ELEC50009 Information Processing

Lab 4

AWS Connectivity, Client-Server, NoSQL Database

This lab assumes that you have created an AWS Free Tier account. You will also provide you with AWS academy accounts to use in your projects. The steps detailed in this lab are equally applicable to both the free tier and academy accounts.

1 Creating an EC2 Instance

EC2 (Amazon Elastic Compute Cloud) is a service that provides resizable virtual servers in the cloud for running applications. We can think of it as a remote computer running our apps in the cloud, which we can connect to from our local machine.

AWS exposes a **public IP address** for this computer, which we can then connect to using protocols such as TCP, UDP, or SSH, depending on the purpose.

Create an EC2 instance by following the standard steps on AWS. Please ensure the following settings while creating your instance (these must be configured correctly for this lab to run smoothly):

- **Instance Type:** t2.micro (or equivalent)
- **Operating System:** Ubuntu Linux
- **Key Pair:** Create and download a .pem file (used for secure SSH login)
- **Network:** Public IPv4 address enabled
- **Security Group:** Allow all inbound traffic (for simplicity in this lab)
- **Storage:** Default settings

As AWS frequently updates the layout and wording of its interface, you may need to explore the console slightly to ensure that the instance is created with the correct settings. Once the instance has been set up, make sure that it is running and note down its public IP address. We will now log into this instance using SSH from our local computer.

2 Logging into the EC2 Instance

In the following, we will be using WSL on Windows, or the Terminal on macOS, to log into the EC2 instance. Let's do this in a step-by-step way:

- **Step 1 – Locate your key file:** Make sure the .pem key file you downloaded when creating the EC2 instance is present in your current working directory.
- **Step 2 – Set permissions on the key file:** SSH requires the key file to have restricted permissions. Run:

```
$ chmod 400 your-key.pem
```

- **Step 3 – Connect to the EC2 instance using SSH:** Use the following command. Here IP1 is the public IP of your EC2 instance.

```
$ ssh -i your-key.pem ubuntu@IP1
```

- **Step 4 – Accept the host key (first connection only):** When prompted, type yes and press Enter.

- **Step 5 – Verify the connection:** Once logged in, run the following command to confirm that the instance is responding:

```
$ hostname
```

- **Step 6 – Exit the EC2 instance:** Exit the SSH session and return to your local terminal by running:

```
$ exit
```

3 Transferring Files

We will use another protocol, scp, to transfer files between our local machine and the EC2 instance. You can use scp later to transfer your Python scripts and other files to the EC2 instance, and back to your local machine. The steps below demonstrate the basic process using a dummy text file.

- **Step 1 – Create a dummy text file on your local machine:** First create a file with some text in it. Run:

```
$ echo "Hello from my local machine" > test.txt
```

- **Step 2 – Copy the file to the EC2 instance:** Use `scp` to transfer the file to the home directory of the `ubuntu` user. Run:

```
$ scp -i your-key.pem test.txt ubuntu@IP1:~
```

- **Step 3 – Log into the EC2 instance:** Connect to the instance using `ssh`. Run:

```
$ ssh -i your-key.pem ubuntu@IP1
```

- **Step 4 – View the file on the EC2 instance:** Once logged in, display the contents of the file to confirm the transfer. Run:

```
$ cat test.txt
```

- **Step 4.5 – Exit the EC2 instance:** Exit the SSH session and return to your local terminal by running:

```
$ exit
```

- **Step 5 – Copy the file back to your local machine:** From your local terminal, use `scp` to copy the file back from the EC2 instance and save it with a new name. Run:

```
$ scp -i your-key.pem ubuntu@IP1:~/test.txt test_2.txt
```

- **Step 6 – View the copied file locally:** Display the contents of the copied file on your local machine to confirm the transfer. Run:

```
$ cat test_2.txt
```

4 Client-Server Communication

In this section, we will begin setting up the software environment required for implementing client-server communication. As a first step, we will ensure that Python is installed on the remote EC2 instance, as it will be used to implement both the server and client programs.

4.1 Installing Python on the EC2 Instance

Ubuntu does not always ship with the required version of Python and supporting tools installed by default. The following steps explain how to install Python on the EC2 instance.

