Oracle®

Transaction Manager for Microservices Quick Start Guide

Release 24.2
F57110-06
June 2024

Oracle Transaction Manager for Microservices Quick Start Guide, Release 24.2

F57110-06

Primary Author: Sylaja Kannan

Contributing Authors: Tulika Das

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

Contents

1

About MicroTx

Oracle Transaction Manager for Microservices (MicroTx) enables enterprise users to adopt and increase use of microservices architecture for mission-critical applications by providing capabilities that make it easier to develop, deploy, and maintain such applications.

As organizations rush to adopt microservices architecture, they often run into problems associated with data consistency as each microservice typically has its own database. In monolithic applications, local transactions were enough as there were no other sources of data that needed to be consistent with the database. An application would start a local transaction, perform some updates, and then commit the local transaction to ensure the application moved from one consistent state to another. Once the application's state is spread across multiple sources of data, some factors need to be considered. What happens if updates succeed in one microservice, but it fails in another microservice as part of the same request? One solution is to use a distributed transaction that spans the sources of data used by the microservices involved in a request. Oracle Transaction Manager for Microservices provides a transaction coordination microservice and libraries to maintain consistency in the state of microservices participating in a transaction.

1.1 About the Distributed Transaction Protocols

MicroTx supports the following distributed transaction protocols:

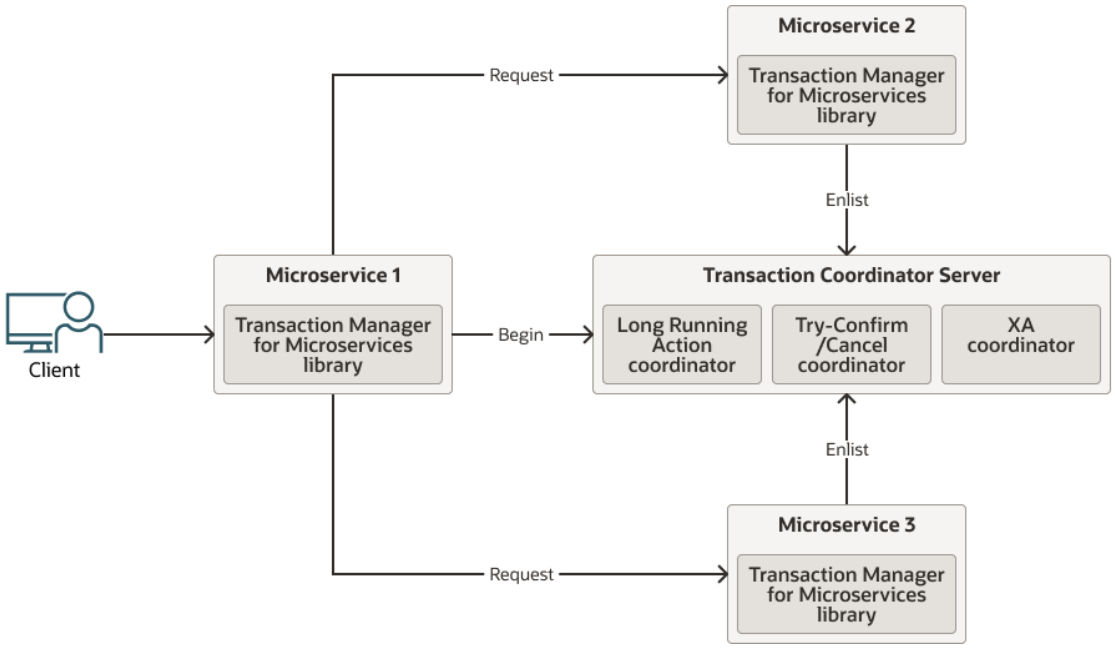
- XA protocol, which is based upon The Open Group's XA specification. For details about the specification, see https://pubs.opengroup.org/onlinepubs/009680699/toc.pdf.
- Saga protocol, which is based on the Eclipse MicroProfile LRA specification. For details about the specification, see https://download.eclipse.org/microprofile/microprofile-lra-1.0-M1/microprofile-lra-spec.html.
- Try-Confirm/Cancel (TCC) protocol

Use XA when strong consistency is required, similar to consistency provided by the local database transactions, where all the ACID properties of a transaction are present. For example, financial applications. Use the Saga protocol for transactions that may take a long time to complete. You can use the Saga protocol to mitigate locking issues. The TCC protocol fits well for applications that use a reservation model, such as airline seats or hotel rooms. Both Saga and TCC support long running transactions. Saga is far more general, but requires application specific actions for both completing a successful Saga and compensating a failed Saga. Whereas, compensation in TCC is performed by deleting the reservation, and then returning whatever was reserved to the pool of available resources.

1.2 Components of MicroTx

MicroTx contains two components: the transaction coordinator and the MicroTx library.

MicroTx, a containerized microservice, runs along with your application microservices. The following figure shows how the components of MicroTx interact with your application microservices.



Transaction Coordinator Server

The transaction coordinator manages transactions amongst the participant services.

MicroTx supports internal memory, Oracle Database, and etcd as a data store for persistence of transaction state.

MicroTx library

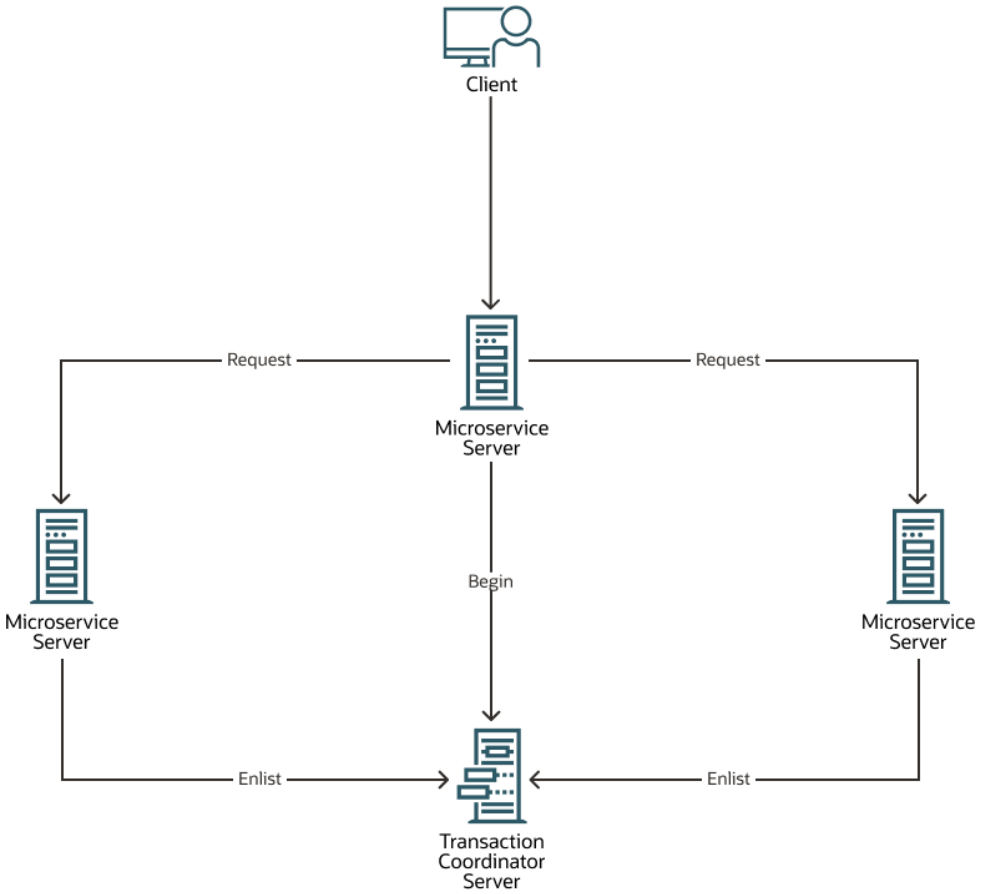
Application microservices provide the business logic and demarcate transaction boundaries. These services participate in a distributed transaction. They use MicroTx APIs to manage their distributed transactions.

Application developers use different parts of the MicroTx client library depending on the following factors:

- The development framework of the microservice, such as Helidon or Node.js.
- The selected transaction protocol, such as XA, Saga, or TCC.
- Whether the application initiates a transaction or participates in the transaction.
  - Transaction initiator service - These applications start and end a transaction. In the preceding figure, Microservice 1 is the transaction initiator service and it sends a request to MicroTx to begin the transaction.
  - Transaction participant service - These applications only join the transaction. They do not initiate the transaction. In the preceding figure, Microservice 2 and Microservice 3 are the transaction participant services that are involved in the transaction.

