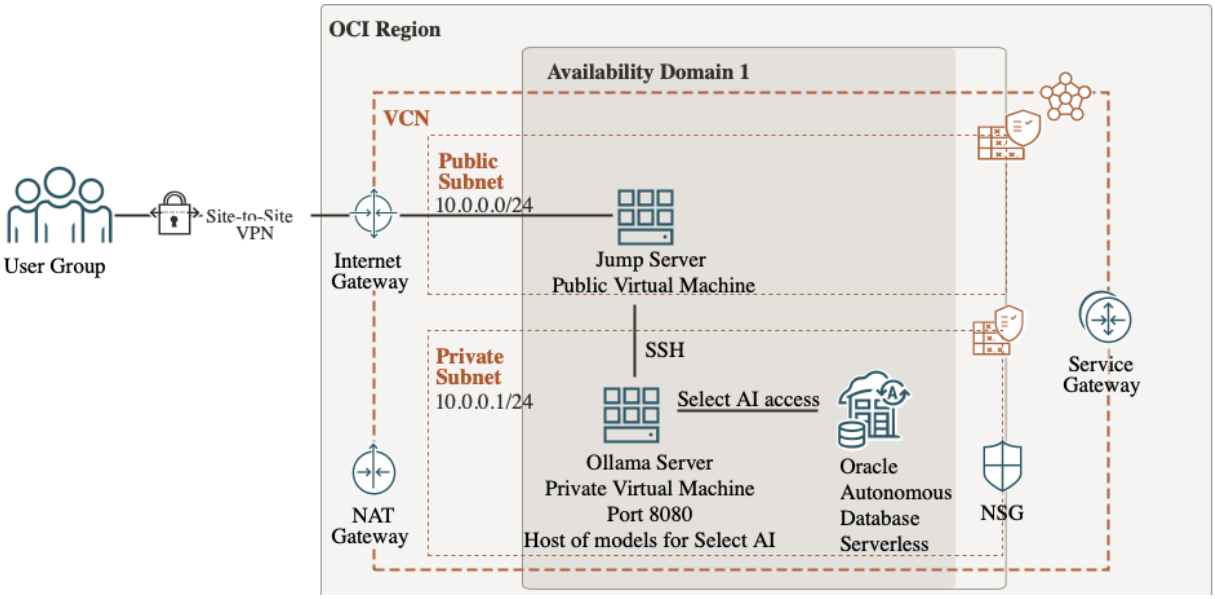
Setting up a private endpoint for AI models using Ollama and Llama.cpp

Contents

Introduction

This document shows how to configure Select AI to work with privately-hosted Ollama and Llama.cpp LLMs and transformers for organizations needing to address security concerns through private AI processing with Autonomous Database Serverless. With this setup, all processing happens within an isolated network environment behind a private endpoint in OCI. This feature is supported on Oracle Autonomous Database 19c and 23ai. These instructions were tested using Oracle Linux 8 and 9.

Architecture Overview



The configuration addresses a need for a secure environment through:

- Autonomous Database Serverless runs behind a private endpoint
- Your AI model (using Ollama) runs on a VM behind the private endpoint
- Select AI can communicate with your privately hosted model without internet exposure
- All components remain isolated within your Oracle Virtual Cloud network

Public Subnet:

- Jump server (public VM) with internet access for secure entry point
- SSH tunneling for secure access to internal resources
- Outbound internet connectivity for software installation

Private Subnet:

- Autonomous Database with private endpoint
- Private VM for running Ollama LLMs and transformers
- Internal communication only within VCN

Network Components (OCI):

- Internet Gateway: enables internet access for resources in the public subnet
- Service Gateway: allows private resources to access Oracle Cloud services without using internet
- Security lists: act as virtual firewalls to control inbound and outbound traffic to and from resources in a subnet
- Route Tables: define rules to direct traffic from subnets to various destinations (e.g., internet gateway, service gateway, etc.)

This architecture isolates the database and Ollama models from the internet while enabling secure access through controlled entry points.

SECTION 1: Core Setup - Database Access and Testing

Network Infrastructure Setup

CIDR (Classless Inter-Domain Routing) defines the range of IP addresses available within a network. In this setup, begin by creating a Virtual Cloud Network (VCN) with a **/16 CIDR block** (e.g., 10.0.0.0/16). A **/16 block allows 65,536 IP addresses** because it reserves 16 bits (2^{16}) for host addressing, giving you the flexibility to create multiple smaller subnets without running out of space.