- **Step 1 – Log into the EC2 instance:** Connect to your EC2 instance using SSH, as described in the previous section.
- **Step 2 – Update the package list:** Before installing any software, update the package index. Run:

```
$ sudo apt update
```

- **Step 3 – Install Python:** Install Python 3 and the package manager `pip`. Run:

```
$ sudo apt install -y python3 python3-pip
```

- **Step 4 – Verify the installation:** Confirm that Python is installed correctly by checking its version. Run:

```
$ python3 --version
```

4.2 A Simple UDP Client and Server in Python

We now implement a minimal UDP-based client–server example. The server listens on port `12000`. The client sends a string to the server, and the server checks whether the string is written entirely in uppercase letters. The server then replies accordingly.

Consider the client and server codes given below.

UDP Server (`udp_server.py`)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('0.0.0.0', serverPort))
print("UDP server running and listening on port", serverPort)
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    text = message.decode()
    if text.isupper():
        reply = "ALL CAPS"
    else:
        reply = "NOT ALL CAPS"
    serverSocket.sendto(reply.encode(), clientAddress)
```

UDP Client (`udp_client.py`)

```
from socket import *
serverIP = "IP1" #The public IP of your EC2 instance
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input("Enter a string: ")
clientSocket.sendto(message.encode(), (serverIP, serverPort))
reply, _ = clientSocket.recvfrom(2048) #ignoring sender IP/Port
print("Server response:", reply.decode())
clientSocket.close()
```

Remember to replace `IP1` with the public IP address of your EC2 instance. Notice that the server binds to `0.0.0.0`; this means it listens on all network interfaces and can accept incoming UDP packets from external machines, not just from the local host.

4.3 Running the Client–Server Example

We now deploy the UDP server on the EC2 instance and the UDP client on the local machine.

- **Step 1 – Create the files locally:** On your local machine, create the two Python files shown above using a suitable editor.
- **Step 2 – Transfer the server file to the EC2 instance:** Use `scp` to copy the server program to the EC2 instance (you may place it in another directory if you wish):

  ```
  $ scp -i your-key.pem udp_server.py ubuntu@IP1:~
  ```

- **Step 3 – Run the server on the EC2 instance:** Log into the EC2 instance using SSH, then start the server:

  ```
  $ python3 udp_server.py
  ```

  Leave this terminal running.

- **Step 4 – Run the client on your local machine:** From a different local terminal, run:

  ```
  $ python3 udp_client.py
  ```

  Try entering both uppercase and lowercase strings and observe the server's response.

5 Setting up a Database

In this section, we configure the EC2 instance so that it can securely access Amazon's managed NoSQL database service, DynamoDB. In your project, DynamoDB will be used to store application-related data.

5.1 Configuring EC2 to Access DynamoDB

To allow the EC2 instance to communicate securely with DynamoDB, we attach an IAM role to the instance. An IAM role is an AWS identity that grants permissions to an AWS service (such as EC2) so it can securely access other AWS resources without using long-term credentials.

- **Step 1 – Create an IAM role:** Using the AWS web interface, create a new IAM role for EC2. Since the AWS console layout may vary, you may need to explore the interface to locate the IAM service, create a new role, select EC2 as the trusted entity, and proceed through the default setup options.

- **Step 2 – Attach DynamoDB permissions:** Attach a policy that grants access to DynamoDB (for example, `AmazonDynamoDBFullAccess` for this lab).
- **Step 3 – Attach the role to the EC2 instance:** Once the IAM role has been created, it must be attached to the EC2 instance so that programs running on the instance can access DynamoDB. To do this, follow the steps below. Note that the exact layout of the AWS interface may vary slightly, so you may need to explore the interface to locate the relevant options.
  - Go to the EC2 section of the AWS web interface.
  - Select **Instances** from the left-hand menu.
  - Click on your running EC2 instance.
  - Choose **Actions** → **Security** → **Modify IAM role**.
  - From the dropdown list, select the IAM role you created.
  - Click **Update IAM role**.

At this point, the EC2 instance is correctly configured to interact with DynamoDB, and no additional credential setup is required on the machine itself.

5.2 Install boto3

To interact with DynamoDB from Python, we will use `boto3`, the official AWS SDK for Python. The following steps install `boto3` on the EC2 instance.

- **Step 1 – Log into the EC2 instance:** Connect to your EC2 instance using SSH, as described earlier.
- **Step 2 – Install `boto3`:** On Ubuntu, install the AWS SDK for Python using the system package manager. Run:

```
$ sudo apt-get update
$ sudo apt-get -y install python3-boto3
```

- **Step 3 – Verify the installation:** Check that `boto3` is installed correctly by attempting to import it:

```
$ python3 -c "import boto3"
```

If no error is produced, `boto3` has been installed successfully and the EC2 instance is ready to interact with DynamoDB.

5.3 DynamoDB: Basic Use Examples

In this section, we perform basic operations on a DynamoDB table, including table creation, loading sample data, and basic querying. Scripts for additional operations, such as, creating, reading, updating, and deleting items, and advanced features like scanning, have also been provided. When using DynamoDB in your projects, you can use these scripts, as required, as the starting point for your work. They will also be discussed in the lectures. All examples are implemented in Python and adapted from AWS's DynamoDB Developer Guide.

5.3.1 Creating a DynamoDB Table

On EC2, run the code provided in `MoviesCreateTable.py`.

This creates a DynamoDB table called `Movies`.

This table uses a composite primary key consisting of a partition key (`year`) and a sort key (`title`). The `boto3.resource` call provides an object-oriented interface to DynamoDB in the specified AWS region. Provisioned throughput defines the maximum read and write capacity allowed for the table.

Run the script to create the table.

Use the python interpreter directly from the terminal to verify access to DynamoDB from the EC2 instance. The following should show all existing tables.

```
$ python3
>>> import boto3
>>> db = boto3.resource('dynamodb', region_name='us-east-1')
>>> print(list(db.tables.all()))
>>> exit()
```

5.3.2 Loading Data into an Existing Table

On EC2, run the code provided in `MoviesLoadData.py` to load movie data into the existing `Movies` table.

The movie data is available in `moviedata.json` and is supplied to `boto3` in JSON format (see Figure 1). Each item represents a movie and uses the same composite primary key defined earlier: the partition key `year` and the sort key `title`. Additional movie details are stored in the `info` attribute as a nested JSON object.

```
{
    "year" : 2013,
    "title" : "Turn It Down, Or Else!",
    "info" : {
        "directors" : [
            "Alice Smith",
            "Bob Jones"
        ],
        "release_date" : "2013-01-18T00:00:00Z",
        "rating" : 6.2,
        "genres" : [
            "Comedy",
            "Drama"
        ],
        "image_url" : "http://ia.media-
imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
        "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
        "rank" : 11,
        "running_time_secs" : 5215,
        "actors" :| [
            "David Matthewman",
            "Ann Thomas",
            "Jonathan G. Neff"
      ]
    }
}
```

Figure 1: JSON format used for loading movie data into the `Movies` table.

5.3.3 Creating and Reading Individual Items

You can add a new data item into an existing table by providing the primary key and the associated data.

Run the code provided in `MoviesItemOps01.py` to create a new data item.

Similarly, run the code provided in `MoviesItemOps02.py` to read an item from the table.

In both cases, read the code and observe the output to examine its effect.

5.3.4 Scripts for Further Operations

Further operations, including deletion of items, querying, and scanning, as described in the lecture, are available in the script files: `MoviesItemOps03.py`, `MoviesItemOps04.py`, `MoviesItemOps05.py`, `MoviesQuery01.py`, `MoviesQuery02.py`, and `MoviesScan.py`.

You may not need to use all of these for your projects, however, the ones related to deletion and querying can be use as staring points for further work. These can all be run and tested on the data already added to the Movies table.

6 Exercises **(Optional)**

Implement the following.

1. Measure the average round-trip time (RTT) between your local machine (Python UDP client) and the EC2 instance (Python UDP server) by sending a single integer. The RTT is the time taken for the integer to travel from the client to the server and back. Repeat this process 500 times and print the running average.
2. Write Python scripts to perform the following operations on the fully populated `Movies` table in DynamoDB.
    - Print complete information on the movie "After Hours" released in the year 1985.
    - Print all movies released before the year 2000.

■