1.3 How MicroTx Works

Here's a typical transaction workflow when you use MicroTx. The following figure shows how MicroTx communicates with your application microservices to handle transactions.



1. Application developers use functions present in the MicroTx library with their application code.
2. When a microservice or client initiates a transaction, it calls functions in the MicroTx library to start a distributed transaction.
3. MicroTx library includes headers that enable the participant services to automatically enlist in the transaction.
4. After all the tasks associated the original request made by the initiator service are complete, the initiator service requests the transaction coordinator to either commit or roll back all the changes.
5. The transaction coordinator sends a call to each participant service to either commit or roll back the changes made by the participants as part of the distributed transaction.

1.4 Use MicroTx Library with Application Code

To use MicroTx to manage the transactions in your application, you need to make a few changes to your existing application code to integrate the functionality provided by the MicroTx libraries.

Let's use a sample Java application to understand the changes that you, as an application developer need to make. The sample application is a banking teller application which transfers an amount from one department to another. The sample XA application code files are available in the `xa` folder in the `microtx-samples` GitHub repository. The MicroTx library is already integrated with the sample application code.

The MicroTx library for Java performs the following functions:

- Enlists the participant service with the Transaction Coordinator in the transaction.
- Injects an `XADataSource` object for the participant application code to use through dependency injection, and then calls `start()` on the associated `XAResource`. Participant microservices, those microservices called in the context of an XA transaction, must use an XA-compliant data source. In Java this means using an `XADataSource` object.
  The MicroTx libraries automatically inject the configured data source into the participant services, so the application developer must add the `@Inject` or `@Context` annotation to the application code. The application code runs the DML using this connection.
- Calls the resource managers to perform operations.

This highlighted code lines, in bold, in the following code snippet of the sample XA application, in Java, show the changes or additions that are typically made.

```
package com.oracle.mtm.sample.data;
...
/**
 * Include the Transaction Manager for Microservices library files.
 */

import com.oracle.mtm.sample.Configuration;
import com.oracle.mtm.sample.entity.Account;
import oracle.trm.jta.common.TrmSQLConnection;

/**
 * Service that connects to the accounts database and provides methods to
interact with the accounts
 */
@RequestScoped
public class AccountsService implements IAccountsService {

    /**
     * The database connection injected by the Transaction Manager for
Microservices library.
     * Use this connection object to execute SQLs (DMLs) within the
application code.
     */
    @Inject
    @TrmSQLConnection
    private Connection connection;

    @Inject
    private Configuration config;
...
    @Override
```

```
    public boolean withdraw(String accountId, double amount) throws
SQLException {
        String query = "UPDATE accounts SET amount=amount-? where
account_id=?";
        /**
         * Use the connection object that the Transaction Manager for
Microservices library
         * injects in the application code.
         */
       PreparedStatement statement = connection.prepareStatement(query);
        statement.setDouble(1, amount);
        statement.setString(2, accountId);
        return statement.executeUpdate() > 0;
    }
...

}
```

The sample code has been truncated with … to improve readability. To view the entire sample code, see the sample XA application code files that are available in the installation bundle in the GitHub repository.

2
About the `runme.sh` Script

Use the `runme.sh` script file to install Transaction Manager for Microservices (MicroTx) in runtime platform, and then quickly run sample applications.

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

- Considerations
  Consider the points discussed in this section before you use the `runme.sh` script file to run sample applications in your test environment.

2.1 Considerations

Consider the points discussed in this section before you use the `runme.sh` script file to run sample applications in your test environment.

Required Environment Details

The `runme.sh` script file is a bash script, so you can run it in environments that support Bash shell.

To run the script and sample applications, ensure that at least 6114 MB of memory is available on your local machine.

Supported Languages

Use the `runme.sh` script file to run sample applications coded in the following languages:

- TypeScript or JavaScript for Node.js
- Java

Supported Platforms

Use the `runme.sh` script file to run sample applications in the following development environments:

- Docker
- Minikube
- Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)

Supported Resource Manager

Before you run the XA sample application, set up an Oracle Database as resource manager for the transaction participant services. In XA transactions, MicroTx client libraries need to access the resource manager's client libraries.

Supported Identity Providers

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

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

This guide provides information about creating an access token using Oracle IAM and Oracle IDCS. See Use Oracle Identity Providers.

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.

3
Prerequisites

Before you begin, download the Transaction Manager for Microservices (MicroTx) installation bundle to your local system.

If you plan to run XA sample application, set up the resource managers for the transaction participant services.

- Download the Installation Bundle
  Perform the following steps to download the MicroTx installation bundle to your local system:
- Clone the Sample Code Files
  Clone the sample application files from the git repo to your local system.
- Set Up XA-Compliant Resource Managers
  If you want to run sample applications that use the XA transaction protocol, then you must set up XA-compliant resource managers and create tables with sample values. Skip this step if you don't want to run sample applications that use XA.

3.1 Download the Installation Bundle

Perform the following steps to download the MicroTx installation bundle to your local system:

1. Download the MicroTx installation bundle (.zip file) from https://www.oracle.com/database/transaction-manager-for-microservices/.
2. Unzip the MicroTx installation bundle.

   ```
   unzip otmm-<version>.zip
   ```

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

   ```
   ls -lR otmm-<version>
   ```

The following folders are available.

- `lib`: This folder contains the MicroTx library files. You must use these library files in your application code to use MicroTx to manage transactions amongst your application microservices.
- `otmm`: This folder contains the MicroTx image and `YAML` files which you can use to install and configure MicroTx.
- `console`: This folder contains the image file for the MicroTx console.

3.2 Clone the Sample Code Files

Clone the sample application files from the git repo to your local system.

1. Clone the git repo from https://github.com/oracle-samples/microtx-samples.
2. View the available files and folders.

This contains the source code for sample applications for different transaction protocols: XA, Saga, and TCC.

3.3 Set Up XA-Compliant Resource Managers

If you want to run sample applications that use the XA transaction protocol, then you must set up XA-compliant resource managers and create tables with sample values. Skip this step if you don't want to run sample applications that use XA.

You can use any Oracle Database. For example, Autonomous Transaction Processing (ATP) Database instances in Oracle Cloud, an Oracle Database running inside a Kubernetes cluster, or an on-premises database. Ensure that application, when it is deployed, can access the database.

Only if you use an Autonomous Database instance, perform the following steps to get 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 location of the wallet file as you will have to provide this later.

Create database and tables with sample values

Run SQL script files to test the sample XA applications, create database and tables with sample values for both the department applications. The SQL script files are available in the `microtx-samples` GitHub repository. Run the SQL script using a client tool with which you connect to the database. To run the scripts, you must either be a database administrator or have the required privileges to create and run DMLs. You'll also need to provide database credentials to establish a connection with the database and run the SQL script.

To use the SQL script to create a database, a table, and populate it with sample values:

1. Open the `department.sql` file in any code editor and replace `<password>` with the database administrator's password. This file is located in the `xa/java/department-helidon` folder in the `microtx-samples` GitHub repository.
2. Run the `department.sql` file by connecting to Oracle Database using SQL developer or SQL plus.
   This creates a database with the name `department_helidon` and a table with the name `accounts`. It also populates the `accounts` table with sample values.
3. Open the `department.sql` file in any code editor and replace `<password>` with the database administrator's password. This file is located in the `xa/java/department-spring` folder in the `microtx-samples` GitHub repository.
4. Run the `department.sql` file by connecting to Oracle Database using SQL developer or SQL plus.

This creates a database with the name `department_spring` and a table with the name `accounts`. It also populates the `accounts` table with sample values as provided in the following table.

Account_ID	Amount
account1	1000
account2	2000
account3	3000
account4	4000
account5	5000

4

Quick Start with Docker

You can install Docker on your local machine, and then use the `runme.sh` script file to install Transaction Manager for Microservices (MicroTx) and run sample applications.

After MicroTx is installed, the transaction coordinator runs in Docker and the sample apps run in your local machine as operating system processes.

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

The `runme.sh` script runs the microservices in a non-secure mode.

> **Note:**
>
> As you run the `runme.sh` script only in test or development environments, you do not need to provide any authentication details.

