Oracle®

MicroTx Installation and Configuration Guide

Release 26.1
G54905-01
May 2026

Oracle MicroTx Installation and Configuration Guide, Release 26.1

G54905-01

Primary Author: Sylaja Kannan

Contributing Authors: Tulika Das

Contributors: Todd Little, Deepak Goel, Brijesh Kumar Deo, Bharath MC, Pruthvithej R, Satyanarayana Chillale, Atul Dhiman, Tushar Shaily, Chandrashekar Venkatachar, Deepak Kesawani, Himanshu Gaur, Shivanshu Singh

Contents

4 Install on a Kubernetes Cluster

1

About Transaction Manager for Microservices

Oracle Transaction Manager for Microservices (MicroTx) empowers enterprises to accelerate their adoption of microservices architectures for mission-critical applications by delivering a comprehensive set of tools that simplify the development, deployment, and ongoing maintenance of such applications. MicroTx enhances application reliability, scalability, and security, making it ideal for complex, modern enterprise environments.

MicroTx Distributed Transactions ensures transactional consistency across distributed microservices applications running in Kubernetes clusters. Supporting multiple distributed transaction protocols, including XA (eXtended Architecture), Long Running Actions (LRA), and Try-Confirm/Cancel (TCC). It provides enterprises with a robust, highly available, and secure solution for managing transactions in even the most demanding, large-scale deployments. See About MicroTx Distributed Transactions.

MicroTx Workflows offers a no-code enterprise platform for designing and orchestrating sophisticated agentic AI workflows. With specialized AI tasks, such as GenAI single-shot prompts, agent integration, and LLM-powered planning, as well as profile-based connectivity with LLMs, MCP servers, third-party tools, and scheduled workflows, MicroTx Workflows enables enterprises to securely and reliably build advanced AI-driven processes, backed by seamless Oracle Database integration and strong enterprise-grade security controls. See About MicroTx Workflows

Topics

- High-Level Tasks to Install MicroTx
  Use the high-level list of tasks as a reference to install, configure, and use Transaction Manager for Microservices (MicroTx).
- Quick Start
  The quickest way to get started is to run sample applications while using MicroTx to manage the workflows and transactions.

1.1 High-Level Tasks to Install MicroTx

Use the high-level list of tasks as a reference to install, configure, and use Transaction Manager for Microservices (MicroTx).

Task	Description	See
Review the environment requirements, supported platforms, and prerequisite configurations before proceeding with the installation.	Plan the installation and setup of MicroTx based on your business requirements.	Plan
Download the installation bundle.	The installation bundle contains the MicroTx image and other required files.	Download the Installation Bundle
Complete the authentication and authorization requirements.	Set up an identity provider and create an access token.	About Authentication and Authorization
Push MicroTx images to Docker registry, provide configuration information, and then install MicroTx.	Install MicroTx on a Kubernetes cluster. Provide configuration information in the `values.yaml` file.	Install on a Kubernetes Cluster

Task	Description	See
Access MicroTx.	Verify that the MicroTx components were installed successfully and access the services.	Access MicroTx

1.2 Quick Start

The quickest way to get started is to run sample applications while using MicroTx to manage the workflows and transactions.

You can run a script to quickly deploy MicroTx on your local machine, and then run sample applications. See https://github.com/oracle-samples/microtx-samples/tree/main/docs/quickstart.

> ⚠ **Caution**
>
> Run this script only in test or development environments. Do not use this script in production environments.

2
Plan

Consider the points discussed in this section to plan the installation and setup of Transaction Manager for Microservices (MicroTx).

- Supported Container Platforms
- Supported Authorized Cloud Environments
- Supported Languages and Frameworks
- Supported Data Stores
  MicroTx uses a data store to persist transaction records, service logs, and metadata.
- Supported Identity Providers
- About Authentication and Authorization
  Authentication ensures that only authorized individuals can access MicroTx, the microservices, transaction, and data. Authorization provides access control to system privileges and data. This builds on authentication to ensure that individuals get appropriate access.

2.1 Supported Container Platforms

You can deploy MicroTx Workflows on Kubernetes.

You can deploy MicroTx Distributed Transactions coordinator on Docker or Kubernetes cluster. MicroTx Distributed Transactions is tested with the following platforms:

- Kubernetes 1.21.x. Use any Kubernetes distribution that is compatible with Kubernetes 1.21.x.
- Docker 20.10.x. Use any operating system that supports Docker 20.10.x or a compatible version.

2.2 Supported Authorized Cloud Environments

You can use the Transaction Manager for Microservices Enterprise Edition licensing in the following authorized cloud environments.

- Amazon Web Services – Kubernetes clusters on Amazon Elastic Compute Cloud (EC2) and Amazon Relational Database Service (RDS)
- Microsoft Azure Platform - Kubernetes clusters on Azure virtual machines, Azure Kubernetes Services (AKS), Azure Arc, and Azure Stack
- Oracle Cloud Infrastructure (OCI) - Kubernetes Engine (OKE), which enables you to deploy, manage, and scale containerized applications on OCI. See Overview of Kubernetes Engine (OKE).

For more information about licensing information, see https://www.oracle.com/a/ocom/docs/corporate/oracle-software-licensing-basics.pdf.

2.3 Supported Languages and Frameworks

Use MicroTx Distributed Transactions to ensure transactional consistency across microservices applications implemented in the following languages:

- TypeScript or JavaScript for Node.js
- Java 11 (for applications built with frameworks, such as Helidon 2.x and WebLogic Server)
- Java 17 (for applications built with frameworks, such as Helidon 3.x, Helidon 4.x, Spring Boot 3.x, and Micronaut 4.2.1 or later)
- Python 3.11 or later

MicroTx Distributed Transactions supports Node.js and Java for all the transaction protocols and supports Python only for TCC.

Supported Java Frameworks

The MicroTx Distributed Transactions libraries are available for Spring REST applications and JAX-RS applications that use the XA, Saga, and TCC transaction protocols. MicroTx Distributed Transactions library is also available for Micronaut applications that use the Saga transaction protocol and ORDS applications that use the XA transaction protocol.

The MicroTx Distributed Transactions XA library is available for the following Java frameworks:

- Spring Boot 3.x, which includes Hibernate, EclipseLink, and MyBatis 3.5.x applications
- Helidon 2.x, 3.x, and 4.x
- Oracle WebLogic Server 14. See Integrating XA Global Transactions Between WebLogic Server and Helidon Using MicroTx in *Integrating Oracle WebLogic Server with Helidon*.
- Oracle Tuxedo 22c
- Oracle REST Data Services (ORDS) 19c
- Micronaut 4.2.1 or later

2.4 Supported Data Stores

MicroTx uses a data store to persist transaction records, service logs, and metadata.

Supported Data Store for MicroTx Workflows

MicroTx Workflows supports Oracle Database 19c and later as the data store.

Supported Data Store for MicroTx Distributed Transactions

MicroTx Distributed Transactions coordinator supports the following data stores.

- Oracle Database 19c and later
- etcd
- embedded LevelDB-based local storage
- in-memory storage

Oracle Database and etcd are centralized data stores. They provide shared, durable storage that can be accessed by all coordinator replicas. Use these storage types when you need a common external store for transaction state, recovery, and coordination across replicas.

The embedded LevelDB-based local storage type uses an embedded LevelDB-based store on the coordinator instance. When session affinity is enabled, requests for a transaction are routed to the same coordinator replica, allowing the embedded database to act as local persistent storage. This avoids the network hop and contention associated with centralized stores such as Oracle Database or etcd, and can improve transaction processing performance. The embedded database is suitable for production deployments that use session affinity and are designed around local coordinator storage.

The in-memory storage type stores transaction data only in memory. Use it for local development, testing, and samples. It is not intended for production because data is lost when the coordinator restarts.

For Oracle Database, you can connect to an on-premises Oracle Database or to an Oracle Cloud Infrastructure Database service. Supported Oracle Database environments include:

- Oracle On-Premises Database
- Autonomous Database for Transaction Processing and Mixed Workloads, shared or dedicated
- Bare Metal and Virtual Machine DB Systems in Oracle Cloud Infrastructure
- Oracle Exadata Cloud Service
- Oracle Exadata Cloud@Customer
- Oracle Real Application Clusters (RAC) 19c

2.5 Supported Identity Providers

You can use the following identity providers to create the authentication information and secure communication.

- Oracle IAM
- Keycloak
- Microsoft Azure Active Directory and Active Directory

This guide provides information about creating an access token using Oracle IAM.

If you want to use Keycloak or Microsoft AD as the identity provider, refer to their product documentation for information about setting up the identity provider and creating an access token.

2.6 About Authentication and Authorization

Authentication ensures that only authorized individuals can access MicroTx, the microservices, transaction, and data. Authorization provides access control to system privileges and data. This builds on authentication to ensure that individuals get appropriate access.

- About Access and Refresh Tokens
  Use access and refresh tokens to ensure that only authenticated users can access MicroTx and to permit only administrative users or users that originally initiated the transaction to manage a transaction.
- About Encrypting and Storing Tokens
  To support asynchronous calls, MicroTx stores the access and refresh tokens, and then uses them in asynchronous calls.

- About the Oracle_Tmm_Tx_Token Transaction Token
  Enable the creation and propagation of the transaction token to ensure that only authorized users have access to the service. When you set `transactionTokenEnabled` to `true` in the YAML file, MicroTx Distributed Transactions creates a new token called `Oracle_Tmm_Tx_Token`, which is a signed transaction token.
- Overview of Role-Based Access Control for MicroTx Workflows
  MicroTx Workflows uses hierarchical role-based access control (RBAC) to secure APIs and operational capabilities. These mappings ensure that users are granted appropriate permissions in MicroTx Workflows based on the roles assigned by your identity provider. This supports robust authorization controls for both system administrators and regular users of MicroTx Workflows.

2.6.1 About Access and Refresh Tokens

Use access and refresh tokens to ensure that only authenticated users can access MicroTx and to permit only administrative users or users that originally initiated the transaction to manage a transaction.

Use an identity provider to create an access token and a refresh token. When you send a new REST API request, such as a request to book a trip, you must pass the access and refresh tokens in the request header.

Access Token

When you enable authentication, you must pass the access token in the `authorization` header with every request. MicroTx enforces JWT-based authentication and validates the access token in all incoming requests against the public key. It also validates all the calls sent from the MicroTx library to the transaction coordinator. MicroTx checks that the user who passes the access token has the required system privileges to perform the operation. This ensures that only authorized users can access the MicroTx APIs.

When you enable authorization checks at coordinator and if you do not provide the access token when you send the request, the transaction is rejected as there is no access token.

Refresh Token

Refresh token is used to refresh an expired access token. Asynchronous calls or transactions could span a few minutes or hours. For example, you use the Saga transaction protocol to book a hotel and flight. It can take a few minutes for the user to complete the bookings. However, the access token could expire before the user completes the transaction. When you specify the URL and client ID of the identity provider in the YAML file, MicroTx provides the refresh token to the identity provider and gets a new access token.

2.6.2 About Encrypting and Storing Tokens

To support asynchronous calls, MicroTx stores the access and refresh tokens, and then uses them in asynchronous calls.

To encrypt the tokens, create encryption keys. MicroTx encrypts the tokens and stores it. When there is an asynchronous call from MicroTx to a participant service, MicroTx fetches the encrypted token, decrypts it, and then attaches the token to the request header.

MicroTx encrypts the access and refresh tokens, and then uses it later while making calls to participant services. For each transaction, MicroTx generates a new value for the initialization vectors. Each transaction record contains the encrypted metadata information, such as key version and initialization vector value.

2.6.3 About the Oracle_Tmm_Tx_Token Transaction Token

Enable the creation and propagation of the transaction token to ensure that only authorized users have access to the service. When you set `transactionTokenEnabled` to `true` in the YAML file, MicroTx Distributed Transactions creates a new token called `Oracle_Tmm_Tx_Token`, which is a signed transaction token.

The following steps describe how MicroTx Distributed Transactions creates the `Oracle_Tmm_Tx_Token` transaction token and propagates it in the subsequent communication between the participant services and MicroTx Distributed Transactions.

1. When a user begins a transaction, the transaction initiator service sends a request to MicroTx Distributed Transactions.
2. MicroTx Distributed Transactions responds to the transaction initiator and returns `Oracle_Tmm_Tx_Token` in the response header.
   The MicroTx Distributed Transactions library creates this token based on the public-private key pair that you provide. You don't have to create the `Oracle_Tmm_Tx_Token` transaction token or pass it in the request header.

   MicroTx Distributed Transactions works with multiple headers and tokens. For the sake of simplicity, we are limiting our discussion to the `Oracle_Tmm_Tx_Token` transaction token in this section.
3. For all the subsequent calls from the participant services to the transaction coordinator, the MicroTx Distributed Transactions library passes `Oracle_Tmm_Tx_Token` in the request header.

To enable propagation of the transaction token in a Kubernetes Cluster, see Transaction Token Properties.

2.6.4 Overview of Role-Based Access Control for MicroTx Workflows

MicroTx Workflows uses hierarchical role-based access control (RBAC) to secure APIs and operational capabilities. These mappings ensure that users are granted appropriate permissions in MicroTx Workflows based on the roles assigned by your identity provider. This supports robust authorization controls for both system administrators and regular users of MicroTx Workflows.

The roles are ordered by privilege as follows:

READ_ONLY_USER < **USER** < **METADATA_MANAGER** < **ADMIN** < **SUPER_ADMIN**

Higher-privileged roles automatically inherit permissions granted to lower-level roles. The `WORKER` role is a specialized runtime role used for task-processing APIs and must be explicitly included in higher-role mappings when task operations are required.

`READ_ONLY_USER` provides observability-only access to workflows, metadata, connectors, storage, scheduler information, and version APIs. `USER` extends these capabilities by allowing workflow execution, workflow updates, storage operations, notifications, conversational APIs, and MicroTx runtime operations. `METADATA_MANAGER` is intended for workflow and platform administrators who manage metadata, connectors, schedulers, and workflow bulk operations. `ADMIN` provides administrative access to queues, events, secrets, and other platform-level APIs. `SUPER_ADMIN` has unrestricted access to all APIs and operations across the platform.

In general:

- GET and READ operations require `READ_ONLY_USER`

- Workflow execution and runtime POST operations require `USER`
- Metadata and scheduler management require `METADATA_MANAGER`
- Administrative and infrastructure operations require `ADMIN`
- Full platform access requires `SUPER_ADMIN`
- Task-processing APIs require `WORKER`

This model enables separation of duties between platform administrators, workflow developers, runtime users, and automated worker services while ensuring controlled privilege escalation through inheritance.

3
Prepare

Before you install Transaction Manager for Microservices (MicroTx), set up a transaction store, identity provider, and optionally, a load balancer.

- Set up a data store, which persists data, for MicroTx Distributed Transactions and MicroTx Workflows.
- Set up an identity provider. Use the OpenID Connect JWT tokens to authenticate and authorize user access to MicroTx Distributed Transactions.
- Optionally, set up a load balancer.

Topics:

3.1 Download the Installation Bundle

Perform the following steps to download the Transaction Manager for Microservices (MicroTx) installation bundle to your local system:

1. Visit https://www.oracle.com/database/transaction-manager-for-microservices/.
2. Click **Download** to download either Oracle Transaction Manager for Microservices Enterprise Edition 26.1 or Oracle Transaction Manager for Microservices Free 26.1. You are redirected to Oracle Software Delivery Cloud.
3. Download the Transaction Manager for Microservices installation bundle using the Oracle Download Manager or click the ZIP file to download it.
4. Create a new directory in your local machine.

5. Extract the contents of the ZIP file to the new directory that you have created. Unzip the Transaction Manager for Microservices installation bundle.

   ```
   unzip MicroTx-<version>_<edition>.zip
   ```

   Where, <*edition*> refers to the Free or Enterprise Edition.

6. Run the following command to view the list of files that are extracted.

   ```
   ls -l MicroTx-<version>_<edition>
   ```

3.2 Download the MicroTx images from Oracle Container Registry

The MicroTx images are based on x86-64 or ARM architecture.

To access Oracle Container Registry, you must have an Oracle Single Sign-On account.

1. Go to Oracle Container Registry.
2. In the search box, type **MicroTx**.

   Information about three images are displayed in the Search Results page.
3. To download images for MicroTx Free, complete the following tasks.
   a. In the Search Results page, click **otmm**. The search results page with details of images for MicroTx Free are displayed.
   b. In the **Tags** section of the page, you can view the available versions of MicroTx. Run one of the following commands:
      - To download the latest version of the MicroTx Workflows image, run the following command:

        ```
        docker pull container-registry.oracle.com/database/microtx-workflow:latest
        ```

      - To download the latest version of the MicroTx console image, run the following command:

        ```
        docker pull container-registry.oracle.com/database/microtx-console:latest
        ```

      - To download the latest version of the MicroTx Distributed Transactions coordinator image, run the following command:

        ```
        docker pull container-registry.oracle.com/database/microtx-coordinator:latest
        ```

4. To download images of the MicroTx Distributed Transactions coordinator and MicroTx console, complete the following tasks:
   a. In the Search Results page, click **microtx-ee-coordinator**. The search results page with details of images for MicroTx Enterprise Edition are displayed.

b. In the **Tags** section of the page, you can view the available versions of MicroTx. Run one of the following commands:

- To download the latest version of the MicroTx Workflows (Enterprise Edition) image, run the following command:

```
docker pull container-registry.oracle.com/database/microtx-ee-workflow:latest
```

- To download the latest version of the MicroTx (Enterprise Edition) console image, run the following command:

```
docker pull container-registry.oracle.com/database/microtx-ee-console:latest
```

- To download the latest version of the MicroTx Distributed Transactions (Enterprise Edition) coordinator image, run the following command:

```
docker pull container-registry.oracle.com/database/microtx-ee-coordinator:latest
```

3.3 Set Up Oracle Identity Providers

Use Oracle IAM as an identity provider to manage access to your application.

If you want to use Keycloak or Microsoft AD as the identity provider, refer to their product documentation for information about setting up the identity provider and creating an access token.

- Use Oracle IAM as Identity Provider
  You can use Oracle IAM as identity provider to manage access to your application.
- Run the Discovery URL
  After setting up the identity provider, run the Discovery URL in any browser to note down the values that you must provide in the `values.yaml` file for authentication purposes.
- Create an Access Token
  This topic provides details to create an access token when you use Oracle IAM as the identity provider.

3.3.1 Use Oracle IAM as Identity Provider

You can use Oracle IAM as identity provider to manage access to your application.

1. In the Oracle Cloud Infrastructure console, add your application as a confidential application. For information to add a confidential application and configure application details and the display settings, see Adding a Confidential Application in *Oracle Cloud Infrastructure documentation*.
2. While adding a confidential application, perform the following tasks:
   a. On the **Configure OAuth** pane, under **Resource server configuration**, select **No resource server configuration**.
   b. On the **Configure OAuth** pane, under **Client configuration**, select **Configure this application as a client now**.
   c. Select the following options as the allowed grant types under **Authorization**.

- **Resource owner**
- **Client credentials**
- **JWT assertion**
- **Refresh token**
- **Authorization code**
- **Implicit**
- **Allow non-HTTPS URLs**

d. Enter HTTP URLs for the redirect URL, logout URL, or post logout redirect URL fields in the following format.

- **Redirect URL**: `http://<console-app-ip>/oidc/redirect`.
- **Post-logout redirect URL**: `http://<console-app-ip>/consoleui/index.html`
- **Logout URL**: `http://<console-app-ip>/oidc/logout`

e. Select **Client type** as **Confidential**.

After completing the configuration, click **Submit**.

3. Under **General Information**, note down the values for **Client ID** and **Client secret**. You will provide these values later to generate token and to deploy the application.
4. Create users and groups in the **User Management** tab of the selected identity domain, and then add users to groups. See Managing Oracle Identity Cloud Service Users and Groups in the Oracle Cloud Infrastructure Console.
5. Add users and groups to the confidential application that you have created. See Assigning Users to Custom Applications and Assigning Groups to Custom Applications,
6. Run the following command to configure a custom claim in the Oracle IAM Identity Domain so that user group information is automatically included in OAuth or OIDC tokens.

```
curl -sS -X POST "https://idcs-77c....identity.oraclecloud.com:443/
admin/v1/CustomClaims" \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  --data-binary '{
    "schemas": ["urn:ietf:params:scim:schemas:oracle:idcs:CustomClaim"],
    "name": "group_roles",
    "value": "$user.groups.*.display",
    "expression": true,
    "mode": "always",
    "tokenType": "AT",
    "allScopes": true
  }'
```

Typically, you need to perform this operation only once to configure the identity domain. After configuration, every issued token for authorized users contains the user's group information. This command also displays the names of the groups based on the configuration details. Note down the names of the groups as you will provide this later in the `values.yaml` file as the names of roles for MicroTx Workflows.

7. Run the following command to generate an access token for any user of the confidential application that you have created.

```
curl -i
     -H "Authorization:Basic <base64 of client_id:client_secret>"
     -H "Content-Type: application/x-www-form-urlencoded;  charset=UTF-8"
     --request POST https://idcs-77c1....identity.oraclecloud.com:443/
oauth2/v1/token
     -d
"grant_type=password&scope=urn:opc:idm:__myscopes__&username=qauser_conduct
or_admin&password=<some_value>"
```

3.3.2 Run the Discovery URL

After setting up the identity provider, run the Discovery URL in any browser to note down the values that you must provide in the `values.yaml` file for authentication purposes.

