Deploying and securing a Flask app on OCI Compute

Liana Lixandru

2024-02-19

Securing applications on Compute in OCI

Creating an Always Free compute instance

The first step in our exercise is to create an Always Free instance on Compute where we can build and run our application.

Creating a VCN

1. Open the navigation menu, click **Networking**, and then click **Virtual cloud networks**.
2. Under **List Scope**, select a compartment that you have permission to work in.
3. Click **Start VCN WIzard**, and select **Create VCN with Internet Connectivity**.
4. Enter a name (`vcn-flask-demo`) and a CIDR Block (ex. `172.16.0.0/24`).

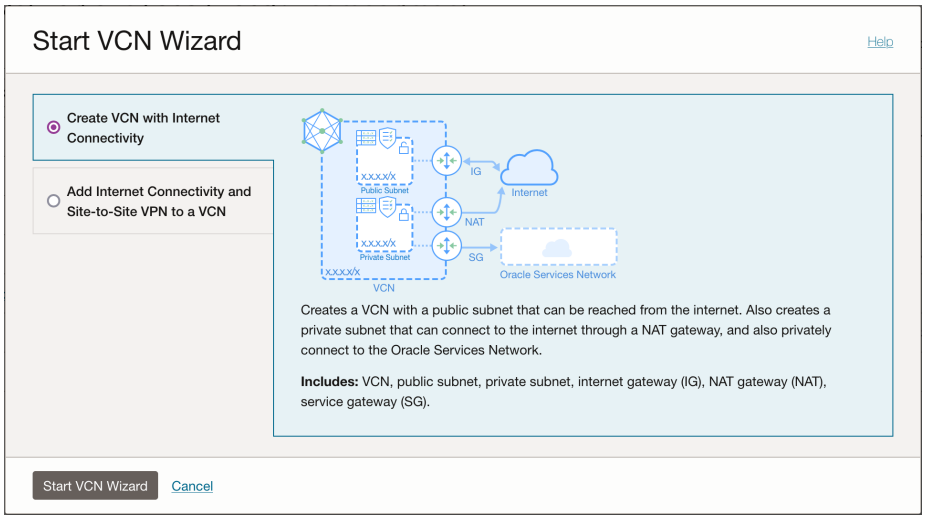

Figure 1: Running the VCN Wizard

5. Add the CIDR Blocks for the two subnets:

- **Public**: `172.16.0.0/26`
- **Private**: `172.16.0.64/26`

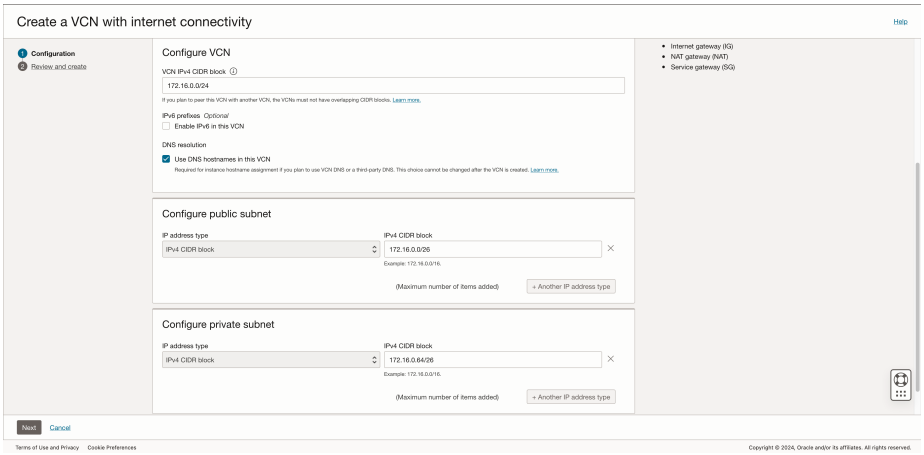

Figure 2: Adding CIDR Blocks for the two subnets

5. Click on **Next**, and then **Create**.

Once the creation is complete, click on View VCN to see the newly created resources.

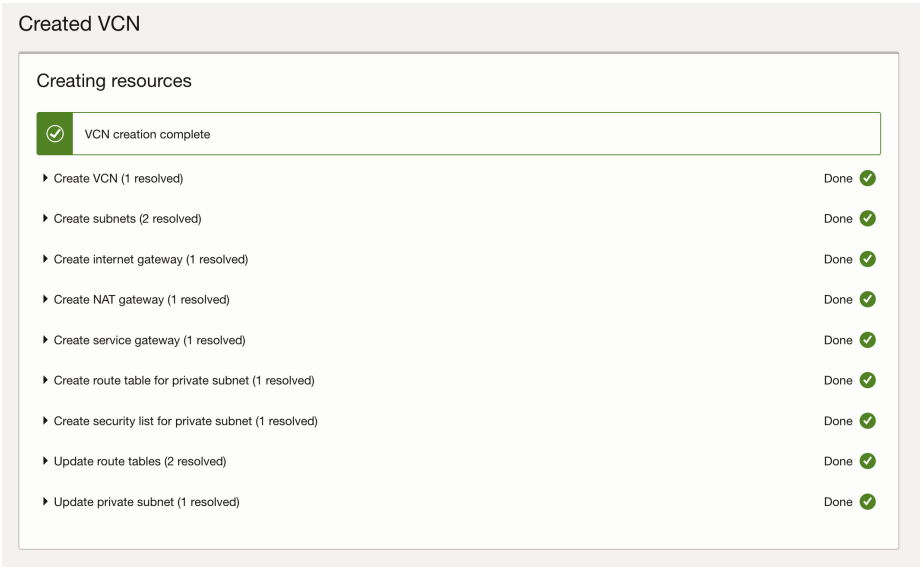

Figure 3: VCN creation complete.png

6. Navigate to the Security List and add ports tcp/80 and tcp/443 to the allowed ports.

Creating a private compute instance And now finally we can create our Compute instance. For that, we will open the menu in the OCI console,

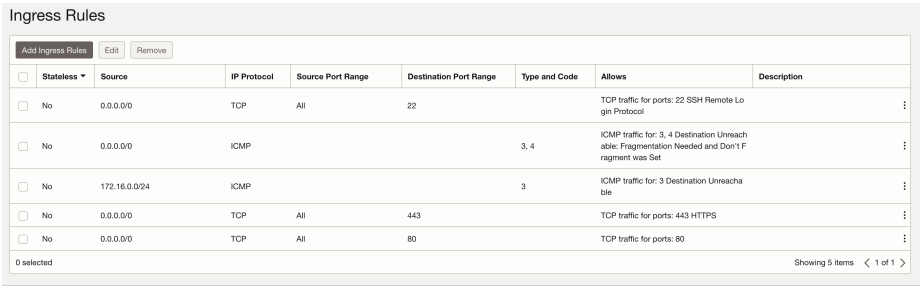

Figure 4: Open up ports 80 and 443

navigate to **Compute - Instances** and then click on **Create Instance**.