To organize the network and apply different access and security controls, divide the /16 range into smaller **/24 subnets**, each containing **256 IP addresses** because a /24 reserves 8 bits (2^{8}) for hosts. This approach keeps the network manageable and allows you to assign specific subnets for different purposes, such as public-facing resources or private database instances. **For more information, refer to the** OCI Networking Overview**.**

Create Virtual Cloud Network (VCN)

- In the Oracle Cloud Infrastructure (OCI) console, navigate to Networking > Virtual Cloud Networks
- Click "Create VCN"
- Enter name (e.g., "ADB-Private-VCN"), compartment, and CIDR block (e.g., 10.0.0.0/16)
- Click "Create VCN"

Create Subnets

Create Public Subnet

- In your VCN, click "Subnets" tab > "Create Subnet"
- Name your subnet (e.g., "Public-Subnet")
- Choose a CIDR block (e.g., 10.0.0.0/24)
- Set access type to "Public Subnet"
- Click "Create Subnet"

Create Private Subnet

- Click "Create Subnet" again
- Name your subnet (e.g., "ADB-Private-Subnet")
- Choose a CIDR block (e.g., 10.0.1.0/24)
- Set access type to "Private Subnet"
- Click "Create Subnet"

Configure Security Lists

Public Subnet Security List

- In your VCN, click "Security Lists" > "Create Security List"
- Name: "Public-Subnet-Security-List"
- Populate the public subnet security list with the following rules

Rule Type	Source/Destination	Protocol	Port	Description
Ingress	0.0.0.0/0	TCP	22	SSH access from internet
Egress	0.0.0.0/0	All	All	Allow all outbound traffic
Egress	10.0.1.0/24	TCP	443	HTTPS to Autonomous Database subnet

Private Subnet Security List

- Create another security list named "Private-Subnet-Security-List"
- Populate the private subnet security list with the following rules

Rule Type	Source/Destination	Protocol	Port	Description
Ingress	0.0.0.0/0	TCP	22	SSH access from public subnet
Ingress	10.0.0.0/16	TCP	1521-1522	Database connections from VCN
Ingress	10.0.1.0/24	TCP	443	HTTPS from public VM subnet
Ingress	10.0.1.0/24	TCP	80	HTTP traffic within private subnet
Ingress	10.0.0.0/24	TCP	80	HTTP from public subnet
Ingress	10.0.0.0/24	TCP	8080	HTTP Ollama traffic from public VM
Egress	0.0.0.0/0	TCP	All	Allow outbound traffic to Oracle Services

Associate Security Lists with Subnets

- Go to "Subnets" > select your **public** subnet > "Edit" > select "Public-Subnet-Security-List"
- Go to "Subnets" > select your **private** subnet > "Edit" > select "Private-Subnet-Security-List"

Create Gateways

Create Internet Gateway

- In VCN, go to Gateways tab > "Create Internet Gateway"
- Name your gateway (e.g., "Internet-Gateway")
- Confirm if it's enabled (this is the default)
- Click "Create Internet Gateway"

Create Service Gateway

- In VCN, go to Gateways > "Create Service Gateway"
- Name your gateway (e.g., "ADB-Service-Gateway")
- Select "All [region] Services in Oracle Services Network"
- Click "Create Service Gateway"

Configure Route Tables

Public Subnet Route Table

- In VCN, go to Routing tab > "Create"
- Name: "Public-Route-Table"
- Click "Create Route Table"
- Once created, click on the new route table
- Click "Add Route Rules"
- Add route rule for internet access:
    - Target Type: Internet Gateway
    - Destination CIDR: 0.0.0.0/0
    - Target Internet Gateway: Select your internet gateway
    - Description: "Route to internet"
- Click "Add Route Rules"

Private Subnet Route Table

- Go back to Routing tab > "Create"
- Name: "Private-Route-Table"
- Click "Create Route Table"
- Once created, click on the new route table
- Click "Add Route Rules"
- Add route rule for Oracle services:
    - Target Type: Service Gateway
    - Destination Service: All [region] Services in Oracle Services Network
    - Target Service Gateway: Select your service gateway
    - Description: "Route to Oracle Services"
- Click "Add Route Rules"