To run the Discovery URL and note down the required identity provider information:

1. Run the Discovery URL in any browser.

   Syntax of Discovery URL

   ```
   https://<tenant-base-url>/.well-known/openid-configuration
   ```

   Example Discovery URL

   ```
   https://idcs-a83e....identity.oraclecloud.com/.well-known/openid-
   configuration
   ```

   The example tenant base URL has been truncated with ellipses (...) for readability. When you run this command in your environment, copy the actual value.

   A list of values is displayed.

2. Note down the values for the `issuer` and `jwksUri` fields. You will need to provide these values in the `values.yaml` file. For example:

   ```
   issuer: "https://identity.oraclecloud.com"
   jwksUri: "https://idcs-a83e....identity.oraclecloud.com:443/admin/v1/
   SigningCert/jwk"
   ```

3. Note down the value of the `token_endpoint` field. You will provide this information in the `values.yaml` file as value for the `tmmConfiguration.identityProvider.identityProviderUrl` property.

   ```
   token_endpoint: "https://idcs-a83e.....identity.oraclecloud.com/oauth2/v1/
   token"
   ```

3.3.3 Create an Access Token

This topic provides details to create an access token when you use Oracle IAM as the identity provider.

If you want to use Keycloak or Microsoft AD as the identity provider, refer to their product documentation for information about setting up the identity provider and creating an access token.

API calls to the service require a valid access token. Create an access token which you can specify in subsequent API calls to the service. In addition to the access token, you can also specify the refresh token in subsequent API calls to the service. MicroTx uses the refresh token to refresh an expired access token.

Before you begin, ensure that you have set up your identity provider and noted down the values for client ID, client secret, and the domain URL.

1. Create an access token. See Example Authorization Flow in *REST API for Oracle Identity Cloud Service*.
2. Store the access token and refresh tokens in environment variables, as shown in the following example for a Linux host.

   ```
   export TOKEN="eyJ4Lm..."
   export REFRESH_TOKEN="ey5Gkr..."
   ```

3. Store the authentication cookie in an environment variable, as shown in the following example for a Linux host.

   ```
   export OTMM_COOKIE="eyJh...x_THw"
   ```

   The example value has been truncated with ellipses (...) for readability.

After you obtain the OAuth 2.0 tokens, use the tokens in the `authorization` and `refresh-token` headers while making subsequent API calls to the service.

3.4 Prepare a Kubernetes Cluster

In Kubernetes, you can install Transaction Manager for Microservices (MicroTx) within a service mesh or without it.

The installation bundle provides example Helm charts with sample values which you can use as a reference to install MicroTx. This section provides instructions to install MicroTx on a Kubernetes cluster with and without using the Istio service mesh.

You can create a similar configuration to install MicroTx in other supported environments. If you are using another service mesh in a Kubernetes cluster, create your own Helm charts.

Istio is a service mesh that provides a separate infrastructure layer to handle inter-service communication. Network communication is abstracted from the services themselves and is handled by proxies. Envoy is the proxy that is deployed as a sidecar inside the microservices container. All communication inside the service mesh is done through the Envoy proxies.

Topics:

- Considerations for Deployment on Kubernetes
  Consider the following factors while deploying MicroTx on Kubernetes.

- Create a Kubernetes Cluster
  Create a Kubernetes cluster or use an existing one. You will install MicroTx onto this cluster.
- Install the Required Software for Kubernetes
  Before installing MicroTx in Kubernetes, you must install and configure the required software on your local machine.
- Install and Configure Istio
  Install Istio, 1.27 or later versions, onto the Kubernetes cluster with the default Istio profile.
- Create a Kubernetes Secret with SSL Details for Istio
  To enable access to Istio using the HTTPS protocol, you must create a Kubernetes secret that contains details of an SSL key and certificate. MicroTx uses this information to access Istio using HTTPS.
- Create a Kubernetes Secret to Access Docker Registry
  When you install the application using Helm, use a Kubernetes secret to provide the authentication details to pull an image from the remote repository.
- Authenticate and Authorize
  Authentication ensures that only authorized individuals get access to the system and data. Authorization provides access control to system privileges and data. This builds on authentication to ensure that individuals get appropriate access.

3.4.1 Considerations for Deployment on Kubernetes

Consider the following factors while deploying MicroTx on Kubernetes.

The installation bundle provides reference Helm charts and this document provides details for a sample deployment of MicroTx in a Kubernetes cluster with Istio service mesh. If you are using another service mesh in a Kubernetes cluster, create your own Helm charts.

Supported Kubernetes Platforms

Deploy MicroTx in a Kubernetes cluster that is running in your data center or your cloud environment. MicroTx is tested with Kubernetes 1.21.x or compatible versions on the following platforms:

- Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE). See Creating a Kubernetes Cluster in *Oracle Cloud Infrastructure documentation*.
- Minikube
- Oracle Linux Container Native Environment

Deployment Across Multiple Kubernetes Clusters

You can deploy your application microservices and MicroTx within a single Istio service mesh in a single Kubernetes cluster.

When your application microservices are distributed across multiple Kubernetes cluster, or if you want MicroTx to communicate with Oracle Database or Tuxedo, then you can deploy MicroTx in a separate Kubernetes cluster. In such a scenario, each Kubernetes cluster will contain an Istio service mesh. You will have to configure ingress and egress gateways to enable communication between multiple Istio services meshes.

3.4.2 Create a Kubernetes Cluster

Create a Kubernetes cluster or use an existing one. You will install MicroTx onto this cluster.

Before you begin, you must plan the environment in the following way:

- Decide if you require a single-node or a multinode Kubernetes cluster to host MicroTx. Oracle recommends that you create at least a single-node cluster in development environments and at least a three-node cluster in production environments.
- Identify the different components in your environment. If your microservices are running in a Kubernetes cluster, you can install MicroTx in the same cluster or a different cluster. If your microservices are distributed across multiple Kubernetes clusters or if you want MicroTx to communicate with components such as Oracle Database or Tuxedo, which are not part of any Kubernetes cluster, create a Kubernetes cluster to host MicroTx.

3.4.3 Install the Required Software for Kubernetes

Before installing MicroTx in Kubernetes, you must install and configure the required software on your local machine.

Perform the following steps to install the required software and configure the environment in your local machine:

1. Install and configure Kubernetes command-line interface (Kubectl), 1.21.x or later versions, to create and manage your deployments in your Kubernetes cluster. See https://kubernetes.io/docs/tasks/tools/.
2. Install the latest version of Helm 3.x on your local machine. For more information, see https://helm.sh/docs/intro/install/.

   Use Helm to make deployments easier as you can run a single command to install applications and resources into Kubernetes clusters. Helm interacts with the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources.
3. Create a namespace. The following command creates a namespace with the name `MicroTx`.

   Sample Command

   ```
   kubectl create ns otmm
   ```

   Sample Response

   ```
   namespace/otmm created
   ```

   Note down the name of the namespace. Later, you will deploy MicroTx in this namespace.

3.4.4 Install and Configure Istio

Install Istio, 1.27 or later versions, onto the Kubernetes cluster with the default Istio profile.

> **Note**
>
> Skip this section if you don't want to install MicroTx in an Istio service mesh.

Oracle recommends using the default Istio profile for production environments. Additionally, create logs that you can use for audits and enable distributed tracing to monitor and troubleshoot microservices-based distributed systems, such as monitoring distributed transactions, analyzing the root cause, analyze service dependency, and optimize performance or latency. For more information, see https://istio.io/latest/docs/setup/additional-setup/config-profiles/.

Perform the following steps to install and configure Istio on your local machine:

1. Download Istio and add the `istioctl` client tool which is located in the `bin` folder to the `PATH` for your workstation. See https://istio.io/latest/docs/setup/additional-setup/download-istio-release/.
2. Run prerequisite checks to validate if the cluster meets Istio install requirements.

   ```
   istioctl x precheck
   ```

   The following message is displayed. You can proceed with the next step and install Istio if there are no issues.

   ```
   No issues found when checking the cluster. Istio is safe to install or
   upgrade!
   ```

3. Run the following commands to install and configure Istio in Kubernetes.
   a. Run the following command to install Istio on Kubernetes with the default Istio profile.

      ```
      istioctl install --set meshConfig.accessLogFile=/dev/stdout \
          --set meshConfig.accessLogEncoding=JSON \
          --set meshConfig.enableTracing=true \
          --set meshConfig.defaultConfig.tracing.sampling=100.0
      ```

      Only in Oracle Linux 8 environments, run the following command to pass an additional flag, `--set components.cni.enabled=true`. For example:

      ```
      istioctl install --set meshConfig.accessLogFile=/dev/stdout \
          --set meshConfig.accessLogEncoding=JSON \
          --set meshConfig.enableTracing=true \
          --set meshConfig.defaultConfig.tracing.sampling=100.0 \
          --set components.cni.enabled=true
      ```

   b. Label the namespace that you have created with `istio-injection=enabled` to put automatic sidecar injection into effect. The following command labels the `otmm` namespace:
      Sample Command

      ```
      kubectl label namespace otmm istio-injection=enabled
      ```

      Sample Response

      ```
      namespace/otmm labeled
      ```

3.4.5 Create a Kubernetes Secret with SSL Details for Istio

To enable access to Istio using the HTTPS protocol, you must create a Kubernetes secret that contains details of an SSL key and certificate. MicroTx uses this information to access Istio using HTTPS.

Before you begin, ensure that you have installed Istio in the Kubernetes cluster. See Install the Required Software for Kubernetes.

To create a Kubernetes secret with the details of the SSL certificates:

1. Identify the location of the SSL certificates that you want to use.
2. Import the SSL certificates to the Kubernetes namespace where you have installed Istio. Run the following command to create a Kubernetes secret with details of the SSL certificate.

   Syntax

   ```
   kubectl create secret tls tls-credential --key=ssl-key-file-path --
   cert=ssl-cert-file-path -n istio-system
   ```

   Example

   ```
   kubectl create secret tls tls-credential --key=~/certificates/
   example.dev.key --cert=~/certificates/example.dev.crt -n istio-system
   ```

   Where,

   - `tls` is the type of the secret.
   - *`tls-credential`* is the name of the Kubernetes secret that you want to create.
   - *`ssl-key-file-path`* is the location of the SSL certificate key file.
   - *`ssl-cert-file-path`* is the location of the SSL certificate file.
   - *`istio-system`* is the namespace in which you have installed Istio. The default namespace is `istio-system`. If you have installed Istio in another namespace, run the `kubectl get ns` command to find all the namespaces in the cluster.

Note down the name of the Kubernetes secret as you will need to provide this detail in the `values.yaml` file to install MicroTx.

3.4.6 Create a Kubernetes Secret to Access Docker Registry

When you install the application using Helm, use a Kubernetes secret to provide the authentication details to pull an image from the remote repository.

The Kubernetes Secret contains all the login details you provide if you were manually logging in to the remote Docker registry using the `docker login` command, including your credentials.

1. Create a secret by providing the credentials on the command-line by using the following command.

   ```
   kubectl create secret docker-registry NAME --docker-server=SERVER --docker-
   username=USERNAME --docker-password=PASSWORD --docker-email=EMAIL --
   namespace=NAMESPACE
   ```

Where,

- `NAME`: Name of the Kubernetes secret that you want to create. Note down this name as you will use this name later in the manifest file to refer to the secret.
- `SERVER`: Name of your private Docker registry. The format varies based on your Kubernetes platform. For example, the format of the user name in Oracle Cloud Infrastructure environment is `<region-key>.ocir.io`.
- `USERNAME`: User name to access the remote Docker registry. The format varies based on your Kubernetes platform. For example, the format of the user name in Oracle Cloud Infrastructure environment is `<tenancy-namespace>/<oci-username>`.
- `PASSWORD`: Password to access the remote Docker registry.
- `EMAIL`: Email ID for your Docker registry.
- `NAMESPACE`: Namespace where you want to deploy MicroTx.

Example

Use the following command to create a Kubernetes secret with the name *regcred* in the `otmm` namespace.

```
kubectl create secret docker-registry regcred --docker-server=iad.ocir.io
--docker-username=mytenancy/myuser --docker-password=pwd --docker-
email=myuser@example.com --namespace=otmm
```

2. Note down the name of the secret that you have created. You will need to provide this value later.
3. Close the terminal.

   When you type secrets at the command line, the command line may store the secrets in your shell history unprotected. The secrets might also be visible to other users on your machine during the time that kubectl is running. To overcome this issue, you can close the terminal after creating the secret.

You can also create a secret based on existing credentials. See https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/#registry-secret-existing-credentials. For information about pulling images from OCI Kubernetes Engine (OKE), see Pulling Images from Container Registry during Kubernetes Deployment in *Oracle Cloud Infrastructure Documentation* .

3.4.7 Authenticate and Authorize

Authentication ensures that only authorized individuals get access to the system and data. Authorization provides access control to system privileges and data. This builds on authentication to ensure that individuals get appropriate access.

- Generate a Kubernetes Secret for an Encryption Key
  To support asynchronous calls, MicroTx stores the authorization and refresh tokens. To store the tokens, encrypt them as you can't store the tokens directly. To encrypt the tokens, create encryption keys.
- Create a Key Pair for Transaction Token
  The application supports including a MicroTx signed transaction token which is unique to each MicroTx transaction.

3.4.7.1 Generate a Kubernetes Secret for an Encryption Key

To support asynchronous calls, MicroTx stores the authorization and refresh tokens. To store the tokens, encrypt them as you can't store the tokens directly. To encrypt the tokens, create encryption keys.

MicroTx encrypts the tokens using the encryption keys that you provide. When there is an asynchronous call from MicroTx to participant services, MicroTx fetches the encrypted token, decrypts it, and then attaches the token to the authorization header.

You must generate an encryption key, and then add the key to a Docker secret if you have enabled the `authTokenPropagationEnabled` property under `authorization`. The encryption key that you generate must have the following attributes.

- Symmetric algorithm: AES-256
- Cipher mode: AES in GCM mode
- Key length: 32 bytes
- Length of initialization vectors: 96 bits

MicroTx encrypts the access and refresh tokens, and then uses it later while making calls to participant services. For each transaction, MicroTx generates a new value for the initialization vectors. Each transaction record contains the encrypted metadata information, such as key version and initialization vector value.

Generate a Kubernetes Secret for an Encryption Key for MicroTx Workflows

1. Run the following command to generate an encryption key, that is 32-bytes long.

   ```
   openssl rand -hex 32 > encryption.key
   ```

   This creates an encryption key file named `encryption.key`.
2. Use the encryption key file to create a Kubernetes secret. You must create this secret in the namespace where you want to install MicroTx Workflows. The following sample command creates a Kubernetes secret with the name `encryption-secret-key1` in the `otmm` namespace.

   ```
   kubectl create secret generic encryption-secret --from-file=encryption.key
   -n otmm
   ```

3. Note down the name of the Kubernetes secret and the file name stored in the secret. You will provide these values for the `secretName` and `fileName` fields under `workflow.server.encryption` in the `values.yaml` file.

Generate a Kubernetes Secret for an Encryption Key for MicroTx Distributed Transactions

1. Run the following command to generate an encryption key with a key length of 32 bytes.

   ```
   openssl rand -hex 16
   ```

   Note down the value that is generated. For example, `e9f0adab17c0180425147166c2ff1cd3`.

2. Create a Kubernetes secret while using the encrypted key that you have generated as the value. You must create this secret in the namespace where you want to install MicroTx.

   The following sample command creates a Kubernetes secret with the name `encryption-secret-key1` in the `otmm` namespace.

   ```
   kubectl create secret generic encryption-secret-key1 \ --from-
   literal=secret='e9f0adab17c0180425147166c2ff1cd3' -n otmm
   ```

3. Note down the name of the Kubernetes secret and its version. You will provide these values for the `secretKeyName` and `version` fields in the `values.yaml` file.

   The following code snippet provides sample values for the `encryption` field in the `values.yaml` file. The sample values in this example are based on the values used in the sample commands in this topic.

   ```
   encryption:
     encryptionSecretKeyVersion: "1"
     encryptionSecretKeys:
         - secretKeyName: "encryption-secret-key0"
           version: "0"
         - secretKeyName: " encryption-secret-key1"
           version: "1"
   ```

3.4.7.2 Create a Key Pair for Transaction Token

The application supports including a MicroTx signed transaction token which is unique to each MicroTx transaction.

When you set `transactionTokenEnabled` to true, MicroTx creates a new token called `tmm-tx-token`, which is a signed transaction token. When the transaction initiator begins a request, the MicroTx responds with the `tmm-tx-token`. To secure calls from the participant services to MicroTx, the MicroTx library passes `tmm-tx-token` in the request header. You don't have to create the `tmm-tx-token` transaction token or pass it in the request header. The MicroTx library creates this token based on the private-public key pair that you provide.

The transaction token that you generate must have the following attributes:

- Asymmetric algorithm: RSA 3072
- Key length: 3072 bits
- Hash algorithm: SHA256

Before you begin, ensure that you have installed OpenSSL.

1. Create RSA private key with key length as 3072 bits by using the following command:

   ```
   openssl genrsa -aes256 -out private.pem 3072
   ```

2. Enter a pass phrase at the command prompt, and then press enter. Remember the pass phrase as you will have to provide it later.

   A new file called `private.pem` is created in the current working folder. This file contains the RSA private key value.

3. Create a RSA public key for the private key that you have generated. Use the following command:

The following command creates a new file called `public.pem` in the current working folder. This file contains the RSA public key value.

```
openssl rsa -in private.pem -outform PEM -pubout -out public.pem
```

4. Run the following command to base64 encode the `private.pem` file.

   Example command

   ```
   base64 private.pem
   ```

   The base64-encoded value of the `private.pem` file is returned.

   Example response

   ```
   LS0tLS...LS0tLQo=
   ```

   The example response has been truncated with ellipses (...) for readability.

   Note down the base64-encoded value of the `private.pem` file.

5. Create a Kubernetes secret with the base64-encoded value of the `private.pem` file.

   The following command creates a Kubernetes secret with the name `TMMPRIVKEY1` in the `otmm` namespace, where you want to install MicroTx.

   ```
   kubectl create secret generic TMMPRIVKEY1 \ --from-
   literal=secret='LS0tLS...LS0tLQo=' -n otmm
   ```

   Note down the name of the Kubernetes secret. You will need to provide this value later in the `values.yaml` file.

6. Run the following command to base64 encode the `public.pem` file.

   Example command

   ```
   base64 public.pem
   ```

   The base64-encoded value of the `public.pem` file is returned.

   Example response

   ```
   LS0tLS...LS0tCg==
   ```

   The example response has been truncated with ellipses (...) for readability.

   Note down the base64-encoded value of the `public.pem` file.

7. Create a Kubernetes secret with the base64-encoded value of the `public.pem` file.

   The following command creates a Kubernetes secret with the name `TMMPUBKEY1` in the `otmm` namespace.

   ```
   kubectl create secret generic TMMPUBKEY1 \ --from-
   literal=secret='LS0tLS...LS0tCg==' -n otmm
   ```

Note down the name of the Kubernetes secret. You will need to provide this value later in the `values.yaml` file.

8. Create Kubernetes secret with the value as private key pass phrase that you had provided in step 2.

   The following command creates a Kubernetes secret with the name `TMMPRIVKEYPASSWD1` and key pass phrase as <*pph...*> in the `otmm` namespace.

   ```
   kubectl create secret generic TMMPRIVKEYPASSWD1 \ --from-
   literal=secret='<pph...>' -n otmm
   ```

   Where, <*pph...*> is the private key pass phrase. Replace this with a value specific to your environment.

   > **Note**
   >
   > Do not base64-encode the key pass phrase, as you must enter the key pass phrase in plain-text format.

   Note down the name of the Kubernetes secret. You will need to provide this value later in the `values.yaml` file.

3.5 Set Up Access to MicroTx Web Console

Complete all the tasks in this section to ensure that users can access the MicroTx web console. Skip this section if you don't want to set up the web console.

Topics:

- Specify the Admin Role in YAML file
  Create and assign the administrator role to users in your identity provider. After creating the admin role, update the YAML file with the name of the admin role.
- Create a Secret with Identity Provider Client Credentials
  To provide users the capability to log in to the MicroTx console, you must set up an identity provider. Create a Kubernetes secret to provide the client credentials of the identity provider to the MicroTx.
- Create a Secret with Cookie Encryption Password for Kubernetes
  In Kubernetes environment, the MicroTx Distributed Transactions console requires the session cookie to be encrypted. You must provide a cookie encryption password through a Kubernetes secret.
- Deploy Kubernetes Metrics Server
  To collect metrics about the health of the MicroTx Distributed Transactions coordinator and console, deploy the Metrics Server.

3.5.1 Specify the Admin Role in YAML file

Create and assign the administrator role to users in your identity provider. After creating the admin role, update the YAML file with the name of the admin role.

Administrators can use the MicroTx console and MicroTx Distributed Transactions REST APIs to view and manage *all* transactions. Other users can manage and view only the distributed transactions that they have initiated; they cannot view the transactions initiated by other users.

1. Create an administrator role in your identity provider.

   Note down the name of the role. You will need to provide this as the value of the `commonConfiguration.security.identityProvider.roleMappings.coordinator.adminUserRoles` property in the `values.yaml` file.

2. Assign the administrator role to users.

   For information about creating administrator role and assigning it to a user, refer to your identity provider documentation.

3. Note down the path to the administrator role from the JWT access token. You will provide this value for the `commonConfiguration.security.identityProvider.roleMappings.coordinator.adminUserRolesPath` property in the `values.yaml` file.

   - The following sample code snippet provides the role paths in the JWT access token for Oracle IAM. For JWT token in Oracle IAM, the roles are present under `userAppRoles`. Based on the sample provided below the value for the `adminUserRolesPath` property, in the `values.yaml` file, is `userAppRoles`.

