Oracle®

Transaction Manager for Microservices Image for OCI Marketplace User's Guide

Release 24.2
F89543-03
June 2024

Oracle Transaction Manager for Microservices Image for OCI Marketplace User's Guide, Release 24.2

F89543-03

Primary Author: Sylaja Kannan

Contributing Authors: Tulika Das

Contributors: Brijesh Kumar Deo, Bharath MC, Shivanshu Singh

Contents

1

Transaction Manager for Microservices Images for OCI

Learn about the Oracle Transaction Manager for Microservices (MicroTx) image for Oracle Cloud Infrastructure (OCI) and how to use this image to create MicroTx instances in (OCI).

Topics:

1.1 About Transaction Manager for Microservices Images for OCI

Utilizing the Transaction Manager for Microservices (MicroTx) image for OCI, you can create Transaction Manager for Microservices domain nodes in the Oracle Cloud Infrastructure compute instances, with the entitlement to use MicroTx software.

Transaction Manager for Microservices image for OCI is preinstalled with the Oracle Transaction Manager for Microservices Free 23.4.1. It contains the following directories:

- `$HOME/OTMM/otmm-23.4.1` contains the MicroTx distribution package.
- `$HOME/OTMM/otmm-23.4.1/samples/xa/java/bankapp` contains the source code and other files for the Bank and Stock-Trading application.
- `$HOME/OTMM/otmm-23.4.1/samples/xa/java/bankapp/Helmcharts` contains the Helm Chart with details to deploy the Bank and Stock-Trading application.

Shapes for Oracle Transaction Manager for Microservices image for OCI Marketplace

You can use any of the following compute shapes to create a Transaction Manager for Microservices instance in OCI.

Standard

```
VM.Standard1.4, VM.Standard1.8, VM.Standard1.16, VM.Standard.B1.4,
VM.Standard.B1.8, VM.Standard.B1.16, VM.Standard2.4, VM.Standard2.8,
VM.Standard2.16, VM.Standard2.24, VM.Standard.E2.4, VM.Standard.E2.8
```

Flexible

```
VM.Standard.E3.Flex (OCPU constraint: 4-64, Memory constraint: 24-1024 GB);
VM.Standard.E4.Flex (OCPU constraint: 4-64, Memory constraint: 24-1024 GB);
VM.Standard3.Flex (OCPU constraint: 4-32, Memory constraint: 24-512 GB)
```

Optimized

```
VM.Optimized3.Flex (OCPU constraint: 4-18, Memory constraint: 24-256 GB)
```

Dense

```
VM.DenseIO1.4, VM.DenseIO1.8, VM.DenseIO1.16, VM.DenseIO2.8, VM.DenseIO2.16,
VM.DenseIO2.24
```

1.2 Minimum System Requirements

To create a Transaction Manager for Microservices instance in OCI and run sample applications, you must meet the minimum system requirements.

Ensure that at least 4 OCPUs, 24 GB memory, and 128 GB of bootable storage volume is available in your Oracle Cloud Infrastructure tenancy.

1.3 Software Requirements

MicroTx coordinator is a containerized application and can run on any container-based platforms, such as Docker and Kubernetes.

Table 1-1 Software Requirements for MicroTx

Component	Requirement
Java 2 JRE for the Java run-time environment	Tested with JRE 17.0.4 and OpenJDK 17
Java 2 Software Development Kit (SDK) for the Java development environment	Tested with JRE 17.0.4 and OpenJDK 17

For more additional information, see Oracle Transaction Manager for Microservices Developer Guide.

2
Create an Instance

Learn how to create a Transaction Manager for Microservices instance in OCI using the Transaction Manager for Microservices image for OCI.

You can create an instance using the Transaction Manager for Microservices image available in OCI Marketplace in two ways. Either use the Compute instance console or create an instance directly by using the image from OCI Marketplace.

Topics:

2.1 Create Compartment Policies

You must be granted management access to Marketplace applications and Resource Manager if you are not an Oracle Cloud Infrastructure administrator. Transaction Manager for Microservices (MicroTx) image for OCI allows you to create compute instances.

The following are sample policies:

```
Allow group
        MyGroup to use app-catalog-listing in compartment MyCompartmentAllow
group
        MyGroup to manage instance-family in compartment MyCompartmentAllow
group
        MyGroup to manage orm-family in compartment MyCompartmentAllow group
        MyGroup to manage virtual-network-family in compartment
MyNetworkCompartment
```

> **See Also:**
>
> Refer to Common Policies in the Oracle Cloud Infrastructure documentation, Managing Policies.

2.2 Create an Instance Using the Compute Instance Console

The Compute instance console enables you to create a new instance using Transaction Manager for Microservices image for OCI.

To create an instance using the Compute instance console:

1. Sign in to the *Oracle Cloud Infrastructure* console.
2. Click Navigation from the menu, select **Compute** and under the **Compute** group, click **Instances**.
3. Click **Create Instance**.
   If required, you can modify the name of the instance.
4. Select the compartment in which to create the instance.
5. Under **Placement**, select the **Availability Domain** for creating the instance. Click **Show advanced options**.
6. Under **Image and Shape**, click **Change Image** and follow the instructions:
   a. From the **Image source** drop-down, select *Marketplace*, and you can either use **Search** or select the Transaction Manager for Microservices image for OCI.
   b. Review the *terms and conditions* and select the **Oracle Terms of Use** check box and click **Select Image**.
7. Under **Image and Shape**, click **Change Shape**. Select the **Instance Type** and select the shape.
   While creating a virtual machine, under **Shape series**, select a processor group, and then select a shape. To know the supported shapes, see About Transaction Manager for Microservices Images for OCI.
8. Configure the network for the instance. Click **Show advanced options** to specify advanced network settings.
9. Under **Add SSH keys**, generate a key, upload your public key, or paste the keys.
10. Under **Boot Volume**, specify the size and encryption options for the instance's boot volume.
11. Click **Show advanced options** to configure advanced settings.
12. Click **Create**.
    For more details, see Creating a Linux instance in *Oracle Cloud Infrastructure documentation*.

2.3 Create an Instance Using the Image in Marketplace

Use the Transaction Manager for Microservices for OCI image available on Oracle Cloud Marketplace to create a new instance.

