Project 1 - Implementing Authentication and Access Control for a Web Application

Network Security - Spring 2025

Due Date: February 13, 2025 (11:55 pm)

Contents

1 Introduction

The goal of this project is to implement authentication and access control mechanisms for a **web application for an electricity distribution billing system**. The application is built using a **three-layer architecture**, as illustrated in Figure 1. This architecture consists of the following components:

- **Web Server**: Responsible for handling HTTP requests from a web browser and sending back results in form of HTML documents. The web server used in this project is **Nginx**.
- **Application Server**: Manages business logic and serves as an intermediary between the web server and the database. The application server used in this project is **Uvicorn**, which hosts Python applications developed using the **FastAPI framework**.
- **Database Server**: Stores and manages the application's data. The database server utilized is **Oracle Database**.

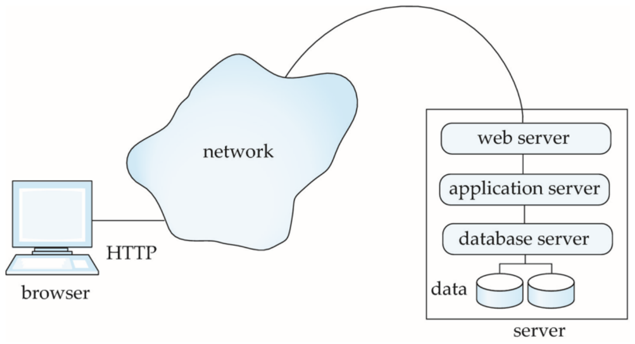


Figure 1: Three-layer web application architecture (source: Database System Concepts, 7th Edition. Silberchatz, Korth, and Sudarshan)

You are provided the functional implementation of this web application (details in Section 6.1), but it lacks basic security features, such as user authentication, or access control management. Your job is to deploy this application on the Oracle cloud and implement the authentication and access control capabilities as discussed in Section 6. This introductory project will also help you to get comfortable with the Oracle Cloud environment and the tools that you will be using throughout the course.

2 Overview

The project consists of primarily the following four tasks:

1. Create an *Oracle Cloud account.*
2. Set up a cloud Autononmous Transaction Processing (ATP) database.
3. Set up a cloud Virtual Machine.
4. Implement authentication and access control mechanisms in the application.

You will be required to submit the source code and some screenshots showing proof of the working setup (details in Section 7). The project submission deadline is **Thursday February 13, 2025 by 11:55 pm**.

3 Part 1: Oracle Cloud Account Creation

The provided application will be deployed, via a cloud virtual machine, that uses a cloud database. In order to access Oracle Cloud resources to deploy the application, you need to create an Oracle Cloud account first. You may have already received an email from Oracle containing the steps to create your account. Here's a brief rundown of the steps required to create an Oracle Cloud account. The accompanying screenshots can be found in the appendix section.

1. Go to `https://signup.oraclecloud.com/`.
2. You will be greeted with a page asking for your *Country*, *First Name*, *Last Name*, and your *Email address*. Fill in the required information and click on the "*Next*" button. **Make sure to use your LUMS email address.** Figure 2 shows the sign up page. Click on "Verify my email" once you have entered your details.
3. You might be greeted with a "Select offer" page. Select the only option available (providing free credits for a year) and click on the *Next* button. Figure 3 shows the offer selection page.
4. After email verification, you will be led to the "Account Information" page. Fill in the required information and click on the *Next* button. **Make sure to select "Individual" as your customer type.** You can set any region as your home region, but preferably select the region closest to you for better performance.
5. After account information, you will need to provide your address information. Fill in the required information, agree to the terms and conditions, and click on the "*Create Account*" button.
6. You will receive an email from Oracle Cloud with your credentials (Cloud account name, and your username). You will need these credentials to log in to Oracle Cloud. On your first login, you might be prompted to set up a 2FA (Two Factor Authentication) method. Follow the instructions to set up 2FA. It is highly recommended to set up 2FA for security reasons.
7. You will be prompted to set up bypass keys. PLEASE MAKE SURE TO SAVE THESE KEYS SOMEWHERE. In the event you change your 2FA device, you won't be able to log in to your account, and hence you will be locked out. The bypass keys will help you in such a situation.

4 Part 2: Create an Oracle Cloud Database

Your Oracle Cloud account lets you create 2 instances of Autonomous Transaction Processing (ATP) databases. The application uses OracleDB to store data, therefore, you will be creating a database to use with the application.

1. On the "Get Started" page, scroll down to the "Launch resources" section and click on the "*Create an ATP database*" button. Figure 4 shows the "Launch resources" section. This will lead you to the page to create an ATP database.
2. On the "Create Autonomous Database" page, fill in the required information. Keep the default "*compartment*" option. Make sure it is your cloud account username. Set a "*Display Name*" and a "*Database Name*" for your ATP instance. Figure 5 shows an example.
3. Choose the workload type to be "*Transaction Processing*" which is the default option. Keep the default deployment type, which is "*Serverless*".
4. In the "Configure the Database" section, toggle the "*Always Free*" option to be enabled. This will ensure that you are using the free tier of Oracle Cloud. Figure 6 shows the "Configure the Database" section. Leave the rest of the options to be default in this section.
5. In the "Create administrator credentials" section, set up a password for the admin user.

6. Keep the default options in the "Choose network access" section. Figure 7 shows the default options in this section.
7. Keep the defaults for the next sections. Click on the "*Create Autonomous Database*" button. This will create your ATP database. It might take a few minutes for the database to be created and then started. You will be then led to your ATP database's details page.

After these steps, you will have successfuly created an ATP database.

4.1 Creating a non-admin user

Before proceeding, it is advisable to create a non-admin user for your database. In order to create a new user, follow these steps:

1. On the ATP database details page, click on the "*Database Actions*" and then "*Database Users*" from the dropdown. Figure 8 shows the "Database Actions" dropdown.
2. On the "Database Users" page, click on the "*Create User*" button.
3. Fill in the required information. Make sure that your username follows the following convention:

   `s<your_roll_number>`

   So, if your roll number is `24100173`, your username should be `s24100173`. **Make a note of the username and password that you set for this user. You will need these credentials to connect to the database.**
4. Allocate some quota on tablespace data for this user, by clicking on the dropdown and choosing a value (500M suggested). Also, enable the "Graph", "OML", and the "Web Access" toggles for this user. Figure 9 shows an example of creating a user with the preferred options.
5. Once done, click on the "*Create User*" button, and the user will be created.