     ```
     "userAppRoles": [
         "Identity Domain Administrator"
       ]
     ```

   - The following sample provides the role paths in the access token for Keycloak. Based on the sample provided below the value for the `adminUserRolesPath` property, in the `values.yaml` file, is `realm_access, roles`.

     ```
     # Admin role path in the JWT token from root.
         # Ex: For keycloak payload, roles are present under realm_access
         # {
         #   "realm_access": {
         #     "roles": [
         #       "admin"
         #     ]
         #   }
         # }
     ```

3.5.2 Create a Secret with Identity Provider Client Credentials

To provide users the capability to log in to the MicroTx console, you must set up an identity provider. Create a Kubernetes secret to provide the client credentials of the identity provider to the MicroTx.

Before you begin, ensure that you have set up your identity provider and noted down the values for client ID and client secret.

To create a Kubernetes secret with the identity provider client credentials:

1. Launch a terminal and enter the following commands to base64 encode the client ID and client secret.

   ```
   echo -n "clientid" | base64 -w 0
   echo -n "clientSecret" | base64 -w 0
   ```

Replace `clientid` and `clientSecret` with the values in your environment.

> **Note**
>
> For Linux, add `-w 0` to the command to remove line breaks.

The base64 encoded value of the client ID and client secret is returned. Note down these values as you will need it later.

2. Open the `console-identity-client-secret.yaml` file in any text editor. This file is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/helmcharts/resources` folder. Update the values of the client ID and password, and then save the changes.

```
apiVersion: v1
    kind: Secret
    metadata:
      name: console-identity-client-secret
    type: Opaque
    data:
      clientId: base64_encoded_clientId
      clientPassword: base64_encoded_clientSecret
```

Where,

- *`console-identity-client-secret`* is the name of the Kubernetes secret that you want to create. Note down this name as you will have to provide it later in the `values.yaml` file.
- *`base64_encoded_clientId`* and *`base64_encoded_clientSecret`* are the base64 encoded values of the client ID and client secret that you have generated in the previous step.

Replace these with values specific to your environment.

3. Run the following command to create a Kubernetes secret in the namespace where you want to install MicroTx.

Command syntax

```
kubectl apply -f <filename> -n <namespace>
```

The following sample command creates a Kubernetes secret with the name `console-identity-client-secret` in the `otmm` namespace with the details that you have provided in the `console-identity-client-secret.yaml` file.

```
kubectl apply -f console-identity-client-secret.yaml -n otmm
```

Note down the name of the secret, `console-identity-client-secret`. You'll provide this name as the value for the `commonConfiguration.security.clientSecretName` property in the `values.yaml` file.

3.5.3 Create a Secret with Cookie Encryption Password for Kubernetes

In Kubernetes environment, the MicroTx Distributed Transactions console requires the session cookie to be encrypted. You must provide a cookie encryption password through a Kubernetes secret.

To encrypt the MicroTx Distributed Transactions console session cookie, create a Kubernetes secret with the cookie encryption password:

1. Launch a terminal and enter the following commands to base64 encode the cookie encryption password.

   ```
   echo -n "cookie-encryption-password" | base64 -w 0
   ```

   Replace `cookie-encryption-password` with a password of your choice.

   > **Note**
   >
   > For Linux, add `-w 0` to the command to remove line breaks.

   The base64 encoded value of the password is returned. Note down this value as you will need it later.

2. Paste the following code in any text editor or reuse the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/helmcharts/resources/cookie-encryption-secret.yaml` file.

   ```
   apiVersion: v1
       kind: Secret
       metadata:
         name: console-cookie-encryption-password-secret
       type: Opaque
       data:
         secret: base64_encoded_cookieEncryptionPassword
   ```

   Where,

   - *`console-cookie-encryption-password-secret`* is the name of the Kubernetes secret that you want to create. Note down this name as you will have to provide it later in the `values.yaml` file.
   - *`base64_encoded_cookieEncryptionPassword`* is the base64 encoded value of the password that you have generated in the previous step.

3. Save the file as a YAML file. For example, `cookieEncryptionPassword.yaml`.
4. Run the following command to create a Kubernetes secret in the namespace where you want to install MicroTx Distributed Transactions.

   Command syntax

   ```
   kubectl apply -f <filename> -n <namespace>
   ```

The following sample command creates a Kubernetes secret with the name `console-cookie-encryption-password-secret` in the `MicroTx` namespace with the details that you have provided in the `cookieEncryptionPassword.yaml` file.

```
kubectl apply -f cookieEncryptionPassword.yaml -n otmm
```

Note down the name of the secret, `console-cookie-encryption-password-secret`. You'll provide this name as the value for the `console.tmmConsoleConfiguration.cookieEncryptionPasswordSecretName` property in the `values.yaml` file.

3.5.4 Deploy Kubernetes Metrics Server

To collect metrics about the health of the MicroTx Distributed Transactions coordinator and console, deploy the Metrics Server.

- Run the following command to deploy the Kubernetes Metrics Server:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/
releases/latest/download/components.yaml
```

For more information about the Metrics Server, see https://github.com/kubernetes-sigs/metrics-server/tree/master?tab=readme-ov-file#kubernetes-metrics-server.

3.6 Set Up Oracle Database as Data Store

Set up a data store for MicroTx Distributed Transactions and MicroTx Workflows to store logs.

MicroTx Workflows supports only Oracle Database as data store.

MicroTx Distributed Transactions supports etcd and Oracle Database as data store. Skip this section if you want to set up etcd as data store. See Set Up etcd as Data Store for MicroTx Distributed Transactions.

Topics:

- Prerequisites
  Create separate schemas for MicroTx Distributed Transactions and MicroTx Workflows to persist data in Oracle Database.
- Grant Privileges to Database User
  The MicroTx Distributed Transactions coordinator and MicroTx Workflows create a stored procedure when the service starts. If the Oracle Database user does not have the permission to create and run stored procedures, the service does not start.
- Get Autonomous Database Client Credentials
  MicroTx supports using Oracle Database as a data store to keep track of transaction information.
- Create Tables in Oracle Database
  Create the tables and schema required by MicroTx Distributed Transactions in Oracle Database using a SQL script only if the Oracle Database user does not have the required privileges, such as `ALTER` and `DELETE`. MicroTx Distributed Transactions creates tables automatically if the user has `ALTER` and `DELETE` privileges.
- Create a Kubernetes Secret for Oracle Database Credentials
  MicroTx Distributed Transactions and MicroTx Workflows support using Oracle Database as a persistent store to keep track of logs, metadata, and other operation details.

3.6.1 Prerequisites

Create separate schemas for MicroTx Distributed Transactions and MicroTx Workflows to persist data in Oracle Database.

Before setting up Oracle Database as data store for MicroTx Distributed Transactions and MicroTx Workflows, complete the following prerequisites:

- Install and configure a data store. Ensure that you set up the required networking rules to allow communication between the transaction coordinator and the data store. For details about setting up the Oracle Database, refer to the documentation that is specific to the database that you want to set up.
- Ensure that you have the required permissions to create tables in the database. When you install MicroTx, the service creates the required tables in the database. So MicroTx requires certain details about the database.
- If you use Oracle Database for persistence of data in both MicroTx Distributed Transactions and MicroTx Workflows, avoid using the same schema user for both. Create separate schemas for MicroTx Distributed Transactions and MicroTx Workflows. You can use the same Oracle Database instance, but ensure that the user schema is different.

3.6.2 Grant Privileges to Database User

The MicroTx Distributed Transactions coordinator and MicroTx Workflows create a stored procedure when the service starts. If the Oracle Database user does not have the permission to create and run stored procedures, the service does not start.

Perform this task only if you use Oracle Database as the data store. Grant the following privileges to the user who has access to the data store.

- To grant the privileges to create and run stored procedures, run the following command.

  Sample Command

  ```
  GRANT CREATE SESSION, CREATE TABLE, CREATE PROCEDURE, EXECUTE ANY
  PROCEDURE, CREATE VIEW, CREATE SEQUENCE, CREATE TYPE TO <data_store_user>;
  ```

  Where, <*data_store_user*> is the name of the user who has access to the data store.
- To grant tablespace which is required to create tables and procedures, run the following command. The following command grants unlimited tablespace privilege to the specified user. Modify the quota as per your requirement.

  Sample Command

  ```
  GRANT UNLIMITED TABLESPACE TO <data_store_user>;
  ```

3.6.3 Get Autonomous Database Client Credentials

MicroTx supports using Oracle Database as a data store to keep track of transaction information.

Skip this task if you are not using an Autonomous Database instance.

If you are using an Autonomous Database instance, perform the following steps to retrieve the Oracle client credentials (wallet files):

1. Download the wallet from the Autonomous Database instance. See Download Client Credentials (Wallets) in *Using Oracle Autonomous Database on Shared Exadata Infrastructure*.
   A ZIP file is downloaded to your local machine. Let's consider that the name of the wallet file is `Wallet_database.zip`.
2. Unzip the wallet file.

   ```
   unzip Wallet_database.zip
   ```

   The files are extracted to a folder. Note down the name of this folder. You will need to provide it in the next steps.
3. Create a configuration map to store the location of the folder where you have extracted the wallet files.
   Perform this step only if you want to deploy MicroTx in a Kubernetes cluster.

   Ensure that you create the configuration map in the namespace where you want to deploy MicroTx.

   ```
   kubectl create configmap db-wallet-configmap --from-file=/
   Wallet_database_folder/ -n otmm
   ```

   Where,

   - *db-wallet-configmap* is the name of the configuration map that you want to create. Note down this name as you will need to provide this name in the `values.yaml` file while deploying MicroTx.
   - *Wallet_database_folder* is the folder where you have extracted the contents of the zipped wallet file.
   - *otmm* is the namespace where you want to deploy MicroTx.

   Replace these values with values that are specific to your environment.
4. Perform the following steps to create the connection string for the data store of MicroTx Distributed Transactions.
   a. Create the connection string to the data store in Oracle Database.

      If you are using a non-autonomous Oracle Database (a database that does not use a credential wallet), use the following format to enter the connection string:

      ```
      <publicIP>:<portNumber>/<database unique name>.<host domain name>
      ```

      For example, `123.213.85.123:1521/CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com`.
   b. Append `&wallet_location=/app/Wallet` to the connection string that you have created in the previous step. For example:

      ```
      tcps://adb.us-ashburn-1.oraclecloud.com:1522/
      bfeddi_adw1_medium.adb.oraclecloud.com?
      retry_count=20&retry_delay=3&wallet_location=/app/Wallet
      ```

      Where, `/app/Wallet` is the location where you have downloaded the wallet file.

5. Perform the following steps to create the database connection string for the data store of MicroTx Workflows.
   a. Create the connection string to the data store in Oracle Database.

      If you are using a non-autonomous Oracle Database (a database that does not use a credential wallet), use the following format to enter the connection string:

      ```
      jdbc:oracle:thin:@//<publicIP>:<portNumber>/<database unique
      name>.<host domain name>
      ```

      The following example is a sample connection string to a database instance without using wallet.

      ```
      jdbc:oracle:thin:@//oracle-db-oracle-db23c-free.oracledb-
      free.svc.cluster.local:1521/FREEPDB1
      ```

      The following example is a sample connection string to an ATP database instance without using wallet.

      ```
      jdbc:oracle:thin:@tcps://adb.us-ashburn-1.oraclecloud.com:1522/
      microtx26ai_tpurgent.adb.oraclecloud.com?
      retry_count=20&retry_delay=3&connection_timeout=30000&tcp_keepalive=true
      ```

   b. Append `&wallet_location=/app/oracle/Wallet` to the connection string that you have created in the previous step. For example:

      ```
      jdbc:oracle:thin:@tcps://adb.us-ashburn-1.oraclecloud.com:1522/
      microtx26ai_tpurgent.adb.oraclecloud.com?
      retry_count=20&retry_delay=3&wallet_location=/app/oracle/wallet
      ```

6. Note down the connection strings as you'll have to provide this value later in the Helm chart, `values.yaml` file.

Next, based on the environment in which you want to install MicroTx Distributed Transactions, create a Docker secret or Kubernetes secret to provide the Oracle Database login details.

3.6.4 Create Tables in Oracle Database

Create the tables and schema required by MicroTx Distributed Transactions in Oracle Database using a SQL script only if the Oracle Database user does not have the required privileges, such as `ALTER` and `DELETE`. MicroTx Distributed Transactions creates tables automatically if the user has `ALTER` and `DELETE` privileges.

Before you begin, ensure that the Oracle Database is running and SQL*Plus or any other Oracle-compatible SQL tool is available to run SQL script.

To create tables and schema required by Transaction Manager for Microservices - Distributed Transactions in Oracle Database:

1. Run the `Setup_Microtx.sql` file, which is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/coordinator/tmm/schema_dba` folder.
2. Create an entry in the table for the latest release. Run the SQL files located in the `schema_dba` folder in a chronological order, starting from `24.2.1_MicroTx.sql`.

3.6.5 Create a Kubernetes Secret for Oracle Database Credentials

MicroTx Distributed Transactions and MicroTx Workflows support using Oracle Database as a persistent store to keep track of logs, metadata, and other operation details.

You must provide the Oracle Database credentials secret in the `values.yaml` file. MicroTx uses the credentials secret to establish a connection to the database after the service is installed.

If you are using an Autonomous Database instance, ensure that you have downloaded the wallet and created a configuration map before you begin with the following steps. See Get Autonomous Database Client Credentials.

Create a Kubernetes secret for MicroTx Workflows

To create a Kubernetes secret for MicroTx Workflows with the Oracle Database login details:

1. Launch a terminal and enter the following commands to base64 encode the client ID and client secret.

   ```
   echo -n "database_username" | base64 -w 0
   echo -n "database_password" | base64 -w 0
   ```

   Replace *database_username* and *database_password* with the values in your environment.

   > **Note**
   >
   > For Linux, add `-w 0` to the command to remove line breaks.

   The base64 encoded value of the database username and password is returned. Note down these values as you will need it later.

2. Paste the following code in any text editor.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: oracledb-credentials-workflow
   type: Opaque
   data:
     username: base64_encoded_database_username
     password: base64_encoded_database_password
   ```

   Where,

   - *oracledb-credentials-workflow* is the name of the Kubernetes secret that you want to create. Note down this name as you will have to provide it later in the `values.yaml` file.
   - *base64_encoded_database_username* and *base64_encoded_database_password* are the base64 encoded values of database username and password that you have generated in the previous step. Replace these with values specific to your environment.

3. Save the file as a YAML file. For example, `oracledb-secret.yaml`.

4. Run the following command to create a Kubernetes secret in the namespace where you want to install MicroTx.

   Command syntax

   ```
   kubectl apply -f <filename> -n <namespace>
   ```

   The following sample command creates a Kubernetes secret with the name `oracledb-credentials-workflow` in the `otmm` namespace with the details that you have provided in the `oracledb-secret.yaml` file.

   ```
   kubectl apply -f oracledb-secret.yaml -n otmm
   ```

5. Note down the name of the Kubernetes secret, `oracledb-credentials-workflow`, that you have created. You will need to provide this name in the `values.yaml` file while deploying MicroTx Workflows.

Create a Kubernetes secret for MicroTx Distributed Transactions

To create a Kubernetes secret for MicroTx Distributed Transactions with the Oracle Database login details:

1. Create a Kubernetes secret with the Oracle Database login details. Ensure that you create the Kubernetes secret in the namespace where you want to deploy MicroTx Distributed Transactions.

   The following command creates a Kubernetes secret with the name `db-secret` in the `otmm` namespace with the password of user `acme`. When you run this command in your environment, replace these values with values specific to your environment.

   ```
   kubectl create secret generic db-secret \
       --from-literal=secret='{"password":"*****", "username":"acme"}' -n otmm
   ```

2. Note down the name of the Kubernetes secret that you have created. You will need to provide this name in the `values.yaml` file while deploying MicroTx Distributed Transactions.

Update the `values.yaml` with the name of the Kubernetes secret that you have created to store the Oracle Database credentials and the connection string. Additionally, provide the name of the configuration map if you are using an Autonomous Database instance.

3.7 Set Up etcd as Data Store for MicroTx Distributed Transactions

Create a data store for MicroTx Distributed Transactions to store the transaction logs. You can use either etcd or Oracle Database as the data store.

Skip this section if you want to:

- Set up Oracle Database as data store. See Set Up Oracle Database as Data Store.
- Install only Transaction Manager for Microservices - Workflows (MicroTx Workflows), which does not support etcd as data store.

Before installing MicroTx Distributed Transactions, you must install and configure a data store. Ensure that you set up the required networking rules to allow communication between the transaction coordinator and the data store.

Ensure that you have the required permissions to create tables in the database. When you install MicroTx Distributed Transactions, the service creates the required tables in the database. So MicroTx Distributed Transactions requires certain details about the database.

Topics:

- Generate RSA Certificates for etcd
  You must provide etcd credentials and etcd endpoints in the `YAML` file for the transaction coordinator. MicroTx Distributed Transactions uses this information to establish a connection to the database after the service is installed.
- Create a Kubernetes Secret for etcd
  You must provide etcd credentials and etcd endpoints in the `values.yaml` file. MicroTx Distributed Transactions uses this information to establish a connection to etcd after the service is installed.

3.7.1 Generate RSA Certificates for etcd

You must provide etcd credentials and etcd endpoints in the `YAML` file for the transaction coordinator. MicroTx Distributed Transactions uses this information to establish a connection to the database after the service is installed.

Skip this step if you are not using etcd as the transaction store.

Before you begin, complete the following tasks:

- Install CFSSL tool. See https://github.com/cloudflare/cfssl. This topic provides sample commands to create certificates using the CFSSL tool. You can use this tool or any other tool of your choice to generate certificates.
- Install and configure the etcd database. For information to create an etcd data store, see https://etcd.io/docs/.
- Enable TLS on etcd for additional security and provide the certificate details in the `YAML` file for the transaction coordinator.

To create certificates and identify the etcd endpoints:

1. Create a directory.

   The following sample code creates a directory named, `cfssl`.

   ```
   mkdir cfssl
   cd cfssl
   ```

   Note the path of this directory as you will create all the certificates inside it.
2. Run the following command to identify the external IP address of the etcd database server.

   Run the following command only if you want to install MicroTx Distributed Transactions in a Kubernetes cluster.

   ```
   kubectl get svc
   ```

   Sample output

   ```
   NAME          TYPE           CLUSTER-IP      EXTERNAL-IP
   PORT(S)             AGE

   etcd          ClusterIP      None            <none>           4002/
   ```

```
TCP,4003/TCP   5h8m

etcd-client   LoadBalancer   192.0.2.83    198.51.100.1
4002:32135/TCP      5h8m
```

3. Note down the external IP address.

   You will provide this value to generate the server certificate and as the etcd endpoints in the `YAML` file for the transaction coordinator.

4. Run the following command to initialize certificate authority.

   ```
   echo '{"CN":"CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert -
   initca - | cfssljson -bare ca -
   ```

   This command creates three files in the current working directory: `ca-key.pem`, `ca.csr`, and `ca.pem` files.

5. Run the following command to configure the certificate authority options.

   Sample command

   ```
   echo '{"signing":{"default":{"expiry":"43800h","usages":["signing","key
   encipherment","server auth","client auth"]}}}' > ca-config.json
   ```

   Where, the output is written to the `ca-config.json` file.

   You can modify values for `expiry` and `usages`. For more information about these attributes, refer to the CFSSL documentation.

6. Generate the server certificate.

   a. Run the following command to assign the IP address of the etcd database server to the variable `ADDRESS`. When you run this command in your environment, replace the sample value with a value specific to your environment.

      ```
      export ADDRESS=192.0.2.82
      ```

   b. Run the following command to assign the name of the etcd database server to the variable `NAME`. This is the server Common Name (CN) that is required to generate the server certificate. When you run this command in your environment, replace the sample value with a value specific to your environment.

      ```
      export NAME=server
      ```

   c. Run the following command to generate the server certificate.

      ```
      echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' |
      cfssl gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -
      hostname="$ADDRESS" - | cfssljson -bare $NAME
      ```

   This command creates three files in the current working directory: `server-key.pem`, `server.csr`, and `server.pem` files.

7. Generate the client certificate. While generating the client certificate, you don't need to specify an IP address for the client certificate host.

a. Run the following command to assign a name to the variable `NAME`. This is the server Common Name (CN) that is required to generate the client certificate. You can provide any value to identify the client certificate.

```
export NAME=client
```

b. Run the following command to generate the client certificate.

```
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' |
cfssl gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -
hostname="$ADDRESS" - | cfssljson -bare $NAME
```

This command creates three files in the current working directory: `client-key.pem`, `client.csr`, and `client.pem` files.

8. Run the following command to protect the client certificate with a password.

```
openssl rsa -passout pass:<your_password> -aes256 -in client-key.pem -out
client-ekey.pem
```

Replace, *<your_password>* with a password for your client private key file. Remember the password that you provide as you'll have to provide it in the next step.

The `client-ekey.pem` file is created in the current working directory. You will need to provide the contents of the `client-ekey.pem` file and password in the next step.

9. In any text editor, create a JSON file which contains the contents of the `client-ekey.pem`, `client.pem`, and the password that you have used to protect the client certificate.