To create an instance:

1. Sign in to the *Oracle Cloud Infrastructure* console.

2. Click the navigation menu, select **Marketplace**, and then click **All Applications**.
3. Select the Transaction Manager for Microservices image for OCI.
4. From the **Version** drop down, select the image build version.
   Every image is built for a specific operating system, which may not support all operating systems.
5. Review the *terms and conditions* and select the **Oracle Terms of Use** check box.
6. Click **Launch Instance**.
7. Select the compartment in which you want to create the instance.
8. Under **Placement**, select the **Availability Domain** for creating an instance.
   Click **Show advanced options** to specify capacity type and fault domain.
9. Under **Image and Shape**, click **Change Shape**. Select the **Instance Type** and select the shape.
   While creating a virtual machine, under **Shape series**, select a processor group, and then select a shape. To know the supported shapes, see About Transaction Manager for Microservices Images for OCI.
10. Configure the network for the instance and click **Show advanced options** to specify advanced network settings.
11. Under **Add SSH keys**, generate a key, upload your public key, or paste the keys.
12. Under **Boot Volume**, specify the size and encryption options for the instance's boot volume.
13. Click **Show advanced options** to configure advanced settings.
14. Click **Create**.
    For more details, see Creating a Linux instance in *Oracle Cloud Infrastructure documentation*.

2.4 Access the MicroTx Instance in OCI

When you create an instance using an image, ensure that you provide an SSH key. You require this SSH key to access the instance and launch it.

To access the Transaction Manager for Microservices instance in OCI:

1. Use the SSH command to connect to an instance as the `opc` user.
   - Connect using the SSH private key if the private key associated with the public key was used during provisioning. For example:

     ```
     ssh -i <private_key> opc@<ip_address>
     ```

   - Connect using the SSH public key if the public key was provided during provisioning. Log in directly with the following command:

     ```
     ssh opc@<ip_address>
     ```

     For detailed instructions, see Accessing Your Instance.
2. Once you are logged in as the `opc` user, switch to the `oracle` user.

   ```
   sudo su - oracle
   ```

3. Create an ingress rule to enable traffic to port 6080. A VNC server is configured on this VM. To access the instance using a VNC session, you must add port 6080 to the ingress rules. Skip this step if an ingress rule that permits traffic over port 6080 already exists.
   a. Go to **Networking** >> **Virtual Cloud Networks**
   b. Choose your network.
   c. Under **Resources**, select **Security Lists**.
   d. Click **Default Security Lists** under **Create Security List**.
   e. Click **Add Ingress Rule**.
   f. Enter the following details:
      - Source Type: CIDR
      - Source CIDR: 0.0.0.0/0
      - IP Protocol: TCP
      - Source Port Range: All (keep default)
      - Destination Port Range: **6080**
      - Description: noVNC Remote Desktop
   g. Click **Add Ingress Rules**.
4. Set a new password to access the VNC session.

   ```
   echo "<PASSWORD>" | vncpasswd -f >/home/oracle/.vnc/passwd
   ```

   Where, *<PASSWORD>* is the new password that you want to set. Ensure that the password meets the password requirements set by VNC server.
5. Restart the VNC server on the VM.

   ```
   sudo systemctl restart vncserver_oracle@:1.service
   ```

6. Run the following command to confirm that the VNC server is in the running state.

   ```
   sudo systemctl status vncserver_oracle@:1.service
   ```

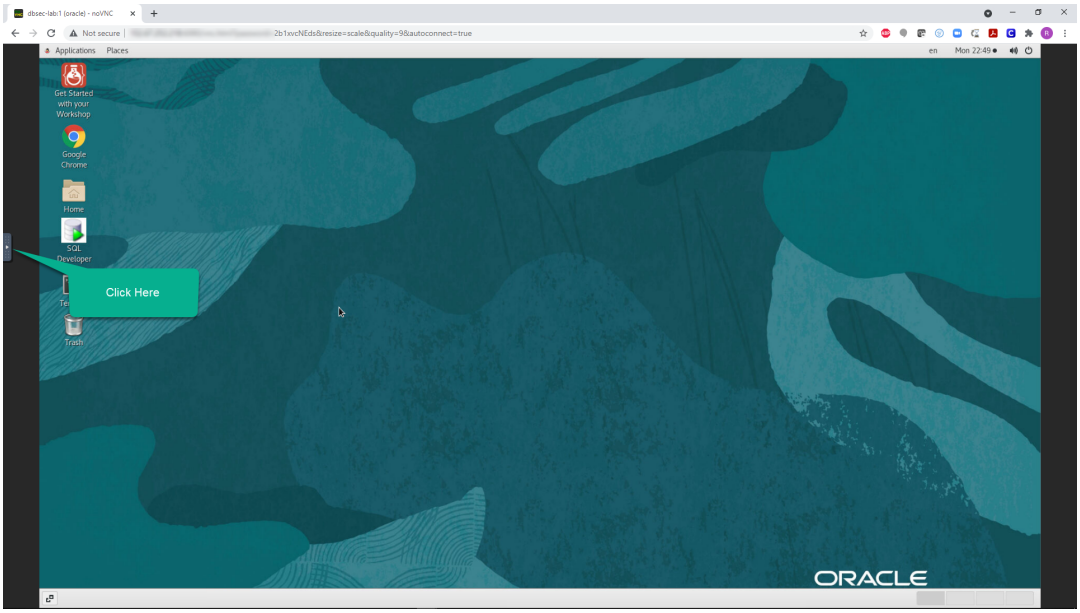
7. Open the following link in a new browser to login and access the VNC session. Use the password that you have specified in an earlier step.

   ```
   http://<ip_address>:6080/vnc.html?
   password=<PASSWORD>&resize=scale&quality=9&autoconnect=true&reconnect=true
   ```

2.5 Enable Copy/Paste from Local Computer to Remote Desktop

You might need to copy text, such as commands from the guide, from your local computer to the remote desktop. While such direct copy/paste isn't supported, you may proceed as indicated below to enable an alternative *local-to-remote clipboard* with input text field.