Associate Route Tables with Subnets

- Navigate back to your VCN main page (click on VCN name)
- Go to Subnets tab > select your public subnet > "Edit"
- In "Route Table" section, select "Public-Route-Table" > "Save Changes"
- Go to Subnets tab > select your private subnet > "Edit"
- In "Route Table" section, select "Private-Route-Table" > "Save Changes"

Database Creation

Creating the Autonomous Database with Private Endpoint

- Navigate to Oracle Database > Autonomous Database
- Click "Create Autonomous Database"

Fill in Basic Information

- Display name and database name
- Choose compartment (same as VCN)
- Select workload type (ATP/ADW)
- Choose "Shared Infrastructure" deployment

Configure Database

- Select CPU and storage capacity
- Set admin password (remember this for Oracle Machine Learning access)
- Choose license type

Network Access Section

- Select "Private endpoint access only"
- Choose your VCN and private subnet
- Optionally provide a private endpoint name and hostname prefix
- Review settings and click "Create Autonomous Database"

Compute Instance Setup

Create Public VM (Jump Server)

- Navigate to Compute > Instances > "Create Instance"
- Name: "Public-Jump-Server"
- Select your compartment
- Choose availability domain
- Select shape (e.g., VM.Standard.E4.Flex)
- Configure networking:
    - VCN: Select your VCN
    - Subnet: Select your public subnet
    - Assign public IPv4 address: Yes
- Add SSH keys (upload your public key)
- Click "Create"

Create Private VM (Model Server)

- Create another instance named "Private-VM"
- Configure networking:
    - VCN: Select your VCN
    - Subnet: Select your private subnet
    - Assign public IPv4 address: No
- Add the same SSH keys
- Click "Create"

Verify Instance Configuration

- Check that the public VM shows both private IP (10.0.0.x) and public IP

Check that the private VM shows only private IP (10.0.1.x)

Database Client Access Configuration

Set up Database Client Access

Connect to Public VM SSH to the public VM using its public IP

```
ssh -i your-private-key.pem opc@<public-vm-ip>
```

Connect to Private VM from Public VM

- Copy your private key to the public VM securely
- From public VM, SSH to private VM using its private IP

```
ssh -i your-private-key.pem opc@<private-vm-ip>
```

Install and Configure SQLcl on Jump VM for Database Access

Connect to Jump VM

```
ssh -i your-key.pem opc@<your-jump-vm-ip>
```

Download and install SQLcl

```
wget https://download.oracle.com/otn_software/java/sqldeveloper/sqlcl-latest.zip
unzip sqlcl-latest.zip
export PATH=$PATH:~/sqlcl/bin
```

Connect to Autonomous Database from Jump VM

Configure wallet

1. Download wallet from Autonomous Database console:
    - Go to Oracle Database → Autonomous Database → [Your Autonomous Database instance]
    - Click "DB Connection"
    - Click "Download Wallet"
    - Set a wallet password and download the ZIP file
2. Transfer wallet to Jump VM using WinSCP.
3. Configure wallet on Jump VM:

Create wallet directory

```
mkdir -p ~/wallet
```

Extract the wallet

```
unzip Wallet_*.zip -d ~/wallet
```

Set TNS_ADMIN environment variable

```
export TNS_ADMIN=~/wallet
```

Check available service names

```
cat ~/wallet/tnsnames.ora
```

Will show your service names like: your-adb-service_high, your-adb-service_medium, your-adb-service_low

4. Connect to Autonomous Database:

Connect using the low service for test

```
sql admin@your-adb-service_low
```

Enter your Autonomous Database admin password when prompted.

Test Database Connection

Once connected, test with simple queries:

```
SQL> SELECT 'Hello from ADB' as message FROM dual;
```

SECTION 2: AI Model Setup - Installing Ollama and Llama.cpp

Set up Nginx Reverse Proxy for the model

Run model on port 8080 and use a reverse proxy to expose on 80 for Select AI. This configuration assumes Ollama and Llama.cpp are not running the same VM.