The `client.pem` file contains the client certificate and the `client-ekey.pem` file contains the key.

a. Copy the contents of the `client.pem`, the client public key file, as the value of the `cert` field.

b. Copy the contents of the `client-ekey.pem`, the client private key file, as the value of the `key` field.

c. Enter the password for the client private key file that you have provided in the previous step as the value of the `keyPassword` field.

d. Replace all the new lines with the newline character `\n`.

e. Create a JSON file with the edited values. The following code shows a sample JSON file. The sample values have been truncated with ellipses (...) for readability.

```
{
"cert":"-----BEGIN CERTIFICATE-----\nMIIDOjCC...\nBQAwD..jHPs=\n-----END
CERTIFICATE-----",
"key":"-----BEGIN RSA PRIVATE KEY-----\nProc-Type: 4,ENCRYPTED\nDEK-Info:
AES-256-CBC,1870...\n\nNb...\n-----END RSA PRIVATE KEY-----",
"keyPassword":"<your_password>"
}
```

10. Validate and then save the JSON file. Remember the name of the JSON file as you have to provide the name of the file and its location. Let's consider that you save the JSON file as `etcdecred.json`.

3.7.2 Create a Kubernetes Secret for etcd

You must provide etcd credentials and etcd endpoints in the `values.yaml` file. MicroTx Distributed Transactions uses this information to establish a connection to etcd after the service is installed.

Before you begin, generate RSA certificates for etcd and create a JSON file with the contents of the generated certificates. See Generate RSA Certificates for etcd.

If you plan to deploy etcd and MicroTx Distributed Transactions within the same Kubernetes cluster, then it is optional for you to configure etcd with TLS. When etcd is configured with TLS, you must provide the certificate details in the `values.yaml` file for the transaction coordinator.

To create Kubernetes secret and Kubernetes configuration map:

1. Create a Kubernetes secret with the content available in the JSON file that you have created. Ensure that you create the Kubernetes secret in the namespace where you want to deploy MicroTx Distributed Transactions.

   ```
   kubectl create secret generic etcd-cert-secret \
       --from-file=location of etcdecred.json -n otmm
   ```

   Where,

   - *`etcd-cert-secret`* is the name of the Kubernetes secret that you want to create. Note down this name as you will need to provide this name in the `YAML` file to install MicroTx Distributed Transactions.
   - *`location of etcdecred.json`* is the location of the JSON file that you have created in the previous step.
   - *`otmm`* is the namespace where you want to deploy MicroTx Distributed Transactions.

2. Create a configuration map for the `ca.pem` file, which you had created previously while initializing the certificate authority. Ensure that you create the configuration map in the namespace where you want to deploy MicroTx Distributed Transactions.

   ```
   kubectl create configmap etcd-ca-cert-map --from-file=location of ca.pem -
   n otmm
   ```

   Where,

   - *`etcd-ca-cert-map`* is the name of the configuration map that you want to create. Note down this name as you will have provide this name in the `values.yaml` file for MicroTx Distributed Transactions.
   - *`location of ca.pem`* is the location of the `ca.pem` file.
   - *`MicroTx`* is the namespace where you want to deploy MicroTx Distributed Transactions.

You will need to provide the etcd endpoints, certificate, Kubernetes secret, and Kubernetes configuration map that you have created in the `values.yaml` file. The following code snippet provides sample value which are based on the values used in the commands in this topic.

```
storage:
    type: etcd
    etcd:
```

```
endpoints: "https://198.51.100.1:4002"
skipHostNameVerification: "false"
credentialSecret:
  secretName: "etcd-cert-secret"
  secretFileName: "etcdecred.json"
cacertConfigMap:
  configMapName: "etcd-ca-cert-map"
  configMapFileName: "ca.pem"
```

If you do not provide the correct IP address for the `endpoints` field, then host verification fails when you install MicroTx Distributed Transactions. To bypass the host verification in development environments, you can set `skipHostNameVerification` to `true` in the `values.yaml` file of MicroTx Distributed Transactions.

> ⚠ **Caution**
>
> You must set the `skipHostNameVerification` field to `false` in production environments.

4

Install on a Kubernetes Cluster

Install Transaction Manager for Microservices (MicroTx) on a Kubernetes cluster.

Before you begin, ensure that you have completed the prerequisites. See Prepare.

Topics

- Push Images to a Remote Docker Repository
  The installation bundle that you have downloaded to your local system contains Docker images of MicroTx Workflows, MicroTx Distributed Transactions coordinator, and MicroTx console.
- Configure the values.yaml File
  The installation bundle contains `values.yaml` file, the manifest file of the application, which contains the deployment configuration details for MicroTx.
- Install MicroTx
  Use Helm charts to install MicroTx on a Kubernetes cluster.
- Access MicroTx
  To access MicroTx, specify the port number, host name, and protocol that you want to use to access. Oracle recommends that you use HTTP protocol only in test or development environments. In production environments, you must use HTTPS protocol.
- Check the Server Health
  After installing MicroTx, validate that the installation was completed successfully.
- Find IP Address of Istio Ingress Gateway
  Before you start a transaction, you must note down the external IP address of the Istio ingress gateway.

4.1 Push Images to a Remote Docker Repository

The installation bundle that you have downloaded to your local system contains Docker images of MicroTx Workflows, MicroTx Distributed Transactions coordinator, and MicroTx console.

Load these image to your local repository, and then push the images to a remote Docker repository. Kubernetes pulls these images from the remote repository to install the services.

If you are using Oracle Cloud Infrastructure Registry, see Push an Image to Oracle Cloud Infrastructure Registry. If you are using other Kubernetes platforms, use the instructions provided in this section.

Before you begin, complete the following tasks:

- Identify a remote private repository to which you want to upload the container image. You can create a new remote Docker repository or use an existing one. Use a private repository to limit access. When you use a remote Docker repository, you have to push images to the remote Docker repository only once, while you can pull an image multiple times onto any Kubernetes cluster that you create.
- Create a Kubernetes secret to access the remote Docker repository. See Create a Kubernetes Secret to Access Docker Registry.

Perform the following steps to push the Docker images of MicroTx Distributed Transactions transaction coordinator, MicroTx console, and MicroTx Workflows to a remote Docker repository:

1. Provide credentials to log in to the remote private repository to which you want to push the image.

   ```
   docker login <repo>
   ```

   Provide the login credentials based on the Kubernetes platform that you are using.

2. Load the images to the local Docker repository.

   a. Load the MicroTx Workflows image to the local Docker repository. This image is located in the *installation_directory*/MicroTx-*RELEASE*/workflow-server/container-images folder. Select an image based on your requirement. The following code loads the AMD image.

      ```
      cd installation_directory/MicroTx-RELEASE/workflow-server/container-images
      docker load < workflow-server-amd.tar.gz
      ```

      The following message is displayed when the image is loaded.

      ```
      Loaded image: workflow-server-amd
      ```

   b. Load the MicroTx Distributed Transactions transaction coordinator image to the local Docker repository. This image is located in the *installation_directory*/MicroTx-*RELEASE*/coordinator/container-images folder. Select an image based on your requirement. The following code loads the AMD image.

      ```
      cd installation_directory/MicroTx-RELEASE/coordinator/container-images
      docker load < transaction-coordinator-amd.tar.gz
      ```

      The following message is displayed when the image is loaded.

      ```
      Loaded image: transaction-coordinator-amd
      ```

   c. Load the MicroTx console image to the local Docker repository. The console image is located in the *installation_directory*/MicroTx-*RELEASE*/console/container-images folder.

      ```
      cd installation_directory/MicroTx-RELEASE/console/container-images
      docker load < console-amd.tar.gz
      ```

      The following message is displayed when the image is loaded.

      ```
      Loaded image: console-amd
      ```

3. Use the following commands to specify a unique tag for the images that you want to push to the remote Docker repository.

   Syntax

   ```
   docker tag local_image_tag remote_image_tag
   ```

   Where,

- *local_image_tag* is the tag with which the image is identified in your local repository.
- *remote_image_tag* is the tag with which you want to identify the image in the remote Docker repository.

Sample Commands

```
docker tag workflow-server-amd <region-key>.ocir.io/otmmrepo/workflow-
server-amd
docker tag transaction-coordinator-amd <region-key>.ocir.io/otmmrepo/
transaction-coordinator-amd
docker tag console-amd <region-key>.ocir.io/otmmrepo/console-amd
```

Where, `<region-key>.ocir.io/otmmrepo` is the remote Docker registry to which you want to push the image files. Provide the registry details based on your environment.

4. Push the Docker images from your local repository to the remote Docker repository.

   Syntax

   `docker push remote_image_tag`

   Sample Commands

   ```
   docker push <region-key>.ocir.io/otmmrepo/workflow-server-amd
   docker push <region-key>.ocir.io/otmmrepo/transaction-coordinator-amd
   docker push <region-key>.ocir.io/otmmrepo/console-amd
   ```

Note down the tag of the Docker image in the remote Docker repository. You'll need to enter this tag while pulling the image from the remote Docker repository.

For more information about container registry in Oracle OCI, see Container Registry Concepts in *Oracle Cloud Infrastructure Documentation*.

4.2 Configure the values.yaml File

The installation bundle contains `values.yaml` file, the manifest file of the application, which contains the deployment configuration details for MicroTx.

Replace the sample values in the `values.yaml` file to provide the environment details, image details, and configuration details to deploy MicroTx.

While deploying MicroTx to a Kubernetes cluster, Helm pulls the MicroTx images from the remote Docker registry. In the `values.yaml` file, specify the image to pull and the credentials to use while pulling the images.

To provide configuration details for MicroTx:

1. Open the `values.yaml` file in any code editor. This file is located in the *`installation_directory`*`\MicroTx-`*`RELEASE`*`\deployment-descriptors\helmcharts\microtx` folder. This file contains sample values.
2. Replace the sample values with values that are specific to your environment.

   The tables in this section describe the properties for the environment, storage, authorization, authentication, and other configuration details that are required to install MicroTx.
3. Save your changes.

Topics

- Namespace Configuration
- Istio Details
  If you are using an Istio service mesh, provide details about the Istio ingress gateway that you have set up in the `values.yaml` file.
- Common Configuration
  In the `values.yaml` file, under `commonConfiguration` provide information about the environment details in which you want to install MicroTx Distributed Transactions, and then configure security.
- MicroTx Distributed Transactions Coordinator Configuration
  In the `values.yaml` file, under `coordinator`, provide configuration information to install the MicroTx Distributed Transactions coordinator to manage your transactions.
- Security Configuration Properties
  Under `security`, enter property values for the JSON Web Token (JWT) which MicroTx uses for authentication.
- Console Configuration Properties
  Under `tmmConsoleConfiguration`, specify the properties for the MicroTx console for operations and workflow management.
- MicroTx Workflows Server Configuration
  In the `values.yaml` file, under `workflow.server`, provide information about the environment details in which you want to install MicroTx Workflows.

4.2.1 Namespace Configuration

Property	Description
`enableIstio`	Determines whether the MicroTx deployment uses the Istio service mesh or the NGINX Ingress load balancer. When set to `true`, Istio service mesh features and the Istio load balancer are enabled. When set to `false`, Istio is disabled, and traffic is routed through the NGINX Ingress load balancer instead.
`namespaceOverride`	Enter a value to override the default namespace for MicroTx deployment. If you do not provide a value, the default namespace is `otmm` when you deploy MicroTx in an Istio service mesh and `otmm-k8s` if you don't deploy in Istio service mesh.

4.2.2 Istio Details

If you are using an Istio service mesh, provide details about the Istio ingress gateway that you have set up in the `values.yaml` file.

Skip providing values for the properties mentioned in the following table if you aren't using an Istio service mesh.
Before you begin, ensure that you have set up and configured Istio. See Install the Required Software for Kubernetes. You must also set `enableIstio` to `true`.

Property	Description
`istioSystemNameSpace`	The namespace in which you have installed Istio. The default namespace is `istio-system`. If you have installed Istio in another namespace, run the following command to find all the namespaces in the cluster. `kubectl get ns`
`istioIngressGateway.name`	Enter the name of the Istio ingress gateway that you have created. For example, `ingressgateway`. To find the name of the Istio ingress gateway, run the following command and from the response note down the value for the `istio` label. `kubectl describe service/istio-ingressgateway -n istio-system`
`istioIngressGateway.tlsEnabled`	Set this to `true` to enable access to the service using the HTTPS protocol. When you set this to `true`, you must provide details of the Kubernetes secret that contains the SSL key and certificate to access Istio using HTTPS. **Note** You must set this value to `true` in production environments.
`istioIngressGateway.credentialName`	You must specify a value for this property if `tlsEnabled` is set to `true`. Enter the name of the Kubernetes secret that you have created to enable access to Istio using the HTTPS protocol. See Create a Kubernetes Secret with SSL Details for Istio. For example, `tls-credential`.
`istioIngressGateway.hosts`	Enter the external IP Address of Istio ingress gateway or the name of the hosts. If you are using a load balancer or have multiple hosts, enter a comma-separated list of host names or IP addresses. See Find IP Address of Istio Ingress Gateway.
`istioIngressGateway.createOtmmGatewayAlias`	Set this to true to create a gateway named `otmm-gateway` for backward compatibility.
`istioIngressGateway.port`	Specify the istio ingress port number. To retrieve the istio ingress gateway port number, run the following command. `kubectl get svc -n istio-system` Ensure that the required networking rules are set up to ensure that traffic is permitted to and from this port to MicroTx.
`istioIngressGateway.protocol`	Specify the protocol you want to use to access Istio. Possible values are `http` and `https`, when `istioIngressGateway.tlsEnabled` is set to `true`. See https://istio.io/latest/docs/ops/configuration/traffic-management/protocol-selection/.

Sample Property Values for Istio

The following code snippet provides sample property values for Istio.

```
istioSystemNameSpace: istio-system
istioIngressGateway:
  name: ingressgateway
  tlsEnabled: "true"
  credentialName: tls-credential
  hosts: 192.0.2.1
  port: 443
  protocol: https
```

4.2.3 Common Configuration

In the `values.yaml` file, under `commonConfiguration` provide information about the environment details in which you want to install MicroTx Distributed Transactions, and then configure security.

Property	Description
`externalURL.protocol`, `externalURL.host`, and `externalURL.port`	Enter the details required to create the external URL to access MicroTx from outside the Kubernetes cluster where you have deployed the service. Enter the protocol, host, and port number to access the Istio ingress gateway. See Access MicroTx.

4.2.4 MicroTx Distributed Transactions Coordinator Configuration

In the `values.yaml` file, under `coordinator`, provide configuration information to install the MicroTx Distributed Transactions coordinator to manage your transactions.

If you do not want to install MicroTx Distributed Transactions coordinator, set the `deploy` property as `false` and skip this section.

Property	Description
`deploy`	Enter `true` to deploy MicroTx Distributed Transactions coordinator. Enter `false` when you do not want to deploy MicroTx Distributed Transactions coordinator.
`tmmReplicaCount`	Specify the number of MicroTx Distributed Transactions pod replicas that you want to create. Oracle recommends a minimum of 3 replicas for production environments. Development or test environments may be able to use fewer replicas.
`resources`	Tune requests or limits according to workload throughput and cluster quotas. For more information about the limits enforced by Kubernetes, see Requests and limits in *Kubernetes documentation*.

Topics

- Image Properties
  Under `tmmImage`, provide information about the MicroTx Distributed Transactions Docker image. It is mandatory to provide values for these properties.
- MicroTx Distributed Transactions Coordinator Properties
  Under `tmmConfiguration`, provide information to configure MicroTx Distributed Transactions coordinator.

- Retry Setting Properties
  Enter details for the transaction coordinator to retry sending a request.
- Caching Properties
  You can enable caching to improve performance. It optimizes the read and write operations for the transaction logs that are stored in etcd or Oracle Database.
- Data Store Properties
  MicroTx Distributed Transactions uses a data store to persist transaction records, service logs, and metadata.
- Encryption Key Properties
  Under `coordinator.tmmConfiguration.encryption`, specify the encryption key that MicroTx Distributed Transactions uses to encrypt the access and refresh tokens. You must provide values for these properties if you have enabled `authTokenPropagationEnabled` under `commonConfiguration.security.authorization`.
- Transaction Token Properties
  Under `transactionToken`, specify the key pair that you want to use for transaction token.

4.2.4.1 Image Properties

Under `tmmImage`, provide information about the MicroTx Distributed Transactions Docker image. It is mandatory to provide values for these properties.

Property	Description
`image`	Enter the tag of the MicroTx Distributed Transactions image that you have pushed to the remote repository. For example, `oracle-tmm:`*`RELEASE`*. See Push Images to a Remote Docker Repository.
`imagePullPolicy`	Enter `Always` to ensure that the image is pulled during the installation. See Image pull policy in *Kubernetes documentation*.
`imagePullSecret`	Specify the name of the Kubernetes secret that you have created. This secret is used to pull the Docker images from the remote repository. For example, `regcred`. See Create a Kubernetes Secret to Access Docker Registry.

The following code snippet provides sample values for `tmmImage`.

```
tmmImage:
  image: oracle-tmm:RELEASE
  imagePullPolicy: Always
  imagePullSecret: regcred
```

4.2.4.2 MicroTx Distributed Transactions Coordinator Properties

Under `tmmConfiguration`, provide information to configure MicroTx Distributed Transactions coordinator.

Property	Description
`tmmAppName`	Enter the name of the MicroTx Distributed Transactions application that you want to create. When you install MicroTx Distributed Transactions, Helm creates the MicroTx Distributed Transactions service with the name that you specify. Note down this name as you will need to provide it later. For example, `tmm-app`.

Property	Description
`tmmId`	Enter a value to uniquely identify each instance of MicroTx Distributed Transactions coordinator that you install. The unique identifier must have 5-characters and can contain only alphanumeric characters (a-z, A-Z, and 0-9). For example, `TMM01`. Use this ID to identify MicroTx Distributed Transactions coordinator when there are multiple installations. You cannot use this ID to differentiate between replicas of a single instance of MicroTx Distributed Transactions coordinator installation as all the replicas have the same ID. You can't change this value after installing MicroTx Distributed Transactions coordinator.
`port`	Enter the port over which you want to internally access MicroTx Distributed Transactions coordinator within the Kubernetes cluster where you will install this service. Create the required networking rules to permit inbound and outbound traffic on this port. Note down this number as you will need to provide it later. For example, `9000`.
`xaCoordinator`, `lraCoordinator`, or `tccCoordinator`	Set `enabled: "true"` for the transaction protocols that your microservices use. MicroTx Distributed Transactions coordinator supports three distribution transaction protocols: XA, Saga, and TCC. If you want to nest an XA transaction within an Saga transaction, set `enabled: "true"` for both `xaCoordinator` and `lraCoordinator`.
`narayanaLraCompatibilityMode`	Only for the Saga transaction protocol. Set `enabled` to `true` only when you want to use Saga participant applications that were implemented to work with the Narayana LRA Coordinator and now would participate in Saga transactions using MicroTx Distributed Transactions. Enable this mode to ensure that the Saga APIs return the same response data that Narayana LRA Coordinator APIs return.
`sessionAffinity`	Set this to `true` to enable session affinity for transaction participant services and the transaction initiator service. When you enable caching, you must also enable session affinity for the coordinator.
`metrics.enabled`	Set this to `true` so that Prometheus can scrape the metrics logs of the MicroTx Distributed Transactions coordinator. By default, this is set this to `true`.
`logging.level`	Enter one of the following types to specify the log level for MicroTx Distributed Transactions: • `info`: Logs events that occur during the normal operation of the MicroTx Distributed Transactions. This setting logs the least amount of information. This is the default setting. • `warning`: Logs events that may cause potentially harmful situations. • `error`: Logs events to indicate that there is an issue that requires troubleshooting. • `debug`: Logs all the events. Use this setting when you want to debug an issue.
`logging.httpTraceEnabled`	Set this to `true` to log all the HTTP request and responses in MicroTx Distributed Transactions when you want to debug. When you set this value as `true`, you must also set the `level` as `debug`.

4.2.4.3 Retry Setting Properties

Enter details for the transaction coordinator to retry sending a request.

Property	Description
`maxRetryCount`	The maximum number of times that the transaction coordinator retries sending the same request again in case of any failures. For example, 10.
`minRetryInterval`	The minimum interval, in milliseconds, after which the transaction coordinator retries sending the same request again in case of any failures. The default value is 1000 ms.
`maxRetryInterval`	The maximum retry interval, in milliseconds, before which the transaction coordinator retries sending the same request again in case of any failures. For example, 10000.
`skipVerifyInsecureTLS`	Oracle recommends that you set this value to `false` and set up a valid certificate signed by trusted authorities for secure access. When you set this value to `false`, the transaction coordinator accesses the participant applications over the HTTPS protocol with a valid certificate signed by trusted authorities. The default value is `false`. If you set this value to `true`, the transaction coordinator can access the participant application's callback URL, without a valid SSL certificate, in an insecure manner. **⚠ Caution** Do not set this value to `true` in production environments.
`httpClientTimeoutInSecs`	Specify the maximum amount of time, in seconds, for which the HTTP callback API requests sent by the MicroTx Distributed Transactions coordinator to the participant services remains active. Enter an integer between 0 to 900. The default value is 180 seconds and the maximum value is 900 seconds. If you set this value to 0, then MicroTx Distributed Transactions does not enforce any limit. When the coordinator sends a HTTP callback API request to the participant services, the participant services must respond within the time period that you specify. If the participant service does not respond within the specified time period, the HTTP request sent by the coordinator times out.
`countersUpdateInterval`	Enter a value between 30 to 1800 seconds to specify data collection time interval or the rate at which information about the most recent transactions is updated on the MicroTx Distributed Transactions console. If you enter 300 seconds, information about the most recent transactions is updated every 300 seconds on the console. The default value is 120 seconds. You can also update this value in the MicroTx Distributed Transactions console by changing the value in the **Interval** box.
`notificationSpikeTimeInSecs`	Time, in seconds, for which a notification is visible in Prometheus.

4.2.4.4 Caching Properties

You can enable caching to improve performance. It optimizes the read and write operations for the transaction logs that are stored in etcd or Oracle Database.