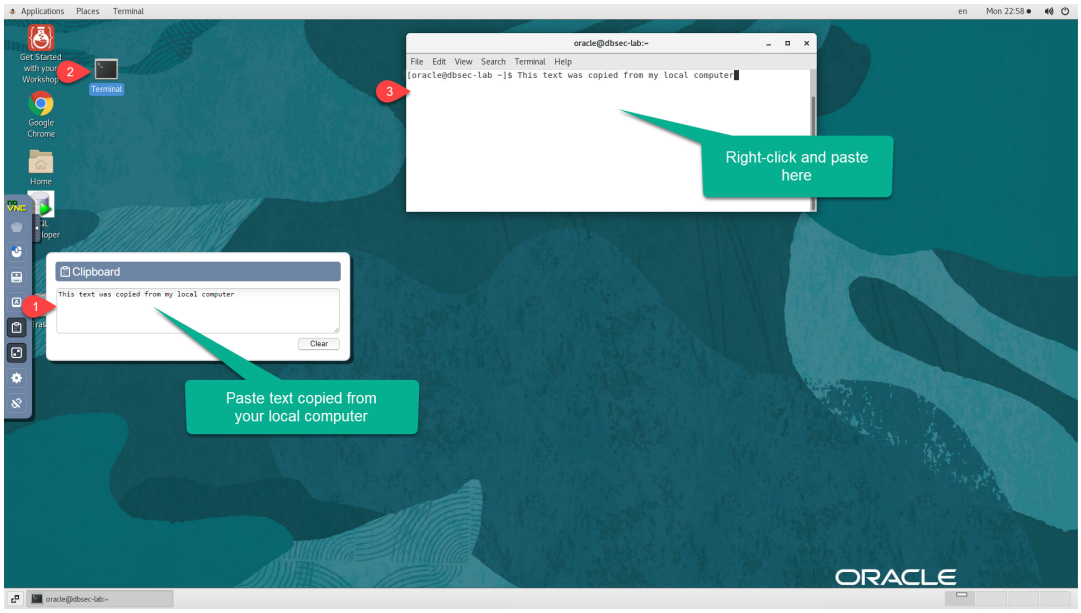
1. Click the small gray tab on the middle-left side of your screen to open the control bar.



2. Click the clipboard icon.

3. Copy some text from your local computer as illustrated below and paste it into the clipboard widget. Open the desired application, such as Terminal, and use mouse controls

to paste the text.



Note:

Ensure that you initialize your clipboard with Step 2 shown in the screenshot above before opening the target application in which you intend to paste the text. Otherwise will find the *paste* function in the context menu is grayed out when you attempt to paste for the first time.

3

Run the Bank and Stock-Trading Application

Learn how to use MicroTx to maintain data consistency across several microservices by deploying and running the Bank and Stock-Trading 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 Bank and Stock-Trading application contains several microservices and it uses distributed, two-phase commit transaction (XA). Run the Bank and Stock-Trading application to purchase and sell stocks as well as transfer money from one account to another. When you run the Bank and Stock-Trading application, you will be able to see how MicroTx ensures consistency of transactions across the distributed microservices and their resource managers. You will also integrate MicroTx with the Kubernetes ecosystem by using tools, such as Kiali and Jaeger, to visualize the flow of requests between MicroTx and the microservices.

Running a sample application is the fastest way for you to get familiar with MicroTx.

Topics:

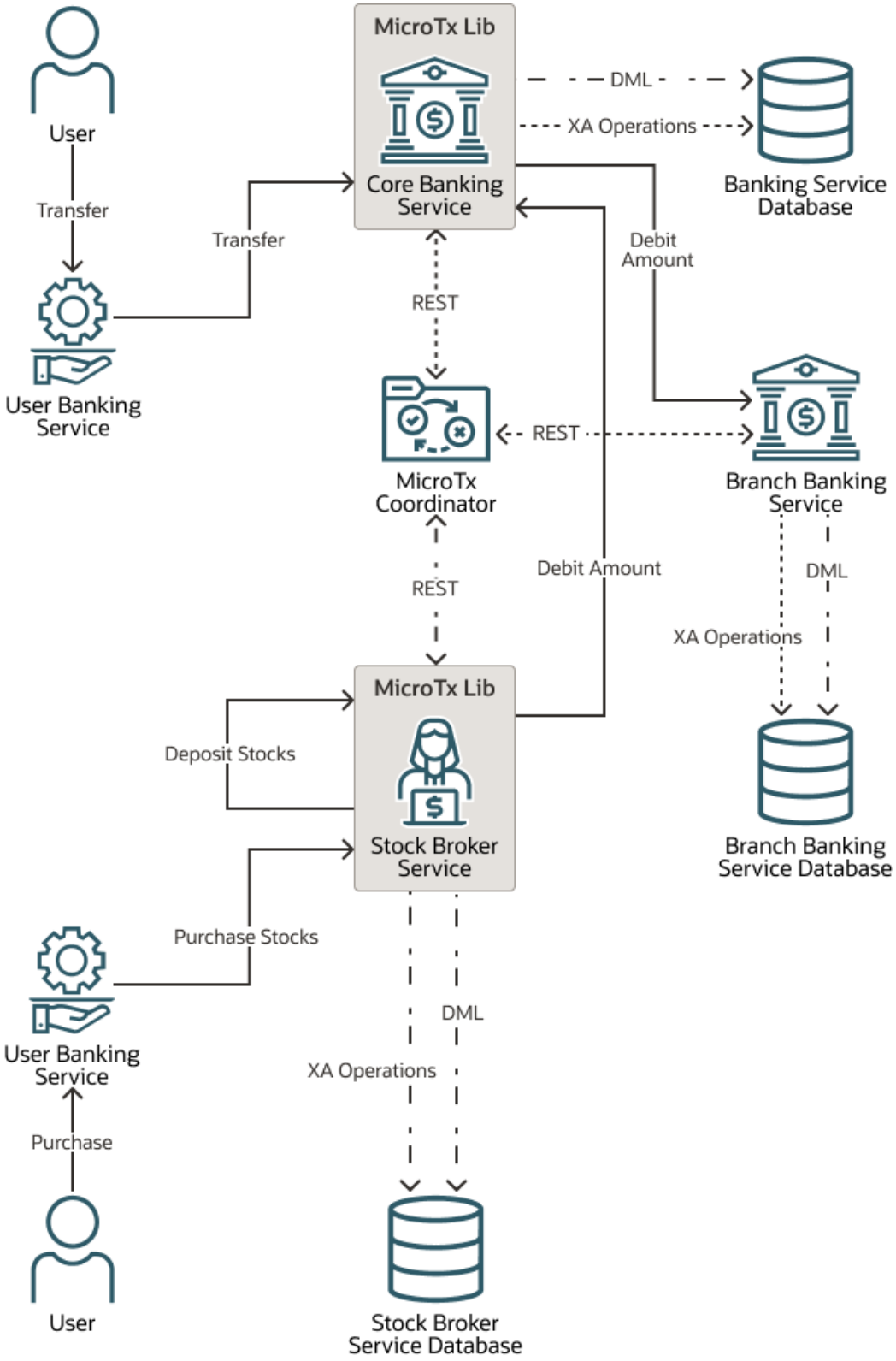
requests flow between MicroTx and the microservices. Running sample applications is the fastest way for you to get familiar with MicroTx.

