Metropolitan State University
ICS 432 - 01: Distributed and Cloud Computing
Fall 2021

Lab 11: NoSQL Databases: AWS DynamoDB and GCP Firestore
Total points: 25
Out: Saturday, November 13, 2021
Due: 11:59 PM on Friday, November 19, 2021

What to submit?

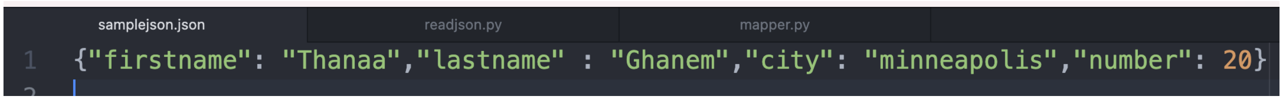
To complete this lab:

- Read this lab assignment carefully.
- At various parts of the lab, you are asked to take screen shots of your work. Open a word document and paste the screen shots in this document in the same order as mentioned in the lab. Make sure to highlight the screen shot number.
- After you complete all the lab exercises, upload the word document to the designated D2L folder by 11:59 PM on Friday, November 19, 2021.

NOTE: On Windows machines, you may consider using Snip & Sketch for screenshot handling. the lab, delete the project to save your credits.

Exercise 1: Reading JSON Documents using Python

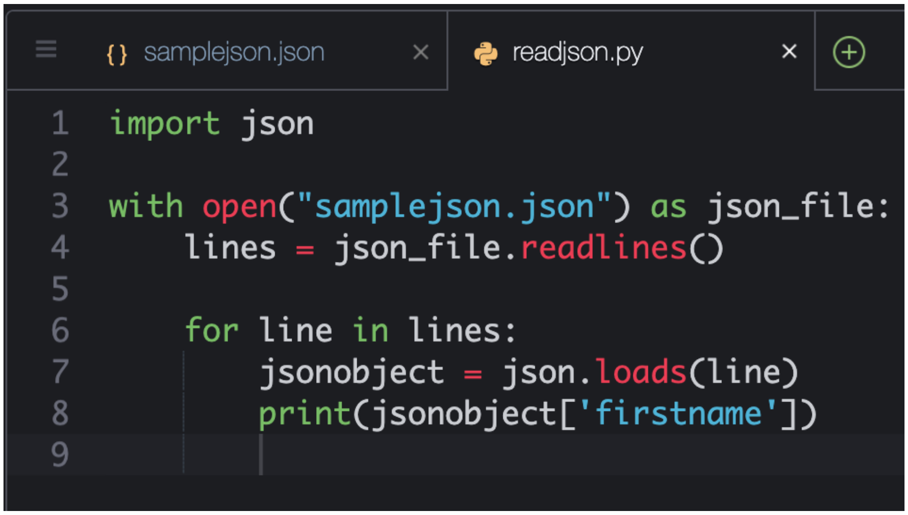
1- Log in to the AWS Management Console using your AWS Educate or AWS Academy Learner Lab account.
2- Open Cloud9, create a folder called Lab11NoSQL.
3- Create a file called `samplejson.json` and write the following JSON object in that file. Use your information instead of mine.



```
{"firstname": "Thanaa","lastname" : "Ghanem","city": "minneapolis","number": 20}
```

4- Examine the following code that reads and parses json objects. The code does the following:
   a. Imports `json` library: `import json`
   b. Opens the json file: `with open("samplejson.json") as json_file:`
   c. Read the whole file: `lines = json_file.readlines()`
   d. Parses the file one line at a time: `for line in lines:`

e. Loads each line into a json object: `jsonobject = json.loads(line)`
f. Prints the value of `'firstname'`: `print(jsonobject['firstname'])`



```python
import json

with open("samplejson.json") as json_file:
    lines = json_file.readlines()

    for line in lines:
        jsonobject = json.loads(line)
        print(jsonobject['firstname'])

```

5- In Cloud9, create a file called `readjson.py` and write the code in the above figure. Run the code to make sure it works as expected. Note, you may get an error about file cannot be found if you run the code by clicking on Run. To avoid this issue, in Cloud9 command window, use `cd` to go inside the Lab11NoSQL folder and run the code using the following command:

```
python readjson.py
```

6- Add lines to your code to print the values of `lastname` and `city`.
7- Add two more lines to your `samplejson.json` file.
8- Run `readjson.py` again.