4.2 Getting the connection wallet

On the ATP database details page, click on the "*Database Connection*" button. This will lead you to a page as shown in Figure 10 . Click on the "*Download Wallet*" button to download the wallet. You will be prompted to set a password for the wallet before downloading the wallet.

Download the wallet, extract, and keep the contents of the wallet in a directory on your local machine. Make note of this path, as this wallet will be pushed to the server later (in section 5.2) to enable the application's communication with the database.

5 Part 3: Cloud VM and application deployment

Your Oracle account allows you to create a maximum of 2 VMs, as part of the benefits of your student account. You will be using one of these VMs to deploy the application. In order to set up the VM, you will need to follow the steps given below.

5.1 Setting up the VM

1. Log in to your Oracle cloud account.
2. You can create a new VM by heading to your "Compute instances" dashboard, or via the hamburger menu -> Compute -> Instances.
3. Input relevant details like your VM's name etc.

4. Make sure that you are choosing **Canonical Ubuntu 22.04**. (not the minimal one).
5. Keep the default shape, however, if it's not available, choose a different shape. **Make sure that you are choosing an x86 arch** based shape (Intel or AMD). Non default shapes will cost you your free credits, so keep an eye on that.
6. Make sure **to save the private key**. Don't lose that, as you won't get it later.
7. Make sure that the permissions on the downloaded private key file are correctly configured, i.e. only you can read it. You can find instructions in the official Oracle docs for your relevant platform here : (for Windows users, instructions for OpenSSH are relevant) `https://docs.oracle.com/en-us/iaas/Content/GSG/Tasks/testingconnection.htm`.
8. Follow the instructions to then correctly configure the file permissions, and to connect to the server. Make sure you can log in to your server, over SSH using the following command (for all OSs)[1] (get the IP address from the created VM's dashboard):

```
$ ssh -i <private-key-path> ubuntu@<ip-address>
```

9. Once logged in, you will be greeted with a Bash shell. First, make sure that the system is up to date by running the following commands:

```
$ sudo apt update
$ sudo apt upgrade
```

   Press `Y` when prompted to install the updates. Most likely, the kernel will also get updated. *If you are greeted with a screen asking about which services to restart, you can just press ENTER to continue.* Once the updates are installed, you will need to reboot the system. You can do this by running:

```
$ sudo reboot
```

10. Once the system reboots, you can log back in and proceed with the next steps. Wait for a few minutes for the reboot to finish, in order to log back in.
11. **Important:** Oracle cloud VMs exist in a Virtual Cloud Network (VCN). By default, the VCN is configured to not allow any incoming HTTP or HTTPS traffic. You will need to configure the VCN to allow incoming traffic on the ports used by these protocols, specifically port 80 for HTTP, and port 443 for HTTPS. These ports can be opened by configuring the Security Lists for the VCN. The following steps will guide you on how to open these ports:
12. On the VM's dashboard, click on link next to **virtual cloud network**, in the Instance details section.
13. Now click on Security Lists on the left navigation bar for the VCN.
14. Click on the Default Security List.

[1]**A little sidenote on conventions:** Throughout the document, you will encounter multiple commands to be run on the terminal. For these commands, the following conventions are used:

(a) `$` is used to denote the terminal prompt (can be a local shell, or the remote shell accessed via SSH). You don't need to type this symbol in.

(b) Anything enclosed in <> is a placeholder. You need to replace the placeholder with the actual value for your case. For example, if you see `ssh -i <path-to-key> ubuntu@<ip-address>`, you need to replace `<ip-address>` with the actual IP address of your VM, and `<path-to-key>` with the path to your private key file. Note that the <> are not to be included in the command.

(c) `#` is used to denote a comment. Anything following a `#` is a comment and is not to be typed in.

15. Here you need to open port 80. Click on + Another Ingress Rule and add the following values as shown below. I believe the values are very much self-explanatory, but in case you have any questions, feel free to ask.

    Source Type: `CIDR`

    Source CIDR: `0.0.0.0/0`

    IP Protocol: `TCP`

    Source Port Range: `All`

    Destination Port Range: `80`

    Click on Add Ingress Rules at the bottom.

16. Similarly, we need to open port 443. Click on + Another Ingress Rule and add the following values as shown below.

    Source Type: `CIDR`

    Source CIDR: `0.0.0.0/0`

    IP Protocol: `TCP`

    Source Port Range: `All`

    Destination Port Range: `443`

    Click on Add Ingress Rules at the bottom.

5.2 Application deployment

The `files/application` directory contains the source code for the application server. The changes to the source code, in order to implement the requirements will be done in this directory.

The `files/setup` directory contains the setup scripts that will help you set up the environment as a web server, that can serve our application on a public IP address.

In order to get started, make sure your Virtual Machine is up and running and you are able to connect to it.

1. Unzip the archive you downloaded from LMS. Begin by copying the entire project directory to your server. For example, use the following command to copy the project directory to your server. This command copies the directory (from your local machine) and pastes it in the home directory of the remote machine. You can of course change where to put the files:

```
$ scp -i <path-to-ssh-key> -r <path-to-project-files-dir> ubuntu@<ip-address>:/
    home/ubuntu
$ ssh -i <path-to-ssh-key> ubuntu@<ip-address> "ls /home/ubuntu"
files/          # the directory containing the project files
...
```

2. Unzip the wallet archive (downloaded in section 4.2). Inside the wallet directory, open the file named `sqlnet.ora`, with any text editor, and replace the value of the key `DIRECTORY=` with `/home/ubuntu/wallet` (keep the quotes). This is the path on the remote server, where this wallet folder will be put. Please only change this value if you decide to put the wallet somewhere else, and modify the `env.sh` accordingly.