- Deploy Kiali and Jaeger (Optional)
  Optionally, you can use Kiali and Jaeger to track and trace distributed transactions in MicroTx through visualization. Use distributed tracing to track how requests flow between MicroTx and the microservices.
- View Service Mesh graph and Distributed Traces (Optional)
  To visualize what happens behind the scenes and how a request processed by the distributed services and MicroTx, you can use the Kiali and Jaeger Dashboards that you started in the previous task.

3.1 About the Bank and Stock-Trading Application

The Bank and Stock-Trading application demonstrates how you can develop microservices that participate in a distributed transaction while using MicroTx to coordinate the requests. You can use the application to withdraw or deposit an amount, as well as buy and sell stocks. Since financial applications that move funds require strong global consistency, the application uses XA transaction protocol.

The following figure shows the various microservices in the Bank and Stock-Trading application. Some microservices connect to an Autonomous Transaction Processing Serverless (ATP-S) instance or resource manager. Resource managers manage stateful resources such as databases, queuing or messaging systems, and caches.



Key:

- Application REST Calls
- MicroTx REST Calls for Transfer
- MicroTx REST Calls for Purchase of Stocks

When a user purchases stocks using the Stock Broker service, the application withdraws money from the Core Banking Service and deposits an equivalent number of stocks by creating an XA transaction. Within the XA transaction, all actions such as purchase, sale, withdraw, and deposit either succeed, or they all are rolled back in case of a failure of any one or more actions.

Participant microservices use the MicroTx client libraries which registers callbacks and provides implementation of the callbacks for the resource manager. As shown in the image, MicroTx communicates with the resource managers to commit or roll back the transaction. MicroTx connects with each resource manager involved in the transaction to prepare, commit, or rollback the transaction. The participant service provides the credentials to the coordinator to access the resource manager. The Stock Broker, Core Banking, Branch Banking, and User Banking microservices are based on Spring Boot 3.x. The MicroTx Spring Boot starter client library files, which are available for Spring REST apps, are already integrated with all the microservices.

During a transaction, each microservice also makes updates to a resource manager to track the change in the amount and stocks. When you run the Bank and Stock-Trading application, you will be able to see how MicroTx ensures consistency of transactions across the distributed microservices and their resource managers.

- The MicroTx coordinator manages transactions amongst the participant services.
- The Stock Broker microservice initiates the transactions, so it is called a transaction initiator service. The user interacts with this microservice to buy and sell shares. When a new request is created, the helper method that is exposed in the MicroTx library runs the `begin()` method to start the transaction. This microservice also contains the business logic to issue the commit and roll back calls. After initiating the transaction, the Stock Broker service also participates in the transaction. After starting a transaction to buy or sell shares, the Stock Broker service also participates in the transaction to deposit or withdraw the shares from a user's account. It uses resources from the Stock Broker Service ATP instance.
- The Core Banking, Branch Banking, and User Banking services participate in the transactions related to the trade in stocks, so they are called participant services. They do not initiate the transaction to buy or sell stocks. The MicroTx library includes headers that enable the participant services to automatically enlist in the transaction. These microservices expose REST APIs to get the account balance and to withdraw or deposit money from a specified account. Core Banking and Branch Banking services also use resources from the Banking Service and Branch Banking Service ATP instances respectively.

The service must meet ACID requirements, so withdraw amount, transfer amount, deposit stocks, sell stocks, debit amount, or credit amount are called in the context of an XA transaction.

3.2 Configure Minikube and Start a Tunnel

Configure Minikube and then start a tunnel between Minikube and the Kubernetes cluster.

Before you begin, ensure that can access the Transaction Manager for Microservices instance that have created in OCI using the Transaction Manager for Microservices image for OCI. See Create an Instance.

1. If you have connected to your instance as an `opc` user through an SSH terminal using auto-generated SSH Keys, then you must switch to the `oracle` user before you begin.

   ```
   sudo su - oracle
   ```

2. At the command prompt, run the following command to start Minikube.

   ```
   minikube start
   ```

3. Run the following command in a new terminal to start a tunnel between Minikube and the Kubernetes cluster. Keep this terminal window open.

   ```
   minikube tunnel
   ```

4. Enter the password to access your local machine if you are prompted to enter your password.
5. In a new terminal, run the following command to note down the external IP address of the Istio ingress gateway.

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

   From the output note down the value of `EXTERNAL-IP`, which is the external IP address of the Istio ingress gateway. You will provide this value in the next step. If the `EXTERNAL-IP` is in the `pending` state, ensure that the Minikube tunnel is running before proceeding with the next steps.

   Example output

   ```
   $ kubectl get svc istio-ingressgateway -n istio-system
   NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)                                      AGE
   istio-ingressgateway   LoadBalancer   [redacted]    [redacted]    15021:31294/TCP,80:30627/TCP,443:32267/TCP   7d11h
   ```

   Let's consider that the external IP in the above example is `192.0.2.117`.
6. 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=192.0.2.117
   ```

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

When you start Minikube, all the service pods start running. When Minikube starts, it also starts the Database service.

3.3 Configure Keycloak

The Bank and Stock-Trading Application console uses Keycloak to authenticate users.

1. Run the following command to note down the external IP address and port to access Keycloak.

   ```
   kubectl get svc -n keycloak
   ```

   From the output note down the value of `EXTERNAL-IP` and `PORT(S)`, which is the external IP address and port of Keycloak. You will provide this value later.

   ```
   [oracle@tmmlabserv1:~]$ kc get svc -n keycloak
   NAME       TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
   keycloak   LoadBalancer   [redacted]      [redacted]      8080:30589/TCP   7d20h
   ```

   Let's consider that the external IP in the above example is `198.51.100.1` and the port is `8080`.

2. Run the following command to run the `reconfigure-keycloak.sh` script from the `$HOME` directory.