The `runme.sh` script installs MicroTx, builds the Docker images, and then installs the sample application. You can also run the sample applications without automating these steps using the `runme.sh` script file. See Deploy Sample Applications in *Transaction Manager for Microservices Developer Guide*.

- Set Up the Required Software
  Before you begin, ensure that the following software is available on your local machine where you want to run Docker.
- Run XA Sample Applications
  Run the XA sample application to transfer an amount from one department to another and to understand how you can use MicroTx to coordinate XA transactions. The MicroTx library files are already integrated with the sample application code.
- Run Saga Sample Applications
  Run the Saga sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.
- Run TCC Sample Applications
  Run the TCC sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

4.1 Set Up the Required Software

Before you begin, ensure that the following software is available on your local machine where you want to run Docker.

- Docker version 20.10.x. See https://docs.docker.com/engine/install/.
- npm version 7.x or later. See https://nodejs.org/en/download/.
- Maven version 3.6 or later. See https://maven.apache.org/download.cgi.
- Java JDK version 11 or later. See https://www.oracle.com/java/technologies/downloads/.
- cURL. See https://curl.se/download.html.
- jq. See https://stedolan.github.io/jq/download/.

Additionally, ensure that ports 8080 to 8083 are free as MicroTx and sample applications use these ports for communication.

4.2 Run XA Sample Applications

Run the XA sample application to transfer an amount from one department to another and to understand how you can use MicroTx to coordinate XA transactions. The MicroTx library files are already integrated with the sample application code.

The sample application demonstrates how you can develop microservices that participate in XA transactions while using MicroTx to coordinate the transactions. When you run the Teller application, it withdraws money from one department and deposits it to another department by creating an XA transaction. Within the XA transaction, all actions such as withdraw and deposit either succeed, or they all are rolled back in case of a failure of any one or more actions. For details about the sample XA application, see About the Sample XA Application in *Transaction Manager for Microservices Developer Guide*.

Ensure that you have completed the prerequisites, such as setting up the resource manager. See Prerequisites.

To run the sample XA application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. If you ran the `runme.sh` script file earlier and you want to reuse the environment details that you had provided earlier, type **Yes**.

   If you use this option, then you don't need to provide details for the applications to connect with its resource manager.

3. Type **1** to run sample applications on Docker in your local machine.

   The script loads the Docker image of MicroTx, and then installs MicroTx. After completing the installation, the script runs the transaction coordinator and displays the URL of the transaction coordinator.

4. Type **1** to run sample applications and MicroTx in non-secure mode in your test environment.

You may have to wait for a few seconds while the script loads the images of the sample applications, and then installs it.

When the script displays the full path that you must set for the `KUBECONFIG` environment variable, you can perform the next step.

5. Type **1** to run the sample application that uses the XA transaction protocol.
6. Provide details for the Department One application to connect with its resource manager.
   a. If you use Oracle Autonomous Database as the resource manager, enter the path to the Oracle Autonomous Database wallet that you have previously downloaded and extracted to your local machine. For example, *installation_directory*`/xa/java/department-helidon/Database_Wallet`.

      If you are using another Oracle Database, press Enter as you don't need to provide details of the wallet.

   b. Enter 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>
        ```

        For example:

        ```
        jdbc:oracle:thin:@123.213.85.123:1521/
        CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com
        ```

      - If you are using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String in *Using Oracle Blockchain Platform*.
      - If you are using Oracle Autonomous Transaction Processing, use the following format to enter the connection string:

        ```
        jdbc:oracle:thin:@tcps://<host>:<port>/<service_name>?
        wallet_location=<wallet_dir>
        ```

        You can find the required details, such as host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the wallet.

        For example:

        ```
        jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:7777/
        unique_connection_string_low.adb.oraclecloud.com?
        wallet_location=Database_Wallet
        ```

   c. Enter the user name to access the Oracle Database, such as `SYS`.
   d. Enter the password for the Oracle Database user.

   The script installs and runs the Department One application.
7. Provide details for the Department Two application to connect with its resource manager.
   a. If you use Oracle Autonomous Database as the resource manager, enter the path to the Oracle Autonomous Database wallet that you have previously downloaded and

extracted to your local machine. For example, *installation_directory*/xa/java/department-helidon/Database_Wallet.

If you are using another Oracle Database, press Enter as you don't need to provide details of the wallet.

b. Enter 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>
  ```

  For example:

  ```
  jdbc:oracle:thin:@123.213.85.123:1521/
  CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com
  ```

- If you are using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String in *Using Oracle Blockchain Platform*.
- If you are using Oracle Autonomous Transaction Processing, use the following format to enter the connection string:

  ```
  jdbc:oracle:thin:@tcps://<host>:<port>/<service_name>?
  wallet_location=<wallet_dir>
  ```

  You can find the required details, such as host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the wallet.

  For example:

  ```
  jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:7777/
  unique_connection_string_low.adb.oraclecloud.com?
  wallet_location=Database_Wallet
  ```

c. Enter the user name to access the Oracle Database, such as `SYS`.

d. Enter the password for the Oracle Database user.

The script installs and runs the Department Two application.

8. Type **1** to run the Teller application to transfer an amount from Department One to Department Two by creating an XA transaction.
9. Enter the account number from which you want to withdraw an amount. The sample table contains the following account numbers: `account1` to `account5`. If you do not enter an account number and press enter, the default value is `account1`.

   The account balance is displayed.
10. Enter the name of the account to which you want to deposit the amount. The sample table contains the following account numbers: `account1` to `account5`. If you do not enter an account number and press enter, the default value is `account2`.

    The account balance is displayed.

11. Enter the amount that you want to transfer. For example, 300. If you do not enter an amount and press enter, the default value is `100`.

    The account balance of both accounts after the transaction are displayed on the screen. You can compare the earlier account balance with the current balance to ensure that the amount has been transferred.

12. Press any key to exit.

13. Type **1** to stop running all the microservices in the sample application and uninstall it.

14. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

4.3 Run Saga Sample Applications

Run the Saga sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

The sample application demonstrates how you can develop microservices that participate in Saga transactions while using MicroTx to coordinate the transactions. When you run the application, it makes a provisional booking by reserving a hotel room and flight ticket. Only when you provide approval to confirm the booking, the booking of the hotel room and flight ticket is confirmed. If you cancel the provisional booking, the hotel room and flight ticket that was blocked is released and the booking is canceled. By default, the hotel and flight service permits only three confirmed bookings. To enable you to test the failure scenario, the services reject any additional booking requests that are made after three confirmed bookings. This leads to the cancellation (compensation) of a provisionally booked hotel or flight within the trip and the trip is not booked. For details about the sample Saga application, see About the Sample Saga Application in *Transaction Manager for Microservices Developer Guide*.

To run the sample Saga application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **1** to run sample applications on Docker in your local machine.

   The script loads the Docker image of MicroTx, and then installs MicroTx. After completing the installation, the script runs the transaction coordinator and displays the URL of the transaction coordinator.

3. Type **1** to run sample applications and MicroTx in non-secure mode in your test environment.

   You may have to wait for a few seconds while the script loads the images of the sample applications, and then installs it.

   When the script displays the full path that you must set for the `KUBECONFIG` environment variable, you can perform the next step.

4. Type **2** to run the sample application that uses the Saga transaction protocol.

   The script installs the sample application.

5. Type **1** to confirm that you want to run the Saga sample application, and then press Enter.

   The sample application provisionally books a hotel room and a flight ticket and displays the details of the provisional booking. In case of any issues, the provisional booking is not made and the status displayed is `Failed Trip Booking`.

6. Confirm or cancel the provisional booking. Type **1** to confirm a successful provisional booking or type **2** to cancel a provisional booking, and then press Enter.

   If you type **1**, your booking is confirmed and information about your confirmed booking is displayed.

7. Press any key to exit.

8. Type **1** to stop running all the microservices in the sample application and uninstall it.

9. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.

4.4 Run TCC Sample Applications

Run the TCC sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

The sample TCC application implements a scenario where the travel agent microservice books a trip, flight booking service books a flight, and the hotel booking microservice books a hotel. The travel agent service accesses both the flight and hotel booking services. When a customer books a flight and a hotel, the booking is reserved until either the customer completes the payment and confirms the booking. In case of any failure, the reserved resources are canceled and the resources are returned back to the inventory. For details about the sample TCC application, see About the Sample TCC Application in *Transaction Manager for Microservices Developer Guide*.

To run the sample TCC application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **1** to run sample applications on Docker in your local machine.

   The script loads the Docker image of MicroTx, and then installs MicroTx. After completing the installation, the script runs the transaction coordinator and displays the URL of the transaction coordinator.

3. Type **1** to run sample applications and MicroTx in non-secure mode in your test environment.

You may have to wait for a few seconds while the script loads the images of the sample applications, and then installs it.

When the script displays the full path that you must set for the `KUBECONFIG` environment variable, you can perform the next step.

4. Type **3** to run the sample application that uses the TCC transaction protocol.
5. Type **1** to run Java applications or type **2** to run Node.js applications.

   The script installs and then runs the three sample microservices: Flight booking, Hotel booking, and Travel agent.
6. Type **y** to confirm that you want to run the TCC sample application, and then press **Enter**.

   The sample application reserves a hotel room and a flight ticket and displays the reservation details.
7. Confirm or cancel the booking. Type **y** to confirm the booking or type **n** to cancel the booking, and then press **Enter**.

   If you type **y**, the booking is confirmed and details about the confirmed booking are displayed.

   If you type **n**, the Travel Agent microservice cancels the reserved resources and returns the resources back to the inventory.
8. Press any key to exit.
9. Type **1** to stop running all the microservices in the sample application and uninstall it.
10. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

5
Quick Start with Minikube

Follow the instructions in this section to configure Minikube, install Transaction Manager for Microservices (MicroTx), and then run a sample application.

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

The `runme.sh` script runs the microservices in a non-secure mode.

> **Note:**
>
> As you run the `runme.sh` script only in test or development environments, you do not need to provide any authentication details.

The `runme.sh` script installs MicroTx, builds the Docker images, and then installs the sample application. You can also run the sample applications without automating these steps using the `runme.sh` script file. See Deploy Sample Applications in *Transaction Manager for Microservices Developer Guide*.

- Set Up the Required Software
  Before you begin, ensure that the following software is available on your local system.
- Run XA Sample Applications
  Run the XA sample application to transfer an amount from one department to another and to understand how you can use MicroTx to coordinate XA transactions. The MicroTx library files are already integrated with the sample application code.
- Run Saga Sample Applications
  Run the Saga sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.
- Run TCC Sample Applications
  Run the TCC sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

5.1 Set Up the Required Software

Before you begin, ensure that the following software is available on your local system.

1. Install Docker version 20.10.x. See https://docs.docker.com/engine/install/.
2. After installing Docker, install Minikube. See https://minikube.sigs.k8s.io/docs/start/.

3. Run the following command to download Istio.

   ```
   curl -sL https://istio.io/downloadIstioctl | sh -
   ```

   When you run the `runme.sh` script, it installs Istio.

4. Add the istioctl client tool to the `PATH` environment variable of your local system. The following example specifies the a sample value. Provide the path based on your environment.

   ```
   export PATH=$HOME/.istioctl/bin:$PATH
   ```

5. Install the following required software.
   - npm version 7.x or later. See https://nodejs.org/en/download/.
   - Maven version 3.6 or later. See https://maven.apache.org/download.cgi.
   - Java JDK version 11 or later. See https://www.oracle.com/java/technologies/downloads/.
   - cURL. See https://curl.se/download.html.
   - jq. See https://stedolan.github.io/jq/download/.
6. Install and configure Kubernetes command-line interface (Kubectl), 1.21.x or later versions, to work with your Kubernetes cluster. See https://kubernetes.io/docs/tasks/tools/.

   Use `Kubectl` to create and manage your deployments. `Kubectl` uses the Kubernetes APIs to interact with the cluster.
7. 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.
8. Ensure that Java Development Kit (JDK) is installed on your local system, and then run the following commands in the Bash shell to set the following environment variables.

   ```
   export JAVA_HOME=jdk-install-dir
   export PATH=$JAVA_HOME/bin:$PATH
   ```

5.2 Run XA Sample Applications

Run the XA sample application to transfer an amount from one department to another and to understand how you can use MicroTx to coordinate XA transactions. The MicroTx library files are already integrated with the sample application code.

The sample application demonstrates how you can develop microservices that participate in XA transactions while using MicroTx to coordinate the transactions. When you run the Teller application, it withdraws money from one department and deposits it to another department by creating an XA transaction. Within the XA transaction, all actions such as withdraw and deposit either succeed, or they all are rolled back in case of a failure of any one or more actions. For details about the sample XA application, see About the Sample XA Application in *Transaction Manager for Microservices Developer Guide*.

Before you begin, complete the following tasks:

- Complete the prerequisites and set up the required software. See Prerequisites.
- Set up resource managers for the two transaction participant services to run XA applications. Set up Oracle Database as the resource manager. See Set Up XA-Compliant Resource Managers.
- Note down the details required to connect to the database, such as credentials and connection string.

To run the sample XA application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **2** to run sample applications in the Minikube environment.

   The script sets up Minikube, configures Istio service mesh, loads the Docker image of MicroTx, and then installs MicroTx. After completing the installation, the script runs the transaction coordinator and provides the URL of the transaction coordinator.

3. Type **1** to run sample applications and MicroTx in non-secure mode in your test environment.

   You may have to wait for a few seconds while the script loads the images of the sample applications, and then installs it.

   When the script displays the full path that you must set for the `KUBECONFIG` environment variable, you can perform the next step.

4. Copy the command returned by the script, and then run the command in a new terminal. Set `KUBECONFIG` to configure Kubectl to run commands on Minikube.

   The script displays the actual command which you can copy and run as it is in a new terminal. The following is a sample command. The actual command that the script returns depends on the value of the `$HOME` variable.

   ```
   export KUBECONFIG=$HOME/.kube/minikube
   ```

5. Run the following command in a new terminal to start a tunnel between Minikube and the Istio ingress gateway.

   ```
   $ minikube tunnel
   ```

   If prompted, enter the password to access your local system, and then press any key to continue running the script.

6. Type **1** to run the sample application that uses the XA transaction protocol.
7. Provide details for the Department One application to connect with its resource manager.
   a. If you use Oracle Autonomous Database as the resource manager, enter the path to the Oracle Autonomous Database wallet that you have previously downloaded and extracted to your local machine. For example, *installation_directory*`/xa/java/department-helidon/Database_Wallet`.

      If you are using another Oracle Database, press Enter as you don't need to provide details of the wallet.

   b. Enter 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>
  ```

  For example:

  ```
  jdbc:oracle:thin:@123.213.85.123:1521/
  CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com
  ```

- If you are using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String in *Using Oracle Blockchain Platform*.
- If you are using Oracle Autonomous Transaction Processing, use the following format to enter the connection string:

  ```
  jdbc:oracle:thin:@tcps://<host>:<port>/<service_name>?
  wallet_location=<wallet_dir>
  ```

  You can find the required details, such as host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the wallet.

  For example:

  ```
  jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:7777/
  unique_connection_string_low.adb.oraclecloud.com?
  wallet_location=Database_Wallet
  ```

c. Enter the user name to access the Oracle Database, such as `SYS`.

d. Enter the password for the Oracle Database user.

The script installs and runs the Department One application.

8. Provide details for the Department Two application to connect with its resource manager.

   a. If you use Oracle Autonomous Database as the resource manager, enter the path to the Oracle Autonomous Database wallet that you have previously downloaded and extracted to your local machine. For example, *`installation_directory`*`/xa/java/department-helidon/Database_Wallet`.

      If you are using another Oracle Database, press Enter as you don't need to provide details of the wallet.

   b. Enter 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>
        ```

For example:

```
jdbc:oracle:thin:@123.213.85.123:1521/
CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com
```

- If you are using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String in *Using Oracle Blockchain Platform*.
- If you are using Oracle Autonomous Transaction Processing, use the following format to enter the connection string:

```
jdbc:oracle:thin:@tcps://<host>:<port>/<service_name>?
wallet_location=<wallet_dir>
```

You can find the required details, such as host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the wallet.

For example:

```
jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:7777/
unique_connection_string_low.adb.oraclecloud.com?
wallet_location=Database_Wallet
```

c. Enter the user name to access the Oracle Database, such as `SYS`.

d. Enter the password for the Oracle Database user.

The script installs and runs the Department Two application.

9. Type **1** to run the Teller application to transfer an amount from Department One to Department Two by creating an XA transaction.
10. Enter the account number from which you want to withdraw an amount. The sample table contains the following account numbers: `account1` to `account5`. If you do not enter an account number and press enter, the default value is `account1`.