3. Next, copy the contents of the wallet (including the updated `sqlnet.ora` file) to the server, placing it inside the `home` directory:

```
$ scp -i <path-to-ssh-key> -r <path-to-wallet-folder> ubuntu@<ip-address>:/home/
    ubuntu/wallet
```

4. Login to your server via SSH, as described in the previous section. In the steps above, commands were being run locally, the commands from the next steps would be run on the server, after logging in via SSH.

```
$ ssh -i <path-to-ssh-key> ubuntu@<ip-address>
```

5. Go to the home directory and generate a self-signed certificate by running the following command. Note that this is the certificate, and the key used for the HTTPS server deployment.

```
$ cd ~
$ openssl req -x509 -newkey rsa:4096 -nodes -out /home/ubuntu/cert.pem -keyout /
    home/ubuntu/key.pem -days 365
$ ls /home/ubuntu/*.pem
/home/ubuntu/cert.pem  /home/ubuntu/key.pem
```

6. Navigate to the project files directory, and make the setup scripts executable by running:

```
$ cd ~/files
$ ls
application  env.sh  setup
$ cd setup
$ chmod +x *.sh
```

7. Execute each setup script one by one as follows:

```
$ ./python_setup.sh
$ ./oic_setup.sh
$ ./nginx_setup.sh
```

8. The server will reboot after running these scripts. Wait for 1–2 minutes, log back in, navigate to the project directory, and update `env.sh` with the correct Database credentials. Note that you can use the `nano` text editor, which works in the terminal too. Open the file using `nano env.sh`. Make sure that the variable `TNS_ADMIN` is correctly set to the directory containing your wallet.

9. Go back to the files directory, and source the `env.sh` file. Also, activate the virtual environment (made by setup scripts):

```
$ ls
application  env.sh  setup
$ nano env.sh
$ source env.sh ; source .venv/bin/activate
```

10. Finally, go to the `application` directory and start the FastAPI server (using either of the following commands):

```
$ cd application
$ fastapi dev app.py    # runs the server in development mode (preferable during
    development)
$ fastapi run app.py    # runs the server in production mode (preferable once
    development is finished)
```

This starts up the application server. If everything was set up correctly, the application should now be live at the link: `https://<vm-server-ip>`, served using `nginx` as the reverse proxy. You can test your deployment by visiting this link and should be greeted with the application welcome page.

You will of course be greeted with SSL certificate errors, which you can safely ignore. Note that this is happening because the server is presenting a self-signed (as done earlier) certificate, which is not trusted. For the purpose of our labs, it is recommended to use Google Chrome, as the testing was done using that browser, and provides a standard for everyone.

6 Part 4: Securing the application

In this part, you will be adding basic security features to the provided application.

6.1 Application details

The application is a minimal web application that serves as the dashboard for the online services provided by an electricity supply company. The dashboard offers three options:

1. **Bill Retrieval**: This allows any user to retrieve the complete bill, with all the relevant details, from the database.
2. **Bill Payment**: This option allows a user to pay the dues against an electricity bill. The option is only meant for a bank cashier, who is supposed to process payments.
3. **Bill Adjustment**: This option allows a user to adjust the bill amount. This option is only meant for a DISCO employee/officer who can adjust someone's bill in case of billing errors.

The current implementation provides all the functional features, however, it is severely lacking in security. All the endpoints are available to everyone on the internet, and thus, anyone can access them, and do malicious activities, such as paying your bill without actually making any payment or making unauthorized adjustments to the bills.

In order to secure the access to application, the application developers decided to use an external identity provider, which we can call **CS473 ID provider** for now. This authentication server, is up and running, and available at the `AUTH_SERVER_IP` address from the `env.sh` file. You will be implementing a primitive version of Oauth2.0 Authorization Code flow, where the application server will redirect the user to the authentication server, and upon successful authentication, the user will be redirected back to the application server, with an authorization token. This token will be used to authenticate the user for the subsequent requests.

Table 1 summarizes the endpoints, their purpose, and the required authorization level.

6.2 Task 1: Implement the authentication and login mechanism

The application doesn't serve its own login page, instead uses a third-party authentication provider (e.g. Google, Meta, etc.) which in this case is CS473 ID provider. The welcome page of the app, sends the user to the `AUTH_SERVER_IP/authenticate`, with extra details such as client-id (which has been pre-registered with the service), and the client's redirect URI. This has been implemented for you already, and the `index.html` file doesn't need any changes.

The ability to register users is a concern of the identity provider and you won't be implementing any such mechanism. For the purposes of testing, the ID provider server includes the following three users:

1. **Username:** `test_u1`, **Password:** `secret`
2. **Username:** `test_u2`, **Password:** `secret`
3. **Username:** `test_u3`, **Password:** `secret`

Once redirected to the Id provider's login page, the user can sign-in with their credentials. Upon successful login, the ID provider uses the client's redirect URI to send an authorization code to the client, which can be

Table 1: A summary of the application interface

Endpoint	**Access Method**	**Description**	**Authorized for**
`/`	GET	The root of the application. Serves a welcome page.	Everyone
`/callback`	POST	This is the endpoint to use, to create a session for a user, and log them in. Also the endpoint, used by the ID provider.	Everyone
`/dashboard`	GET	The home page, or the dashboard, for a logged-in user.	Logged in/authenticated users only.
`/bill-retrieval`	GET	Serves the bill retrieval request page.	Logged in/authenticated users only.
	POST	The bill retrieval request page submits the received form input at this endpoint. The response is a bill details page.	Logged in/authenticated users only.
`/bill-payment`	GET	Serves the bill payment request page.	DISCO Employee and Bank cashier only.
	POST	The bill payment request page submits the received form input at this endpoint. The response is a payment receipt page.	DISCO Employee and Bank cashier only.
`/bill-adjustments`	GET	Serves the bill adjustments request page.	DISCO Employees only
	POST	The bill adjustments request page submits the received form input at this endpoint. The response is an adjustment receipt page.	DISCO Employees only

exchanged for an access token. Table 2 shows the endpoints that are exposed by the authentication server and the expected input.

Table 2: A summary of the authentication interface

Endpoint	**Access Method**	**Description**
`/authenticate`	GET	Serves the authentication page. Allows users to authenticate access to other applications, by logging in.
`/login`	POST	The sign-in page posts its inputs at this endpoint, which validates the credentials and redirects to the client URI, upon successful login.
`/token`	GET	This endpoint is used to receive an access token by providing an authorization code.