```
cd $HOME
sh reconfigure-keycloak.sh
```

3. Sign in to Keycloak. In a browser, enter the IP address and port number that you have copied in a previous step. The following example provides sample values. Provide the values based on your environment.

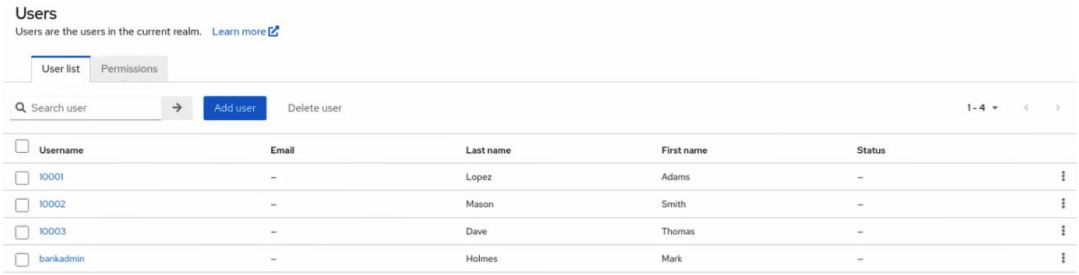
```
http://198.51.100.1:8080
```

This command configures Keycloak and updates the settings to suit the requirements of the application.

4. Click **Administration Console**.
5. Sign in to Keycloak with the initial administrator user name `admin` and password `admin`.

After logging in, we strongly recommend that you reset the password for the `admin` user. For information about resetting the password, see the Keycloak documentation.

6. Select the **MicroTx-BankApp** realm, and then click **Users** to view the list of users in the `MicroTx-BankApp` realm. The `MicroTx-BankApp` realm is preconfigured with these default user names.



Set the password for each user. For information about providing credentials for users, see the Keycloak documentation.

3.4 Verify that All the Resources are Ready

Redeploy the Bank and Stock-Trading application, and then ensure that are the resources are available.

1. Run the `reconfigure-bankapp.sh` script from the `HOME` directory to set the values by required by the application's Helm Chart.

```
$ cd $HOME
$ sh reconfigure-bankapp.sh
```

The Helm Chart is updated with the required values, and then the Bank and Stock-Trading application is redeployed using the latest values.

2. Verify that the Bank and Stock-Trading application has been redeployed successfully.

```
helm list -n otmm
```

Example output

```
$ helm list -n otmm
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
bankapp otmm            1               2023-11-12 17:54:28.36336038 +0000 UTC  deployed        bankapp-1.0.1   1.0.1
otmm    otmm            1               2023-11-09 20:02:09.009490794 +0000 UTC deployed        otmm-23.4.1     23.4.1
```

In the output, verify that the `STATUS` of the `bankapp` is `deployed`.

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

```
kubectl get pods -n otmm
```

Example output

```
kubectl get pods -n otmm
NAME                                    READY   STATUS    RESTARTS         AGE
arizona-branch-bank-7464886fd-zs27x     2/2     Running   0                103m
core-banking-599b7659d4-89srk           2/2     Running   0                103m
otmm-tcs-0                              2/2     Running   0                103m
user-banking-867d4f9657-7gbln           2/2     Running   24 (5m13s ago)   103m
```

4. Verify that the database instance is running. The database instance is available in the `oracledb` namespace. Run the following command to retrieve the list of resources in the `oracledb` namespace and their status.