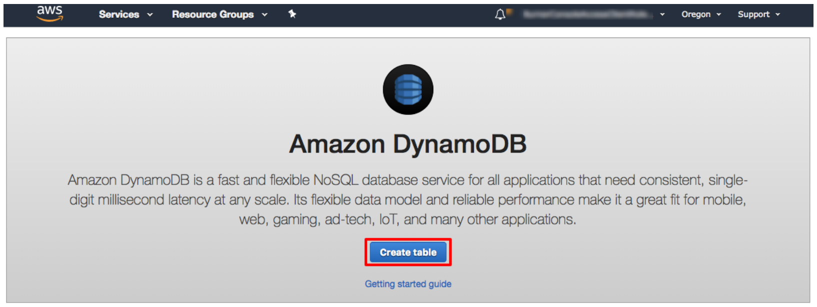
Lab screenshot #1: take a screenshot of your `samplejson.json` file.
Lab screenshot #2: take a screenshot of your `readjson.py` and the output after running the code.

Exercise 2: Working with AWS DynamoDB

Reference: AWS DynamoDB Developer Guide:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/dynamodb-dg.pdf

Part 1: Working with DynamoDB using AWS Console

1. From AWS Console, go to Services → *DynamoDB* and open the DynamoDB console.
2. In the DynamoDB console, choose **Create table**.



Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad-tech, IoT, and many other applications.

4- You are going to create a simple table to store data about students. Type <your-last-name>*Students* in the **Table name** box.

5- Each table in DynamoDB must have a **partition key** that is used to spread data across partitions for scalability. It's important to choose an attribute with a wide range of values and that is likely to have evenly distributed access patterns. Use the following for your table:

   a. **Partition key:** *department* -- type String.
   b. Sort key: lastname -- type String.

6- In the Table settings section, check the box next to **Use default settings**.

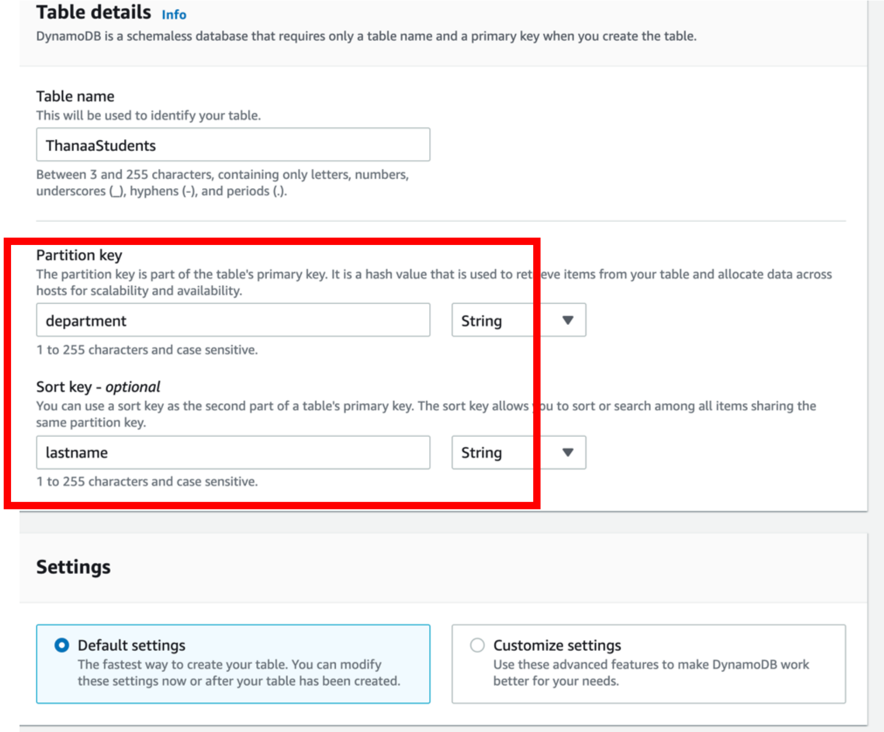


Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

ThanaaStudents

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

department | String ▼

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

lastname | String ▼

1 to 255 characters and case sensitive.