Install nginx

```
sudo dnf install nginx
```

Back up the default nginx configuration

```
sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

Disable the default server block to avoid conflicts

```
sudo vi /etc/nginx/nginx.conf
```

Find the server block (around line 30) and manually add # at the beginning of each line from "server {" to the matching "}".

Create Ollama reverse proxy configuration

```
sudo tee /etc/nginx/conf.d/ollama.conf > /dev/null <<'EOF'
server {
   listen 80;
   location / {
       proxy_pass http://127.0.0.1:8080;
       proxy_http_version 1.1;
       proxy_set_header Upgrade $http_upgrade;
       proxy_set_header Connection 'upgrade';
       proxy_set_header Host $host;
       proxy_cache_bypass $http_upgrade;
   }
}
EOF
```

Test the nginx configuration

```
sudo nginx -t
```

Expected output:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

To reload nginx after changes

```
sudo nginx -s reload
```

Ollama AI Model Setup

Configure Ollama on Private VM for AI model integration with database workflows:

Download on Public VM

Connect to Public VM

```
ssh -i your-key.pem opc@<your-public-vm-ip>
```

Download Ollama binary

```
wget https://ollama.com/download/ollama-linux-amd64.tgz
```

Install Ollama temporarily (for model downloads only):

NOTE: This temporary installation is necessary because 'ollama pull' requires the Ollama service to be running - you cannot download models directly with wget/curl

```
sudo tar -C /usr -xzf ollama-linux-amd64.tgz
```

Start Ollama service

```
ollama serve &
```

Download models (this requires internet)

```
ollama pull llama2
```

Stop Ollama service

```
pkill ollama
```

Package models for transfer

```
tar -czf models.tgz -C ~/.ollama .
```

Transfer to Private VM

Transfer Ollama binary

```
scp -i your-key.pem ollama-linux-amd64.tgz opc@<private-vm-ip>:/home/opc/
```

Transfer models

```
scp -i your-key.pem models.tgz opc@<private-vm-ip>:/home/opc/
```

Install on Private VM

Connect to Private VM

```
ssh -i your-key.pem opc@<private-vm-ip>
```

Install Ollama

```
sudo tar -C /usr -xzf ollama-linux-amd64.tgz
```

Create ollama user

```
sudo useradd -r -s /bin/false -m -d /usr/share/ollama ollama
```

Set up models directory for ollama user

```
sudo mkdir -p /usr/share/ollama/.ollama
tar -xzf models.tgz -C ~/.ollama
sudo cp -r ~/.ollama/* /usr/share/ollama/.ollama/
sudo chown -R ollama:ollama /usr/share/ollama/.ollama
```

Start Ollama as ollama user on port 8080

```
sudo -u ollama OLLAMA_HOST=0.0.0.0:8080 nohup /usr/bin/ollama serve > ~/ollama.log 2>&1 &
```

Configure Private VM Firewall for Cross-Subnet Access

Add Jump VM subnet and Oracle managed network to trusted sources:

On Private VM, allow traffic from public subnet

```
sudo firewall-cmd --permanent --add-source=10.0.0.0/24
```

Add ports 80 and 8080 to allowed services:

```
sudo firewall-cmd --permanent --add-port=80/tcp
sudo firewall-cmd --permanent --add-port=8080/tcp
sudo firewall-cmd --reload
```

Verify the configuration:

```
sudo firewall-cmd --list-sources
sudo firewall-cmd --list-ports
```

Expected output:

- `Sources: 10.0.0.0/24`
- `Ports: 80/tcp 8080/tcp`

Test Installation

Verify Ollama is running on port 8080

```
sudo netstat -tlnp | grep :8080
```

Expected output: `tcp 0 0 0.0.0.0:8080 0.0.0.0:* LISTEN [PID]/ollama`

Test locally

```
curl http://localhost:8080/api/version
```

Expected output: `{"version":"0.7.1"}`

Test connectivity from Jump VM

```
curl http://<private-vm-ip>:8080/api/version
```

Expected output: `{"version":"<version number>"}`

Test model:

```
curl -X POST http://<private-vm-ip>/v1/chat/completions \
     -H "Content-Type: application/json" \
     -d '{
           "model": "llama2",
           "messages": [{"role": "user", "content": "Hello world"}],
           "max_tokens": 100
         }'
```

Expected output: JSON response with "response" field containing model's reply. Example:

```
{"model":"llama2","created_at":"...","response":"\nHello there! How are you
today?...","done":true,...}
```

Llama.cpp AI Model Setup

Download llama.cpp and the model

```
wget https://github.com/ggml-org/llama.cpp/archive/refs/heads/master.zip
wget https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF/resolve/main/llama-2-7b-
chat.Q4_K_M.gguf
```

Copy files to the private VM