Give the instance a name, for example `flask-demo`, then select the image and shape corresponding to Ampere A1 / aarch64. If needed, increase the number of core OCPUs and the memory.

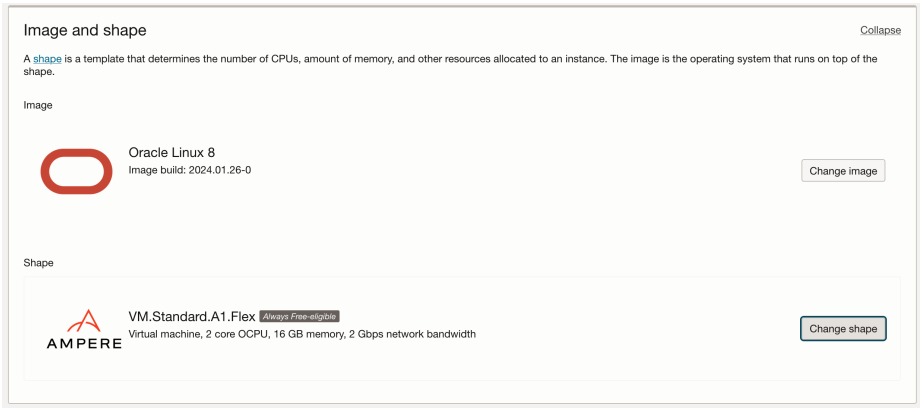

Figure 5: Select the image and shape

Make sure our newly created VCN and the private subnet are selected in the list.

Download the generated SSH keys or add your own to the instance.

And then click on **Create** at the bottom. Your instance will soon be provisioned and we can use it to work on our application.

Connecting to our compute instance through a Bastion In order to connect to our private instance, we will need to set up a tunnel through the Bastion service. Bastions let you create and manage sessions that provide authenticated users with ephemeral, timebound access to resources in the tenancy. They establish secure bridge connections from preconfigured IP addresses to supported target hosts that do not have a public IP address, such as compute instances running an OpenSSH server or autonomous transaction processing databases that support SSH tunneling to an arbitrary port.

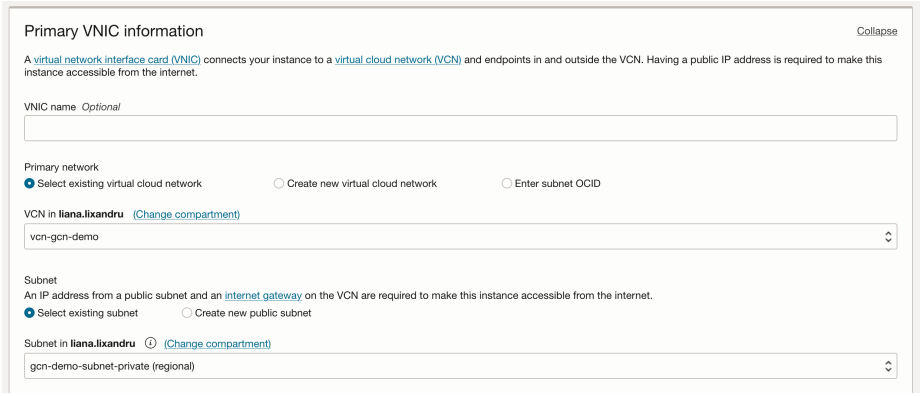

Figure 6: Creating a Compute Instance - selecting the VCN

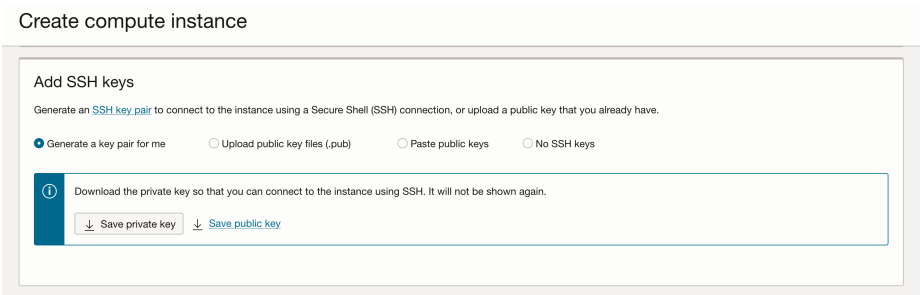

Figure 7: Creating a Compute Instance - saving your SSH keys

Navigate to **Compute - Instances**, click on our freshly-created instance and then move to the **Oracle Cloud Agent** tab. Enable the Bastion plugin.

> It can take up to 10 minutes to enable the Bastion plugin.

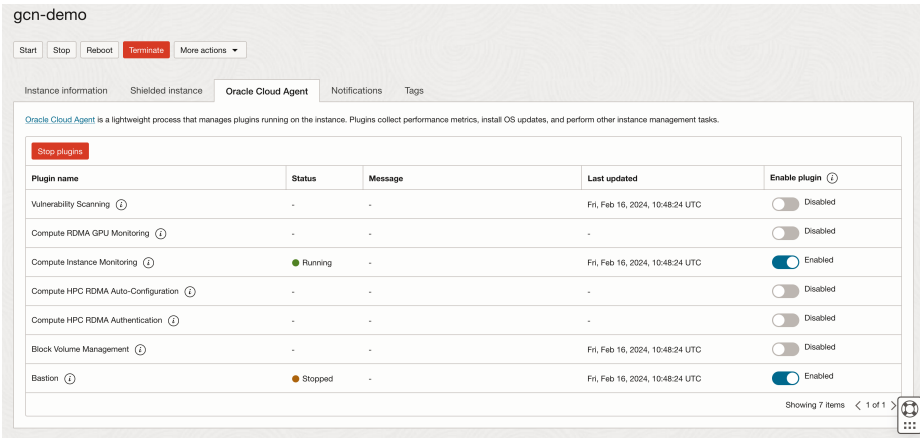

Figure 8: Enabling the Bastion plugin on the Compute instance

Once that is done, in order to set a Bastion up, we need to open the navigation menu, click on **Identity & Security**, and then on **Bastion**. Click on **Create Bastion** on the newly-opened page.

Give it a name, then select our freshly created VCN and subnet from the list, and add `0.0.0.0/0` to the CIDR block allowlist, then click on **Create bastion**.

> To secure your access even further, you can only add your public IP address here, so that your computer is the only one allowed to connect to this bastion.