```
kubectl get pods -n oracledb
```

Example output

```
$ kubectl get pods -n oracledb
NAME                                        READY   STATUS    RESTARTS      AGE
db23cfree-oracle-db23c-free-6b69f68c98-pt99r   1/1     Running   6 (17h ago)   5d10h
```

It usually takes some time for the Database services to start running in the Minikube environment. Proceed with the remaining tasks only after ensuring that all the resources, including the database service, are ready and in the `RUNNING` status and the value of the `READY` field is `1/1`.

3.5 Transfer Funds with the Bank and Stock-Trading Application

Run the Bank and Stock-Trading application to transfer funds and to understand how you can use Transaction Manager for Microservices (MicroTx) to coordinate XA transactions. After running the application, use distributed tracing to understand how the requests flow between MicroTx and the microservices. Running sample applications is the fastest way for you to get familiar with MicroTx.

When you send a request to transfer funds, the Core Banking service and Branch Banking service interact with each other to perform this task. The Core Banking service in turns sends the debit amount request to the Branch Banking service. Once the amount is successfully debited from the specified bank account, it is credited to another bank account. The microservices use the XA protocol and MicroTx to manage the transactions. Within an XA transaction, all actions such as debit amount and credit amount either succeed, or all actions are rolled back in case of a failure of any one or more actions.

Fund transfer happens between the branches. In this sample application, we have only one branch (Arizona), so the Core banking and Branch banking microservices interact with each other.

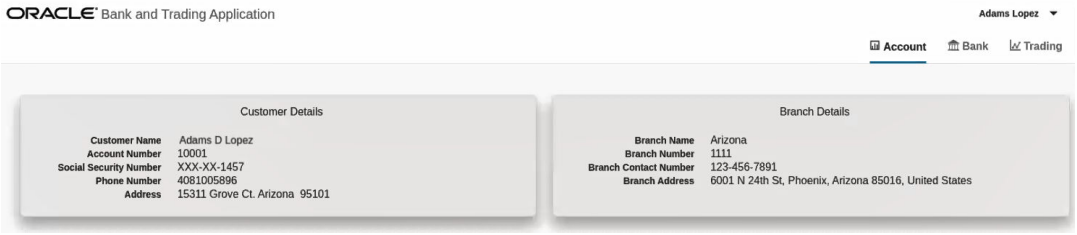
To transfer funds:

1. Access the bank application. In a browser, type `192.0.2.117/bankapp`, where `192.0.2.117` is the external IP address of the Istio ingress gateway which you have noted down earlier after starting a Minikube tunnel.

   The Keycloak login page is displayed.

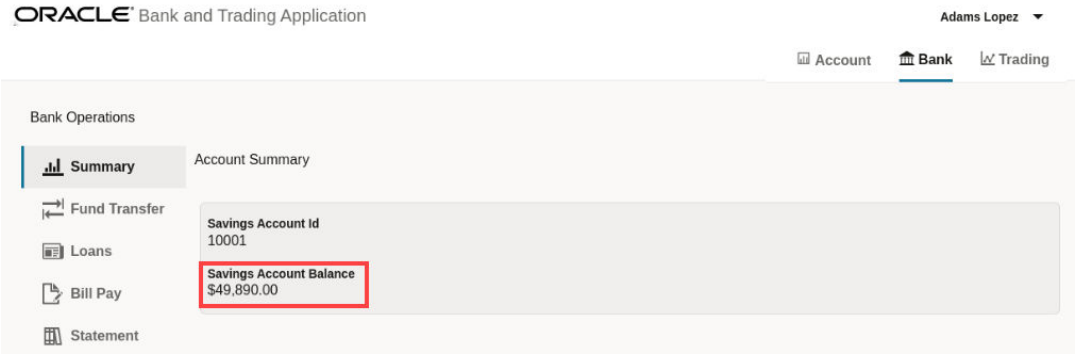
2. Enter the username and password to access the Keycloak instance. Enter the password that you had provided earlier for the preconfigured Keycloak users.

   The Bank and Stock-Trading application's console is displayed as shown in the following figure.



3. Click the **Bank** tab, and then click **Summary**.

   The account summary is displayed as shown in the following image. Note down the account balance.



4. Click **Fund Transfer**, and then click **Transfer Funds**.

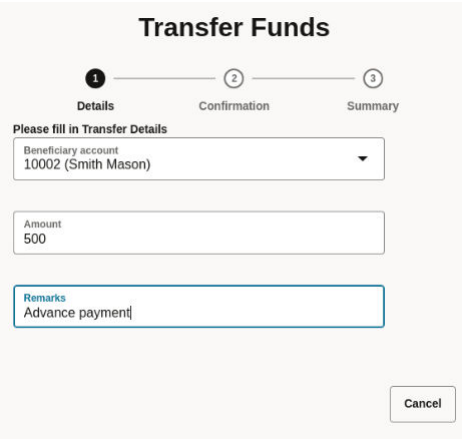
   The **Transfer Funds** dialog box appears.

5. In the **Transfer Funds** dialog box, enter the following details.

   a. Select the name or the account to which you want to transfer the amount.

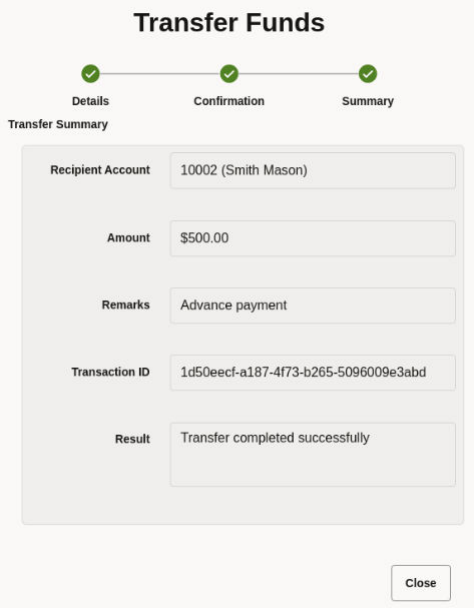
   b. Enter the amount that you want to transfer.

   c. (Optional.) Enter remarks, if any, regarding the transfer.



d. Click **Confirmation**, and then review the details of the transfer.

e. Click **Confirm** to transfer the amount.