    The account balance is displayed.
11. Enter the name of the account to which you want to deposit the amount. The sample table contains the following account numbers: `account1` to `account5`. If you do not enter an account number and press enter, the default value is `account2`.

    The account balance is displayed.
12. Enter the amount that you want to transfer. For example, 300. If you do not enter an amount and press enter, the default value is `100`.

    The account balance of both accounts after the transaction are displayed on the screen. You can compare the earlier account balance with the current balance to ensure that the amount has been transferred.
13. Press any key to exit.
14. Type **1** to stop running all the microservices in the sample application and uninstall it.
15. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.
16. Type **1** to uninstall Istio. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

5.3 Run Saga Sample Applications

Run the Saga sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

The sample application demonstrates how you can develop microservices that participate in Saga transactions while using MicroTx to coordinate the transactions. When you run the application, it makes a provisional booking by reserving a hotel room and flight ticket. Only when you provide approval to confirm the booking, the booking of the hotel room and flight ticket is confirmed. If you cancel the provisional booking, the hotel room and flight ticket that was blocked is released and the booking is canceled. By default, the hotel and flight service permits only three confirmed bookings. To enable you to test the failure scenario, the services reject any additional booking requests that are made after three confirmed bookings. This leads to the cancellation (compensation) of a provisionally booked hotel or flight within the trip and the trip is not booked. For details about the sample Saga application, see About the Sample Saga Application in *Transaction Manager for Microservices Developer Guide*.

To run the sample Saga application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **2** to run sample applications in the Minikube environment.

   The script sets up Minikube, configures Istio service mesh, loads the Docker image of MicroTx, and then installs MicroTx. After completing the installation, the script runs the transaction coordinator and provides the URL of the transaction coordinator.

3. Type **1** to run sample applications and MicroTx in non-secure mode in your test environment.

   You may have to wait for a few seconds while the script loads the images of the sample applications, and then installs it.

   When the script displays the full path that you must set for the `KUBECONFIG` environment variable, you can perform the next step.

4. Copy the command returned by the script, and then run the command in a new terminal. Set `KUBECONFIG` to configure Kubectl to run commands on Minikube.

The script displays the actual command which you can copy and run as it is in a new terminal. The following is a sample command. The actual command that the script returns depends on the value of the `$HOME` variable.

```
export KUBECONFIG=$HOME/.kube/minikube
```

5. Run the following command in a new terminal to start a tunnel between Minikube and the Istio ingress gateway.

```
$ minikube tunnel
```

If prompted, enter the password to access your local system, and then press any key to continue running the script.

6. Type **2** to run the sample application that uses the Saga transaction protocol.

The script installs the sample application.

7. Type **1** to confirm that you want to run the Saga sample application, and then press Enter.

The sample application provisionally books a hotel room and a flight ticket and displays the details of the provisional booking. In case of any issues, the provisional booking is not made and the status displayed is `Failed Trip Booking`.

8. Confirm or cancel the provisional booking. Type **1** to confirm a successful provisional booking or type **2** to cancel a provisional booking, and then press Enter.

If you type **1**, your booking is confirmed and information about your confirmed booking is displayed.

9. Press any key to exit.
10. Type **1** to stop running all the microservices in the sample application and uninstall it.
11. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.
12. Type **1** to uninstall Istio. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

5.4 Run TCC Sample Applications

Run the TCC sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

The sample TCC application implements a scenario where the travel agent microservice books a trip, flight booking service books a flight, and the hotel booking microservice books a hotel. The travel agent service accesses both the flight and hotel booking services. When a customer books a flight and a hotel, the booking is reserved until either the customer completes the payment and confirms the booking. In case of any failure, the reserved resources are canceled and the resources are returned back to the inventory. For details about the sample TCC application, see About the Sample TCC Application in *Transaction Manager for Microservices Developer Guide*.

To run the sample TCC application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **2** to run sample applications in the Minikube environment.

   The script sets up Minikube, configures Istio service mesh, loads the Docker image of MicroTx, and then installs MicroTx. After completing the installation, the script runs the transaction coordinator and provides the URL of the transaction coordinator.

3. Type **1** to run sample applications and MicroTx in non-secure mode in your test environment.

   You may have to wait for a few seconds while the script loads the images of the sample applications, and then installs it.

   When the script displays the full path that you must set for the `KUBECONFIG` environment variable, you can perform the next step.

4. Copy the command returned by the script, and then run the command in a new terminal. Set `KUBECONFIG` to configure Kubectl to run commands on Minikube.

   The script displays the actual command which you can copy and run as it is in a new terminal. The following is a sample command. The actual command that the script returns depends on the value of the `$HOME` variable.

   ```
   export KUBECONFIG=$HOME/.kube/minikube
   ```

5. Run the following command in a new terminal to start a tunnel between Minikube and the Istio ingress gateway.

   ```
   $ minikube tunnel
   ```

   If prompted, enter the password to access your local system, and then press any key to continue running the script.

6. Type **3** to run the sample application that uses the TCC transaction protocol.
7. Type **1** to run Java applications or type **2** to run Node.js applications.

The script installs and then runs the three sample microservices: Flight booking, Hotel booking, and Travel agent.

8. Type **y** to confirm that you want to run the TCC sample application, and then press **Enter**.

   The sample application reserves a hotel room and a flight ticket and displays the reservation details.

9. Confirm or cancel the booking. Type **y** to confirm the booking or type **n** to cancel the booking, and then press **Enter**.

   If you type **y**, the booking is confirmed and details about the confirmed booking are displayed.

   If you type **n**, the Travel Agent microservice cancels the reserved resources and returns the resources back to the inventory.

10. Press any key to exit.
11. Type **1** to stop running all the microservices in the sample application and uninstall it.
12. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.
13. Type **1** to uninstall Istio. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

6
Quick Start with OKE

Follow the instructions in this section to install Transaction Manager for Microservices (MicroTx) in Oracle Container Engine for Kubernetes (OKE) and run a sample application.

The script deploys the sample applications on a single node in the Kubernetes cluster on which you have deployed MicroTx.

In the test environment, create at least one node in the Kubernetes cluster to host MicroTx. MicroTx supports Kubernetes 1.21.x or later versions.

The `runme.sh` script installs MicroTx, builds the Docker images, and then installs the sample application. You can also run the sample applications without automating these steps using the `runme.sh` script file. See Deploy Sample Applications in *Transaction Manager for Microservices Developer Guide*.

- Set Up the Required Software
  You must complete the following tasks before you begin running the sample applications in OKE.
- Run XA Sample Applications
  Run the XA sample application to transfer an amount from one department to another and to understand how you can use MicroTx to coordinate XA transactions. The MicroTx library files are already integrated with the sample application code.
- Run Saga Sample Applications
  Run the Saga sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.
- Run TCC Sample Applications
  Run the TCC sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

6.1 Set Up the Required Software

You must complete the following tasks before you begin running the sample applications in OKE.

1. Run the following command to download Istio.

   ```
   curl -sL https://istio.io/downloadIstioctl | sh -
   ```

   When you run the `runme.sh` script, it installs Istio.
2. Add the istioctl client tool to the `PATH` environment variable of your local system. The following example specifies the a sample value. Provide the path based on your environment.

   ```
   export PATH=$HOME/.istioctl/bin:$PATH
   ```

3. Install the following required software.
   - npm version 7.x or later. See https://nodejs.org/en/download/.
   - Maven version 3.6 or later. See https://maven.apache.org/download.cgi.
   - Java JDK version 11 or later. See https://www.oracle.com/java/technologies/downloads/.
   - cURL. See https://curl.se/download.html.
   - jq. See https://stedolan.github.io/jq/download/.
   - OpenSSL. See https://www.openssl.org/.
4. Install and configure Kubernetes command-line interface (Kubectl), 1.21.x or later versions, to work with your Kubernetes cluster. See https://kubernetes.io/docs/tasks/tools/.
   Use `Kubectl` to create and manage your deployments. `Kubectl` uses the Kubernetes APIs to interact with the cluster.
5. Install the latest version of Helm 3.x on your local machine. See https://helm.sh/docs/intro/install/.
6. Install Oracle Cloud Infrastructure (OCI) CLI. Ensure that OCI CLI is configured to connect to the Kubernetes cluster. See https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/cliinstall.htm.
7. Create a Kubernetes Cluster with OKE.
   a. Log in to the Oracle OCI cloud console. See https://www.oracle.com/in/cloud/sign-in.html.
   b. Create a Kubernetes cluster in the OKE environment. See Quick Create Workflow to Create a Cluster in *Oracle Cloud Infrastructure documentation*.
   c. Set up local access to the Kubernetes cluster that you have created so that you can access your OKE cluster environment from your local machine. See Setting Up Local Access to Clusters in *Oracle Cloud Infrastructure documentation*.
8. Create an access token. To create an access token using Oracle IAM and Oracle IDCS, see Use Oracle Identity Providers and Create an Access Token. 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.
9. Set up resource managers for the two transaction participant services to run the sample XA application. Set up Oracle Database as the resource manager. See Set Up XA-Compliant Resource Managers.
10. Ensure that Java Development Kit (JDK) is installed on your local system, and then run the following commands in the Bash shell to set the following environment variables.