```
scp -i /home/opc/omlpem.pem master.zip opc@10.0.1.125:/home/opc/
scp -i /home/opc/omlpem.pem llama-2-7b-chat.Q4_K_M.gguf opc@10.0.1.125:/home/opc/
```

SSH into the VM

```
ssh -i /home/opc/omlpem.pem opc@10.0.1.125
```

Unzip llama.cpp

```
unzip master.zip
cd llama.cpp-master
```

Install build dependencies and nginx

```
sudo dnf install cmake gcc-c++ make -y
sudo dnf install libcurl-devel -y
sudo dnf install nginx -y
```

Build llama.cpp

```
mkdir build
cd build
cmake ..
cmake --build .
```

Prepare the models folder

```
mkdir -p ../models
mv ../llama-2-7b-chat.Q4_K_M.gguf ../models/
```

Test llama.cpp using the CLI

```
./bin/llama-cli -m ../models/llama-2-7b-chat.Q4_K_M.gguf -p "Hello world"
```

Start llama-server on port 8080

```
cd /home/opc/LLAMA__CPP_FILES_PRIVATE_VM/llama.cpp-master/build

nohup ./bin/llama-server -m ../models/llama-2-7b-chat.Q4_K_M.gguf --port 8080 > llama-server.log
2>&1 &
```

Verify llama-server is running

```
ps aux | grep llama-server
```

Test via curl on the private and public VMs

```
curl -X POST 10.0.1.125/v1/completions \
     -H "Content-Type: application/json" \
     -d '{"prompt":"Hi, how are you?", \
          "max_tokens":50, \
          "model":"llama-2-7b-chat.Q4_K_M.gguf"}'

Expected response: “I'm doing well, thank you for asking! How can I help you today?”
```

SECTION 3: Oracle Machine Learning Notebook integration (optional)

Oracle Machine Learning (OML) Configuration

Additional Security List Rules for OML

Add HTTPS access rule to your private subnet security list:

- Navigate to your VCN > Security Lists > select private subnet security list
- Click "Add Ingress Rule"
- Configure HTTPS access rule:
    - Source Type: CIDR
    - Source CIDR: 10.0.1.0/24 (your private subnet CIDR)
    - IP Protocol: TCP
    - Source Port Range: All (leave blank)
    - Destination Port Range: 443
    - Stateless: No
    - Description: "HTTPS access for OML notebooks"
- Click "Add Ingress Rule"

Accessing OML Notebooks via SSH Tunneling

Access OML through SSH tunneling from your local Windows machine to the Jump VM, which can reach both the internet and your private database.

Configure PuTTY SSH Tunnel

PuTTY Session Settings:

- Host Name: your-jump-vm-public-ip
- Port: 22
- Connection type: SSH

SSH > Auth Section:

- Private key file: Browse to your working PPK file

SSH > Tunnels Section:

- Source port: 443
- Destination: your-adb-hostname.adb.region.oraclecloudapps.com:443
- Select "Local"
- Click "Add"

Important Notes:

- Use the Autonomous Database hostname from the "Private endpoint URL" field in the OCI Console, not the IP address
- Run PuTTY as Administrator (required for binding to port 443)
- Save the configuration and connect

Access OML

1. Keep PuTTY session open (maintains the tunnel)
2. Open browser on your Windows machine
3. Navigate to: https://localhost/oml/
4. Login with Autonomous Database credentials:
    - Tenant: Your full tenancy OCID (ocid1.tenancy.oc1.....)
    - Database name, username, and password

Test Network Connectivity

- From Jump VM, test Autonomous Database connectivity using hostname:

```
curl -k -I https://your-adb-hostname.adb.region.oraclecloudapps.com/oml/
```

If connection times out, verify security list rules are configured correctly.

Select AI Testing

Testing Select AI with Private Model

After completing the infrastructure setup, verify that Select AI can communicate with your private Ollama instance.

Under Autonomous Database ADMIN:

Create database user and apply grants:

```
CREATE USER SELECT_AI_USER IDENTIFIED BY "<password>";
GRANT DWROLE TO SELECT_AI_USER;
GRANT OML_DEVELOPER TO SELECT_AI_USER;
ALTER USER SELECT_AI_USER GRANT CONNECT THROUGH OML$PROXY;
ALTER USER SELECT_AI_USER QUOTA UNLIMITED on DATA;