After the application withdraws the specified amount from account 10001 and deposits the amount in account 10002, the **Transaction ID** and **Result** are displayed on the screen.

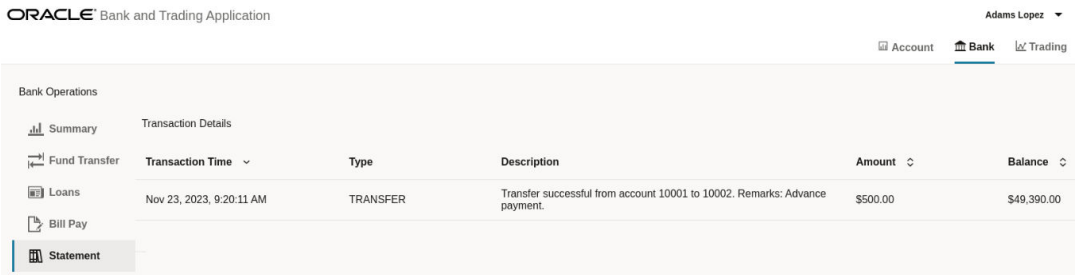


f. Click **Close** to close the **Transfer Funds** dialog box.

6. Click **Summary** to view the updated balance in the account summary.

The following image shows that the amount has reduced by 500, the amount that you have transferred.

7. Click **Statement** to view the transaction statement as shown in the image below.



3.6 Trade Stocks with the Bank and Stock-Trading Application

Run the Bank and Stock-Trading application to purchase and sell stocks and to understand how you can use Transaction Manager for Microservices (MicroTx) to coordinate XA transactions. After running the application, use distributed tracing to understand how the requests flow between MicroTx and the microservices. Running sample applications is the fastest way for you to get familiar with MicroTx.

When you send a request to purchase stocks, the Stock Broker service debits the required amount from the Core Banking service. The Core Banking service in turns sends the debit amount request to the Branch Banking service. Once the amount is successfully debited from your bank account, the Stock Broker service purchases the stocks and deposits the purchased stocks into your account. The microservices use the XA protocol and MicroTx to manage the transactions. Within an XA transaction, all actions such as debit amount and deposit stocks either succeed, or all actions are rolled back in case of a failure of any one or more actions.

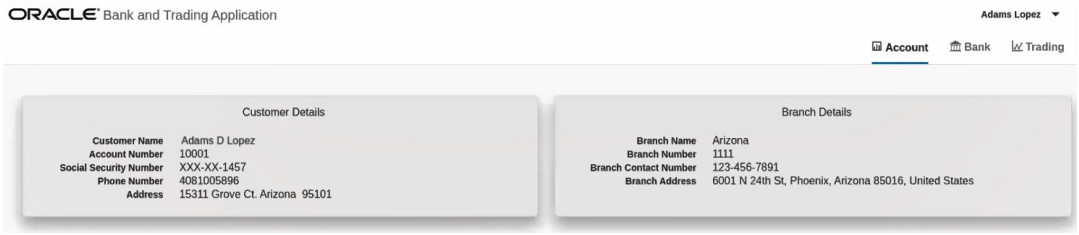
To purchase stocks:

1. Access the bank application. In a browser, type `192.0.2.117/bankapp`, where `192.0.2.117` is the external IP address of the Istio ingress gateway which you have noted down earlier after starting a Minikube tunnel.

   The Keycloak login page is displayed.

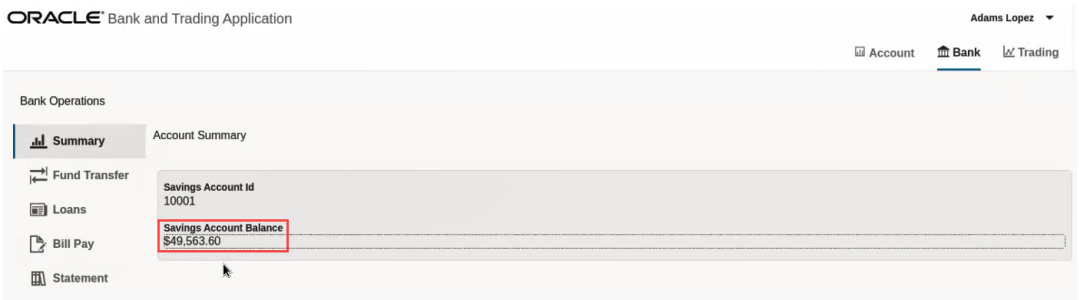
2. Enter the username and password to access the Keycloak instance. Enter the password that you had provided earlier for the preconfigured Keycloak users.

   The Bank and Stock-Trading application's console is displayed as shown in the following figure.



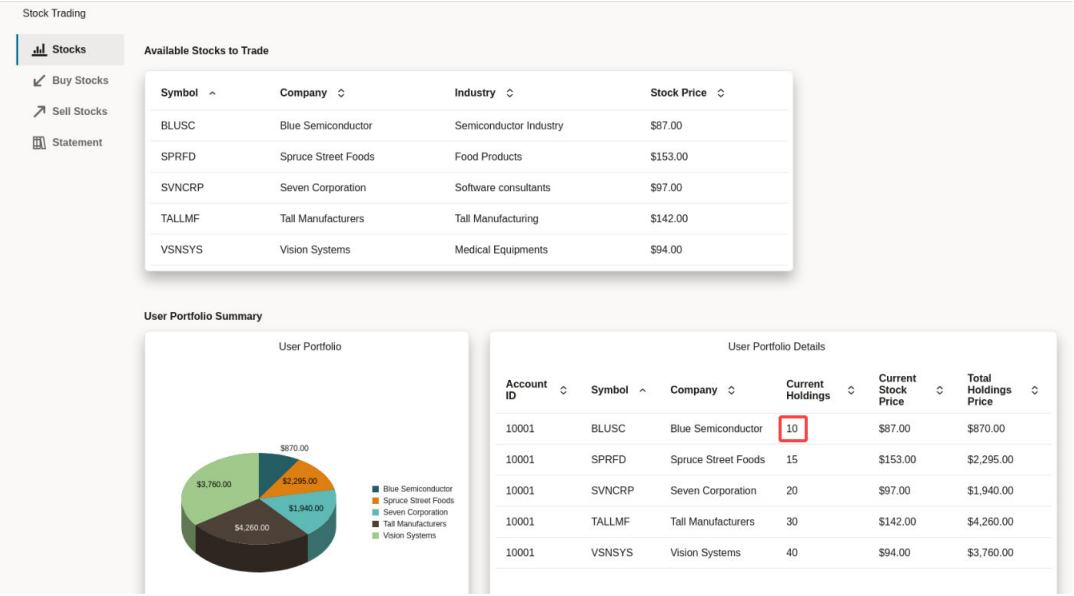
3. Click the **Bank** tab, and then click **Summary**.

   The account summary is displayed as shown in the following image. Note down the account balance.

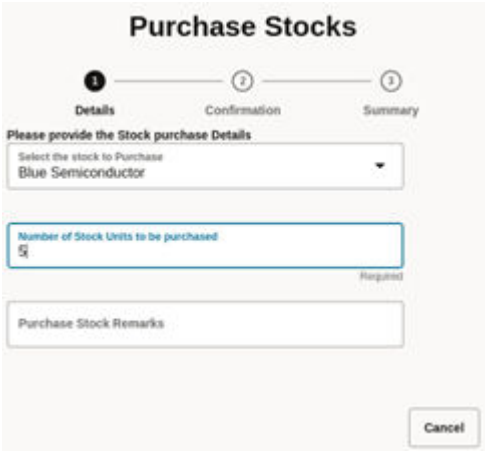


4. Click the **Trading** tab.

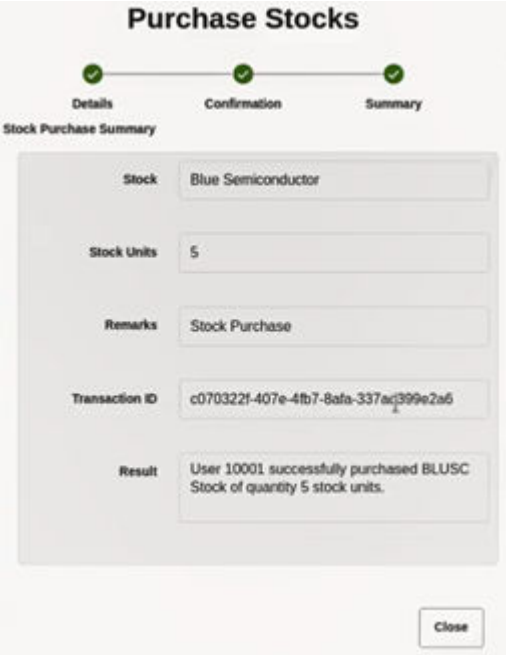
   The Stock Trading page is displayed. Identify the stock that you want to purchase and the number of units of the stock that are currently available in your account. For example, let's consider that you want to purchase shares of the Blue Semiconductor. The following image shows that you have 10 shares of Blue Semiconductor.



5. Click **Buy Stocks**, and then click **Buy Stocks**.
6. In the **Purchase Stocks** dialog box, enter the following details.
   a. Select the stock that you want to purchase.
   b. Enter the number of units of the stock that you want to purchase.
   c. (Optional.) Enter remarks, if any, regarding your purchase.



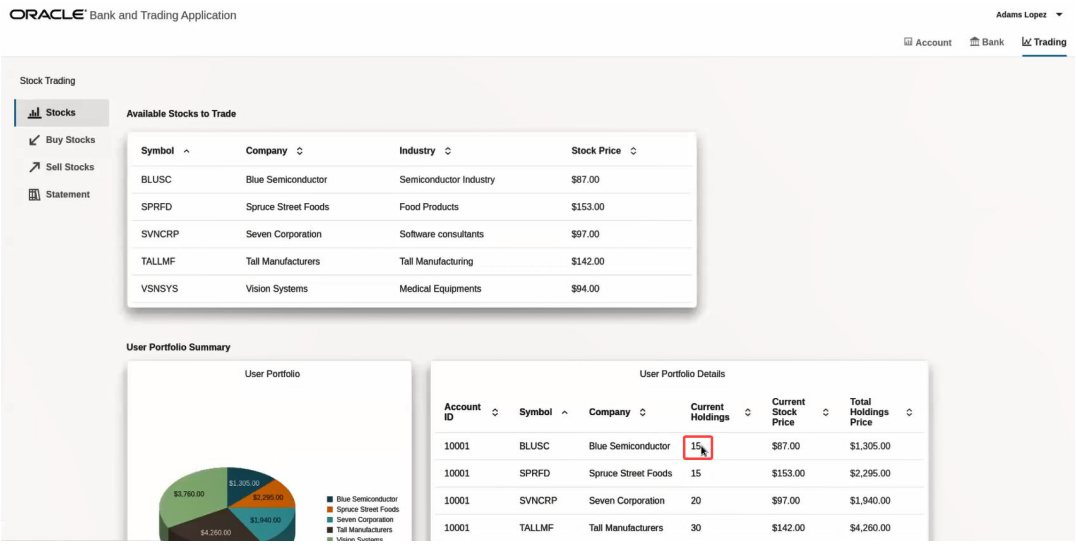
   d. Click **Confirmation**, and then review the details of the purchase.