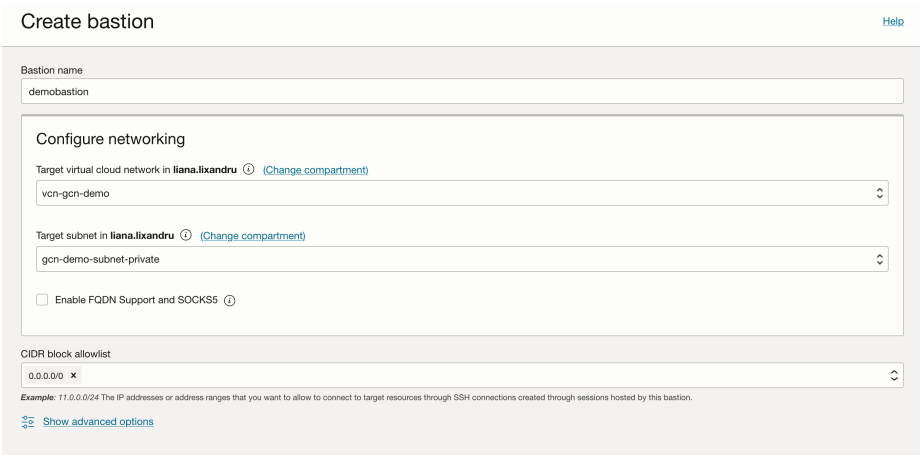

Figure 9: Creating a Bastion

Once the bastion is active, click on it and then on **Create Session** so we can open a session to our compute instance.

- **Session type**: Managed SSH session
- **Session name**: give it a new name or leave it as default
- **Username**: ubuntu (we used an Ubuntu image for our instance)
- Create and download new SSH keys for this session in the Add SSH key section, or add a SSH key that you already have.

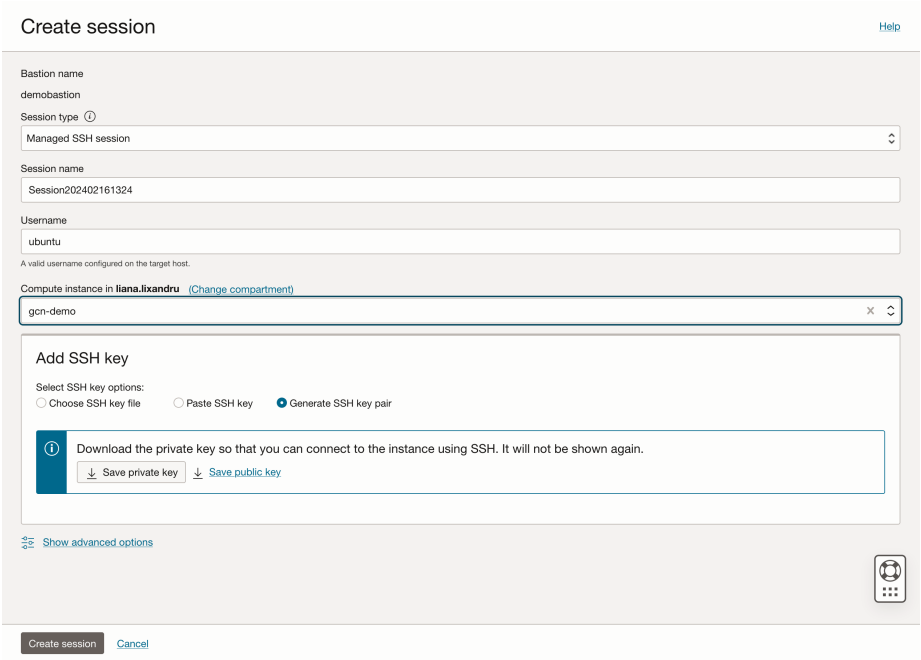

Figure 10: Creating the session

Once the session has been created, we can copy the SSH command directly from the menu.

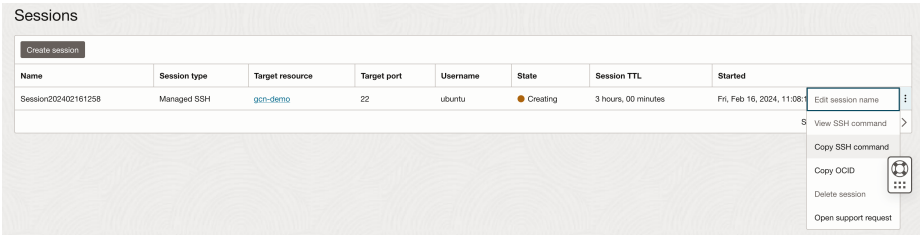

Figure 11: Copying the Bastion SSH command

Connecting from Visual Studio Code In VS Code, navigate to Remote SSH, and open up the command palette. Select Open SSH Configuration File from the list so we can add our connection in.

In our `.ssh/config` file, we need to add our details in as it follows:

```
Host flask-demo-bastion
    HostName [PRIVATE IP]
```

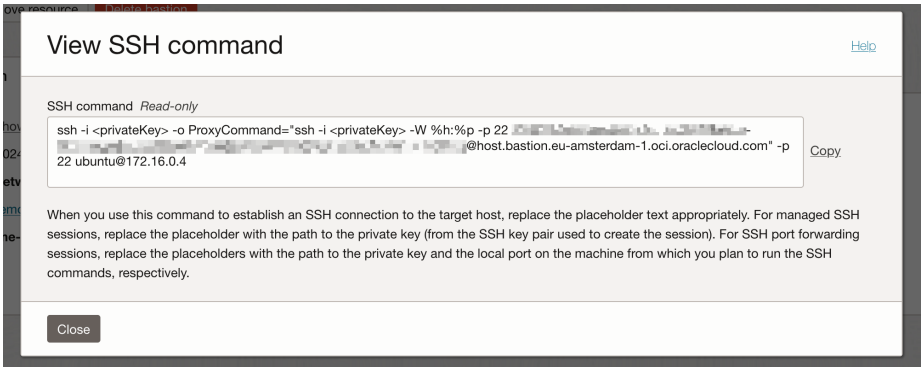

Figure 12: Viewing the SSH command

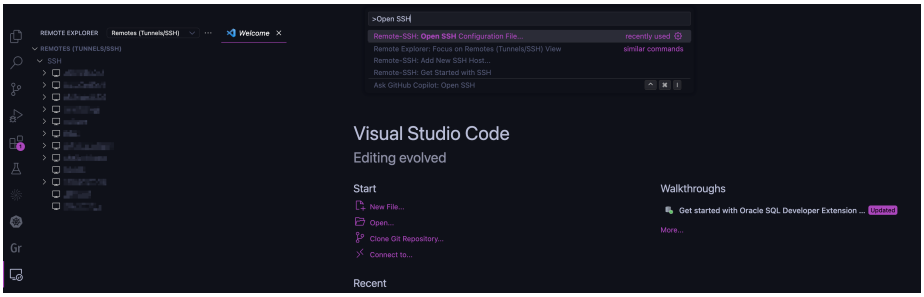

Figure 13: Open SSH configuration file.png

```
    User ubuntu
    IdentityFile ~/.ssh/bastion-key
    ProxyCommand ssh -i ~/.ssh/bastion-key -W %h:%p -p 22 ocid1.bastionsession.oc1.[REGION
```

Save the file and close it, then refresh the Remotes list. Our newly added connection should appear right at the top. Click on **Connect in current window**.

