica Plan	`plansecret`
Unica Platform	`platsecret`

- Update the secret arn, secret name, and key in the following files (see the example for reference):
  - `values.yaml`



```
awssecrets:
  secrets:
      iamserviceaccountforawssecretmanager: "suite-unica11"
      kubernetessecret: "application-api-key"
  platsecret:
      platformdatabasepassword: "PLATFORM_DATABASE_PASSWORD"
      secretsmanagerarnplat: "arn:aws:secretsmanager:ap-south-1:385481138434:secret:platsecret1-7dBQks"
```

- `deployment.yaml`



```
- name: {{ .Values.image.awssecrets.platsecret.platformdatabasepassword }}
  valueFrom:
    secretKeyRef:
      name: {{ .Values.image.awssecrets.secrets.kubernetessecret }}
      key: {{ .Values.image.awssecrets.platsecret.platformdatabasepassword }}
```

Implementing AWS Secret Manager

To implement AWS Secret Manager on your setup, complete the following steps:

1. Using a command line interface, restrict access to your pods using IAM roles for service accounts. Alternatively, you can also restrict access using a console.
2. To turn on Open ID Connect (OIDC), run the following eksctl command:

```
eksctl utils associate-iam-oidc-provider --region=<REGION> --cluster=<CLUSTERNAME> --approve
```

Note:

- You must run the earlier mentioned command only once.
- In the command, mentioned earlier, replace `<REGION>` and `<CLUSTERNAME>` with relevant and appropriate values.

3. For retrieving secrets from AWS Secret Manager, create a policy by running the following command:

```
aws iam create-policy --policy-name <my-policy> --policy-document file://policy
```

A sample policy file follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [x`
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:ap-south-1:385481138434:secret:*"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
}
```

4. Create a service account role to associate the policy (created in *Step 2*) with your service account. To create a service account, run the following command:

```
eksctl create iamserviceaccount --name <SERVICE_ACCOUNT_NAME> --namespace <NAMESPACE> --cluster
 <CLUSTERNAME> --attach-policy-arn <IAM_policy_ARN> --approve --override-existing-serviceaccounts
```

Note: In the command, mentioned earlier, replace `<NAMESPACE>`, `<CLUSTERNAME>`, `<IAM_policy_ARN>`, and `<SERVICE_ACCOUNT_NAME>` with relevant and appropriate values.

5. To install the Kubernetes secrets store CSI driver, using helm with syncSecret.enabled=true, run the following commands:

   a. Run the following command:

   ```
   helm repo add secrets-store-csi-driver
    https://kubernetes-sigs.github.io/secrets-store-csi-driver/charts
   ```

   b. If you do not require a periodical pull of updated secrets, initialize the driver by running the following command:

   ```
   helm install csi-secrets-store secrets-store-csi-driver/secrets-store-csi-driver --set
    syncSecret.enabled=true --namespace kube-system
   ```

   c. If you want to turn on automated rotation for the driver, using the rotation reconciler feature which is currently in alpha, run the following command:

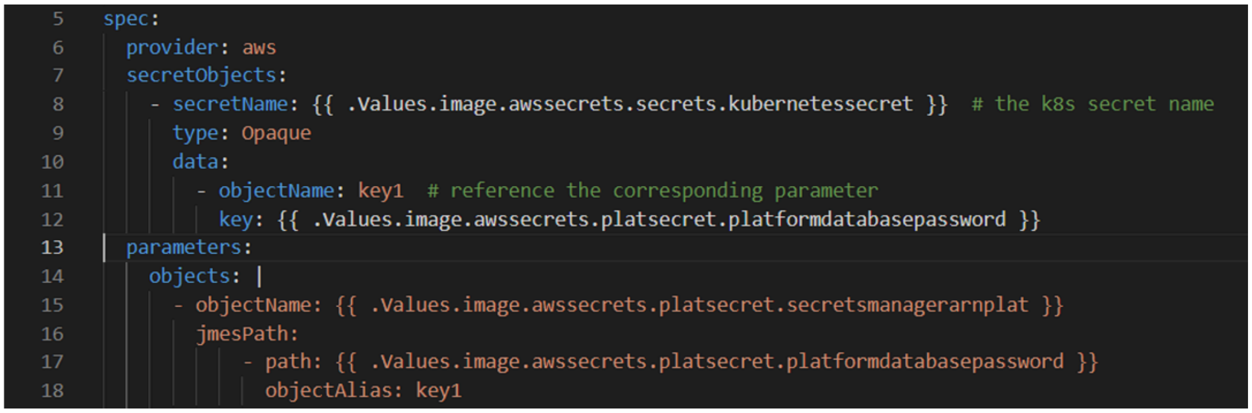
   ```
   helm -n kube-system install csi-secrets-store secrets-store-csi-driver/secrets-store-csi-driver
    --set enableSecretRotation=true --set rotationPollInterval=3600s
   ```

   Note: You can adjust the rotation intervals, as per your requirements, to find an appropriate balance between API call cost consideration and rotation frequency

6. To install the ASCP, run the following command:

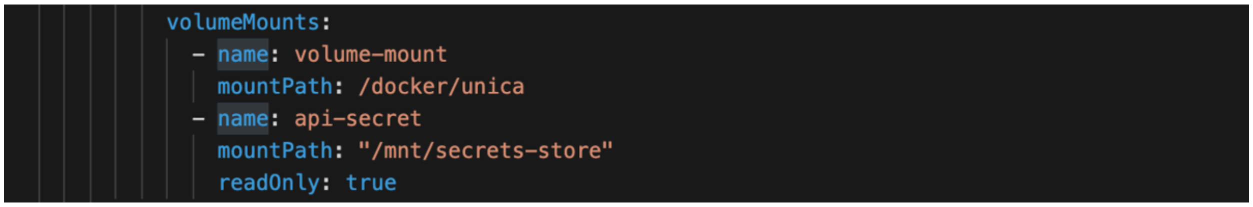
```
kubectl apply -f
 https://raw.githubusercontent.com/aws/secrets-store-csi-driver-provider-aws/main/deployment/aws-provide
r-installer.yaml
```

7. Create the custom resource **SecretProviderClass** and deploy it to sync with AWS secret with Kubernetes. For details, access the `spc.yaml` inside the Unica helm chart.

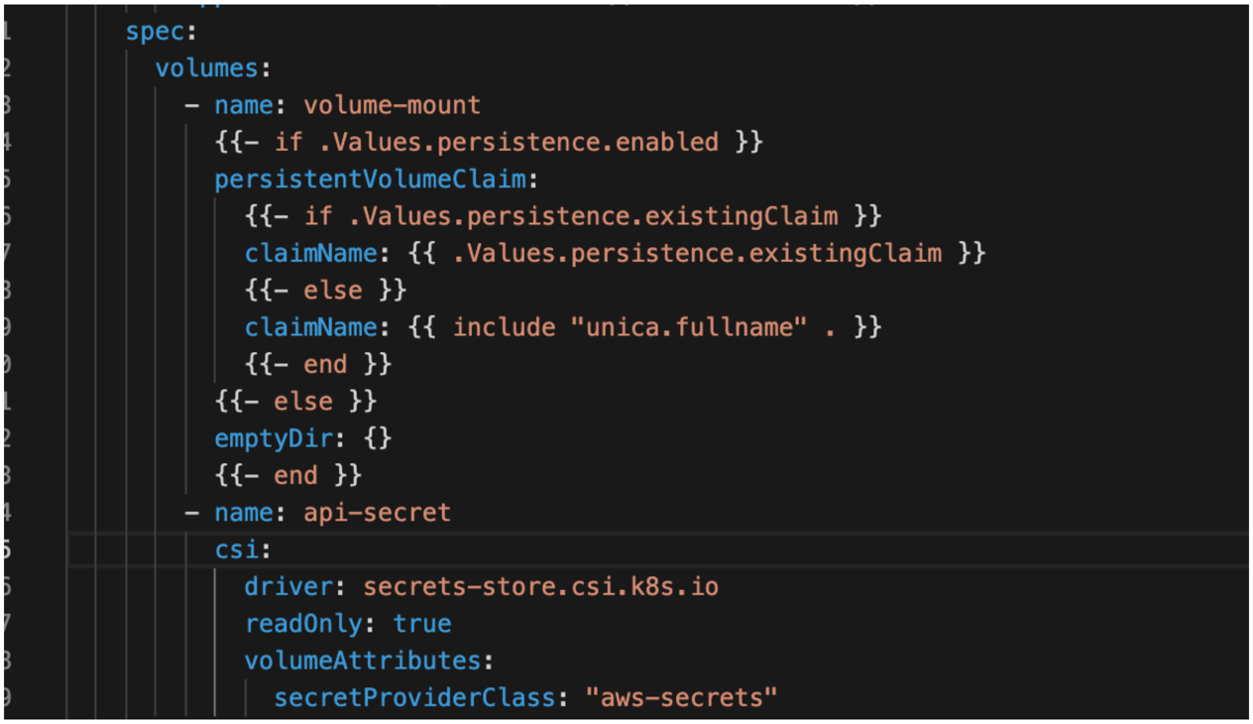