A step-by-step guide to the authentication flow is as follows:

1. User lands on the welcome page (of the client/application), and clicks on the sign-in using an external ID provider.

2. The client redirects the user to the sign-in page of the ID provider, with important details like client-id and redirect URI.

3. The user signs in with their credentials.

4. The form is submitted to the ID provider, which verifies the credentials, and sends an authorization code to the client's redirect URI. In our case, this is the `/callback` endpoint, defined in the `application/app.py` file. You will be implementing this endpoint.

5. The `/callback` endpoint uses the authorization code to request an access token from the ID provider. This is done by sending a request to the `/token` endpoint of the ID provider. Include the following JSON data in your request:

   ```
   {
       "code": <code-received-from-id-provider>,
       "client_id": <client-id(defined in app.py)>,
       "client_secret": <client-secret(defined in app.py),
       "grant_type": "authorization_code"
   }
   ```

   Make sure to not change the keys (as well as the `grant_type`), as this is the interface expected by the `/token` endpoint. Both the client ID, and the client secret are defined inside the `env.sh` file, and are included in the `app.py` file.

6. The successful response from the ID provider looks like as follows:

   ```
   {
       "token": <JWT-sent-by-client>,
       "token-type": "Bearer"
   }
   ```

7. Decode the JWT token and extract the username from the `sub` field. Other fields are also included in the token, observe the token structure to understand them. Make sure that the token is also not expired. Finally, make sure to verify the authenticity of the token using the secret key defined in the `env.sh` file. This is done to make sure that the token was indeed signed by the ID provider, and hasn't been tampered with.

6.2.1 Create a user session and set a session cookie

Once the token is successfully validated, the server should create a session for the user. This session should be managed using a session cookie set in the response. The server should keep track of the active sessions internally, either using a database, or some other internal structure. This can be used to provide the ability to sign out and end the session. The documentation at `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies` serves as an excellent introduction to cookies, and how to use them. Make sure to set up your cookies to allow cross-site inclusion too, (which is done by setting the value of the attribute `"samesite"` to `none` instead of the browser default `lax`). You will see in the next project, how this can be exploited for malicious purposes.

Finally, the user should be redirected to the `/dashboard` endpoint, which is meant for the authenticated users only.

Note that this will be all done using the `/callback` endpoint, defined in the `application/app.py` file.

6.2.2 Implement the validate_session() check

If the session cookie is successfully set by the response from `/callback` endpoint of the app, at the user side the authentication token should be automatically included in further requests to the server in order to access the app functionality. The function `validate_session()`, inside the file `application/app.py` should be implemented to check the validity of this token.

The function is defined as follows:

```python
# Validates the session token, and returns a dictionary containing username
async def validate_session(session_token: str | None = Cookie(default=None)):
```

It accepts a session token, derived from the cookie, (or `None` if no cookie was included). This function can be used to protect the application resources meant for the auhtenticated users only, as for example, shown here for the case of the `/dashboard` endpoint (from the same `app.py` file).

```python
@app.get("/dashboard", response_class=HTMLResponse)
async def get_dashboard(request: Request, user: dict = Depends(validate_session)):
```

The `validate_session` function gets called as a dependency automatically upon invocation of this endpoint, and returns a dictionary, containing username of the logged-in user, derived from the token. The function must:

1. Validate the token using the secret key, and the algorithm defined in the `env.sh` file.
2. Make sure that the auth token itself hasn't expired.
3. Make sure that the session corresponding to this user, and token is not inactive.
4. If all conditions are met, the function returns the dictionary, with the username of the authenticated user.
5. The function should however, raise error, or HTTPexception to notify about the authentication failure.

Protect all the endpoints, which are not meant for everyone, as specified in Table 1, using the `validate_session` dependency, using the example for the `/dashboard` endpoint.

6.3 Task 2: Implement the Access Control mechanisms

In the application, our user-based login system only allows access to resources by authenticated users. However, the application has to implement its own internal access control policy, to control "unauthorized" accesses of the authenticated users. For example, a user who is logged in as a customer, should not be able to access the interface to pay or adjust bills. Similarly, the bank cashier, authorized only for payments, should not be able to make adjustments.

6.3.1 Access Control Framework

For the purposes of enforcing the access control policy, the `AccessController` class defined in the file `/application/access_ctrl.py` must be used. The class can enforce a control policy, defined in a configuration file, that can be loaded using the `load_config` method. Your job is to implement the `is_allowed()` method, which accepts a "role" name, and the "resource" name. The method verifies the access of the given role, to the given resource by consulting its permission matrix, which is defined in the configuration file. A sample configuration file is given as follows:

```
[roles]
customer
admin
```

```
[resources]
/dashboard
/checkout
/delete-user

[permissions]
1,1,0
1,1,1
```

The permissions matrix is an $M \times N$ matrix, where M is the number of roles, and N is the number of resources. Therefore, each row specifies access to the protected resources by a given role. (1 if access is allowed, 0 otherwise). Note that the rows specify roles, in the same order defined in the `[roles]` section, and the columns specify which resource, in the same order (from right to left) as defined in the `[resources]` section. So, the above permissions matrix says that the customer role can access all endpoints except the `/delete-user`, while the admin role can access all the endpoints.

We can define the following roles for our application:

1. `bank_cashier`
2. `DISCO_employee`
3. `customer`

You will have to define your own policy, based on these roles and the authorizations as summarized in table 1. Put the policy in an `access.cfg` file in the `application` directory, and then load it inside the `app.py`, before enforcing the policy.

6.3.2 Endpoint protection

Once the `is_allowed` method is implemented, use the method to enforce your access policy for each endpoint, that is not meant for everyone to access. Note that `is_allowed` method takes the role (and not the username), which is not provided to the application. For our purposes, we can create a simple username to role mapping, based on the suffix of the username, specified as follows.

Username Suffix	**Assigned Role**
`_u1` e.g. `test_u1`	Customer
`_u2` e.g. `test_u2`	Bank Cashier
`_u3` e.g. `test_u3`	DISCO employee

Table 3: Username-to-Role Mapping

If the username doesn't match any of the three patterns, an exception must be thrown, specifying that the username is invalid.