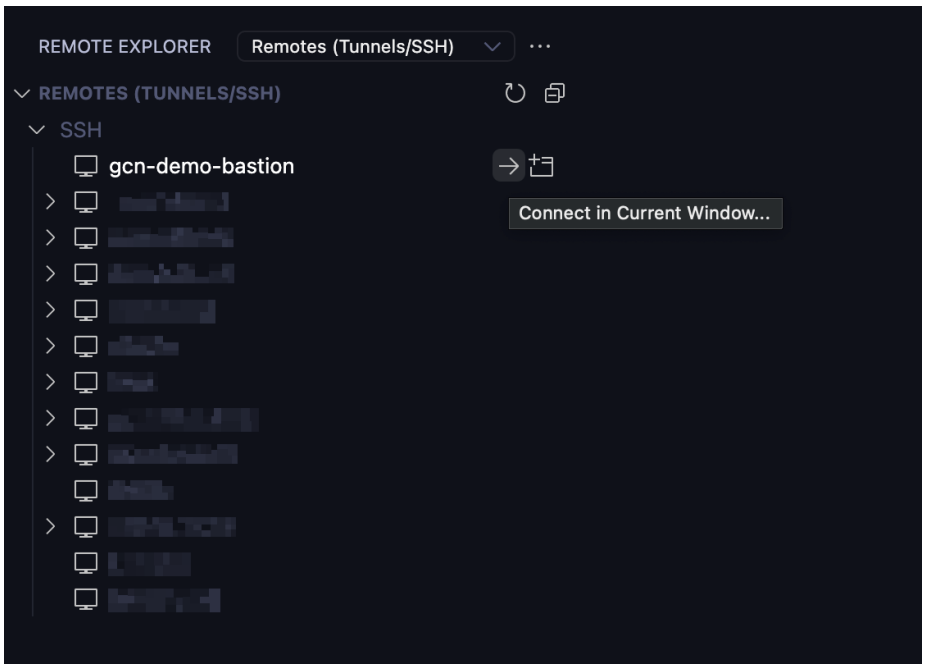

Figure 14: Connect in current window

Editing and building our application

1. Now we can clone the repository containing the source code of our app.

```
git clone https://github.com/llixandru/oci-upload-object-storage-python
```

And follow the guide in the README to set the app up.

Create an API key for testing

1. Navigate to your user profile in the OCI console and click on API Keys.
2. Click on Add API Key and save your private and public keys locally.
3. Copy the details shown in the preview.
4. Create a config file in the VS Code terminal, in your home directory.

```
cd
mkdir .oci
```

5. Upload your private key to the compute instance and place it in the .oci folder.

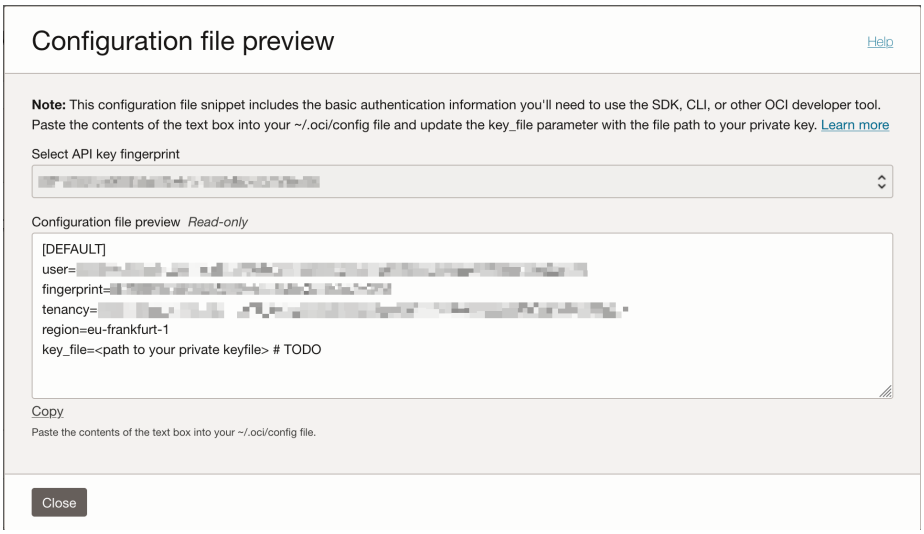

Figure 15: Configuration file preview

```
scp /path/to/oci_api_key.pem flask-demo:/home/ubuntu/.oci/
```

Then, on the compute instance, change the permissions on the key.

```
chmod 400 ~/.oci/oci_api_key.pem
```

6. In VS Code, create a new file. Paste the details of the configuration inside. Set the path to `~/.oci/config` and save it.

Configure the Project For this section, we will need an Oracle Cloud Infrastructure account and an Object Storage bucket created. Let's create the bucket.

1. Open the navigation menu and click **Storage**. Under **Object Storage & Archive Storage**, click **Buckets**.
2. Select the compartment from the list under **List Scope**. All buckets in that compartment are listed in tabular form. This is the compartment where the bucket you create is located.
3. Click **Create Bucket**. The **Create Bucket** dialog box appears.
4. Complete the following:
   - **Bucket Name**: The system generates a default bucket name that reflects the current year, month, day, and time, for example `bucket-20190306202303064****-1359`. If you change this default to any other bucket name, use letters, numbers, dashes, underscores, and periods. Avoid entering confidential information.
   - **Default Storage Tier**: Standard
5. Click **Create**.

The bucket is created immediately and you can start uploading objects to it.