   e. Click **Confirm** to purchase the stocks.
   After the Stock Broker service purchases the stocks and deposits it in your account, the **Transaction ID** and **Result** are displayed on the screen.



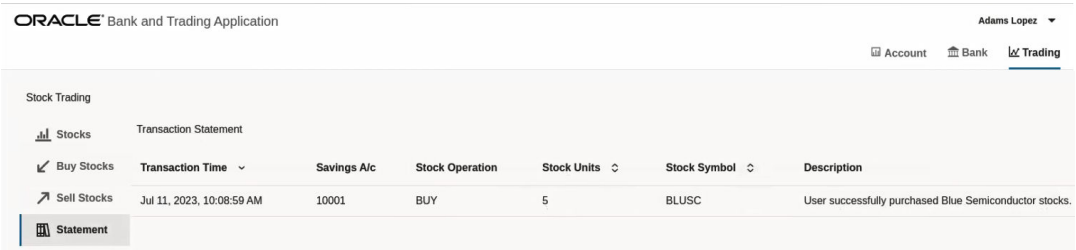
f. Click **Close** to close the **Purchase Stocks** dialog box.

7. Click **Stocks** to view the updated list of stocks.

   The following image shows the number of shares of Blue Semiconductor has increased by 5, the purchased amount, in your account.

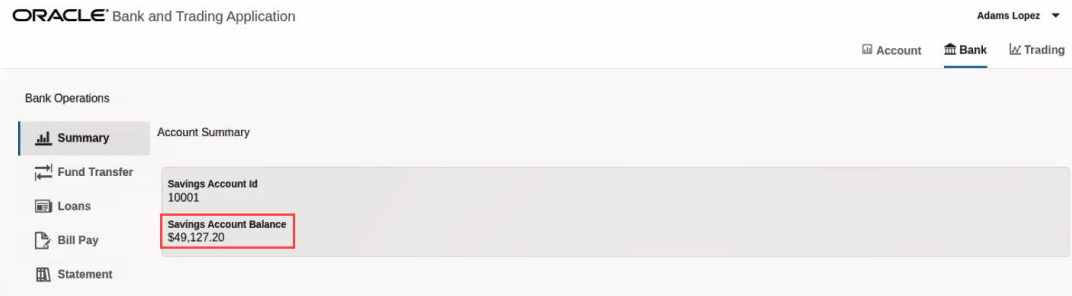


8. Click **Statement** to view the transaction statement as shown in the image below.



9. Click the **Bank** tab, and then click **Summary**.

The account summary is displayed as shown in the following image. Note that the account balance has reduced as the Stock Broker service debits the amount required to purchase the shares from your account.



3.7 Deploy Kiali and Jaeger (Optional)

Optionally, you can use Kiali and Jaeger to track and trace distributed transactions in MicroTx through visualization. Use distributed tracing to track how requests flow between MicroTx and the microservices.

Run the following commands to deploy Kiali and Jaeger in a Minikube cluster.

1. Deploy Kiali.

   ```
   kubectl apply -f https://raw.githubusercontent.com/istio/istio/
   release-1.17/samples/addons/kiali.yaml
   ```

2. Deploy Prometheus. To use Kiali, you must deploy Prometheus and Kiali in the same cluster.

   ```
   kubectl apply -f https://raw.githubusercontent.com/istio/istio/
   release-1.17/samples/addons/prometheus.yaml
   ```

3. Deploy Jaeger.

   ```
   kubectl apply -f https://raw.githubusercontent.com/istio/istio/
   release-1.17/samples/addons/jaeger.yaml
   ```

4. Start the Kiali Dashboard. Run the following command in a new terminal. Ensure that you leave this terminal open. If a new browser opens, close the browser.

   ```
   istioctl dashboard kiali
   ```

   From the output, note down the URL. This is the URL on which you can access the Kiali dashboard in a browser. For example, `http://localhost:20001/kiali`.

5. Start the Jaeger Dashboard. Run the following command in a new terminal. Ensure that you leave this terminal open. If a new browser opens, close the browser.

   ```
   istioctl dashboard jaeger
   ```

   From the output, note down the URL. This is the URL on which you can access the Jaeger dashboard in a browser. For example, `http://localhost:16686`.

3.8 View Service Mesh graph and Distributed Traces (Optional)

To visualize what happens behind the scenes and how a request processed by the distributed services and MicroTx, you can use the Kiali and Jaeger Dashboards that you started in the previous task.

Perform this task only if you have deployed Kiali and Jaeger in your cluster.

1. Open a new browser tab and navigate to the Kiali dashboard URL. For example, `http://localhost:20001/kiali`.
2. Select Graph for the `otmm` namespace.
3. Open the Jaeger dashboard URL in a new browser. For example, `http://localhost:16686`.
4. In the **Service** drop-down list, select `istio-ingressgateway`. A list of traces is displayed where each trace represents a request.
5. Select a trace to view it.