    ```
    export JAVA_HOME=jdk-install-dir
    export PATH=$JAVA_HOME/bin:$PATH
    ```

    The JDK contains keytool, a utility to create and manage certificates. The `runme.sh` script runs the keytool utility to generate a certificate to enable TLS to access MicroTx.
11. Ensure that you have sudo privileges to run commands in the Bash shell.

- Use Oracle Identity Providers
  You can use Oracle Identity Cloud Service (IDCS) or Oracle IAM as an identity provider to manage access to your application.

- Create an Access Token
  This topic provides details to create an access token when you use Oracle IDCS or Oracle IAM as the identity provider.

6.1.1 Use Oracle Identity Providers

You can use Oracle Identity Cloud Service (IDCS) or 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.

Oracle Cloud Infrastructure previously used Oracle IDCS as the identity provider. Now, Oracle Cloud Infrastructure uses Oracle IAM as the identity provider.

To identify if your Oracle Cloud Infrastructure tenancy uses Oracle IDCS or Oracle IAM:

1. Log in to the Oracle Cloud Infrastructure console.
2. Open the navigation menu and click **Identity & Security**.
   - Under **Identity**, if you see **Users and Groups**, your tenancy has not been migrated to Oracle IAM. Your tenancy uses Oracle IDCS.
   - Under **Identity**, if you see **Domains**, your tenancy has been migrated to Oracle IAM.

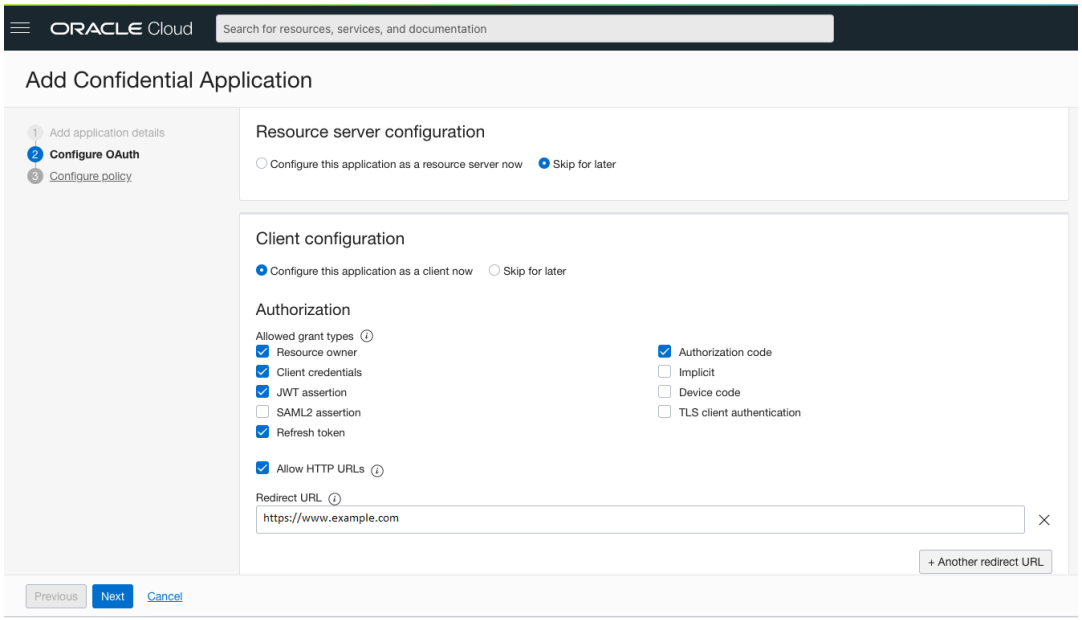
Based on whether your tenancy uses Oracle IDCS or Oracle IAM, you can use the relevant information to create a confidential application and activate it.

- Use Oracle IAM as Identity Provider
  You can use Oracle IAM as identity provider to manage access to your application.
- Use Oracle IDCS as Identity Provider
  You can use Oracle IDCS as identity provider to manage access to your application.

6.1.1.1 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. See Adding a Confidential Application in *Oracle Cloud Infrastructure documentation*.



While adding a confidential application, perform the following tasks:

a. On the **Configure OAuth** pane, under **Resource server configuration**, click **Skip for later**.

b. On the **Configure OAuth** pane, click **Configure this application as a client now**, and then select the following options:

   - **Resource owner**
   - **Client credentials**
   - **JWT assertion**
   - **Refresh token**
   - **Authorization code**
   - **Allow HTTP URLs**: Optional. Select this option only if you want to add a redirect URL without HTTPS. If you don't select this option, only HTTPS URLs are supported.
   - **Add Redirect URL**: Enter the application URL where the user is redirected after authentication.

c. Skip web tier policy configuration.

The application is created.

2. Click **Activate** to activate the application.
3. Under **General Information**, note down the values for **Client ID** and **Client secret**.
4. Click **Users**, and then assign users to the application. See Assigning Users to Custom Applications in *Oracle Cloud Infrastructure documentation*.
5. Open the navigation menu and click **Identity & Security**. Under **Identity**, click **Domains**. Select the identity domain you want to work in.

   The **Domain information** tab of the identity domain is displayed.
6. From this tab, copy the **Domain URL**. For example, `https://idcs-a83e4de370ea4db1b8c703a0b742ce74.identity.oraclecloud.com`. You'll need this information while running the Discovery URL.

7. Enable client access for the signing certificate. By default, access is restricted to only the signed-in users. To access this certificate in Docker, Kubernetes, and Istio, you must enable client access.
   a. Select the identity domain you want to work in and click **Settings** and then **Domain settings**.
   b. Turn on the switch under **Access Signing Certificate** to enable clients to access the tenant signing certificate without logging in to IAM.
   c. Click **Save** to save the default settings.
   d. To check if you can access the certificate without logging in, type the following link in a new browser window.

      ```
      https://<yourtenant>.identity.oraclecloud.com/admin/v1/SigningCert/jwk
      ```

      Where, `<yourtenant>` are the details of your Oracle Cloud Infrastructure tenancy.

      You should be able to open the link without logging in to Oracle Cloud Infrastructure.

6.1.1.2 Use Oracle IDCS as Identity Provider

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

1. In the Oracle Cloud Infrastructure console, add your application as a confidential application. See Adding a Confidential Application in *Administering Oracle Identity Cloud Service*.

   While adding a confidential application, perform the following tasks:
   a. On the **Add Confidential Application** wizard's **Client** page, click **Configure this application as a client now**.
   b. In the **Authorization** section, select the following options:
      - **Resource owner**
      - **Client credentials**
      - **JWT assertion**
      - **Refresh token**
      - **Authorization code**
      - **Redirect URL**: Enter the application URL where the user is redirected after authentication.
   c. Skip the next steps. Use the default selections, and then click **Finish**. The application has been added in a deactivated state.
   d. Record the **Client ID** and **Client Secret** that appear in the **Application Added** dialog box. You will need to provide this information later.
   e. Click **Close**.
   The new application's details page is displayed.
   f. At the top of the page, to the right of the application name, click **Activate** to activate the application.
   g. In the **Activate Application?** dialog box, click **Activate Application**.
2. Click **Users**, and then assign users to the application. See Assign Applications to the User Account in *Administering Oracle Identity Cloud Service*.

3. Enable client access for the signing certificate. By default, access is restricted to only the signed-in users. To allow clients to access the tenant signing certificate and the SAML metadata without logging in to Oracle Identity Cloud Service, perform the following steps.
   a. In the Identity Cloud Service console, expand the **Navigation Drawer**, click **Settings**, and then click **Default Settings**.
   b. Turn on the **Access Signing Certificate** option.
   c. Click **Save** to save the default settings.