6. In our code, back in VS Code, update the env file (`.env`) with the name of the bucket:

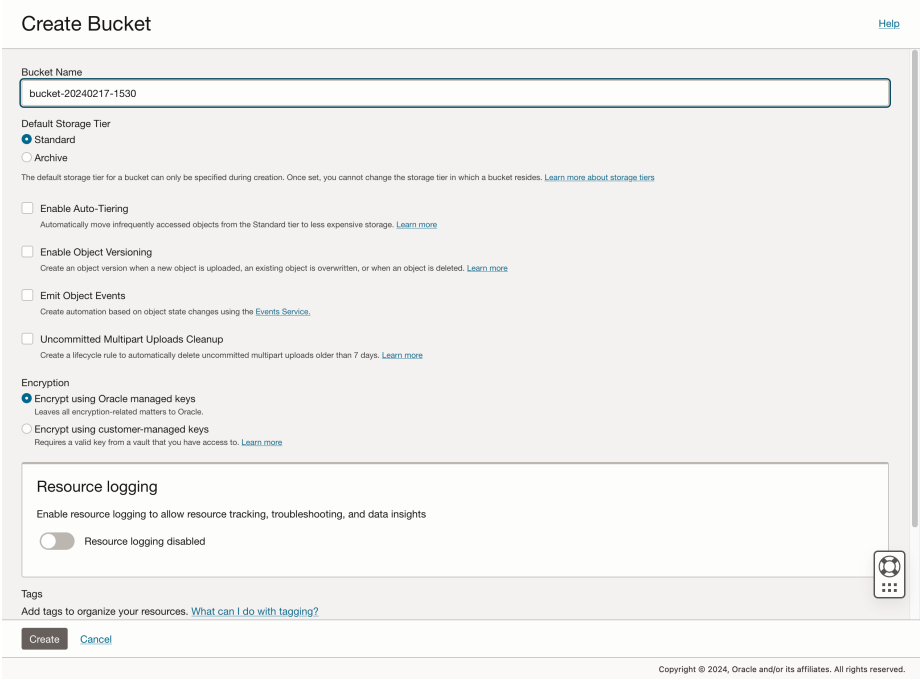

Figure 16: Bucket creation

```
BUCKET_NAME=${OBJECT_STORAGE_BUCKET}
```

Replace `${OBJECT_STORAGE_BUCKET}` with the name of your bucket.

Run the Application To run and test our app, we need to execute the following commands in the folder:

```
python -m venv venv

source venv/bin/activate

pip install -r requirements.txt

python3 app.py
```

The app will start on port `5000`.

Test the Application Locally, in a terminal, let's copy a photo over to the compute instance so we can test our APIs.

```
scp ~/Downloads/my_picture.jpg flask-demo:/home/ubuntu/
```

1. Upload a picture

```
curl -F "fileUpload=@/home/ubuntu/my_picture.jpg" http://localhost:5000/pictures/avatar.jp
```

2. Download a picture

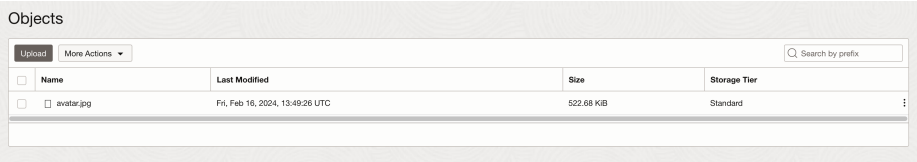

Figure 17: The uploaded picture being shown in the Bucket

```
curl -X GET http://localhost:5000/pictures/avatar.jpg -o downloaded_file.jpg
```

3. Delete a picture

```
curl -X DELETE http://localhost:5000/pictures/avatar.jpg
```

Deploy the Application

1. Create a systemd service file. This file will define the service and how systemd should handle it. The file should be placed in `/etc/systemd/system/` directory and should have a `.service` extension.

```
sudo apt install nano && sudo nano /etc/systemd/system/flask-storage-app.service
```

2. In the service file, define the service using the following template:

```
[Unit]
Description=Gunicorn instance to serve Flask App
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/path/to/your/app
Environment="PATH=/path/to/venv/bin:$PATH"
ExecStart=/path/to/venv/bin/gunicorn --workers 4 wsgi:app

[Install]
WantedBy=multi-user.target
```

Save and close the file (`ctrl+x`, `Y` and `Enter`).

3. Enable and start the service:

```
sudo systemctl daemon-reload

sudo systemctl enable flask-storage-app

sudo systemctl start flask-storage-app

systemctl status flask-storage-app
```

4. Update the private subnet security list to open up port tcp/8000, on which our Gunicorn app listens.

[OPTIONAL] Cloning and creating a new compute instance

In order to clone our work and quickly provision a second compute instance running the same app, we need to follow a few simple steps.

1. Navigate to **Compute** - **Instances** and open our instance.
2. Click on **More Actions** - **Create custom image**, give the image a name and click **Create**. The instance will be temporarily taken down while the custom image is being created.
3. On the **Instances** page, click on **Create instance**. Select our custom image from the list, and add the same SSH key that we used for the previous instance creation.
4. Make sure to enable the Bastion plugin in the Oracle Cloud Agent tab of the new instance if you want to SSH into it later.

Creating a private load balancer

1. In the navigation menu, click **Networking**, then click **Load Balancers**.
2. Click **Create Load Balancer**.
3. Visibility Type: Click **Private Load Balancer**. The LB receives a private IP address from the hosting subnet. The LB acts a front end for internal incoming traffic visible only within the VCN.
4. **Subnet**: Select the name of the VNC and Subnet for the LB from the pull-down menus.
5. Click **Create Load Balancer**.

Secure the function

API Gateway We will create an API Gateway deployment that will call our function. This deployment will be accessible via REST APIs.

The API Gateway service enables you to publish APIs with private endpoints that are accessible from within your network, and which you can expose with public IP addresses if you want them to accept internet traffic. The endpoints support API validation, request and response transformation, CORS, authentication and authorization, and request limiting.

Using the API Gateway service, you create one or more API gateways in a regional subnet to process traffic from API clients and route it to back-end services. You can use a single API gateway to link multiple back-end services (such as load balancers, compute instances, and OCI Functions) into a single consolidated API endpoint.

Accessing our function through API Gateway can quickly and efficiently enhance security for our in several ways:

- **Access Control**: OCI allows you to define security rules and access control policies at the network level using security lists, network security groups, and service gateways.
- **Centralized Access Management**: API Gateway can serve as a centralized entry point for accessing your internal services. You can implement authentication and authorization mechanisms, such as API keys, OAuth tokens, or IAM policies, to control access to your internal APIs.

- **Traffic Inspection and Management**: API Gateway provides features for traffic inspection, request transformation, and rate limiting. You can enforce security policies such as request validation, payload encryption, and content filtering at the API Gateway layer before forwarding requests to your internal services. This adds an additional layer of protection against malicious attacks and helps ensure that only valid and authorized requests reach your backend services.
- **Logging and Monitoring**: API Gateway offers built-in logging and monitoring capabilities, allowing you to track and audit incoming requests, response times, and error rates.

Overall, API Gateway in OCI provides a secure and scalable solution for exposing and managing access to your internal services, while also enabling fine-grained control over security policies, access management, and traffic handling.

Creating an API Gateway

1. Open the navigation menu and click **Developer Services**.
2. Under **API Management**, click **Gateways**.
3. Click **Create Gateway** and specify:
   - a name for the new gateway, such as `apigw-flask-demo`
   - the type of the new gateway as **Public**
   - the name of the compartment in which to create API Gateway resources
   - the name of the VCN to use with API Gateway (we will use the same VCN as the one our cluster is on)
   - the name of the public subnet in the VCN

Creating a deployment on API Gateway Now, we need to move on to the next part of securing our application, which is creating the API Gateway deployment. We will first create it with anonymous access, and then easily and quickly secure it via OAuth2.