7 Submission

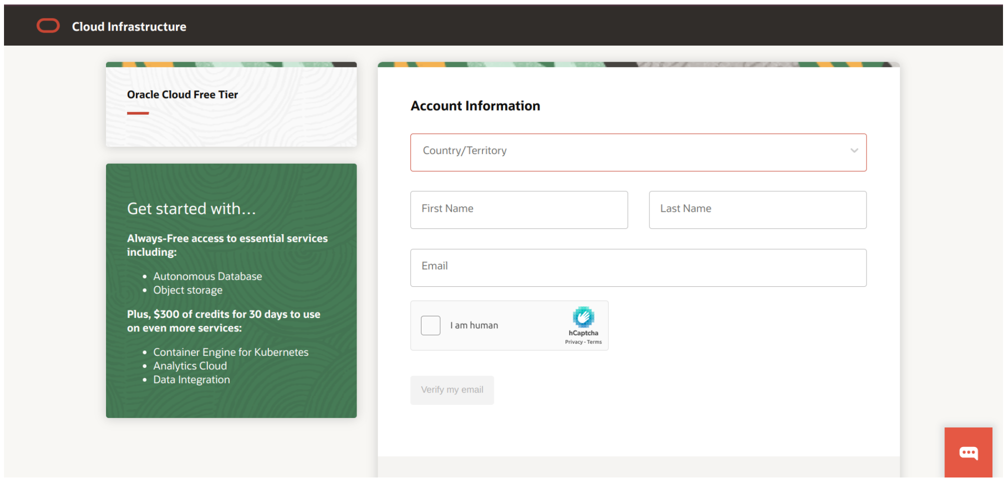
You are required to submit the following via LMS before the deadline:

- A screenshot of the successful deployment of the application, on a valid public IP address, by visiting the application URL in the browser.
- Your updated `application` directory. This includes the updated files in the `templates` directory, and the updated Python files.

Kindly create a ZIP archive of the screenshot, and the updated files and submit it on LMS. The ZIP archive should be named as `<your_roll_number>_lab1.zip`. For example, if your roll number is `24100173`, the ZIP archive should be named `24100173_lab1.zip`.

Appendix

Cloud signup


Cloud Infrastructure

Oracle Cloud Free Tier

Get started with...

Always-Free access to essential services including:

- Autonomous Database
- Object storage

Plus, $300 of credits for 30 days to use on even more services:

- Container Engine for Kubernetes
- Analytics Cloud
- Data Integration

Account Information

Country/Territory

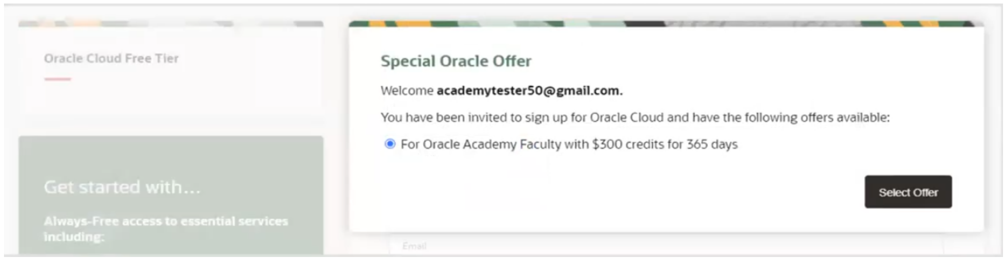
First Name | Last Name

Email

I am human

Verify my email

Figure 2: Signup page


Oracle Cloud Free Tier

Get started with...

Always-Free access to essential services including:

Special Oracle Offer

Welcome **academytester50@gmail.com.**

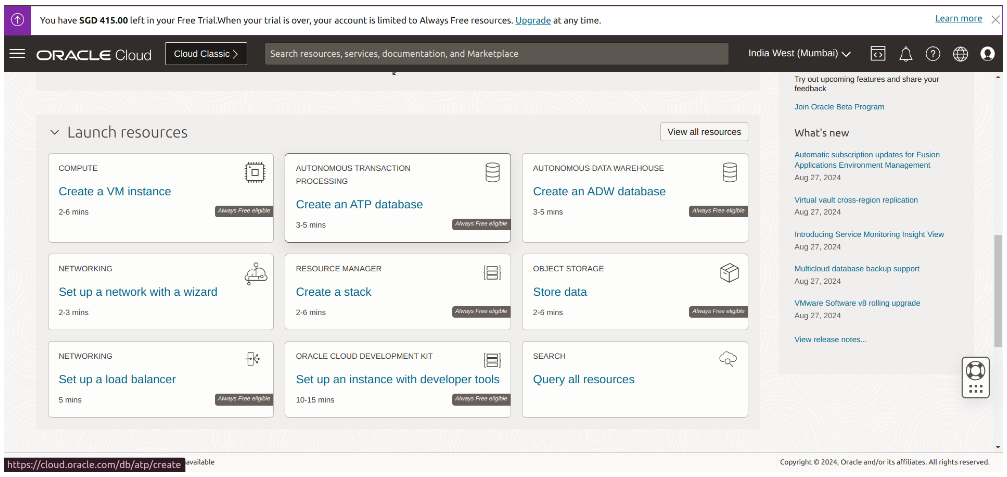
You have been invited to sign up for Oracle Cloud and have the following offers available:

● For Oracle Academy Faculty with $300 credits for 365 days

Select Offer

Figure 3: Offer page

ATP Database creation


You have **SGD 415.00** left in your Free Trial.When your trial is over, your account is limited to Always Free resources. Upgrade at any time. Learn more

ORACLE Cloud | Cloud Classic > | Search resources, services, documentation, and Marketplace | India West (Mumbai)

Launch resources | View all resources

COMPUTE Create a VM instance 2-6 mins Always Free eligible	AUTONOMOUS TRANSACTION PROCESSING Create an ATP database 3-5 mins Always Free eligible	AUTONOMOUS DATA WAREHOUSE Create an ADW database 3-5 mins Always Free eligible
NETWORKING Set up a network with a wizard 2-3 mins	RESOURCE MANAGER Create a stack 2-6 mins Always Free eligible	OBJECT STORAGE Store data 2-6 mins Always Free eligible
NETWORKING Set up a load balancer 5 mins Always Free eligible	ORACLE CLOUD DEVELOPMENT KIT Set up an instance with developer tools 10-15 mins Always Free eligible	SEARCH Query all resources