Property	Description
`enabled`	Enter `true` to enable local caching for the coordinator. You can set this to `true` only if you use etcd cluster or Oracle Database to store the transaction information. You cannot enable caching if you use internal memory as the transaction store.
`maxCapacity`	Enter a decimal value as the maximum storage capacity of the cache in GB. For example, `1.5`. The default value is `1`. Ensure that your environment has the required memory to support the maximum value that you enter.

The following code snippet provides sample values for the `caching` property in the `values.yaml` file. When you enable caching, you must also enable session affinity for the coordinator.

```
caching:
  enabled: true
  maxCapacity: 1.5
sessionAffinity: true
```

For more information about session affinity, see Enable Session Affinity.

4.2.4.5 Data Store Properties

MicroTx Distributed Transactions uses a data store to persist transaction records, service logs, and metadata.

MicroTx Distributed Transactions coordinator supports the following storage types:

- `db`: Oracle Database 19c and later
- `etcd`: etcd
- `embedded_db`: embedded LevelDB-based local storage
- `memory`: in-memory storage

Oracle Database and etcd are centralized data stores. They provide shared, durable storage that can be accessed by all coordinator replicas. Use these storage types when you need a common external store for transaction state, recovery, and coordination across replicas.

The embedded LevelDB-based local storage type uses an embedded LevelDB-based store on the coordinator instance. When session affinity is enabled, requests for a transaction are routed to the same coordinator replica, allowing the embedded database to act as local persistent storage. This avoids the network hop and contention associated with centralized stores such as Oracle Database or etcd, and can improve transaction processing performance. The embedded database is suitable for production deployments that use session affinity and are designed around local coordinator storage. When you want to use multiple replicas of the transaction coordinator, you must set up an etcd cluster or Oracle Database as the transaction log data store.

The in-memory storage type stores transaction data only in memory. Use it for local development, testing, and samples. If you use internal memory, you can't create multiple replicas of the transaction coordinator. It is not intended for production because data is lost when the coordinator restarts.

Type of Data Store for Transaction Logs and Limit for Record Retention

Under `coordinator.tmmConfiguration.storage`, specify the type of data store that MicroTx Distributed Transactions uses for persistence of transaction state. After specifying the type of data store, you can provide additional details to connect to the external data store.

Only records of transactions that have reached a final state, irrespective of whether it is success or failure, are deleted. When any one of the limits specified in `completedTransactionTTL`, `batchInterval`, or `thresholdFactor` is reached, it triggers the deletion of the completed transaction records.

Completed transaction records are retained for the duration specified by `completedTransactionTTL`. After this TTL expires, records that are in a final state are added to the deletion queue. The coordinator deletes queued records in batches when either the configured `batchInterval` elapses or the number of queued records reaches the threshold calculated from `thresholdFactor`.

Only completed transaction records are deleted. Active or in-progress transaction records are not deleted by this cleanup process.

For etcd storage, completed records are removed automatically using the etcd lease TTL. For other storage types, the coordinator runs the batch deletion loop.

Property	Description
`type`	Enter one of the following values to specify the persistent data that you want MicroTx Distributed Transactions to use to track the transaction information. • `etcd` to use etcd as the data store. You must provide details to connect to the etcd data store in the `storage.etcd` field. • `db` to use Oracle Database as the data store. You must provide details to connect to the Oracle data store in the `storage.db` field. • `memory` to skip entering details to connect to either etcd or Oracle Database and use the internal memory instead. When you use internal memory, all the transaction details are lost every time you restart MicroTx Distributed Transactions. • `embedded_db` for embedded LevelDB-based local storage.
`completedTransactionTTL`	Specify the time to live, in seconds, for a completed transaction record in the transaction data store. After this period expires, the completed transaction record becomes eligible for deletion. The permissible range is 60 to 1200 seconds. If a lower value is configured, it is set to 60. If a higher value is configured, it is set to 1200. The default value is 60 seconds.
`batchInterval`	Specify the interval, in seconds, at which the coordinator checks and deletes queued completed transaction records in batches. If the interval has elapsed and the deletion queue is not empty, the coordinator deletes the queued records from the transaction data store. The default value is 180 seconds.

Property	Description

`thresholdFactor`	Specify an integer multiplier used to calculate the deletion threshold. The threshold is calculated as:
	
	`threshold = batchSize * thresholdFactor`
	
	The current `batchSize` is 1000. For example, if `thresholdFactor` is 2, completed transaction records are deleted when the deletion queue reaches 2000 records.
	The deletion channel capacity is calculated as:
	
	`channelSize = batchSize * thresholdFactor * bufferFactor`
	
	The current `bufferFactor` is 2.
	The default and recommended value is 2. Use the default value, unless you need to retain more queued completed records before triggering deletion. A higher value increases the number of records retained in the deletion queue before threshold-based deletion starts.

Oracle Database as Data Store for Transaction Logs

Under `tmmConfiguration.storage.db`, specify the details to connect to an Oracle Database. Skip this section and do not provide these values if you are connecting to an etcd database or using internal memory.

The following table specifies the properties required to connect to an Oracle Database as the MicroTx Distributed Transactions data store.

Property	Description

`connectionString`	Enter the connection string for the data store in Oracle Database.
	• If you are using a database that uses a credential wallet, such as Oracle Autonomous Transaction Processing, use the following format to enter the connection string:
	`tcps://<database-host>:<database-portNumber>/<service_name>`
	You can find the required details, such as protocol, host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the credential wallet. See Download Client Credentials (Wallets) in *Using Oracle Autonomous Database on Shared Exadata Infrastructure*.
	For example:
	`tcps://adb.us-phoenix-1.oraclecloud.com:1521/sales.adb.oraclecloud.com`
	Additionally, you can pass database timeout parameters, such as `retry_count` and `retry_delay`, by appending it to the connection string. For example,
	`<connection-string>?retry_count=20&retry_delay=3`
	• If you are using a non-autonomous Oracle Database (a database that does not use a credential wallet), use the following format to enter the connection string:
	`<database-host-IPaddress>:<database-portNumber>/<databaseUniqueName>.<hostDomainName>`
	For example:
	`123.213.85.123:1521/CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com`
	Where, `CustDB_iad1vm.sub05031027070.customervcnwith` is the unique name of the database or the service name.
	• If you are using Oracle RAC database, see About Connecting to an Oracle RAC Database Using SCANs.

`credentialSecretName`	Enter the name of the Kubernetes secret that contains the credentials to connect to the Oracle Database. Example, `db-secret`. See Create a Kubernetes Secret for Oracle Database Credentials.
`connectionParams`	Enter a space-separated list of connection parameters. For details about the connection parameters that you can specify, see https://pkg.go.dev/github.com/godror/godror#section-documentation. For example, `poolMinSessions=1 poolMaxSessions=300`

Property	Description
`walletConfigMap.configMapName`	Provide a value for this field only if you connect to a database that uses a credential wallet, such as Autonomous Database. For other databases, you can leave this field empty. Enter the name of the configuration map that you have created for the credential wallet of the Autonomous Database instance. See Get Autonomous Database Client Credentials. For example, `db-wallet-configmap`.

etcd Database as Data Store for Transaction Logs

Under `tmmConfiguration.storage.etcd`, specify the details to connect to an etcd database. Skip this section and do not provide these values if you are connecting to an Oracle Database or are using internal memory.

The following table specifies the properties required to connect to an etcd database as the MicroTx Distributed Transactions data store.

Before you provide this information, ensure that you complete the following tasks and note down the required details.

1. Generate RSA Certificates for etcd
2. Create a Kubernetes Secret for etcd

Property	Description
`endpoints`	Enter the external IP address of the etcd database server. If you have installed the etcd cluster in the Kubernetes cluster where you will install MicroTx Distributed Transactions, then provide the Kubernetes service name and the port of the etcd cluster (nodes) as values. Otherwise, enter a comma-separated list of host names or IP addresses of the etcd cluster nodes along with the ports, such as `198.51.100.1:4002,198.51.100.2:4002,198.51.100.3:4002`.
`skipHostNameVerification`	Set this to `false` to verify the IP address of the etcd database server. If you set this to `true`, then the server host name or IP address is not verified. You can set this field to `true` only for test or development environments. **⚠ Caution** You must set this field to `false` in production environments.
`credentialSecret.secretName`	Enter the path to the Kubernetes secret in the container. The secret contains the client credentials, client key, and the password that you have used to protect the client certificate. For example, `/etc/otmm/etcd-cert-secret`. See Create a Kubernetes Secret for etcd.
`credentialSecret.secretFileName`	Enter the location of the JSON file, that contains client credentials, client key, and the password that you have used to protect the client certificate. For example, `/etc/otmm/etcdecred.json`. See Generate RSA Certificates for etcd.
`cacertConfigMap.configMapName`	Enter the name of the configuration map file, which you had created while initializing the certificate authority. For example, `etcd-ca-cert-map`. See Create a Kubernetes Secret for etcd.

Property	Description
`cacertConfigMap.configMapFileName`	Enter the name of the PEM file that you had created while initializing the certificate authority. For example, `ca.pem`. See Generate RSA Certificates for etcd.

The following code snippet provides sample value which are based on the sample values provided in the Generate RSA Certificates for etcd and Create a Kubernetes Secret for etcd topics.

```
storage:
    type: etcd
    etcd:
      endpoints: "https://198.51.100.1:4002"
      skipHostNameVerification: "false"
      credentialSecret:
        secretName: "etcd-cert-secret"
        secretFileName: "etcdecred.json"
      cacertConfigMap:
        configMapName: "etcd-ca-cert-map"
        configMapFileName: "ca.pem"
```

If you do not provide the correct IP address for the `endpoints` field, then host verification fails when you install MicroTx Distributed Transactions.

4.2.4.6 Encryption Key Properties

Under `coordinator.tmmConfiguration.encryption`, specify the encryption key that MicroTx Distributed Transactions uses to encrypt the access and refresh tokens. You must provide values for these properties if you have enabled `authTokenPropagationEnabled` under `commonConfiguration.security.authorization`.

Property	Description
`encryptionSecretKeyVersion`	Specify the version of the secret key that you want to use for encrypting the transaction tokens from the list of secret keys that you provide.
`encryptionSecretKeys.secretKeys.SecretKeyName`	Specify the name and version of the Kubernetes secrets that contains encryption key as the value. To support the encryption keys rotation, specify multiple encryption keys and their versions.
`encryptionSecretKeys.secretKeys.version`	Enter a natural number as the version of the Kubernetes secret. Ensure that the version is unique for each secret key.

If you create a new Kubernetes secret key, do not delete the entry for the previous secret key immediately. You may delete the old key and the corresponding entry in the `values.yaml` file after a few days because existing transactions may be using the older versions of the key. After a few days, you can update the `values.yaml` file, and then update MicroTx Distributed Transactions.

The following code snippet provides sample values for the `encryption` field in the `values.yaml` file. The sample values in this example are based on the values used in the sample commands in Generate a Kubernetes Secret for an Encryption Key.

```
encryption:
  encryptionSecretKeyVersion: "1"
  encryptionSecretKeys:
    secretKeys:
      - secretKeyName: "encryption-secret-key1"
        version: "1"
      - secretKeyName: "encryption-secret-key2"
        version: "2"
```

4.2.4.7 Transaction Token Properties

Under `transactionToken`, specify the key pair that you want to use for transaction token.

If you set `transactionTokenEnabled` to `true`, it is mandatory to provide values listed in the following table.

Property	Description
`transactionTokenEnabled`	Set this to `true` when you want MicroTx Distributed Transactions to include a signed transaction token, `tmm-tx-token`, in the request header. You don't have to create the `tmm-tx-token` transaction token or pass it in the request header. The MicroTx Distributed Transactions library creates this token based on the private-public key pair that you provide. For more information about creating the key pair, see Create a Key Pair for Transaction Token.
`transactionTokenKeyPairVersion`	Enter the version of the key pair that you want to use for signing and verification of the transaction token. When there are multiple key pairs, you must specify the version of the key pair that you want to use.
`transactionTokenKeyPairs.keyPairs.privateKeyName`	Enter the name of the Kubernetes secret which has the base64-encoded value of the private key.
`transactionTokenKeyPairs.keyPairs.publicKeyName`	Enter the name of the Kubernetes secret which has the base64-encoded value of the public key.
`transactionTokenKeyPairs.keyPairs.privateKeyPasswordName`	Enter the name of the Kubernetes secret which has the value of the pass phrase that you had provided while generating the private key.
`transactionTokenKeyPairs.keyPairs.version`	Enter the version of the private-public key pair that you want to use.

The following code snippet provides sample values for the `transactionToken` field.

```
transactionToken:
  transactionTokenEnabled: "true"
  transactionTokenKeyPairVersion: "1"
  transactionTokenKeyPairs:
    keyPairs:
      - privateKeyName: "TMMPRIVKEY1"
        publicKeyName: "TMMPUBKEY1"
        privateKeyPasswordName: "TMMPRIVKEYPASSWD1"
        version: "1"
```

```
  - privateKeyName: "TMMPRIVKEY2"
    publicKeyName: "TMMPUBKEY2"
    privateKeyPasswordName: "TMMPRIVKEYPASSWD2"
    version: "2"
```

4.2.5 Security Configuration Properties

Under `security`, enter property values for the JSON Web Token (JWT) which MicroTx uses for authentication.

Property	Description
`enabled`	Set this value to `TRUE` to enable authorization checks, such as role based access control (RBAC) for the MicroTx. When enabled, you must provide details of identity provider and all other properties under `security`. **⚠ Caution** You must set this field to `TRUE` in production environments.
`clientSecretName`	Enter the name of the Kubernetes secret that you have created to ensure that users can access the MicroTx console. See Create a Secret with Identity Provider Client Credentials.

- Identity Provider Properties
  Under `identityProvider`, enter property values for the JSON Web Token (JWT) which MicroTx uses for authentication.
- Role Mapping for MicroTx Distributed Transactions Coordinator
  Under `security.identityProvider.roleMappings.coordinator`, map identity provider roles to MicroTx Distributed Transactions roles and permissions.
- Role Mapping for MicroTx Workflows
  Under `commonConfiguration.security.identityProvider.roleMappings.workflow`, map MicroTx Workflows role-based access control (RBAC) roles to corresponding identity provider realm roles.
- Authorization Properties
  MicroTx Distributed Transactions supports authorization across participant services and coordinator by propagating the JWT token in every request.
- Authentication Properties
  Under `authentication`, enable JSON Web Token (JWT) authentication.

4.2.5.1 Identity Provider Properties

Under `identityProvider`, enter property values for the JSON Web Token (JWT) which MicroTx uses for authentication.

When you set `authentication.requestsWithNoJWT` to `DENY`, you must provides values for all the identity provider properties listed in the table below.

However, provide values for the `audience`, `adminUserRoles`, `adminUserRolesPath`, and `clientSecretName` properties to ensure that users can access the MicroTx console. If you don't want to provide access to the MicroTx console, you can skip providing values for these properties.

Property	Description
`serverType`	Enter `idcs` if you are using Oracle IDCS as the identity provider. Otherwise, enter `other`. The default value is `other`.
`scopes`	If the `serverType` is `idcs`, you *must* specify a scope to grant different levels of access. If you are not using Oracle IDCS as the identity provider, do not provide a value for this property. For Oracle IDCS, enter a space-separated list of scopes. The default scope for Oracle IDCS is `openid groups`.
`issuer`	Identifies the JWT token issuer. Enter the URI of the identity server that you have set up. It is the value of the `issuer` field in the Discovery URL. For example, `https://identity.oraclecloud.com`. See Run the Discovery URL.
`jwksUri`	The URL of the identity provider's publicly hosted `jwksUri`, which is used to validate signature of the JWT. The JSON Web Key Set (JWKS) contains the cryptographic keys which are used to verify the incoming JWT tokens. See Run the Discovery URL.
`identityProviderUrl`	Specify the URL of the JWT identity provider. This information is required to create a new access token by using the refresh token. If you do not provide this information, expired access tokens are not auto-refreshed. For example, `http://192.0.2.1:8080/auth/realms/tmmdev` when you use Keycloak as the identity provider. See Run the Discovery URL.
`audience`	Enter the audience of the token. Every JWT is validated to check the audience. You must provide a value for this parameter to access the MicroTx Distributed Transactions console. Note down this value from the JWT access token.

The following code snippet provides sample values for the `authentication` field in the `values.yaml` file. The sample values in this example are based on Run the Discovery URL and Specify the Admin Role in YAML file.

```
identityProvider:
    issuer: "https://identity.oraclecloud.com"
    jwksUri: "https://idcs-a83e4...identity.oraclecloud.com:443/admin/v1/
SigningCert/jwk"
    identityProviderUrl: "https://idcs-a83e4...identity.oraclecloud.com/
oauth2/v1/token"
    clientSecretName: "console-identity-client-secret"
    audience: "account"
    scopes: "idcs"
```

The example tenant base URL, `https://idcs-a83e4...identity.oraclecloud.com`, has been truncated with ellipses (...) for readability. Copy the complete value in your environment.

4.2.5.2 Role Mapping for MicroTx Distributed Transactions Coordinator

Under `security.identityProvider.roleMappings.coordinator`, map identity provider roles to MicroTx Distributed Transactions roles and permissions.

Property	Description
`adminUserRolesPath`	Enter the path to the administrator role in the JWT token. For example, `realm_access, roles`. See Specify the Admin Role in YAML file.
`adminUserRoles`	Enter a comma-separated list of names of the administrator roles that you have configured in the identity provider to grant access to the MicroTx Distributed Transactions administrator APIs. Only the users that are granted this role are permitted to access the console. For example, `admin, consoleadmin`. See Specify the Admin Role in YAML file.

The following code snippet provides sample values for `roleMappings` for MicroTx Distributed Transactions coordinator roles in the `values.yaml` file.

```
roleMappings:
  coordinator:
    adminUserRoles: "admin, consoleadmin"
    adminUserRolesPath: "realm_access, roles"
```

4.2.5.3 Role Mapping for MicroTx Workflows

Under `commonConfiguration.security.identityProvider.roleMappings.workflow`, map MicroTx Workflows role-based access control (RBAC) roles to corresponding identity provider realm roles.

MicroTx Workflows manages a set of internal roles to control access to system capabilities and resources. These roles govern what users and service accounts can see and do within the system, based on the assigned responsibilities. The following table provides details about the internal roles defined in the product under `roles`.

Property	Description
`claimsRolePaths`	Dot-separated path in the JWT token to extract realm roles. For example, `realm_access.roles` for Keycloak.
`roles.superAdmin`	Super user with all control capabilities across all logical groups.
`roles.admin`	Full access to all resources and administrative functions within the logical group.
`roles.user`	Capability to view or initiate the workflow and task definitions.
`roles.readOnlyUser`	View-only access to workflow or task definitions and instances.
`roles.metadataManager`	Access to workflow or task definition creation, update, deletion, and workflow initiation.
`roles.worker`	Service credential for polling and updating tasks.

Sample Code

The following code snippet provides sample values for the `roleMappings` field in the `values.yaml` file.

```
roleMappings:
  workflow:
    claimsRolePaths: "realm_access.roles"
```

```
roles:
  superAdmin: "microtx-conductor-super-admin"
  admin: "microtx-conductor-admin"
  user: "microtx-conductor-user"
  readOnlyUser: "microtx-conductor-read-only-user"
  metadataManager: "microtx-conductor-metadata-manager"
  worker: "microtx-conductor-worker"
```

4.2.5.4 Authorization Properties

MicroTx Distributed Transactions supports authorization across participant services and coordinator by propagating the JWT token in every request.

Under `tmmConfiguration.authorization`, use the `authTokenPropagationEnabled` field to control this function. Configure your identity providers to auto-refresh the expired access tokens at the coordinator.

Property	Description
`enabled`	Set this to `true` to enforce RBAC rules on the MicroTx Distributed Transactions coordinator API access. The first step in authorization is to enforce RBAC controls at the MicroTx Distributed Transactions coordinator. **⚠ Caution** You must set this field to `true` in production environments.
`authTokenPropagationEnabled`	Set this to `true` to enable token propagation to ensure secure communication between the participant services and MicroTx Distributed Transactions. MicroTx Distributed Transactions performs the following actions: • The MicroTx Distributed Transactions libraries propagate the authorization headers in every outgoing call to the MicroTx Distributed Transactions coordinator. The coordinator uses the propagated access token from incoming requests for the authorization checks. • The MicroTx Distributed Transactions coordinator encrypts and stores the access token and refresh token details in the transaction store. These tokens are appended in the callback API calls from the MicroTx Distributed Transactions coordinator to the MicroTx Distributed Transactions library that is present in the participant application. When you enable token propagation, you must provide the details for the encryption keys under the `encryption` property in the `values.yaml` file.

4.2.5.5 Authentication Properties

Under `authentication`, enable JSON Web Token (JWT) authentication.

Property	Description
`requestsWithNoJWT`	Enter `DENY` to enable authorization checks at coordinator, such as role based access control (RBAC). This ensures that each request has a JWT token. Enter `ALLOW` to bypass JWT authentication. This permits requests that do not have JWT tokens. **⚠ Caution** You must set this field to `DENY` in production environments. When set to `DENY`, you must provide values for the identity provider properties. See Identity Provider Properties.

The following code snippet provides sample values for the `authentication` field in the `values.yaml` file.

```
authentication:
  requestsWithNoJWT: DENY
```

4.2.6 Console Configuration Properties

Under `tmmConsoleConfiguration`, specify the properties for the MicroTx console for operations and workflow management.

Ensure that you have completed the following tasks before you provide the property values.