1. Navigate to **Developer Services - API Gateway - Gateways**. Click on our created API Gateway and the **Deployments** tab, which will allow us to create a new deployment.
2. Click on **Create deployment**. Fill in a name for our deployment, and the path prefix, for example `/pictures`.
3. Click on next. This is where we can configure Authentication. We will save this for later, to check how access to our access changes once we add OAuth2 as a security measure.
4. In the **Routes** tab, we need to create our route to the function.
   - **Path**: `/{userId}/{filename}`
   - **Methods**: POST, GET, DELETE
   - **Backend Type**: HTTP
   - **URL**: `http://<compute-or-load-balancer-private-ip>:8000/${request.path[userId]}/$`
5. Click on **Next**, then **Create**. We can now see and copy the publicly accessible endpoint for our application.

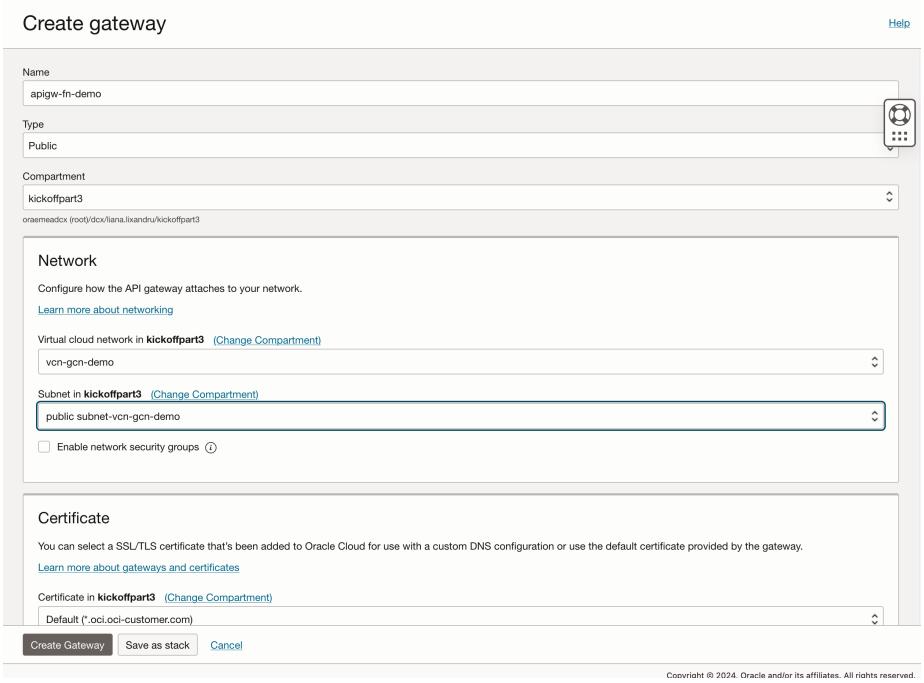

Figure 18: Create a new API Gateway

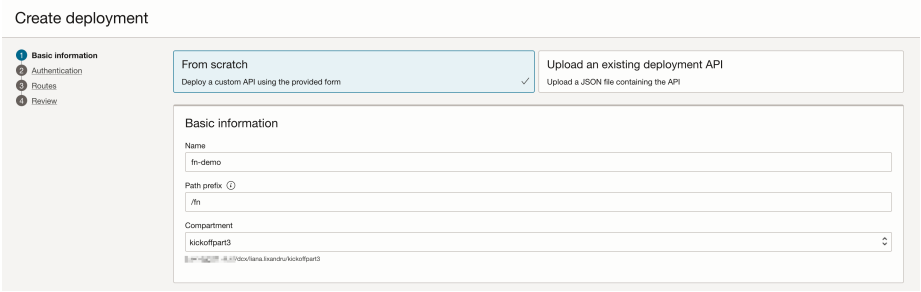

Figure 19: Creating a new deployment on API GW

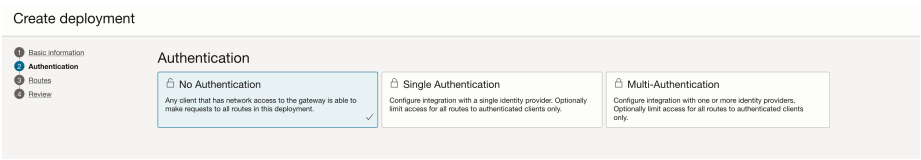

Figure 20: API GW Authentication options

Figure 21: Active deployment on API GW

Testing via REST Once the deployment is active we can copy the endpoint and call our function via REST.

```
curl -F "fileUpload=@/path/to/image/avatar.jpg" https://[API_GW_ID].apigateway.[region].oc

curl https://[API_GW_ID].apigateway.[region].oci.customer-oci.com/pictures/john/avatar.jpg

curl -X "DELETE" https://[API_GW_ID].apigateway.[region].oci.customer-oci.com/pictures/joh
```

OAuth2 OAuth 2.0 provides a robust and standardized framework for securing API calls, protecting sensitive data, and ensuring a positive user experience. It is widely used in modern web and mobile applications to enable secure access to resources while maintaining user privacy and control.

Let's see how we can add it to our API Gateway deployment and how our calls to the OCI Function will change.

Securing the application with IDCS and OAuth2 API Gateway allows us to set up authentication using either a custom code that we can write in Oracle Functions, or an existing identity provider, such as Oracle IAM, Auth0, and so on.

> Let's look at how we can set up an OAuth2 authentication flow in IAM for our newly created endpoint.

Prerequisites for using JWT Tokens

When enabling authentication and authorization using JWTs, you must consider the following:

You need an identity provider (Auth0, Oracle IAM, etc) that can issue JWT tokens for client applications. In API Gateway, you'll have a choice between using Remote JWKS (JSON Web Key Set) or Static Key in the authentication policy for the validation of JWTs:

- **Remote JWKS** will be retrieving the public verification keys from the identity provider at runtime
- **Static Keys** will be using public verification keys already issued by an identity provider and API Gateway will be able to verify JWTs locally without having to contact the identity provider

  In this example, we'll be using **Static Keys**

Obtain JWKS Static Key from IDCS

Permission required for this step only: Oracle IAM Administrator

Before creating the Authentication Policy in API Gateway, we need to obtain the JWKS Static Key from Oracle IAM.

1. From the Oracle IAM Console – **Identity & Security** Menu - **Federation - OracleIdentityCloudService**, click on the *Oracle Identity Cloud Service Console* link:
2. In Oracle IAM, go to **Settings - Default Settings** and **Toggle ON** the **Access Signing Certificate** option, in case it's not activated already.
3. Get the JWKS from IAM.