Try out upcoming features and share your feedback

Join Oracle Beta Program

What's new

Automatic subscription updates for Fusion Applications Environment Management
Aug 27, 2024

Virtual vault cross-region replication
Aug 27, 2024

Introducing Service Monitoring Insight View
Aug 27, 2024

Multicloud database backup support
Aug 27, 2024

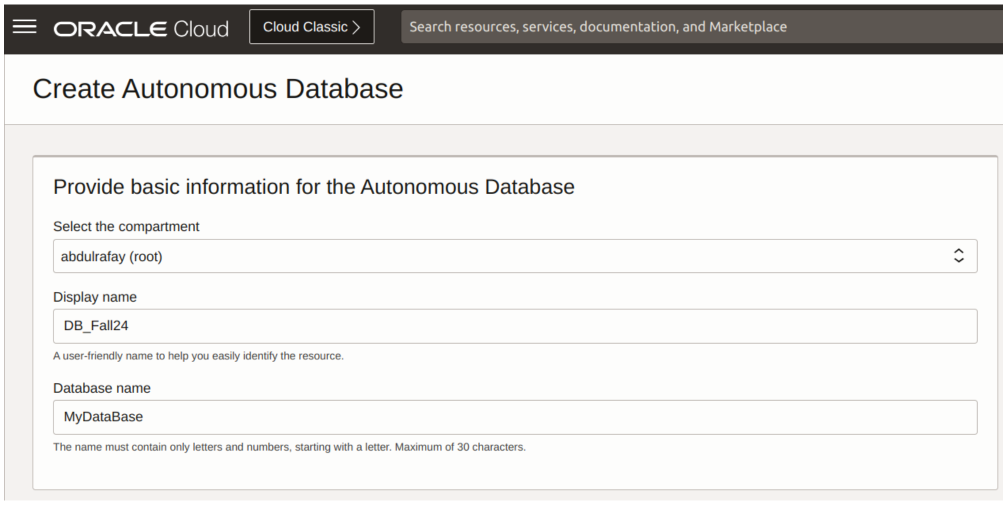
VMware Software v8 rolling upgrade
Aug 27, 2024

View release notes...

https://cloud.oracle.com/db/atp/create available

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

Figure 4: Cloud account Get Started page



Create Autonomous Database

Provide basic information for the Autonomous Database

Select the compartment: abdulrafay (root)

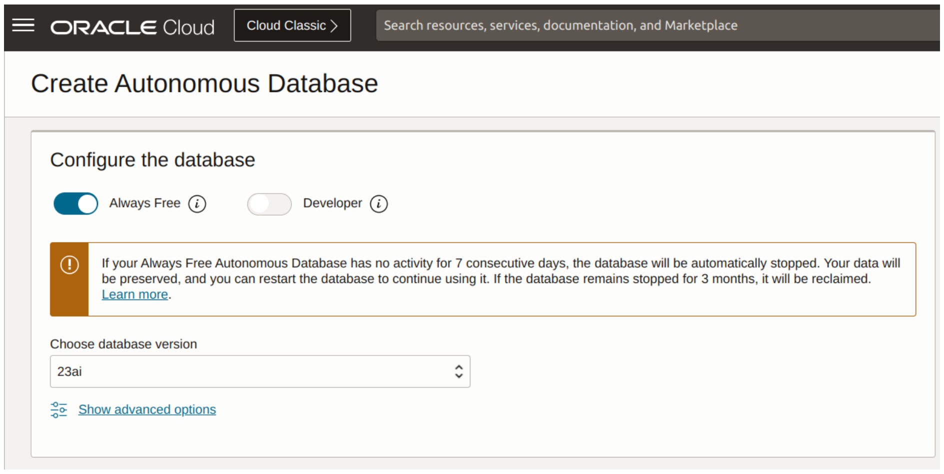
Display name: DB_Fall24

A user-friendly name to help you easily identify the resource.

Database name: MyDataBase

The name must contain only letters and numbers, starting with a letter. Maximum of 30 characters.

Figure 5: ATP Database creation



Create Autonomous Database

Configure the database

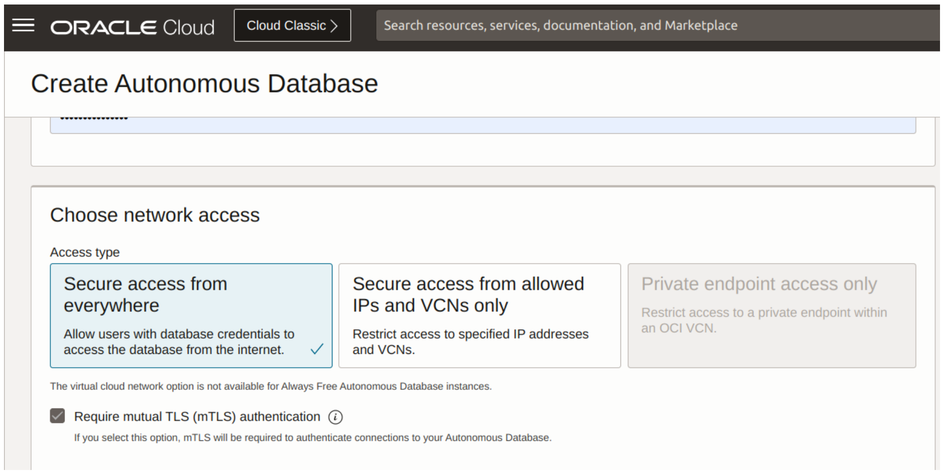
Always Free [✓] Developer

If your Always Free Autonomous Database has no activity for 7 consecutive days, the database will be automatically stopped. Your data will be preserved, and you can restart the database to continue using it. If the database remains stopped for 3 months, it will be reclaimed. Learn more.

Choose database version: 23ai

Show advanced options

Figure 6: Configure the DB to use free resources



Create Autonomous Database

Choose network access

Access type

[✓ Secure access from everywhere — Allow users with database credentials to access the database from the internet.]

Secure access from allowed IPs and VCNs only — Restrict access to specified IP addresses and VCNs.

Private endpoint access only — Restrict access to a private endpoint within an OCI VCN.