Settings

● Default settings
The fastest way to create your table. You can modify these settings now or after your table has been created.

○ Customize settings
Use these advanced features to make DynamoDB work better for your needs.

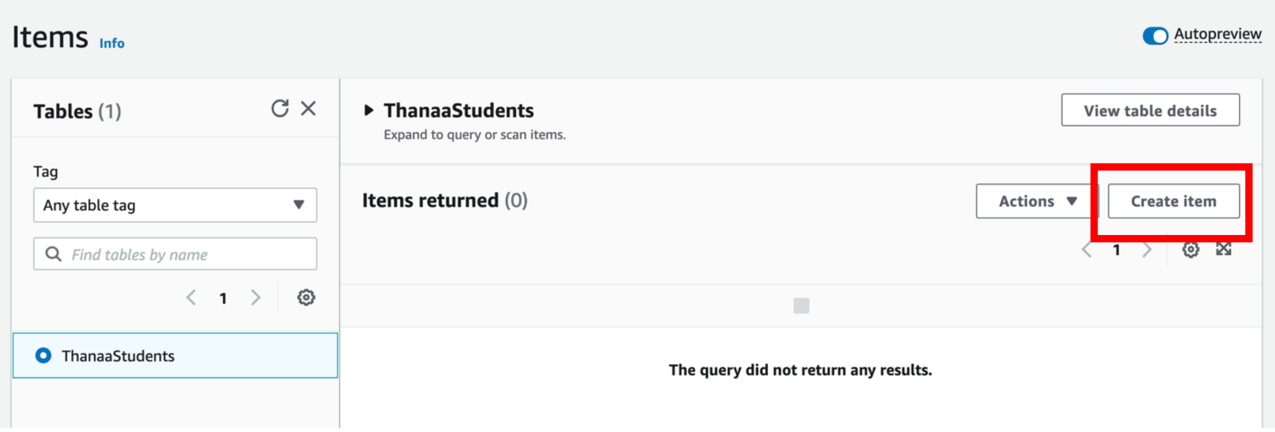
7- Scroll down and click on **Create table.**

8- Wait until the table is successfully created and the table status is Active. Click on the table name.

Lab screenshot #3: take a screenshot to show your table's Overview window.

9- You are going to insert data in the table. From the menu on the left, click on '**Items**' then click on your table name to select it.

10- Click on **'Create Item'.**



The query did not return any results.

11- When the Create Item window is open, you will find the department and lastname attributes as you specified when you created the table. Type **csc** in the department field and your last name in the last name field.

12- To add a new attribute of type String, click on the drop-down menu next to **Add new attribute** and choose String. For the attribute name, enter firstname and add your first name in the attribute value.

13- Insert another attribute called age of type Number and enter a value of your choice.

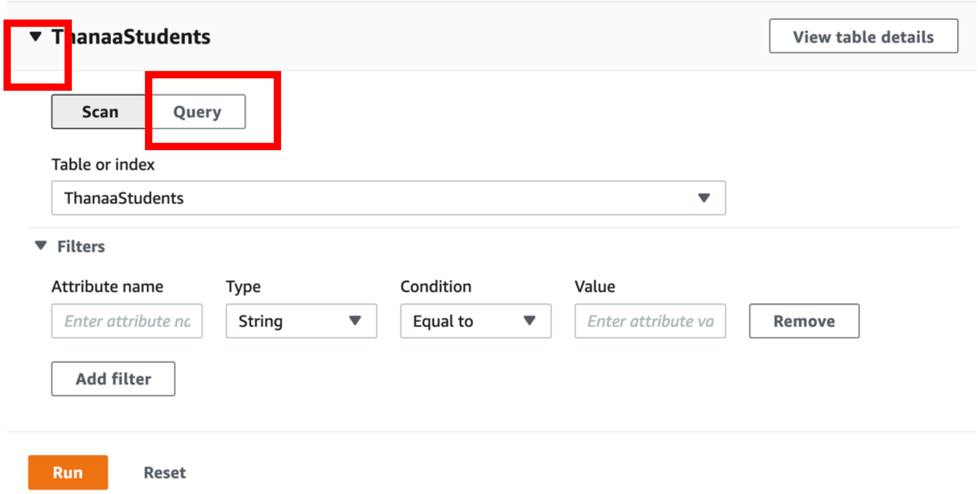
Lab screenshot #4: take a screenshot of the Create item screen after adding all attributes for that item.