`https://idcs-1234xxx.identity.oraclecloud.com/admin/v1/SigningCert/jwk`

When accessing the URL above, you'll get a JSON file with the key as a response – save the response somewhere as we'll need it later.

> Once you've retrieved the JWKS Key, go back to Oracle IAM and **Toggle OFF** the Access Signing Certificate option to prevent unauthorized access.

Create the Authentication Policy Edit the existing API Deployment

Edit the deployment created earlier.

Add an Authentication Policy

In the API Request Policies section of the deployment click **Add** next to **Authentication:**

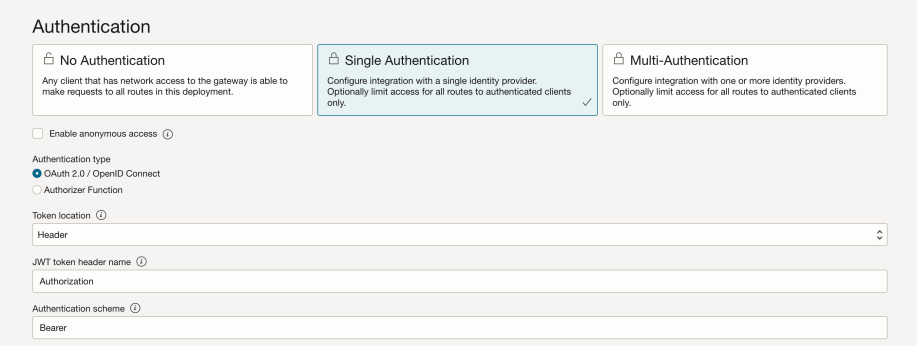

Figure 22: Adding an Authentication Policy

Configure Authentication Policy

Configure the policy as follows:

- **Authentication Type**: JWT
- **Authentication Token**: Header
- **Authentication Scheme**: Bearer
- **Issuers**: https://identity.oraclecloud.com/
- **Audiences**: Specify a value that is allowed in the audience (aud) claim of a JWT to identify the intended recipient of the token. For this example, we'll be setting the audience as `apigw-flask-demo`
- **Type**: Static Keys

- **KEY ID**: SIGNING_KEY
- **Format**: JSON Web Key

Example:

```
{ "kid": "SIGNING_KEY", "kty": "RSA", "use": "sig", "alg": "RS256",
"n": "abcd1234xxxx", "e": "AQAB" }
```

All the values for these parameters can be found in the JWKS we saved earlier. Replace the values with the ones for your instance.

For more info on what each field represents – please check the documentation.

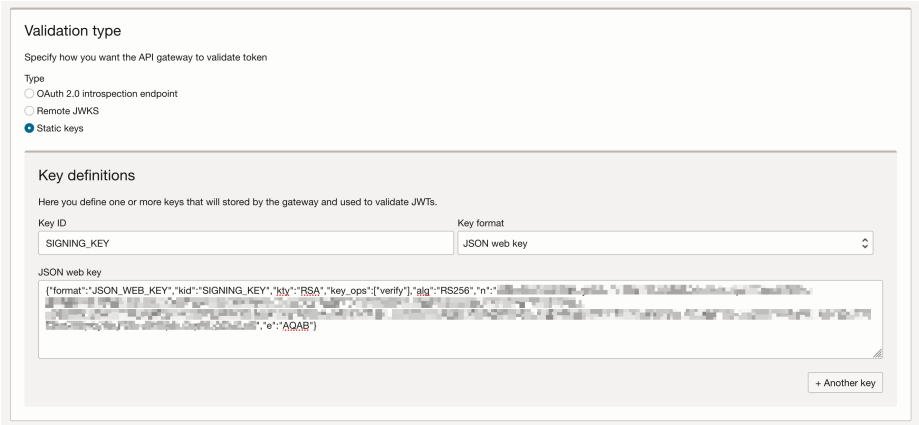

Figure 23: Validation type

Create Oracle IAM applications to generate and validate JWTs We need to create two confidential applications in Oracle IAM to generate and then to validate the tokens:

- A Resource Server – this application will be used to validate the tokens
- A Client Application – this application will be used by a client to obtain the tokens

Create the Resource Server Application

The Resource Server Application will be used for JWT validation.

1. Navigate to Domains, go to **Integrated applications**, click on **Add application** and select **Confidential Application**, then click on **Launch workflow**.
2. Give the Resource Server application a **Name** and click on **Next**
3. **Skip the Client configuration**
4. Select **Configure this application as a resource server**
5. In **Token issuance policy**, select **Specific** resources to be authorized.
6. Add a **Scope** for OAuth. For this example, let's set it to `objectstorage`.
7. Click on **Next** twice, and then on **Finish**
8. **Activate** the Resource Server application.