- Push the MicroTx console image to the remote Docker repository. See Push Images to a Remote Docker Repository.
- Set up an identity provider. See Set Up Oracle Identity Providers.
- Set permissions for administrators. Administrators can access the MicroTx Distributed Transactions console to view and manage transactions by all the users. Non-admin users can only view and manage transactions that they have initiated. They can not view or manage transactions of other users. See Specify the Admin Role in YAML file and Create a Secret with Identity Provider Client Credentials.
- Create a Kubernetes secret to store the cookie encryption password. See Create a Secret with Cookie Encryption Password for Kubernetes

It is mandatory to provide values for all the properties specified in the following table.

Property	Description
`tmmConsoleAppName`	Name of the MicroTx console application that you want to create. Helm creates the console application with the name that you specify. Provide a unique name to identify the application. The name must start and end with an alphanumeric character, be between 4 to 63 characters, and contain only lowercase characters. Optionally, you can use hyphen (-) and dot (.) as special characters. For example, `otmm-console`.

Property	Description
`tmmConsoleImage.image`	Enter the path of the console image in the remote repository. For example, `iad.ocir.io/mytenancy/Console:`*`RELEASE`*. If you don't provide a value, the console application is not deployed.
`tmmConsoleImage.imagePullPolicy`	Enter `Always` to ensure that the image is pulled during the installation. For more information, see Image pull policy in *Kubernetes documentation*.
`tmmConsoleImage.imagePullSecret`	Specify the name of the Kubernetes secret that you want to use to pull the Docker images. See Create a Kubernetes Secret to Access Docker Registry. For example, `regcred`.
`port`	Specify a port number. Ensure that the required networking rules are set up to ensure that traffic is permitted to and from this port to the MicroTx console.
`replicaCount`	Specify the number of MicroTx console replicas that you want to create.
`cookieEncryptionPasswordSecretName`	Specify the name of the Kubernetes secret that you have created to store the cookie encryption password for the Console. See Create a Secret with Cookie Encryption Password for Kubernetes. For example, `console-cookie-encryption-password-secret`.
`resources`	Tune requests or limits according to workload throughput and cluster quotas. For more information about the limits enforced by Kubernetes, see Requests and limits in *Kubernetes documentation*.

The following code snippet provides sample values for `tmmConsoleConfiguration`.

```
tmmConsoleConfiguration:
  tmmConsoleAppName: otmm-console
  # If tmmConsoleImage.image is empty, the console application is not
deployed.
  tmmConsoleImage:
    image: iad.ocir.io/mytenancy/Console:RELEASE
    imagePullPolicy: Always
    imagePullSecret: regcred
  port: 8080
  cookieEncryptionPasswordSecretName: "console-cookie-encryption-password-
secret"
```

4.2.7 MicroTx Workflows Server Configuration

In the `values.yaml` file, under `workflow.server`, provide information about the environment details in which you want to install MicroTx Workflows.

Property	Description
`appName`	Enter the name of the MicroTx Workflows application that you want to install. When you install MicroTx, Helm creates the MicroTx Workflows application with the name that you specify. Note down this name as you will need to provide it later. For example, `workflow-server`.

Property	Description
`replicaCount`	Specify the number of MicroTx Workflows pod replicas that you want to create. Oracle recommends a minimum of 3 replicas for production environments. Development or test environments may be able to use fewer replicas.
`image`	Enter the tag of the MicroTx Workflows image that you have pushed to the remote repository. For example, `workflow-server:1.0-SNAPSHOT`. See Push Images to a Remote Docker Repository.
`imagePullPolicy`	Enter `Always` to ensure that the image is pulled during the installation. For more information, see Image pull policy in *Kubernetes documentation*.
`imagePullSecret`	Specify the name of the Kubernetes secret that you have created. This secret is used to pull the Docker images from the remote repository. For example, `regcred`. See Create a Kubernetes Secret to Access Docker Registry.
`service.port`	Enter the port over which you want to internally access MicroTx Workflows within the Kubernetes cluster where you will install this service. Create the required networking rules to permit inbound and outbound traffic on this port. Note down this number as you will need to provide it later. For example, `9010`.
`config.queue`	Enter the storage type used to persist the MicroTx Workflows queue. The only supported value is `oracledb`.
`config.metadata`	Enter the storage type used to persist metadata. The only supported value is `oracledb`.
`config.indexing.enabled`	Mandatory. Set this to `true` to enable indexing.
`config.indexing.type`	Mandatory. Type `oracledb` as the indexing storage type.
`metrics.enabled`	Set this to `true` so that Prometheus can scrape the metrics of the MicroTx Workflows server. The default value is `true`.
`scheduler.enabled`	Set this to `true` to create, edit, view, or delete schedule definitions in MicroTx Workflows. Schedule definitions enable workflows to be executed automatically based on a configured schedule. The default value is `true`. If set to `false`, workflow executions cannot be scheduled.
`logging.core.level`	Specify one of the following log levels for the core MicroTx Workflows server. • `info`: Logs events that occur during the normal operation of the MicroTx Workflows. This setting logs the least amount of information. This is the default setting. • `warning`: Logs events that may cause potentially harmful situations. • `error`: Logs events to indicate that there is an issue that requires troubleshooting. • `debug`: Logs all the events. Use this setting when you want to debug an issue.
`logging.microtx.level`	Specify the log level for AI-related tasks as `info`, `warning`, `error`, or `debug`. You may want to set the log level to `info` for the MicroTx Workflows server and `debug` for AI-related tasks when troubleshooting or debugging issues in a task.
`resources`	Tune requests or limits according to workload throughput and cluster quotas. For more information about the limits enforced by Kubernetes, see Requests and limits in *Kubernetes documentation*.

- Data Store Properties
  MicroTx Workflows uses Oracle Database as data store to persist metadata and to store logs.
- Encryption Properties
  Under `workflow.server.encryption`, specify the encryption key that MicroTx Workflows uses to encrypt the access and refresh tokens. You must provide values for these properties if you have enabled `authTokenPropagationEnabled` under `tmmConfiguration.authorization`.
- Task Configuration Properties
  Under `workflow.server.taskConfig`, enter task-level property values that are shared across all tasks within a specific task category.
- Event Handler Configuration
  Under `workflow.server.eventHandler`, enable event handlers in MicroTx Workflows to listen to messages sent by Transactional Event Queues (TxEventQ) within MicroTx Workflows or to listen to external message brokers, such as Kafka. By default, event handlers created in MicroTx Workflows do not listen to message brokers.
- File System Storage Properties
  Under `workflow.config.storage.filesystem`, provide information about the file system you mount the MicroTx Workflows server.

4.2.7.1 Data Store Properties

MicroTx Workflows uses Oracle Database as data store to persist metadata and to store logs.

The following table specifies the property values required under `workflow.server.config.storage.oracledb` to connect to an Oracle Database as the MicroTx Workflows data store.

Property	Description
`url`	Enter the connection string for the data store in Oracle Database. See Get Autonomous Database Client Credentials.
`credentialSecretName`	Enter the name of the Kubernetes secret that contains the credentials to connect to the Oracle Database. Example, `db-secret`. See Create a Kubernetes Secret for Oracle Database Credentials.
`walletConfigMap`	Provide a value for this field only if you connect to a database that uses a credential wallet, such as Autonomous Database. For other databases, you can leave this field empty. Enter the name of the configuration map that you have created for the credential wallet of the Autonomous Database instance and not the name of the ZIP file. If you specify a value for this field, ensure that the database `url` you provide ends with ?`wallet_location=/app/oracle/wallet`. See Get Autonomous Database Client Credentials. For example, `db-wallet-configmap`.

4.2.7.2 Encryption Properties

Under `workflow.server.encryption`, specify the encryption key that MicroTx Workflows uses to encrypt the access and refresh tokens. You must provide values for these properties if you have enabled `authTokenPropagationEnabled` under `tmmConfiguration.authorization`.

Property	Description
`enabled`	Set this to `true` to enable encryption. The default value is `true`. You must set this value to true in production environments, where you also enable `security`.

Property	Description
`secretName`	Specify the name of the Kubernetes secrets that contain encryption key as the value. To support the encryption keys rotation, you can specify multiple encryption keys and their versions.
`fileName`	File name as stored in the secret and mounted.

The following code snippet provides sample values for the `encryption` field in the `values.yaml` file. The sample values in this example are based on the values used in the sample commands in Generate a Kubernetes Secret for an Encryption Key.

```
encryption:
      enabled: "true"
      secretName: encryption-secret
      fileName: encryption.key
```

4.2.7.3 Task Configuration Properties

Under `workflow.server.taskConfig`, enter task-level property values that are shared across all tasks within a specific task category.

For example, `txeventq.<property_name>` represents a property that is shared across all TxEventQ Publish tasks. If multiple TxEventQ Publish tasks are running, they share the same cache configuration. MicroTx Workflows uses these properties to optimize connection management and improve task execution performance. These settings are particularly useful in high-traffic environments because reusing cached connections reduces the overhead of creating new connections and improves performance.

Property	Description
`txeventq.requestTimeout`	Specifies the maximum amount of time a task waits for a request to reach the database. If the request does not reach the database within the configured timeout period, the task fails and times out. Default value is 20000 ms.
`txeventq.cacheTime`	Determines how long a cached entry remains valid before it expires and is removed from the cache. It acts as the Time To Live (TTL) for a cache entry. For TxEventQ tasks, the cache entry represents a database connection. For example, if the cache time is set to 1200 milliseconds, the database connection remains in the cache for up to 1200 milliseconds before it expires and is removed. Reusing cached connections helps optimize connection management and improves overall task execution performance. Default value is 120000 ms.
`txeventq.cacheSize`	Determines the maximum number of entries that a cache can contain. It directly correlates to the number of different database connections that can be stored in the cache at a given time. For example, if the cache size is set to 10, the system can cache up to 10 different database connections simultaneously. A larger cache size allows more database profiles or connections to be reused from the cache, which reduces the time required to establish new connections and improves overall task execution performance, especially in high-traffic environments. The default value is 10.
`transaction.readTimeout`	Maximum permissible time, in milliseconds (ms), to read full response after connection. Default value is 180000 ms. You can tune this for large payloads, such as transaction logs.
`transaction.connectTimeout`	Maximum permissible time, in milliseconds (ms), to establish a TCP connection. Default value is 180000 ms.
`grpc.cacheSize`	Determines the number of different connections that can be stored in the cache. Default value is 10.

Property	Description
`grpc.cacheTime`	Defines how long the cached connections remain valid before expiring. Default value is 120000 ms.
`sql.cacheSize`	Determines the maximum number of entries that a cache can contain. It directly correlates to the number of different database connections that can be stored in the cache at a given time. For example, if the cache size is set to 10, the system can cache up to 10 different database connections simultaneously. A larger cache size allows more database profiles or connections to be reused from the cache, which reduces the time required to establish new connections and improves overall task execution performance, especially in high-traffic environments. The default value is 10. Default value is 15.
`sql.cacheTime`	Determines how long a cached entry remains valid before it expires and is removed from the cache. It acts as the Time To Live (TTL) for a cache entry. For SQL tasks, the cache entry represents a database connection. For example, if the cache time is set to 1200 milliseconds, the database connection remains in the cache for up to 1200 milliseconds before it expires and is removed. Reusing cached connections helps optimize connection management and improves overall task execution performance. Default value is 600 s.

The following code snippet provides sample values for all the properties under `workflow.server.taskConfig` in the `values.yaml` file.

```
    # task configuration
    taskConfig:
      # TxEventQ task tuning knobs for timeout/caching behavior.
      txeventq:
        requestTimeout: 20000ms
        cacheTime: 120000ms
        cacheSize: 10
      # HTTP transaction task timeout settings for coordinator interactions.
      transaction:
        readTimeout: 180000ms
        connectTimeout: 180000ms
      # gRPC task connection/result caching controls.
      grpc:
        cacheSize: 10
        cacheTime: 120000ms
      # SQL task cache sizing and TTL controls.
      sql:
        cacheSize: 15
        cacheTime: 600s
```

4.2.7.4 Event Handler Configuration

Under `workflow.server.eventHandler`, enable event handlers in MicroTx Workflows to listen to messages sent by Transactional Event Queues (TxEventQ) within MicroTx Workflows or to listen to external message brokers, such as Kafka. By default, event handlers created in MicroTx Workflows do not listen to message brokers.

Property	Description
`txeventq.enabled`	Set this to `true` so that event handlers in MicroTx Workflows can listen to messages sent by Transactional Event Queues (TxEventQ) within MicroTx Workflows. The default value is `false`.

Property	Description
`kafka.enabled`	Set this to `true` so that event handlers in MicroTx Workflows can listen to external message brokers, such as Kafka. The default value is `false`.
`kafka.bootstrapServers`	Enter a comma-separated list of addresses of bootstrap servers to access Kafka brokers.

The following code snippet provides sample values for `eventHandler` in the `values.yaml` file.

```
eventHandler:
      #Enable txeventQ event handler
      txeventq:
        enabled: "false"
      # Kafka event handler settings
      kafka:
        # Enable Kafka event handler
        enabled: "false"
        # Usage: Comma-separated list of Kafka broker addresses (e.g.,
localhost:9092 or kafka1:9092,kafka2:9092)
        bootstrapServers: "kafka-service.kafka.svc.cluster.local:9092"
```

4.2.7.5 File System Storage Properties

Under `workflow.config.storage.filesystem`, provide information about the file system you mount the MicroTx Workflows server.

You must mount a file system from a PersistentVolumeClaim (PVC) in Kubernetes. For steps to create a PVC in Kubernetes in Oracle Cloud Infrastructure (OCI), see the `README.md` file located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/helmcharts/resources/persistent-volumes` folder. For instructions to create a PVC in other cloud environments, refer to the documentation specific to the cloud provider.

Property	Description
`fileSystem.enabled`	Set this to `true` to enable mounting a file system on the MicroTx Workflows server.
`pvcName`	Name of the PVC used to mount the file system.

The following code snippet provides sample values for `tmmImage`.

```
workflow:
  config:
    storage:
      fileSystem:
        enabled: "false"
        pvcName: wf-fss-pvc
```

When you upload files to Storage using MicroTx Workflows connectors or by using GenAI Ingestion task, files are uploaded to the specified file system.

4.3 Install MicroTx

Use Helm charts to install MicroTx on a Kubernetes cluster.

1. Navigate to the `helmcharts` folder.

   ```
   cd installation_directory/MicroTx-RELEASE/deployment-descriptors/helmcharts
   ```

2. Deploy MicroTx using the configuration details provided in the `values.yaml` file. Run the following command to install MicroTx Workflows, MicroTx Distributed Transactions coordinator, and MicroTx console in the `otmm` namespace.

   Syntax

   ```
   helm install <release name> --namespace <namespace> <chart directory> --
   values <values.yaml>
   ```

   Sample Command

   ```
   helm install tmm-app --namespace otmm tmm/ --values microtx/values.yaml
   ```

   Where,

   - `tmm-app` is the name of the application that you want to create.
   - otmm is the namespace in Kubernetes cluster, where you want to install MicroTx.
   - *installation_directory*`/MicroTx-`*RELEASE*`/deployment-descriptors/helmcharts/microtx` folder contains the `chart.yaml` file for MicroTx.
   - *installation_directory*`/MicroTx-`*RELEASE*`/deployment-descriptors/helmcharts/microtx` folder contains the `values.yaml` file, the application's manifest file, in your local machine. This file contains the deployment configuration details of MicroTx and the console.

   The following message is displayed.

   ```
   NAME: tmm-app
   LAST DEPLOYED: Tue Apr 19 21:14:25 2022
   NAMESPACE: otmm
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   ```

3. Verify that all resources, such as pods and services, are ready. Use the following command to retrieve the list of resources in the namespace `otmm` and their status.

   ```
   kubectl get all -n otmm
   ```

   Sample response

   Some of the values may be truncated with … for the sake of readability. When you run this command in your environment, you will see the entire value.

   ```
   NAME                 READY STATUS     RESTARTS AGE
   pod/otmm-console-7c6b-j6k     2/2   Running     0      38s
   pod/otmm-tcs-0             2/2   Running    0      38s
   pod/workflow-server-f8b7-sq84     2/2   Running     0      38s

   NAME             TYPE       CLUSTER-IP     EXTERNAL-IP PORT(S)
   AGE
   ```

```
service/otmm-console    ClusterIP 10.96.31..... <none>        5001/
TCP,5002/TCP 38s
service/otmm-tcs        ClusterIP 10.96.22..... <none>      9000/TCP 38s
service/workflow-server ClusterIP 10.96.14..... <none>      9010/TCP 38s

NAME                 READY UP-TO-DATE AVAILABLE AGE
deployment.apps/otmm-console    1/1   1      1        38s
deployment.apps/workflow-server 1/1   1      1         38s

NAME                         DESIRED   CURRENT READY   AGE
replicaset.apps/otmm-console-7c656b   1     1     1     38s
replicaset.apps/workflow-server-f8bf7 1     1     1     38s

NAME               READY        AGE
statefulset.apps/otmm-tcs   1/1         38s
```

When you install MicroTx using Helm, Helm deploys MicroTx Workflows, MicroTx Distributed Transactions transaction coordinator, MicroTx console in separate pods. After the installation is complete, you can access the services.

4.4 Access MicroTx

To access MicroTx, specify the port number, host name, and protocol that you want to use to access. Oracle recommends that you use HTTP protocol only in test or development environments. In production environments, you must use HTTPS protocol.

Use the internal URL or external URL to access MicroTx. You will use different URLs depending on whether you want to access MicroTx from within the Kubernetes cluster where you have deployed the service or from a different Kubernetes cluster.

Internal URL to access MicroTx

Use the internal URL to access MicroTx from within the Kubernetes cluster where you have deployed the service. For example, when you have deployed the transaction initiator application and MicroTx in the same Kubernetes cluster.

To access MicroTx, create the URL in the following format:

`http://`*`internalHostname`*`:`*`internalPort`*`/api/v1`

Where,

- *`internalHostname`*: Name that you have entered for the `tmmAppName` property in the `values.yaml` of MicroTx. For example, `tmm-app`.
- *`internalPort`*: Port number that you have entered for the `port` property in the `values.yaml` of MicroTx. For example, `9090`. Ensure that you have set up the required networking rules to permit HTTPS traffic on this port.

Based on the example values provided above, the example URL to access MicroTx is `https://tmm-app:9000/api/v1`.

All communication within a container uses the HTTP protocol as the communication goes through the Envoy proxy, which uses mTLS.

External URL to access MicroTx

Use the external URL to access MicroTx from outside the Kubernetes cluster where you have deployed the service. For example, when you deploy the transaction initiator application and MicroTx in different Kubernetes cluster. In such a scenario, the transaction initiator application uses the external URL to access MicroTx.

To access MicroTx externally, create the URL in the following format:

```
https://externalHostname:externalPort/api/v1
```

Where,

- `https` is the protocol permitted at the Istio ingress gateway. You can specify either `http` or `https`.
- *`externalHostname`*: The IP address of the load balancer of the Istio ingress gateway. See Find IP Address of Istio Ingress Gateway. For example, `192.0.2.1`.
- *`externalPort`*: Port number of the load balancer of the Istio ingress gateway. You must create the required networking rules to permit inbound and outbound traffic on this port. For example, `443`.

Based on the example values provided above, the example URL to access MicroTx is `https://192.0.2.1:443/api/v1`.

4.5 Check the Server Health

After installing MicroTx, validate that the installation was completed successfully.

Run the following commands to check the health of MicroTx Distributed Transactions coordinator and MicroTx Workflows.

1. Run the following command to check the health of MicroTx Distributed Transactions coordinator.

   Command Syntax

   ```
   curl https://externalHostname:externalPort/health
   ```

   To identify values for the *`externalHostname`* and *`externalPort`*, see Access MicroTx.

   Sample output

   If the MicroTx Distributed Transactions coordinator is healthy, the following response is displayed.

   ```
   {
       "coordinators": {
           "xaCoordinator": "Healthy",
           "lraCoordinator": "Healthy",
           "tccCoordinator": "Healthy"
       },
       "version": "",
       "started": "2023-10-11T11:22:44.343082+05:30"
   }
   ```

This indicates that you have successfully deployed MicroTx Distributed Transactions and the service is available to coordinate transactions.

2. Run the following command to check the health of MicroTx Workflows server.

   Command Syntax

   ```
   curl https://externalHostname:externalPort/workflow-server/health
   ```

   To identify values for the *externalHostname* and *externalPort*, see Access MicroTx.

   Sample output

   If the MicroTx Workflows server is healthy, the following response is displayed.

   ```
   {
     "status": "Healthy",
     "version": "26.1.1",
     "started": "2026-05-05T19:48:06.313321333Z"
   }
   ```

   This indicates that you have successfully deployed MicroTx Workflows and the service is available.

4.6 Find IP Address of Istio Ingress Gateway

Before you start a transaction, you must note down the external IP address of the Istio ingress gateway.

You need this information to access the applications.

1. Run the following command to find the external IP address of the Istio ingress gateway.

   Command

   ```
   kubectl get svc istio-ingressgateway -n istio-system
   ```

   Sample Output

   ```
   kubectl get svc istio-ingressgateway -n istio-system
   NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP
   PORT(S)                                      AGE
   istio-ingressgateway   LoadBalancer   10.109........   192.0.2.1
   15021:31695/TCP,80:32333/TCP,443:7777/TCP    44h
   ```

2. From the output note down the value of `EXTERNAL-IP`, which is the external IP address of the Istio ingress gateway, and the port associated with the HTTP or HTTPS traffic, based on the access protocol that you have configured. For example: `https://192.0.2.1:443`.

3. Store the external IP address of the Istio ingress gateway in an environment variable named CLUSTER_IPADDR as shown in the following command.

   ```
   export CLUSTER_IPADDR=192.0.2.1
   ```

   Note that, if you don't do this, then you must explicitly specify the IP address in the commands when required.

5

Upgrade to MicroTx 26.1

MicroTx 26.1 provides additional features.

Run one of the following steps only if you want to upgrade to MicroTx 26.1 to avail the latest features.

Topics

5.1 Upgrade to the Latest Free Release

Upgrade to the latest release of Transaction Manager for Microservices (MicroTx) Free.

To upgrade to the latest release of MicroTx Free from a previous release:

1. Download the files to install the latest release of MicroTx. See Download the Installation Bundle.

   Ensure that you do not overwrite the files that you had downloaded for earlier versions of MicroTx.

   The latest image of the transaction coordinator is available in the *installation_directory*/MicroTx-<*version*>/coordinator/container-images folder. In the next steps, you will use this file to update the existing transaction coordinator image.

2. Load the transaction coordinator image to the local repository, tag the image, and then push the image. See Push Images to a Remote Docker Repository.

3. Update the YAML file for the transaction coordinator with the name of the latest image in the repository. If you want to use the latest features, complete the required tasks to use these features, and then update the YAML file with the property values. See Configure the values.yaml File.

4. If you have installed MicroTx in a Kubernetes cluster, run the following command to complete the upgrade.

   Syntax

   ```
   helm upgrade <release name> --namespace <namespace> <chart directory> --
   values <path_to_updated_values.yaml>
   ```

   The following sample command upgrades the MicroTx Distributed Transactions application named `tmm-app` in the `MicroTx` namespace.

   ```
   helm upgrade tmm-app --namespace MicroTx tmm --values tmm/values1.yaml
   ```

   Where,

   - `tmm-app` is the name of the MicroTx Distributed Transactions coordinator.
   - `MicroTx` is the namespace in Kubernetes cluster, where you have installed MicroTx Distributed Transactions coordinator.
   - *`installation_directory`*`/MicroTx-<`*`version`*`>/MicroTx/helmcharts/tmm` is the folder that contains the `chart.yaml` file for MicroTx Distributed Transactions.
   - *`installation_directory`*`/MicroTx-<`*`version`*`>/MicroTx/helmcharts/tmm/values1.yaml` is the location of the application's updated manifest file in your local machine. This file contains the updated deployment configuration details for MicroTx Distributed Transactions.

5.2 Back Up Cached Maintenance Data

To retain information that's available in the cache, you must run the MicroTx Distributed Transactions transaction coordinator in maintenance mode before upgrading it.

When you enable cache, the cached data is available only locally to optimize the read and write operations. When you start the maintenance mode, transaction logs for all the replicas of the MicroTx Distributed Transactions coordinator are populated in both, the data store and cache.

You can back up cached data only when the following conditions are satisfied. Skip this procedure if the following conditions are not satisfied.

- Caching is enabled for the MicroTx Distributed Transactions transaction coordinator. See Enable Caching.
- MicroTx Distributed Transactions uses Oracle Database or etcd as data store. You cannot enable caching if you use internal memory as the data store.

To run the MicroTx Distributed Transactions transaction coordinator in maintenance mode:

1. Based on your environment, update the value of the namespace where you have deployed MicroTx Distributed Transactions in the `maintenance-broadcast.yaml` file. The `maintenance-broadcast.yaml` is in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/upgrades/maintenance` folder.

2. Run the `maintenance-broadcast.yaml` file in the namespace where you have deployed the MicroTx Distributed Transactions transaction coordinator.

   ```
   cd installation_directory/MicroTx-RELEASE/upgrades/maintenance
   ```

   ```
   kubectl apply -f maintenance-broadcast.yaml
   ```

   The following output is displayed.

   ```
   serviceaccount/otmm-tcs created
   pod/maintenance-job created
   role.rbac.authorization.k8s.io/endpoint-listing-role created
   rolebinding.rbac.authorization.k8s.io/endpoint-listing-role-binding created
   peerauthentication.security.istio.io/otmm-tcs-peerauth created
   ```

3. Clear the outdated cached maintenance data from the maintenance table in data store. This deletes the data which was backed up when you ran the transaction coordinator in the maintenance mode during an earlier upgrade. The `maintenance-script.sh` is in the *installation_directory*/MicroTx-*RELEASE*/upgrades/maintenance folder.

   ```
   kubectl exec -it maintenance-job -n otmm -- /bin/sh -c "$(cat maintenance-
   script.sh)"
   ```

   The following options are displayed.

   ```
   Which URL would you like to hit?
   1) Clear old maintenance data
   2) Start maintenance (5 mins)
   3) Start maintenance with a duration (mins)
   4) Check maintenance status
   Enter your choice (1, 2, 3 or 4):
   ```

   a. Enter **1**.

   > ⚠ **Caution**
   >
   > You must clear the outdated maintenance data before starting the maintenance mode. Do not use this option after starting the maintenance mode. In case of any data corruption in the maintenance mode or if the pods become unresponsive, this option can serve as a recovery point to restart the process.

   b. Enter the name of the MicroTx Distributed Transactions application that you have deployed.

   c. Enter the port over which you want to access the MicroTx Distributed Transactions transaction coordinator.

   Wait for the following message to be displayed, before proceeding to the next step.

   ```
   Maintenance data deleted successfully
   ```

4. Run the script again to start the maintenance mode, and then enter option **2** or **3**.

   ```
   kubectl exec -it maintenance-job -n otmm -- /bin/sh -c "$(cat maintenance-
   script.sh)"
   ```

   - Enter **2** to use the default time buffer of 5 minutes for the ongoing transactions to complete.
   - Enter **3** to use specify a time buffer, in minutes, for the ongoing transactions to complete. You will be prompted to enter a value in minutes. Enter an integer which is more than 5. If you enter a value less than 5 minutes, the script uses the default value of 5 minutes.

   The following message is displayed when the maintenance mode starts.

   ```
   coordinator maintenance process started
   ```

5. Run the script again, and then enter option **4** to check the progress and to confirm that the maintenance mode has completed successfully.

   ```
   kubectl exec -it maintenance-job -n otmm -- /bin/sh -c "$(cat maintenance-
   script.sh)"
   ```

   The following message is displayed when the maintenance mode is completed successfully.

   ```
   maintenance mode completed
   ```

   You may need to run this step a few times if the maintenance mode is still in progress. Proceed with the next step only after ensuring that the maintenance mode has completed successfully.

6. Run the following command to delete the resources created by `maintenance-broadcast.yaml`.

   ```
   kubectl delete -f maintenance-broadcast.yaml
   ```

   The following message is displayed.

   ```
   serviceaccount "otmm-tcs" deleted
   pod "maintenance-job" deleted
   role.rbac.authorization.k8s.io "endpoint-listing-role" deleted
   rolebinding.rbac.authorization.k8s.io "endpoint-listing-role-binding"
   deleted
   peerauthentication.security.istio.io "otmm-tcs-peerauth" deleted
   ```

Next: Use a suitable method to upgrade to the latest release of MicroTx Enterprise Edition. When you upgrade to the latest release, the upgrade process shuts down the older replicas of the MicroTx Distributed Transactions coordinator and brings up new replicas of the MicroTx Distributed Transactions coordinator in a rolling fashion. The cache of the new MicroTx Distributed Transactions coordinator replicas are populated with the copy of the cached maintenance data that's available in the data store.

5.3 Upgrade to the Latest Enterprise Edition in Kubernetes Cluster

Run these steps to upgrade to the latest release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition only if you use Oracle Database to store the transaction logs in the previous release. Run these steps on Kubernetes clusters.

The MicroTx Distributed Transactions coordinator runs in the main container. As part of the upgrade process, MicroTx Distributed Transactions creates a Kubernetes init container. The Kubernetes init container is a specialized container that runs in a pod. The Kubernetes init container starts, completes the prerequisite steps for the upgrade, upgrades the MicroTx Distributed Transactions coordinator, and then terminates when it finishes the upgrade. It uses the MicroTx Distributed Transactions image that the main container also uses. See https://kubernetes.io/docs/concepts/workloads/pods/init-containers/.

Prerequisites

Do not use the instructions in this section in the following scenarios:

- if you cannot grant the required privileges to the database user or if you cannot run the init process. In such scenarios, the database administrator must run SQL scripts to upgrade the Database. See Upgrade to the Latest Enterprise Edition Using SQL Scripts.
- if you used etcd or internal memory to store the transaction logs in a previous release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition.
- if you want to upgrade to the latest release of MicroTx Distributed Transactions Free.
- if you want to upgrade MicroTx Distributed Transactions in Docker. Docker does not support the init container functionality, which is available only in Kubernetes cluster. To upgrade in Docker environments, you must run the init process in an independent container. See Upgrade to the Latest Enterprise Edition in Docker.

To upgrade to the latest release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition:

1. Before you begin, run the MicroTx Distributed Transactions transaction coordinator in maintenance mode to back up cached data. See Back Up Cached Maintenance Data.
2. Grant privileges to create and run stored procedures to the data store user. Perform this task only if you use Oracle Database as the data store. See Grant Privileges to Database User.
3. Ensure that you have the following required privileges to perform this task if you are not a database administrator.
   - EXECUTE
   - INSERT
   - SELECT
   - UPDATE
   - DELETE
   - CREATE
   - ALTER

   MicroTx Distributed Transactions uses these privileges to handle the upgrade.

4. Download the files to install the latest release of MicroTx Distributed Transactions. See Download the Installation Bundle.

   Ensure that you do not overwrite the installer files for earlier versions of MicroTx Distributed Transactions.

   The latest image of the transaction coordinator is available at *installation_directory*/MicroTx-26.1/MicroTx/image/tmm-26.1.tgz. In the next steps, you will use this file to update the existing transaction coordinator image.

5. Load the transaction coordinator image to the local repository, tag the image, and then push the image. See Push Images to a Remote Docker Repository.

6. Create a copy of the `values.yaml` file and name it as `values1.yaml`. Update the `values1.yaml` file with the name of the latest transaction coordinator image in the repository. If you want to use the latest features, complete the required tasks to use these features, and then update the YAML file with the property values. See Configure the values.yaml File.

7. Update the property values for the Kubernetes init container in the `tcs.yaml` file, which is located at *installation_directory*/MicroTx-26.1/MicroTx/image/tcs.yaml. Use this file as a reference when you create your own Helm Chart to upgrade MicroTx Distributed Transactions. The following code snippet provides sample values. Provide the values based on your environment.

```
{{- if .Values.tmmConfiguration.storage.db }}
      {{- if eq .Values.tmmConfiguration.storage.type "db" }}
      initContainers:
        - name : {{ printf "init-%s" .Values.tmmConfiguration.tmmAppName }}
          image: {{ .Values.tmmImage.image }}
          securityContext:
            runAsUser: 1337
          volumeMounts:
            - name: config-volume
              mountPath: /etc/config
            {{-
if .Values.tmmConfiguration.storage.db.walletConfigMap.configMapName }}
            - name: wallet-volume
              mountPath: /etc/config/dbwallet
            {{- end }}
          env:
            - name: CONFIG_MAP_PATH
              value: /etc/config
            - name: INIT_CONTAINER_ENABLED
              value: "True"
            {{-
if .Values.tmmConfiguration.storage.db.credentialSecretName }}
            - name: STORAGE_DB_CREDENTIAL
              valueFrom:
                secretKeyRef:
                  key: secret
                  name:
{{ .Values.tmmConfiguration.storage.db.credentialSecretName }}
            {{- end }}
      {{- end }}
      {{- end }}
```

Parameters	Description
`securityContext:` `  runAsUser: 1337`	When you use a Kubernetes init container with an Istio service mesh, the Kubernetes init container cannot send any outgoing network calls as Istio does not support it. To enable the Kubernetes init container to send outgoing network calls, you must specify the security context. See https://istio.io/latest/docs/setup/additional-setup/cni/#compatibility-with-application-init-containers.
`INIT_CONTAINER_ENABLED`	Set this to `True` to enable the creation of a Kubernetes init container.

8. Run the following command to complete the upgrade.

   Syntax

   ```
   helm upgrade <release name> --namespace <namespace> <chart directory> --
   values <path_to_updated_values.yaml>
   ```

   The following sample command upgrades the MicroTx Distributed Transactions application named `tmm-app` in the `MicroTx` namespace.

   ```
   helm upgrade tmm-app --namespace MicroTx tmm/ --values tmm/ee/values1.yaml
   ```

   Where,

   - `tmm-app` is the name of the MicroTx Distributed Transactions application.
   - `MicroTx` is the namespace in Kubernetes cluster, where you have installed MicroTx Distributed Transactions.
   - *`installation_directory`*`/MicroTx-26.1/MicroTx/helmcharts/tmm` is the folder that contains the `chart.yaml` file for MicroTx Distributed Transactions.
   - *`installation_directory`*`/MicroTx-26.1/MicroTx/helmcharts/tmm/ee/values1.yaml` is the location of the `values1.yaml` file, the application's updated manifest file, in your local machine. This file contains the updated deployment configuration details for MicroTx Distributed Transactions.

5.4 Upgrade to the Latest Enterprise Edition in Docker

Run these steps to upgrade to the latest release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition only if you use Oracle Database to store the transaction logs in the previous release. You can run these steps in the Docker environment.

The MicroTx Distributed Transactions coordinator runs in the main container. As part of the upgrade process, MicroTx Distributed Transactions creates an init process in an independent container. The init process starts, completes the prerequisite steps for the upgrade, upgrades the MicroTx Distributed Transactions coordinator, and then terminates when it finishes the upgrade. It uses the MicroTx Distributed Transactions image that the main container also uses.

Do not use the instructions in this section in the following scenarios:

- if you cannot grant the required privileges to the database user, the database administrator must run SQL scripts to upgrade the Database. See Upgrade to the Latest Enterprise Edition Using SQL Scripts.
- if you used etcd or internal memory to store the transaction logs in a previous release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition.
- if you want to upgrade to the latest release of MicroTx Distributed Transactions Free.
- if you want to upgrade MicroTx Distributed Transactions in Kubernetes clusters. See Upgrade to the Latest Enterprise Edition in Kubernetes Cluster.

To upgrade to Transaction Manager for Microservices - Distributed Transactions 26.1 Enterprise Edition:

1. Before you begin, run the MicroTx Distributed Transactions transaction coordinator in maintenance mode to back up cached data. See Back Up Cached Maintenance Data.
2. Download the files for the latest release of MicroTx Distributed Transactions. See Download the Installation Bundle.

   Ensure that you do not overwrite the files that you had downloaded for earlier versions of MicroTx Distributed Transactions.

   The latest image of the transaction coordinator is available at *installation_directory*/MicroTx-26.1/MicroTx/image/tmm-26.1.tgz. In the next few steps, you will use this file to update the existing transaction coordinator image.
3. Load the MicroTx Distributed Transactions image to the local Docker repository. The MicroTx Distributed Transactions image is located at installation_directory/MicroTx-<*version*>/image/tmm-<*version*>.tgz.

   ```
   cd installation_directory/MicroTx-<version>/MicroTx
   docker load < image/tmm-<version>.tgz
   ```

   The following message is displayed when the image is loaded.

   ```
   Loaded image: tmm:<version>
   ```
4. Set INIT_CONTAINER_ENABLED to True in the tcs.yaml file to enable the creation of an init process. The tcs.yaml file is located in the *installation_directory*/MicroTx-26.1/MicroTx/image folder.
5. Run the following command to initiate an init process in a Docker container to upgrade MicroTx Distributed Transactions.

   Syntax

   ```
   docker container run --name init-otmm-app \
   -v "$(pwd)":/app/config \
   -w /app \
   -p 9000:9000/tcp \
   --env CONFIG_FILE=/app/config/tcs.yaml \
   --network=host \
   -e STORAGE_DB_CREDENTIAL='{"password":"<dbUserpassword>",
   "username":"<DBuserName>"}' \
   -d tmm:26.1
   ```

   Where,

- `--name init-otmm-app` is the name of the MicroTx Distributed Transactions application that you want to upgrade.
- `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
- `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
- `--env CONFIG_FILE=config/tcs.yaml` specifies the location of the `tcs.yaml` file, which contains the coordinator configuration details.
- `-e STORAGE_DB_CREDENTIAL` provides the user name and password to access Oracle Database.
- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

6. Remove the independent container, in which the init process ran, from the list of docker containers in Docker environment. The init process terminates automatically after it finishes the upgrade. Run the following command to remove the independent container.

   ```
   docker container rm <container_ID_of_init_otmm_app>
   ```

7. Set `INIT_CONTAINER_ENABLED` to `False` in the `tcs.yaml` file.
8. Run the following command to start running MicroTx Distributed Transactions in the Docker container.

   Syntax

   The following sample command runs the MicroTx Distributed Transactions application named `otmm-app`.

   ```
   docker container run --name otmm-app \
   -v "$(pwd)":/app/config \
   -w /app \
   -p 9000:9000/tcp \
   --env CONFIG_FILE=/app/config/tcs.yaml \
   --network=host \
   -e STORAGE_DB_CREDENTIAL='{"password":"<dbUserpassword>",
   "username":"<DBuserName>"}' \
   -d tmm:26.1
   ```

   Where,

   - `--name otmm-app` is the name of the MicroTx Distributed Transactions application that you want to create.
   - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
   - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
   - `--env CONFIG_FILE=config/tcs.yaml` specifies the location of the `tcs.yaml` file, which contains the coordinator configuration details.
   - `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

5.5 Upgrade to the Latest Enterprise Edition Using SQL Scripts

Run these steps to upgrade to the latest release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition using SQL scripts only if you used Oracle Database to store the transaction logs in the previous release.

Prerequisites

- Oracle Database is installed and running.
- SQL*Plus or any other Oracle-compatible SQL tool is available to run SQL scripts on the Oracle Database.
- Ensure that you have the following required privileges to perform this task if you are not a database administrator.
  - SELECT privilege on DBA_TABLE
  - CREATE SESSION
  - CREATE TABLE
  - CREATE PROCEDURE
  - EXECUTE ANY PROCEDURE

Do not use the instructions in this section in the following scenarios:

- if you can grant the required privileges, such as `ALTER` and `DELETE` to the database user and if init containers are available in your environment. In such scenarios, skip this section and see Upgrade to the Latest Enterprise Edition in Kubernetes Cluster.
- if you used etcd or internal memory to store the transaction logs in a previous release of Transaction Manager for Microservices - Distributed Transactions Enterprise Edition.
- if you want to upgrade to the latest release of MicroTx Distributed Transactions Free.

To upgrade to Transaction Manager for Microservices - Distributed Transactions 26.1 Enterprise Edition:

1. Before you begin, run the MicroTx Distributed Transactions transaction coordinator in maintenance mode to back up cached data. See Back Up Cached Maintenance Data.
2. Download the files to install the latest release of MicroTx Distributed Transactions. See Download the Installation Bundle.

   Ensure that you do not overwrite the installer files for earlier versions of MicroTx Distributed Transactions.

   The latest image of the transaction coordinator is available at *`installation_directory`*`/MicroTx-26.1/MicroTx/image/tmm-26.1.tgz`. In the next steps, you will use this file to update the existing transaction coordinator image.
3. Load the transaction coordinator image to the local repository, tag the image, and then push the image. See Push Images to a Remote Docker Repository.
4. Update the YAML file for the transaction coordinator with the name of the latest image in the repo. If you want to use the latest features, complete the required tasks to use these features, and then update the YAML file with the property values. See Configure the values.yaml File.
5. Only if you are setting up a fresh Oracle Database, run the `Setup_Microtx.sql` file to create the required tables in the Database. Skip this step if you are upgrading the MicroTx

Distributed Transactions coordinator with an existing Oracle Database. This file is available at *installation_directory*/MicroTx-*RELEASE*/MicroTx/dba_scripts.

6. Run the 24.2.1_MicroTx.sql file to create an entry for the latest release. This file is available at *installation_directory*/MicroTx-*RELEASE*/MicroTx/dba_scripts.
7. Start the transaction coordinator.

A

Install on Docker Compose

You can install MicroTx Distributed Transactions on Docker Compose and run sample applications. For deployment instructions, refer to the `README.md` file in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/docker-compose` folder.

You can create a similar configuration to install MicroTx Distributed Transactions in other supported environments. If you want to install MicroTx Distributed Transactions on a Kubernetes cluster, skip this section and see Install on a Kubernetes Cluster.

> **Note**
>
> The instructions provided in this section are specific to test or development environments. Do not use these instructions to set up and use MicroTx Distributed Transactions in production environments.

Do not use these instructions to install MicroTx Workflows on Docker Compose. Currently, Docker Compose deployment descriptors are provided only for the MicroTx Distributed Transactions coordinator, and not for MicroTx Workflows.

B

Run MicroTx Distributed Transactions in a Docker Container in Local Environment

You can run MicroTx in an independent Docker container in local environments on Windows, macOS, and Linux operating systems.

Do not use these instructions to install MicroTx Workflows on Docker. Currently, Docker deployment descriptors are provided only for the MicroTx Distributed Transactions coordinator, and not for MicroTx Workflows.

Topics

B.1 Run MicroTx Distributed Transactions in a Docker Container on Linux

Use the following commands to run MicroTx Distributed Transactions in an independent Docker container on Linux operating system.

Before you begin, ensure that you have loaded the MicroTx Distributed Transactions Docker image and updated the `tcs.yaml` file. The `tcs.yaml` file is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/docker-compose/config` folder on your local machine. This file contains the deployment configuration details for MicroTx Distributed Transactions. The coordinator image is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/coordinator/container-images` folder on your local machine.

Run MicroTx Distributed Transactions using the configuration details provided in the `tcs.yaml` file. Ensure that the `tcs.yaml` file is present in the current working directory.

1. Use the following command to run MicroTx Distributed Transactions in a Docker container if you have not set up a data store and want to use internal memory.

Sample Command

```
docker container run --name otmm -v "$(pwd)":/app/config \
-w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs.yaml \
--network=host -d tmm:<version>
```

Where,

- `--name otmm` is the name of the container that you want to create.
- `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
- `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
- `--env CONFIG_FILE=config/tcs.yaml` specifies the location of the `tcs.yaml` file, which contains coordinator configuration.
- `--network=host` is a network configuration. The callback URL is an IPv4 address. To retrieve the IPv4 address, run the `ipconfig` command and then note down the value of `IPv4 Address`.
- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

2. Use the following commands to run MicroTx Distributed Transactions in a Docker container when you want to use Oracle Database as a data store to persist the transaction state and to store the transaction logs.
   a. Create a copy of the `tcs.yaml` file and name it as `tcs-db.yaml`.
   b. Open the `tcs-db.yaml` file in any code editor. Under `tmmConfiguration.storage`, specify the `type` as `db`, and then provide details to connect to an Oracle Database under `tmmConfiguration.storage.db`. See Data Store Properties.
   c. Use the following command to run MicroTx Distributed Transactions in a Docker container while providing credentials to access Oracle Database. Replace <*DBpassword*> and <*DBuser*> with the password and user name to access Oracle Database in your environment.

      Sample Command

      ```
      docker container run --name otmm -v "$(pwd)":/app/config \
      -w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs-db.yaml \
      --network=host \
      -e STORAGE_DB_CREDENTIAL='{"password":"<DBpassword>",
      "username":"<DBuser>"}' \
      -d tmm:23.4.2
      ```

      Where,

      - `--name otmm` is the name of the container that you want to create.
      - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
      - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.

- `--env CONFIG_FILE=config/tcs-db.yaml` specifies the location of the `tcs.yaml` file, which contains configuration details for the coordinator and Oracle Database credentials.
- `--network=host` is a network configuration. The callback URL is an IPv4 address.
- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

3. After MicroTx Distributed Transactions is installed, run the MicroTx Distributed Transactions health check API to verify that the MicroTx Distributed Transactions coordinator is up and running.