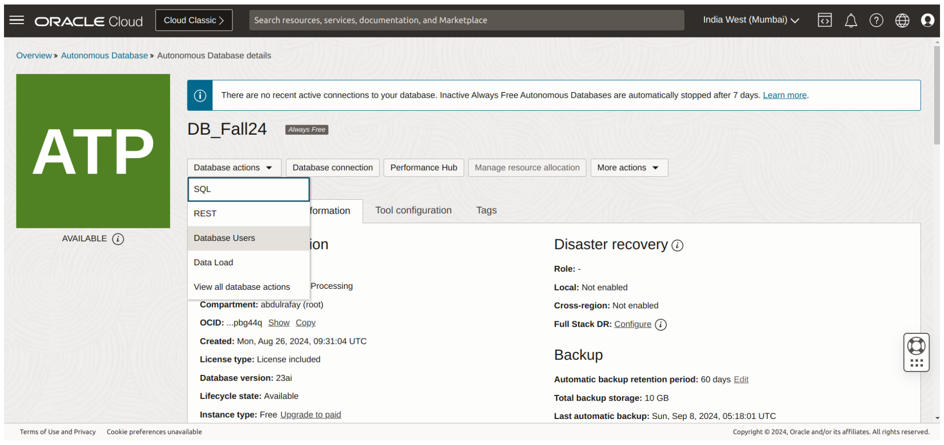
The virtual cloud network option is not available for Always Free Autonomous Database instances.

[✓] Require mutual TLS (mTLS) authentication

If you select this option, mTLS will be required to authenticate connections to your Autonomous Database.

Figure 7: Configure network access

Creating a non-admin user


ORACLE Cloud
Cloud Classic
Search resources, services, documentation, and Marketplace
India West (Mumbai)
Overview » Autonomous Database » Autonomous Database details
ATP
AVAILABLE
Database actions
Database connection
Performance Hub
Manage resource allocation
More actions
SQL
REST
Database Users
Data Load
View all database actions
formation
Tool configuration
Tags
Terms of Use and Privacy
Cookie preferences unavailable
Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

There are no recent active connections to your database. Inactive Always Free Autonomous Databases are automatically stopped after 7 days. Learn more.

DB_Fall24 Always Free

ion

Processing

Compartment: abdulrafay (root)

OCID: ...pbg44q Show Copy

Created: Mon, Aug 26, 2024, 09:31:04 UTC

License type: License included

Database version: 23ai

Lifecycle state: Available

Instance type: Free Upgrade to paid

Disaster recovery

Role: -

Local: Not enabled

Cross-region: Not enabled

Full Stack DR: Configure

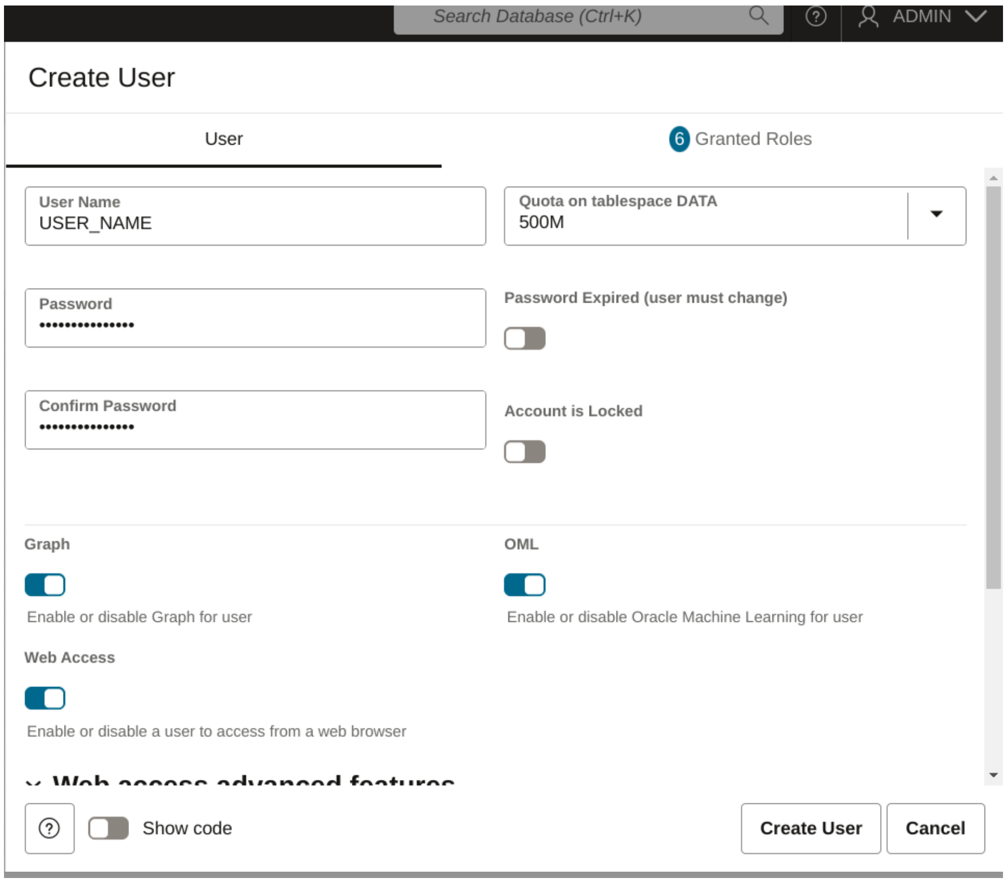
Backup

Automatic backup retention period: 60 days Edit

Total backup storage: 10 GB

Last automatic backup: Sun, Sep 8, 2024, 05:18:01 UTC

Figure 8: DB Actions dropdown



Create User

User Name
USER_NAME

Quota on tablespace DATA
500M

Password
•••••••••••••••

Password Expired (user must change)