Create the Client Application

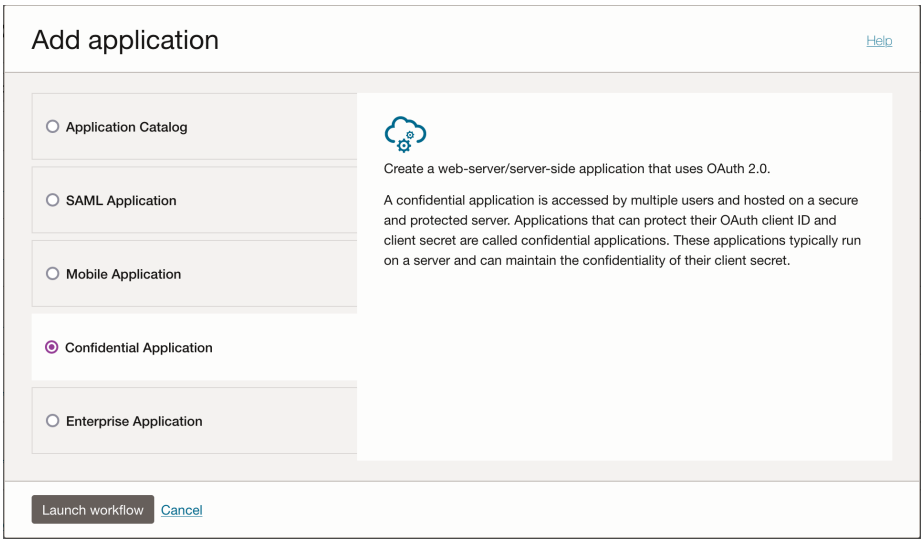

Figure 24: Creating the Confidential Applications

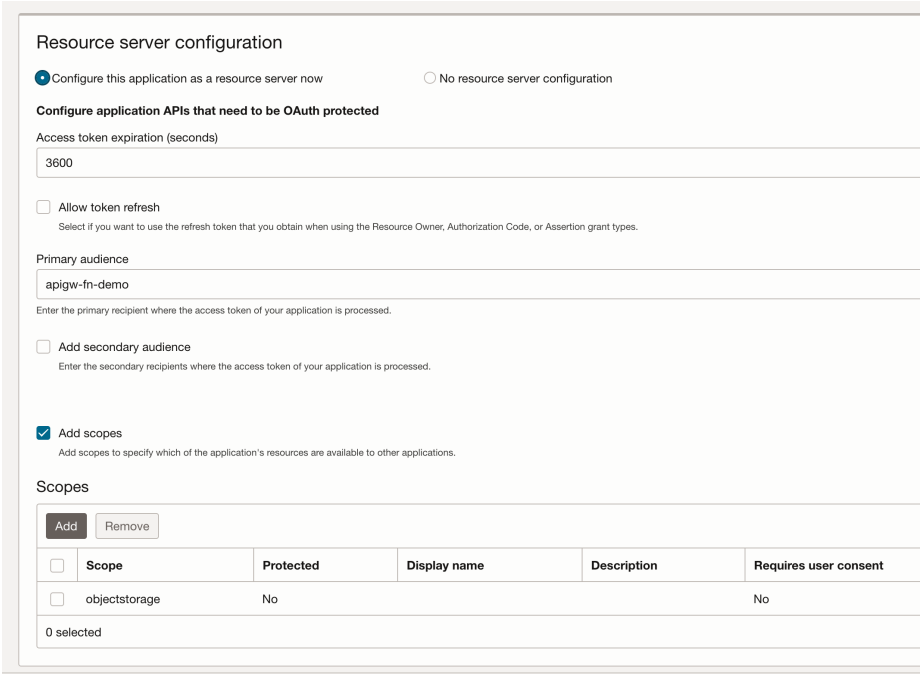

Figure 25: Resource server and scope

The Client Application will be used by the API clients to obtain the tokens.

1. Click again on **Add application** and select **Confidential Application**
2. Give the Client Application a **Name** and click on **Next**
3. **Configure this application as a client**, check **Client Credentials** and **JWT Assertion**
4. In **Token issuance policy** , add the **Scope** defined in our Resource Server app (check **Add resources**, click on **Add scope** below, and search for the Resource Server in the list).
5. Click on **Next** and skip the rest of the screens

   A Client ID and Client Secret for this client application will be generated for you at the end of the process. You will need to save them, as they will be used from the client application, but they can also be consulted afterwards, in the application's definition, on the Configuration tab.

6. **Activate** the application.

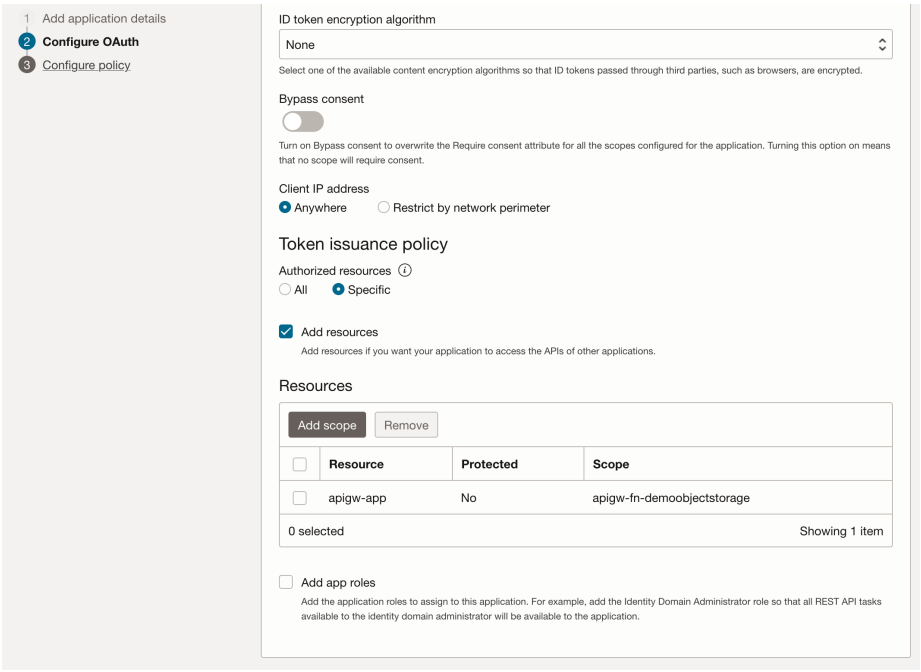

Figure 26: Token issuance policy for the client

Test the service Using cURL, Postman or any other REST client, test out the services. We can retrieve the IAM Domain URL from the Domain configuration.

Calling our endpoint like before will now return a `401 Unauthorized` error.

1. Get the JWT Token from IAM:

```
curl --location --request POST "https://idcs-xxxx.identity.oraclecloud.com/
```

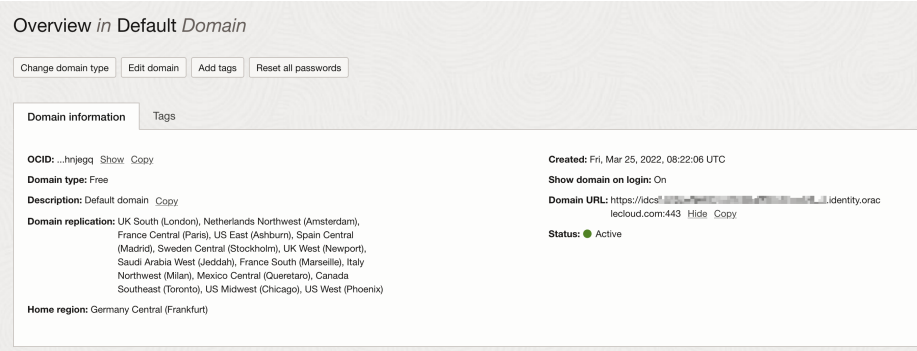

Figure 27: IAM URL retrieval

```
oauth2/v1/token" \
--header "Content-Type: application/x-www-form-urlencoded" \
--data-urlencode "grant_type=client_credentials" \
--data-urlencode "client_id=<client-id>" \
--data-urlencode "client_secret=<client-secret>" \
--data-urlencode "scope=<scope>"
```

2. Call the API Gateway endpoint:

```
curl --location --request POST 'https://<id>.apigateway.<region-name>.
oci.customer-oci.com/pictures/john/avatar.jpg' \
--header 'Authorization: Bearer <TOKEN>' \
--header 'Content-Type: application/json' \
-F "fileUpload=@/path/to/image/avatar.jpg"
```

Thank you for joining us for this webinar. If you want to learn more, don't hesitate to check out the official documentation at https://docs.oracle.com. Happy coding!