6.1.2 Create an Access Token

This topic provides details to create an access token when you use Oracle IDCS or 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 authentication 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. Launch a terminal and enter the following command.

   ```
   echo -n "clientid:clientsecret" | base64 -w 0
   ```

   Where, replace `clientid:clientsecret` with the values in your environment. `-w 0` is added for Linux to the command to remove line breaks.

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

   Based on your environment, you can use any base64 client to encode the `clientid:clientsecret`.

2. Copy the value that is returned. You'll have to provide this value every time you want to create an authentication token.
3. Get an authentication token using the base64-encoded value, as shown in the following cURL command example. Run one of the following commands based on whether you want to generate only the access token or the refresh token as well.
   - The following command creates the access token.

     Command syntax

     ```
     curl -i
     -H "Authorization:Basic {base64 encoded value of clientid:clientsecret}"
     -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
     --request POST https://domain-url/oauth2/v1/token
     -d
     "grant_type=password&username=username&password&scope=urn:opc:idm:__mysc
     opes__"
     ```

Example

```
curl -i
-H "Authorization:Basic ZWY1N2E1OWUyZjY..."
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://idcs-
a83e4de370ea4db1b8c703a0b742ce74.identity.oraclecloud.com/oauth2/v1/
token
-d
"grant_type=password&username=acme@example.com&password&scope=urn:opc:id
m:__myscopes__"
```

- The following command creates the access token and the refresh token.

  Command syntax

  ```
  curl -i
  -H "Authorization:Basic {base64 encoded value of clientid:clientsecret}"
  -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
  --request POST https://domain-url/oauth2/v1/token
  -d
  "grant_type=password&scope=urn:opc:idm:__myscopes__+offline_access&usern
  ame=username&password=password"
  ```

  Example

  ```
  curl -i
  -H "Authorization:Basic ZWY1N2E1OWUyZjY..."
  -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
  --request POST https://idcs-
  a83e4de370ea4db1b8c703a0b742ce74.identity.oraclecloud.com/oauth2/v1/
  token
  -d
  "grant_type=password&scope=urn:opc:idm:__myscopes__+offline_access&usern
  ame=acme@example.com&password=password"
  ```

4. Copy the `access_token` value from the response as shown in the following example.

   Example output

   ```
   {
     "access_token":"eyJ4Lm...",
     "expires_in": 300,
     "refresh_expires_in": 1800,
     "refresh_token": "ey5Gkr...",
     "token_type": "Bearer",
     "not-before-policy": 0,
     "session_state": "c966d...",
     "scope": "profile email"
   }
   ```

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

   Make sure to copy only the actual token, which is the `access_token` and `refresh_token` values between the quotation marks.

5. Store the authentication token and refresh tokens in environment variables, as shown in the following example for a Linux host.

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

6. 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.

6.2 Run XA Sample Applications

Run the XA sample application to transfer an amount from one department to another and to understand how you can use MicroTx to coordinate XA transactions. The MicroTx library files are already integrated with the sample application code.

The sample application demonstrates how you can develop microservices that participate in XA transactions while using MicroTx to coordinate the transactions. When you run the Teller application, it withdraws money from one department and deposits it to another department by creating an XA transaction. Within the XA transaction, all actions such as withdraw and deposit either succeed, or they all are rolled back in case of a failure of any one or more actions. For details about the sample XA application, see About the Sample XA Application in *Transaction Manager for Microservices Developer Guide*.

Before you begin, note down the following information:

- Name of the Oracle Cloud Infrastructure Registry to which you want the script to push the Docker images of the sample applications.
- Details to connect to the database, such as credentials and connection string.
- Complete the prerequisites and set up the required software. See Prerequisites.

To run the sample XA application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

```
cd installation_directory/otmm-<version>
sh runme.sh
```

2. Type **3** to run sample applications in the OKE environment.

   The script installs and configures Istio, and then prints the URL to access the Istio ingress gateway.

3. You can deploy observability consoles, such as Kiali and Jaeger, to track and trace distributed transactions in MicroTx. If these consoles are already deployed in the Istio service mesh, the script skips this step.

   a. Type **1** to deploy the consoles.

   b. Type **1** to confirm that you want to deploy Kiali.

Kiali is deployed, along with the prerequisites such as Prometheus, and then the Kiali dashboard is displayed in the default browser.

c. Type **1** to confirm that you want to deploy Jaeger.
Jaeger is deployed, and then the Jaeger dashboard is displayed in the default browser.

Kiali and Jaeger dashboards are displayed in the default browser only if you are running the `runme.sh` script file on a system with graphical user interface. If you are running the `runme.sh` script on a remote system and you have connected through a console or terminal, then you must set up port forwarding to view the dashboards from your system.

4. Provide the following information to enable the script to create a self-signed certificate. A certificate is required to create a secure connection to access MicroTx using TLS.

   a. Enter the password to access the Bash shell with sudo privileges.

   b. Enter the password to access the Java KeyStore.

   The script accesses the KeyStore and generates a self-signed certificate.

   c. Type **1** to confirm that you trust the certificate.

   d. Re-enter the password to access the Java KeyStore.

   The script adds the certificate to the KeyStore.

5. Specify the name of the Oracle Cloud Infrastructure Registry in the format `<region-key>.ocir.io`.

   For example, `iad.ocir.io`. For information about region keys, see Regions and Availability Domains in *Oracle Cloud Infrastructure documentation*.

6. Specify the name of the repository to which you want to push the Docker image of the sample application in the format `<region-key>.ocir.io/<repository_name>`.

   For example, `iad.ocir.io/otmmrepo`. The `runme.sh` script builds and pushes the images to the specified repository. It also prefixes the specified value to the image name as a tag.

7. Enter user name to access the registry.

8. Enter the password to access the registry.

   The script loads the Docker image of MicroTx, and then installs MicroTx.

9. Type **1** to run the sample application that uses the XA transaction protocol.

10. Provide details for the Department One application to connect with its resource manager.

    a. If you use Oracle Autonomous Database as the resource manager, enter the path to the Oracle Autonomous Database wallet that you have previously downloaded and extracted to your local machine. For example, `installation_directory/xa/java/department-helidon/Database_Wallet`.

    If you are using another Oracle Database, press Enter as you don't need to provide details of the wallet.

    b. Enter 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>
      ```

For example:

```
jdbc:oracle:thin:@123.213.85.123:1521/
CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com
```

- If you are using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String in *Using Oracle Blockchain Platform*.
- If you are using Oracle Autonomous Transaction Processing, use the following format to enter the connection string:

```
jdbc:oracle:thin:@tcps://<host>:<port>/<service_name>?
wallet_location=<wallet_dir>
```

You can find the required details, such as host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the wallet.

For example:

```
jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:7777/
unique_connection_string_low.adb.oraclecloud.com?
wallet_location=Database_Wallet
```

c. Enter the user name to access the Oracle Database, such as `SYS`.

d. Enter the password for the Oracle Database user.

The script installs and runs the Department One application.

11. Provide details for the Department Two application to connect with its resource manager.

a. If you use Oracle Autonomous Database as the resource manager, enter the path to the Oracle Autonomous Database wallet that you have previously downloaded and extracted to your local machine. For example, *`installation_directory`*`/xa/java/department-helidon/Database_Wallet`.

If you are using another Oracle Database, press Enter as you don't need to provide details of the wallet.

b. Enter 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>
```

For example:

```
jdbc:oracle:thin:@123.213.85.123:1521/
CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com
```

- If you are using Oracle Database Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String in *Using Oracle Blockchain Platform*.

- If you are using Oracle Autonomous Transaction Processing, use the following format to enter the connection string:

  ```
  jdbc:oracle:thin:@tcps://<host>:<port>/<service_name>?
  wallet_location=<wallet_dir>
  ```

  You can find the required details, such as host, port, and service name in the `tnsnames.ora` file, which is located in folder where you have extracted the wallet.

  For example:

  ```
  jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:7777/
  unique_connection_string_low.adb.oraclecloud.com?
  wallet_location=Database_Wallet
  ```

c. Enter the user name to access the Oracle Database, such as `SYS`.

d. Enter the password for the Oracle Database user.

The script installs and runs the Department Two application.

12. Type **1** to run the Teller application to transfer an amount from Department One to Department Two by creating an XA transaction.
13. Enter the account number from which you want to withdraw an amount. The sample table contains the following account numbers: `account1` to `account5`. If you do not enter an account number and press enter, the default value is `account1`.

    The account balance is displayed.
14. Enter the name of the account to which you want to deposit the amount. The sample table contains the following account numbers: `account1` to `account5`. If you do not enter an account number and press enter, the default value is `account2`.

    The account balance is displayed.
15. Enter the amount that you want to transfer. For example, 300. If you do not enter an amount and press enter, the default value is `100`.

    The account balance of both accounts after the transaction are displayed on the screen. You can compare the earlier account balance with the current balance to ensure that the amount has been transferred.
16. Press any key to exit.
17. Type **1** to stop running all the microservices in the sample application and uninstall it.
18. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.
19. Type **1** to uninstall Istio. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.

- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

6.3 Run Saga Sample Applications

Run the Saga sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

The sample application demonstrates how you can develop microservices that participate in Saga transactions while using MicroTx to coordinate the transactions. When you run the application, it makes a provisional booking by reserving a hotel room and flight ticket. Only when you provide approval to confirm the booking, the booking of the hotel room and flight ticket is confirmed. If you cancel the provisional booking, the hotel room and flight ticket that was blocked is released and the booking is canceled. By default, the hotel and flight service permits only three confirmed bookings. To enable you to test the failure scenario, the services reject any additional booking requests that are made after three confirmed bookings. This leads to the cancellation (compensation) of a provisionally booked hotel or flight within the trip and the trip is not booked. For details about the sample Saga application, see About the Sample Saga Application in *Transaction Manager for Microservices Developer Guide*.

To run the sample Saga application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **3** to run sample applications in the OKE environment.

   The script installs and configures Istio, and then prints the URL to access the Istio ingress gateway.

3. You can deploy observability consoles, such as Kiali and Jaeger, to track and trace distributed transactions in MicroTx. If these consoles are already deployed in the Istio service mesh, the script skips this step.

   a. Type **1** to deploy the consoles.

   b. Type **1** to confirm that you want to deploy Kiali.
   Kiali is deployed, along with the prerequisites such as Prometheus, and then the Kiali dashboard is displayed in the default browser.

   c. Type **1** to confirm that you want to deploy Jaeger.
   Jaeger is deployed, and then the Jaeger dashboard is displayed in the default browser.

   Kiali and Jaeger dashboards are displayed in the default browser only if you are running the `runme.sh` script file on a system with graphical user interface. If you are running the `runme.sh` script on a remote system and you have connected through a console or terminal, then you must set up port forwarding to view the dashboards from your system.

4. Provide the following information to enable the script to create a self-signed certificate. A certificate is required to create a secure connection to access MicroTx using TLS.

   a. Enter the password to access the Bash shell with sudo privileges.

   b. Enter the password to access the Java KeyStore.

   The script accesses the KeyStore and generates a self-signed certificate.

c. Type **1** to confirm that you trust the certificate.

d. Re-enter the password to access the Java KeyStore.

The script adds the certificate to the KeyStore.

5. Specify the name of the Oracle Cloud Infrastructure Registry in the format `<region-key>.ocir.io`.

For example, `iad.ocir.io`. For information about region keys, see Regions and Availability Domains in *Oracle Cloud Infrastructure documentation*.

6. Specify the name of the repository to which you want to push the Docker image of the sample application in the format `<region-key>.ocir.io/<repository_name>`.

For example, `iad.ocir.io/otmmrepo`. The `runme.sh` script builds and pushes the images to the specified repository. It also prefixes the specified value to the image name as a tag.

7. Enter user name to access the registry.

8. Enter the password to access the registry.

The script loads the Docker image of MicroTx, and then installs MicroTx.

9. Type **2** to run the sample application that uses the Saga transaction protocol.

The script installs the sample application.

10. Type **1** to confirm that you want to run the Saga sample application, and then press Enter.

The sample application provisionally books a hotel room and a flight ticket and displays the details of the provisional booking. In case of any issues, the provisional booking is not made and the status displayed is `Failed Trip Booking`.

11. Confirm or cancel the provisional booking. Type **1** to confirm a successful provisional booking or type **2** to cancel a provisional booking, and then press Enter.

If you type **1**, your booking is confirmed and information about your confirmed booking is displayed.

12. Press any key to exit.

13. Type **1** to stop running all the microservices in the sample application and uninstall it.

14. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.

15. Type **1** to uninstall Istio. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.

6.4 Run TCC Sample Applications

Run the TCC sample application to book a trip and understand how you can use MicroTx to coordinate the transactions. The MicroTx library files are already integrated with the sample application code.

The sample TCC application implements a scenario where the travel agent microservice books a trip, flight booking service books a flight, and the hotel booking microservice books a hotel. The travel agent service accesses both the flight and hotel booking services. When a customer books a flight and a hotel, the booking is reserved until either the customer completes the payment and confirms the booking. In case of any failure, the reserved resources are canceled and the resources are returned back to the inventory. For details about the sample TCC application, see About the Sample TCC Application in *Transaction Manager for Microservices Developer Guide*.

To run the sample TCC application using the `runme.sh` script file:

1. Enter the following commands in a bash shell to run the `runme.sh` script file.

   ```
   cd installation_directory/otmm-<version>
   sh runme.sh
   ```

2. Type **3** to run sample applications in the OKE environment.

   The script installs and configures Istio, and then prints the URL to access the Istio ingress gateway.

3. You can deploy observability consoles, such as Kiali and Jaeger, to track and trace distributed transactions in MicroTx. If these consoles are already deployed in the Istio service mesh, the script skips this step.

   a. Type **1** to deploy the consoles.

   b. Type **1** to confirm that you want to deploy Kiali.
   Kiali is deployed, along with the prerequisites such as Prometheus, and then the Kiali dashboard is displayed in the default browser.

   c. Type **1** to confirm that you want to deploy Jaeger.
   Jaeger is deployed, and then the Jaeger dashboard is displayed in the default browser.

   Kiali and Jaeger dashboards are displayed in the default browser only if you are running the `runme.sh` script file on a system with graphical user interface. If you are running the `runme.sh` script on a remote system and you have connected through a console or terminal, then you must set up port forwarding to view the dashboards from your system.

4. Provide the following information to enable the script to create a self-signed certificate. A certificate is required to create a secure connection to access MicroTx using TLS.

   a. Enter the password to access the Bash shell with sudo privileges.

   b. Enter the password to access the Java KeyStore.

   The script accesses the KeyStore and generates a self-signed certificate.

   c. Type **1** to confirm that you trust the certificate.

   d. Re-enter the password to access the Java KeyStore.

   The script adds the certificate to the KeyStore.

5. Specify the name of the Oracle Cloud Infrastructure Registry in the format `<region-key>.ocir.io`.

   For example, `iad.ocir.io`. For information about region keys, see Regions and Availability Domains in *Oracle Cloud Infrastructure documentation*.

6. Specify the name of the repository to which you want to push the Docker image of the sample application in the format `<region-key>.ocir.io/<repository_name>`.

   For example, `iad.ocir.io/otmmrepo`. The `runme.sh` script builds and pushes the images to the specified repository. It also prefixes the specified value to the image name as a tag.

7. Enter user name to access the registry.

8. Enter the password to access the registry.

   The script loads the Docker image of MicroTx, and then installs MicroTx.

9. Type **3** to run the sample application that uses the TCC transaction protocol.

10. Type **1** to run Java applications or type **2** to run Node.js applications.

    The script installs and then runs the three sample microservices: Flight booking, Hotel booking, and Travel agent.

11. Type **y** to confirm that you want to run the TCC sample application, and then press **Enter**.

    The sample application reserves a hotel room and a flight ticket and displays the reservation details.

12. Confirm or cancel the booking. Type **y** to confirm the booking or type **n** to cancel the booking, and then press **Enter**.

    If you type **y**, the booking is confirmed and details about the confirmed booking are displayed.

    If you type **n**, the Travel Agent microservice cancels the reserved resources and returns the resources back to the inventory.

13. Press any key to exit.

14. Type **1** to stop running all the microservices in the sample application and uninstall it.

15. Type **1** to uninstall MicroTx. If you want use the existing installation to run other sample applications, type **2**.

16. Type **1** to uninstall Istio. If you want use the existing installation to run other sample applications, type **2**.

What's next?

- Use the Kiali dashboard to view how the MicroTx handles the flow of requests between the sample microservices.
- Perform distributed tracing using Jaeger to trace the entire transaction. See Perform Distributed Tracing with Jaeger.
- Run another sample application.
- View the source files of the sample application.
- View the log files to find more details about the transactions.
- Create and run your own application using MicroTx.