GRANT EXECUTE ON DBMS_CLOUD to SELECT_AI_USER;
GRANT EXECUTE ON DBMS_CLOUD_AI to SELECT_AI_USER;
GRANT EXECUTE ON DBMS_CLOUD_PIPELINE to SELECT_AI_USER;
```

Add the user to the Host Access Control List (ACL). Replace SELECT_AI_USER with your username:

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '*.oraclevcn.com',
    ace  => xs$ace_type(privilege_list => xs$name_list('http', 'connect', 'resolve'),
            principal_name => 'SELECT_AI_USER',
            principal_type => xs_acl.ptype_db)
);
END;
/
```

Force traffic through private endpoints instead of public internet:

```
ALTER DATABASE PROPERTY
SET route_outbound_connections = 'ENFORCE_PRIVATE_ENDPOINT';
```

Under the database user, configure Select AI Profile

Ollama

Create a minimal AI profile for using the 'chat' action that points to your private Ollama instance. Replace http://omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com with your private Ollama instance name.

```
BEGIN
    DBMS_CLOUD_AI.drop_profile(profile_name => 'OLLAMA_PRIVATE_PROFILE', force => true);
    DBMS_CLOUD_AI.create_profile(
        profile_name => 'OLLAMA_PRIVATE_PROFILE',
        attributes => '{
            "model": "llama2",
            "provider_endpoint": "http://omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com",
            "conversation": true}');
END;
/
```

Set the active profile:

```
SQL> exec DBMS_CLOUD_AI.SET_PROFILE('OLLAMA_PRIVATE_PROFILE');
```

Test Select AI Query

```
SQL> Select ai chat hello world;
```

Llama.cpp

Create a minimal AI profile for using the 'chat' action that points to your private Llama.cpp instance. Replace http://omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com with your private Ollama instance name.

Create an AI profile

```
BEGIN
    DBMS_CLOUD_AI.drop_profile(profile_name => 'LLAMACPP_PRIVATE_PROFILE', force => true);
    DBMS_CLOUD_AI.create_profile(
         profile name => 'LLAMACPP_PRIVATE_PROFILE',
         attributes => '{
         "model": "llama-2-7b-chat.Q4_K_M.gguf",
         "provider_endpoint": "http://omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com",
         "conversation": true}');
END;
/
```

Set the active profile

```
SQL> exec DBMS_CLOUD_AI.SET_PROFILE('LLAMACPP_PRIVATE_PROFILE');
```

Run a Select AI chat command

```
SQL> select ai chat how are you;
```

```
Expected response: “I'm doing well, thank you for asking! How can I help you today?”
```

Verification

If Select AI returns a response, you have successfully:

- Deployed Autonomous Database with private endpoint
- Set up secure networking between components
- Configured a private AI model (Ollama)
- Integrated Select AI with your private model

Troubleshooting

Network Connectivity Tests

Test DNS resolution from database. Replace 'omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com' with your private instance name.

```
SELECT UTL_INADDR.GET_HOST_ADDRESS('omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com') FROM
dual;
```

Expected: Should return IP address (e.g., 254.37.128.0)

Test port connectivity from database:

```
set serveroutput on
DECLARE
  l_conn UTL_TCP.connection;
BEGIN
  l_conn := UTL_TCP.open_connection(
    remote_host => 'omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com',
    remote_port => 8080,
    tx_timeout  => 60
  );
  DBMS_OUTPUT.PUT_LINE('Port is open');
  UTL_TCP.close_connection(l_conn);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Port is closed: ' || SQLERRM);
END;
/
```

Expected: Should display "Port is open"

Direct AI API Call Test

Test direct API call to Ollama from database. Replace `'omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com'` with your private instance name.

```
set serveroutput on

DECLARE

  l_result  CLOB;

  l_body      JSON_OBJECT_T := JSON_OBJECT_T('{ "model": "llama2", "messages": [{"role":
"user","content": "Hello world"}],"stream": false}');

  l_endpoint  VARCHAR2(2000) :=
'http://omlpevm.omlprivatesubne.omlprivatevcn.oraclevcn.com/v1/chat/completions';

BEGIN

  l_result := DBMS_CLOUD.get_response_text(

                  DBMS_CLOUD.send_request(

                      credential_name => '',

                      uri             => l_endpoint,

                      method          => DBMS_CLOUD.METHOD_POST,

                      body            => l_body.to_blob()

                  )

              );

  DBMS_OUTPUT.PUT_LINE(l_result);

END;

/
```

Expected: Should return JSON response from Ollama model