```
curl --location \
          'http://localhost:9000/health' \
          --header 'Accept: application/json'
```

B.2 Run MicroTx Distributed Transactions in a Docker Container on Windows

Use the following commands to run MicroTx Distributed Transactions in an independent Docker container on Windows operating system. These commands have been tested on PowerShell.

Before you begin, ensure that you have loaded the MicroTx Distributed Transactions Docker image and updated the `tcs.yaml` file. The `tcs.yaml` file is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/docker-compose/config` folder on your local machine. This file contains the deployment configuration details for MicroTx Distributed Transactions. The coordinator image is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/coordinator/container-images` folder on your local machine.

Run MicroTx Distributed Transactions using the configuration details provided in the `tcs.yaml` file. Ensure that the `tcs.yaml` file is present in the current working directory.

1. Use the following command to run MicroTx Distributed Transactions in a Docker container if you have not set up a data store and want to use internal memory.

   Sample Command

```
docker container run --name otmm -v "$(pwd)":/app/config \
-w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs.yaml \
--network=host -d tmm:<version>
```

   Where,

   - `--name otmm` is the name of the container that you want to create.
   - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
   - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
   - `--env CONFIG_FILE=config/tcs.yaml` specifies the location of the `tcs.yaml` file, which contains coordinator configuration.

- `--network=host` or `--add-host host.docker.internal:host-gateway`. Based on your requirements, select a network configuration.
  - If you specify `--network=host`, the callback URL is `host.docker.internal`.
  - If you specify `--add-host host.docker.internal:host-gateway`, the callback URL is an IPv4 address. To retrieve the IPv4 address, run the `ipconfig` command and then note down the value of `IPv4 Address`.
- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

2. Use the following commands to run MicroTx Distributed Transactions in a Docker container when you want to use Oracle Database as a data store to persist the transaction state and to store the transaction logs.
   a. Create a copy of the `tcs.yaml` file and name it as `tcs-db.yaml`.
   b. Open the `tcs-db.yaml` file in any code editor. Under `tmmConfiguration.storage`, specify the `type` as `db`, and then provide details to connect to an Oracle Database under `tmmConfiguration.storage.db`. See Data Store Properties.
   c. Store the credentials to access Oracle Database in an environment variable, `STORAGE_DB_CREDENTIAL`, as shown in the following example. Provide values based on your environment. You'll use this environment variable in the next step.

      ```
      $env:STORAGE_DB_CREDENTIAL='{\"password\":\"<dbUserpassword>\",
      \"username\":\"<DBuserName>\"}'
      ```

   d. Use the following command to run MicroTx Distributed Transactions in a Docker container while providing credentials to access Oracle Database.

      Sample Command

      ```
      docker container run --name otmm -v "$(pwd)":/app/config \
      -w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs-db.yaml \
      --network=host \
      -env STORAGE_DB_CREDENTIAL="$($env:STORAGE_DB_CREDENTIAL)" \
      -d tmm:23.4.2
      ```

      Where,

      - `--name otmm` is the name of the container that you want to create.
      - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
      - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
      - `--env CONFIG_FILE=config/tcs-db.yaml` specifies the location of the `tcs-db.yaml` file, which contains coordinator configuration and credentials to connect to an Oracle Database.
      - `--network=host` or `--add-host host.docker.internal:host-gateway`. Based on your requirements, select a network configuration.
        - If you specify `--network=host`, the callback URL is `host.docker.internal`.

  - If you specify `--add-host host.docker.internal:host-gateway`, the callback URL is an IPv4 address. To retrieve the IPv4 address, run the `ipconfig` command and then note down the value of `IPv4 Address`.

- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

3. After MicroTx Distributed Transactions is installed, run the MicroTx Distributed Transactions health check API to verify that the MicroTx Distributed Transactions coordinator is up and running.

```
curl --location \
          'http://localhost:9000/health' \
          --header 'Accept: application/json'
```

B.3 Run MicroTx Distributed Transactions in a Docker Container on macOS (Intel x86)

Use the following commands to run MicroTx Distributed Transactions in an independent Docker container on macOS (Intel x86) operating system.

Before you begin, ensure that you have loaded the MicroTx Distributed Transactions Docker image and updated the `tcs.yaml` file. The `tcs.yaml` file is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/docker-compose/config` folder on your local machine. This file contains the deployment configuration details for MicroTx Distributed Transactions. The coordinator image is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/coordinator/container-images` folder on your local machine.

Run MicroTx Distributed Transactions using the configuration details provided in the `tcs.yaml` file. Ensure that the `tcs.yaml` file is present in the current working directory.

1. Use the following command to run MicroTx Distributed Transactions in a Docker container if you have not set up a data store and want to use internal memory.

   Sample Command

```
docker container run --name otmm -v "$(pwd)":/app/config \
-w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs.yaml \
--network=host -d tmm:<version>
```

   Where,

   - `--name otmm` is the name of the container that you want to create.
   - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
   - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
   - `--env CONFIG_FILE=config/tcs.yaml` specifies the location of the `tcs.yaml` file, which contains coordinator configuration.
   - `--network=host` is a network configuration. In macOS, the callback URL is `host.docker.internal2`.

- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

2. Use the following commands to run MicroTx Distributed Transactions in a Docker container when you want to use Oracle Database as a data store to persist the transaction state and to store the transaction logs.

   a. Create a copy of the `tcs.yaml` file and name it as `tcs-db.yaml`.

   b. Open the `tcs-db.yaml` file in any code editor. Under `tmmConfiguration.storage`, specify the `type` as `db`, and then provide details to connect to an Oracle Database under `tmmConfiguration.storage.db`. See Data Store Properties.

   c. Use the following command to run MicroTx Distributed Transactions in a Docker container while providing credentials to access Oracle Database. Replace *<DBpassword>* and *<DBuser>* with the password and user name to access Oracle Database in your environment.

      Sample Command

      ```
      docker container run --name otmm -v "$(pwd)":/app/config \
      -w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs-db.yaml \
      --network=host \
      -e STORAGE_DB_CREDENTIAL='{"password":"<DBpassword>",
      "username":"<DBuser>"}' \
      -d tmm:<version>
      ```

      Where,

      - `--name otmm` is the name of the container that you want to create.
      - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
      - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
      - `--env CONFIG_FILE=config/tcs-db.yaml` specifies the location of the `tcs.yaml` file, which contains configuration details for the coordinator and Oracle Database credentials.
      - `--network=host` is a network configuration. In macOS, the callback URL is `host.docker.internal2`.
      - `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

3. After MicroTx Distributed Transactions is installed, run the MicroTx Distributed Transactions health check API to verify that the MicroTx Distributed Transactions coordinator is up and running.

   ```
   curl --location \
            'http://localhost:9000/health' \
            --header 'Accept: application/json'
   ```

B.4 Run MicroTx Distributed Transactions in a Docker Container on macOS (Arm)

Use the following commands to run MicroTx Distributed Transactions in an independent Docker container on macOS of Apple M-series (Arm architecture).

Before you begin, ensure that you have loaded the MicroTx Distributed Transactions Docker image and updated the `tcs.yaml` file. The `tcs.yaml` file is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/deployment-descriptors/docker-compose/config` folder on your local machine. This file contains the deployment configuration details for MicroTx Distributed Transactions. The coordinator image is located in the *`installation_directory`*`/MicroTx-`*`RELEASE`*`/coordinator/container-images` folder on your local machine.

Run MicroTx Distributed Transactions using the configuration details provided in the `tcs.yaml` file. Ensure that the `tcs.yaml` file is present in the current working directory.

1. Create a custom network.

   ```
   docker network create otmm_network
   ```

   Where, *`otmm_network`* is the name of the custom network. Note down this name as you will provide it in the next step.

2. Use the following command to run MicroTx Distributed Transactions in a Docker container if you have not set up a data store and want to use internal memory.

   Sample Command

   ```
   docker container run --name otmm -v "$(pwd)":/app/config \
   -w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs.yaml \
   --network otmm_network -d tmm:<version>
   ```

   Where,

   - `--name otmm` is the name of the container that you want to create.
   - `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
   - `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
   - `--env CONFIG_FILE=config/tcs.yaml` specifies the location of the `tcs.yaml` file, which contains coordinator configuration.
   - `--network otmm_network` is the name of the custom network that you have created in the previous step.
   - `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

3. Use the following commands to run MicroTx Distributed Transactions in a Docker container when you want to use Oracle Database as a data store to persist the transaction state and to store the transaction logs.

   a. Create a copy of the `tcs.yaml` file and name it as `tcs-db.yaml`.

b. Open the `tcs-db.yaml` file in any code editor. Under `tmmConfiguration.storage`, specify the `type` as `db`, and then provide details to connect to an Oracle Database under `tmmConfiguration.storage.db`. See Data Store Properties.

c. Use the following command to run MicroTx Distributed Transactions in a Docker container while providing credentials to access Oracle Database. Replace *<DBpassword>* and *<DBuser>* with the password and user name to access Oracle Database in your environment.

Sample Command

```
docker container run --name otmm -v "$(pwd)":/app/config \
-w /app -p 9000:9000/tcp --env CONFIG_FILE=/app/config/tcs-db.yaml \
--network otmm_network \
-e STORAGE_DB_CREDENTIAL='{"password":"<DBpassword>",
"username":"<DBuser>"}' \
-d tmm:<version>
```

Where,

- `--name otmm` is the name of the container that you want to create.
- `-v "$(pwd)":/app/config` mounts the current directory into container at the `/app/config` path.
- `-w /app` specifies the working directory as `/app`. In this sample command, `/app` remains the default working directory as the volume is also mounted on this directory based on the value provided for the `-v` flag.
- `--env CONFIG_FILE=config/tcs-db.yaml` specifies the location of the `tcs.yaml` file, which contains configuration details for the coordinator and Oracle Database credentials.
- `--network otmm_network` is the name of the custom network that you have created in the previous step.
- `tmm:<version>` is the MicroTx Distributed Transactions Docker image that you have loaded to the local Docker repository.

4. Update the participant service URL in the `application.properties` file of each participant service. For example, `http://host.docker.internal:8083`.

5. After MicroTx Distributed Transactions is installed, run the MicroTx Distributed Transactions health check API to verify that the MicroTx Distributed Transactions coordinator is up and running.

```
curl --location \
     'http://localhost:9000/health' \
     --header 'Accept: application/json'
```

C

Configure Coordinator Using Environment Variables

You can provide configuration details for the transaction coordinator using environment variables or YAML files.

Do not use these instructions for MicroTx Workflows.

Topics

C.1 Provide Configuration Details

You can provide configuration details in environment variables or in a `YAML` file.

To use environment variables to provide configuration details for the MicroTx Distributed Transactions transaction coordinator, see Environment Variables for Transaction Coordinator and skip this section. You can use this option only if you are using MicroTx Distributed Transactions Free.

To provide configuration details for the MicroTx Distributed Transactions transaction coordinator using `YAML` files:

1. Use the `YAML` files, the manifest file of the application, which contains the deployment configuration details for MicroTx Distributed Transactions. To install MicroTx Distributed Transactions on a Kubernetes cluster, provide the configuration details in the `values.yaml` file. See Environment Variables for Transaction Coordinator.
2. You can use the updated `YAML` file to deploy MicroTx Distributed Transactions using `helm install` command in Kubernetes cluster or `docker container run` command in Docker container.
3. In multi-cloud environments, you may not be able to directly pass a `YAML` file. In such scenarios, you can base64-encode the YAML file, and then pass the encoded YAML file value as an environment variable. Provide values for the following environment variables.
   - `CONFIG_FILE`: Name of the `YAML` file that contains the configuration details. For example, `tcs-values.yaml`.
   - `CONFIG_FILE_BASE64`: Base64-encoded value of the file that you have specified in `CONFIG_FILE`.

> **Note**
>
> Based on your environment, consider the character limit on environment variable value. Ensure that the base64-encoded value is not truncated due to any limitations of your environment.

C.2 Environment Variables for Transaction Coordinator

Provide values for the following environment variables to configure the MicroTx Distributed Transactions transaction coordinator. These details are required to deploy MicroTx Distributed Transactions.

Property	Description
`TMM_APPNAME`	Enter the name of the MicroTx Distributed Transactions application that you want to create when you install MicroTx Distributed Transactions. Note down this name as you will need to provide it later. For example, `tmm-app`.
`PORT`	Enter the port over which you want to internally access MicroTx Distributed Transactions within the Kubernetes cluster where you will install this service. Create the required networking rules to permit inbound and outbound traffic on this port. Note down this number as you will need to provide it later. For example, `9000`.
`ID`	Enter a value to uniquely identify each instance of MicroTx Distributed Transactions that you install. The unique identifier must have 5-characters and can contain only alphanumeric characters (a-z, A-Z, and 0-9). For example, `TMM01`. Use this ID to identify MicroTx Distributed Transactions when there are multiple installations. You cannot use this ID to differentiate between replicas of a single instance of MicroTx Distributed Transactions installation as all the replicas have the same ID. You can't change this value after installing MicroTx Distributed Transactions.
`APPLICATION_NAMESPACE`	Specify the namespace in which you want to deploy MicroTx Distributed Transactions.
`LISTEN_ADDR`	Enter the port over which you want to access MicroTx Distributed Transactions. Create the required networking rules to permit inbound and outbound traffic on this port. Note down this number as you will need to provide it later. For example, `0.0.0.0:9000`. Specify the listener address in the format, `<IP_address>:<port>`, as provided in the example.
`INTERNAL_ADDR`	Enter the internal URL to access MicroTx Distributed Transactions from within the environment where you will install the service.
`EXTERNAL_ADDR`	Enter the external URL to access MicroTx Distributed Transactions from outside the environment where you will install the service.

Property	Description
HTTP_CLIENT_TIMEOUT_IN_SECS	Specify the maximum amount of time, in seconds, for which the HTTP callback API requests sent by the MicroTx Distributed Transactions coordinator to the participant services remains active. Enter an integer between 0 to 900. The default value is 180 seconds and the maximum value is 900 seconds. If you set this value to 0, then MicroTx Distributed Transactions does not enforce any limit. When the coordinator sends a HTTP callback API request to the participant services, the participant services must respond within the time period that you specify. If the participant service does not respond within the specified time period, the HTTP request sent by the coordinator times out.
XA_COORDINATOR_ENABLED	Set to `true` if your microservices use the XA transaction protocol. If you want to nest an XA transaction within a Saga transaction, set both XA_COORDINATOR_ENABLED and LRA_COORDINATOR_ENABLED to `true`.
XA_COORDINATOR_TX_MAX_TIMEOUT	Only for the XA transaction protocol. Specify the maximum amount of time, in milliseconds, for which the transaction remains active. If a transaction is not committed or rolled back within the specified time period, the transaction is rolled back. The default value is 600000 ms.
LRA_COORDINATOR_ENABLED	Set to `true` if your microservices use the Saga transaction protocol. If you want to nest an XA transaction within a Saga transaction, set both XA_COORDINATOR_ENABLED and LRA_COORDINATOR_ENABLED to `true`.
TCC_COORDINATOR_ENABLED	Set to `true` if your microservices use the TCC transaction protocol.
NARAYANA_LRA_COMPATIBILITY_MODE	Only for the Saga transaction protocol. Set this property to `true` when you want to use Saga participant applications that were implemented to work with the Narayana LRA Coordinator and now would participate in Saga transactions using MicroTx Distributed Transactions. Enable this mode to ensure that the MicroTx Distributed Transactions Saga APIs return the same response data that Narayana LRA Coordinator APIs return.
LOGGING_LEVEL	Enter one of the following types to specify the log level for MicroTx Distributed Transactions: • `info`: Logs events that occur during the normal operation of the MicroTx Distributed Transactions. This setting logs the least amount of information. This is the default setting. • `warning`: Logs events that may cause potentially harmful situations. • `error`: Logs events to indicate that there is an issue that requires troubleshooting. • `debug`: Logs all the events. Use this setting when you want to debug an issue.
HTTP_TRACE_ENABLED	Set this to `True` to log all the HTTP request and responses in MicroTx Distributed Transactions when you want to debug. If you set this to `True`, you must also set the LOGGING_LEVEL to `debug`.
LOGGING_DEV_MODE	Set this to `True` only in test environments to get more details for debugging purposes. You must set this to `False` in production environments.
MAX_RETRY_COUNT	The maximum number of times that the transaction coordinator retries sending the same request again in case of any failures. The default value is 0.
MIN_RETRY_INTERVAL	The minimum interval, in milliseconds, after which the transaction coordinator retries sending the same request again in case of any failures. The default value is 0.

Property	Description
MAX_RETRY_INTERVAL	The maximum retry interval, in milliseconds, before which the transaction coordinator retries sending the same request again in case of any failures. For example, 10000. The default value is 0.
SKIP_VERIFY_INSECURE_TLS	Oracle recommends that you set this value to `false` and set up a valid certificate signed by trusted authorities for secure access. When you set this value to `false`, the transaction coordinator accesses the participant applications over the HTTPS protocol with a valid certificate signed by trusted authorities. The default value is `false`. If you set this value to `true`, the transaction coordinator can access the participant application's callback URL, without a valid SSL certificate, in an insecure manner. **⚠ Caution** Do not set this value to `true` in production environments.
SERVE_TLS_ENABLED	Set this to `true` to enable TLS to ensure secure communication between participant services and MicroTx Distributed Transactions. You must provide details for the certificate and key file under SERVE_TLS_CERT_FILE and SERVE_TLS_KEY_FILE properties. When you enable TLS, you can access the transaction coordinator over HTTPS. For secure access to MicroTx Distributed Transactions over HTTPS, create a self-signed certificate and note down location of the certificate and private key. For information about creating an SSL certificate, see Guidelines for Generating Self-Signed Certificate and Private Key using OpenSSL in *Security Guide*. If you set this field to `false`, you don't have to provide values for SERVE_TLS_CERT_FILE and SERVE_TLS_KEY_FILE. When you disable TLS, you can access the transaction coordinator over HTTP. You must provide the `internalAddr` and `externalUrl` using HTTP protocol. For example, `http://localhost:9000`. **⚠ Caution** You must set this to `true` in production environments.
SERVE_TLS_CERT_FILE	Path to the TLS certificate, in PEM format, on your local machine.
SERVE_TLS_KEY_FILE	Path to the private key file, in PEM format, which is associated with the certificate on your local machine.
COMPLETED_TX_TTL	The time to live (TTL) in seconds for a completed transaction record in the transaction data store. The permissible range of values is 60 to 1200 seconds. When the specified time period expires, the completed transaction entry is removed from the data store. The default value is 60 seconds.

Property	Description
AUTHENTICATION_ENABLED	Set to `false` to bypass JWT authentication. This permits requests that do not have JWT tokens. Enter `true` if you want all requests to have a JWT token. MicroTx validates the token provided in the request and denies access if the token is invalid. If you set `enabled` to `true`, then you must provide values for the `issuer` and `jwksUri` parameters of the JWT. **⚠ Caution** You must set this property to `true` in production environments.
AUTHORIZATION_ENABLED	Set this to `true` to enable MicroTx Distributed Transactions check the subject in the incoming JWT token. MicroTx Distributed Transactions then tags the subject or user against the transaction ID, and further changes to the transaction is allowed only by the tagged subject or user. If you set this field to `false`, you don't have to provide identity provider details.
IDENTITY_PROVIDER_URL	Specify the URL of the identity provider. This information is required to create a new access token by using the refresh token. If you do not provide this information, expired access tokens are not auto-refreshed.
IDENTITY_PROVIDER_CLIENT_ID	Specify the client ID of the identity provider. This information is required to create a new access token by using the refresh token. If you do not provide this information, expired access tokens are not auto-refreshed.