14- Click on **Create item** to add the first student to the table.

15- Insert three other students in the table. At the end, your table should have two entries with department ***csc*** and two entries with department **math**.

Lab screenshot #5: take a screenshot of your table showing the four items.

16- Now you can issue queries to retrieve information from your table. Click on the arrow next to your table name to expand the query dashboard. Click on the arrow next to **Filters**.



17- To list all students in the **math** department, Click on Query, enter value ‘math’. Click Run to execute the query. Note that search in DynamoDB is case sensitive so use the exact same case as used for data entry in your table.

Lab screenshot #6: take a screenshot showing the results of your query.

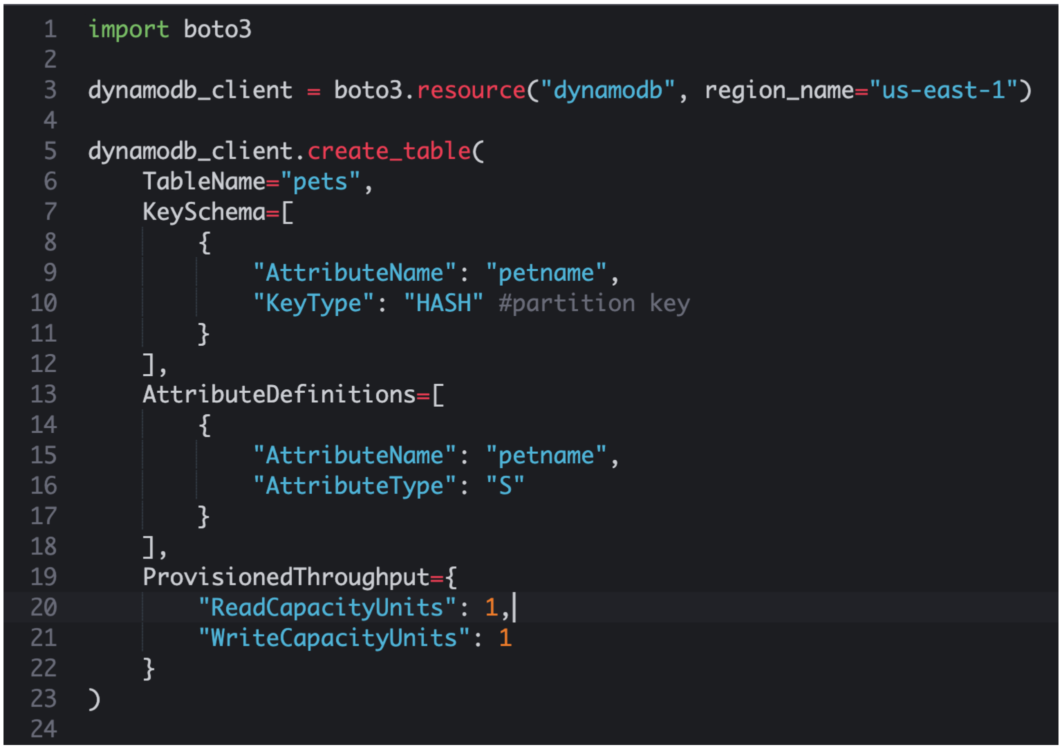
Part 2: Working with DynamoDB using Python SDK

In this part, you will write Python code to interact with DynmoDB using the following 4 tasks:

1- Create a DynamoDB table.
2- Add items to a DynamoDB table.
3- Update an item in a DynamoDB table
4- Populate the table with data from a json file.
5- Create a Global Secondary Index (GSI) on a DynamoDB table.
6- Performing CRUD operations.
7- Clean up.