```
spec:
  provider: aws
  secretObjects:
    - secretName: {{ .Values.image.awssecrets.secrets.kubernetessecret }}  # the k8s secret name
      type: Opaque
      data:
        - objectName: key1  # reference the corresponding parameter
          key: {{ .Values.image.awssecrets.platsecret.platformdatabasepassword }}
  parameters:
    objects: |
      - objectName: {{ .Values.image.awssecrets.platsecret.secretsmanagerarnplat }}
        jmesPath:
          - path: {{ .Values.image.awssecrets.platsecret.platformdatabasepassword }}
            objectAlias: key1
```

8. Configure and deploy the pods to mount the volumes based on the configured secrets.

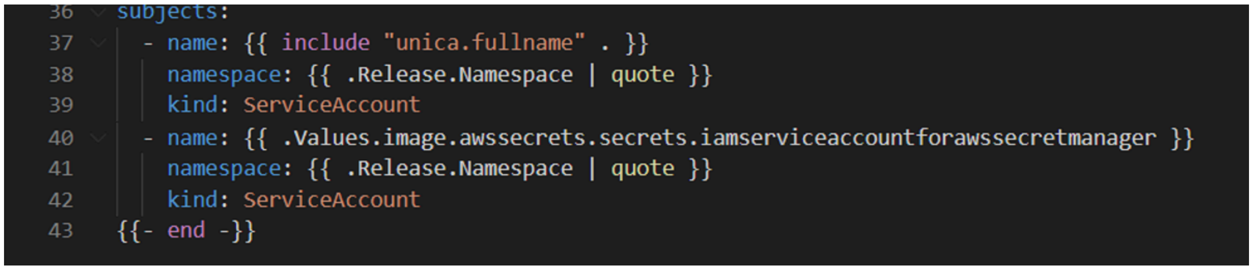


```
volumeMounts:
  - name: volume-mount
    mountPath: /docker/unica
  - name: api-secret
    mountPath: "/mnt/secrets-store"
    readOnly: true
```



```
spec:
  volumes:
    - name: volume-mount
      {{- if .Values.persistence.enabled }}
      persistentVolumeClaim:
        {{- if .Values.persistence.existingClaim }}
        claimName: {{ .Values.persistence.existingClaim }}
        {{- else }}
        claimName: {{ include "unica.fullname" . }}
        {{- end }}
      {{- else }}
      emptyDir: {}
      {{- end }}
    - name: api-secret
      csi:
        driver: secrets-store.csi.k8s.io
        readOnly: true
        volumeAttributes:
          secretProviderClass: "aws-secrets"
```

9. In the `rbac.yaml` file, assign the **ClusterRoleBinding** permissions to the `iamservice` account, created in *Step 3*, for internal Kubernetes communication.



```
subjects:
  - name: {{ include "unica.fullname" . }}
    namespace: {{ .Release.Namespace | quote }}
    kind: ServiceAccount
  - name: {{ .Values.image.awssecrets.secrets.iamserviceaccountforawssecretmanager }}
    namespace: {{ .Release.Namespace | quote }}
    kind: ServiceAccount
{{- end -}}
```

Chapter 12. Enabling Multicast using Weave-Net CNI plugin on AWS EKS cluster

You can enable multicasting on AWS EKS cluster only for Kubernetes versions 1.21 or above,

Before you begin

- Create a role on AWS having the necessary privileges for creating AWS clusters (example: `AWS_EKS_CLUSTER_ROLE`).
- Create a minimum of two subnets within the VPC. You must create the cluster within this VPC.

About this task

To enable multicasting on AWS EKS cluster using Weave-Net CNI plugin, complete the following steps:

1. Use the AWS CLI and create an EKS cluster without any node group.

   Note: Multicasting will not work if you create clusters using AWS web console.

   Sample Command:

   ```
   aws eks create-cluster --region <region-name> --name <cluster-name> --kubernetes-version 1.21 --role-arn
    <full-arn-of-the-role> --resources-vpc-config subnetIds=<subnet-id1>,<subnet-id2>,...<subnet-idn>
   ```
2. Run the following command to delete the aws-node default daemon-set:

   ```
   kubectl delete ds aws-node -n kube-system command
   ```

   Result

   This disables the default vpc-cni plugin.
3. Confirm if your security group allows TCP port `6783` and UDP ports `6783` and `6784`. If your security group does not allow these ports, add the necessary firewall rules to your security groups to allow these ports.
4. Run the following command to delete the `kube-proxy ds`:

   ```
   kubectl delete ds kube-proxy -n kube-system
   ```
5. Run the following command to create an add-on for Kube-proxy:

   ```
   aws eks create-addon --cluster-name <your-cluster-name> --addon-name kube-proxy --resolve-conflicts
    OVERWRITE
   ```

   Result

   This will add the latest kube-proxy add-on to the cluster, based on the Kubernetes cluster version.
6. Run the following command to apply weave-net daemoset:

   ```
   kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d
    '\n')"
   ```
7. Verify the Daemon sets on cluster. There should be two daemon sets for Weave and correspondingly two Kube-proxy daemon sets.
8. Add the node group to the Cluster and wait till the nodes are created and all the required nodes are ready.
9. Deploy the Unica product and verify the Multicasting.

Chapter 13. Uninstalling the chart

1. To uninstall or delete the `my-release` deployment, run the following command:

   ```
   helm delete --purge <releasename>
   ```

2. Delete the persistent volumes.
3. Delete the file systems.

What to do next

If required, clean the persisted data of the database.