Confirm Password
•••••••••••••••

Account is Locked

Graph

Enable or disable Graph for user

OML

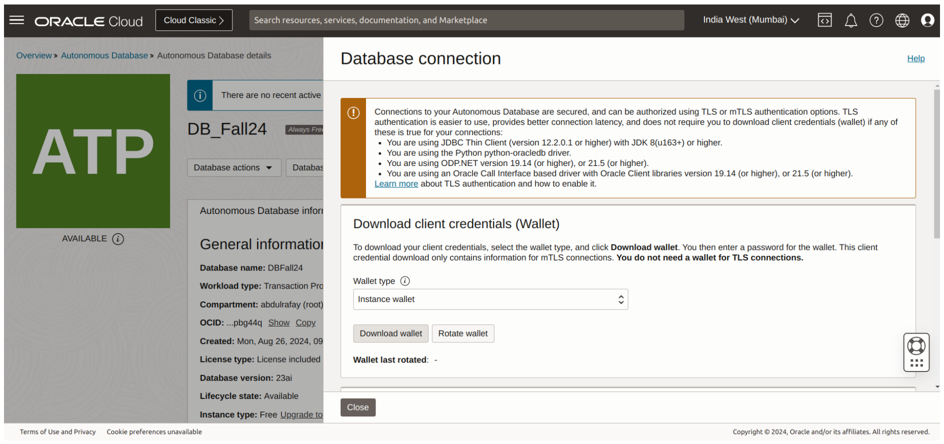
Enable or disable Oracle Machine Learning for user

Web Access

Enable or disable a user to access from a web browser

Web access advanced features

Figure 9: Create a new user


ORACLE Cloud
Cloud Classic
Search resources, services, documentation, and Marketplace
India West (Mumbai)
Overview » Autonomous Database » Autonomous Database details
ATP
AVAILABLE
There are no recent active
DB_Fall24
Database actions
Autonomous Database infor
Help
Download wallet
Rotate wallet
Close
Terms of Use and Privacy
Cookie preferences unavailable
Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

General information

Database name: DBFall24
Workload type: Transaction Pro
Compartment: abdulrafay (root)
OCID: ...pbg44q Show Copy
Created: Mon, Aug 26, 2024, 09
License type: License included
Database version: 23ai
Lifecycle state: Available
Instance type: Free Upgrade to

Database connection

Connections to your Autonomous Database are secured, and can be authorized using TLS or mTLS authentication options. TLS authentication is easier to use, provides better connection latency, and does not require you to download client credentials (wallet) if any of these is true for your connections:

- You are using JDBC Thin Client (version 12.2.0.1 or higher) with JDK 8(u163+) or higher.
- You are using the Python python-oracledb driver.
- You are using ODP.NET version 19.14 (or higher), or 21.5 (or higher).
- You are using an Oracle Call Interface based driver with Oracle Client libraries version 19.14 (or higher), or 21.5 (or higher).

Learn more about TLS authentication and how to enable it.

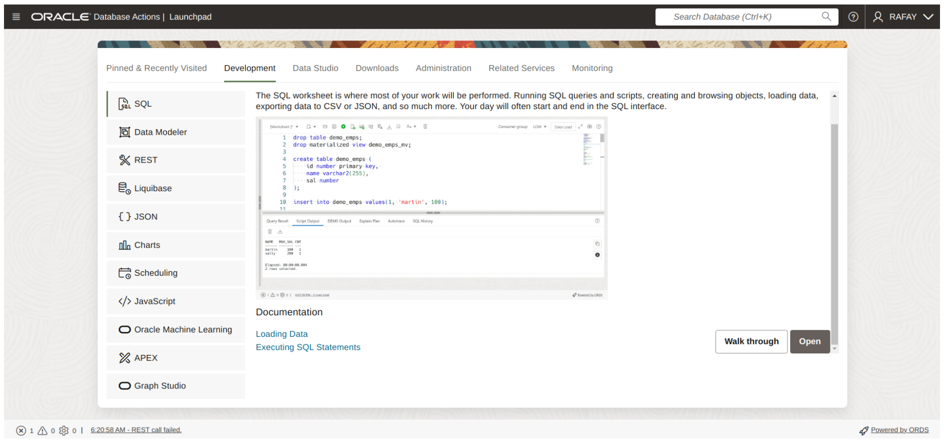
Download client credentials (Wallet)

To download your client credentials, select the wallet type, and click **Download wallet**. You then enter a password for the wallet. This client credential download only contains information for mTLS connections. **You do not need a wallet for TLS connections.**

Wallet type: Instance wallet

Wallet last rotated: -

Figure 10: Getting the Connection Wallet



The SQL worksheet is where most of your work will be performed. Running SQL queries and scripts, creating and browsing objects, loading data, exporting data to CSV or JSON, and so much more. Your day will often start and end in the SQL interface.

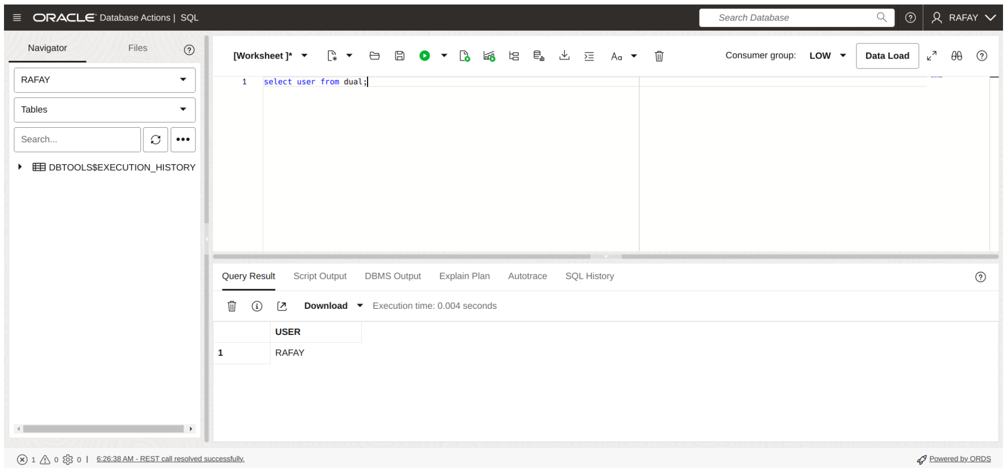
```
drop table demo_emps;
drop materialized view demo_emps_mv;

create table demo_emps (
    id number primary key,
    name varchar2(255),
    sal number
);

insert into demo_emps values(1, 'martin', 100);
```

6:20:58 AM - REST call failed.

Figure 11: Oracle Cloud Web Interface and Developer tools



```
select user from dual;
```

Execution time: 0.004 seconds

	USER
1	RAFAY

6:26:38 AM - REST call resolved successfully.

Figure 12: Web Interface SQL Worksheet