Task 1: Create a DynamoDB table

1- In Cloud9, create a file called `creattable.py`
2- Write the following code to create a table named `pets`. The partition key for the table is an attribute called `petname`. You will not use this table often, so you think it's best to set the read capacity unit (RCU) to 1 and the write capacity unit (WCU) to 1.



```
import boto3

dynamodb_client = boto3.resource("dynamodb", region_name="us-east-1")

dynamodb_client.create_table(
    TableName="pets",
    KeySchema=[
        {
            "AttributeName": "petname",
            "KeyType": "HASH" #partition key
        }
    ],
    AttributeDefinitions=[
        {
            "AttributeName": "petname",
            "AttributeType": "S"
        }
    ],
    ProvisionedThroughput={
        "ReadCapacityUnits": 1,
        "WriteCapacityUnits": 1
    }
)

```

3- In Cloud9's command window, run the `createtable.py` file using the following command:

```
python createtable.py
```

Lab screenshot #7: take a screenshot showing your python code.

4- In a separate browser tab, go to Amazon DynamoDB console and click on **Tables**. If your code worked as expected, you will see that your table has been created in the US East (N. Virginia) Region. Wait for the status your table to change to Active before you proceed.

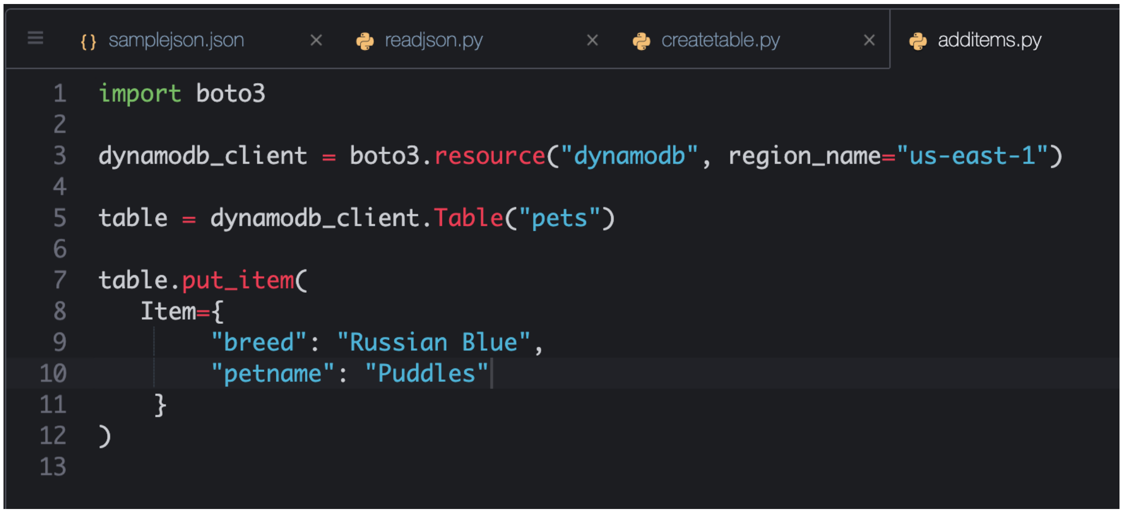
Lab screenshot #8: take a screenshot your DynamoDB console showing the **Overview** of your `pets` table.

Task 2: Add items to a DynamoDB table

1- Refer to the following link for documentation about the method to use to add items to the table:
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html#DynamoDB.Client.put_item

2- In your Lab's folder on Cloud9, create a file called `additems.py`

3- Write the following code to add one item to your `pets` table.



```python
import boto3

dynamodb_client = boto3.resource("dynamodb", region_name="us-east-1")

table = dynamodb_client.Table("pets")

table.put_item(
   Item={
        "breed": "Russian Blue",
        "petname": "Puddles"
    }
)

```

4- Add code to insert two more pets to your table with values of your choice.

5- Run `additems.py`

Lab screenshot #9: take a screenshot of your python code.
Lab screenshot #10: go to DynamoDB console and take a screenshot of your table showing the three pets.

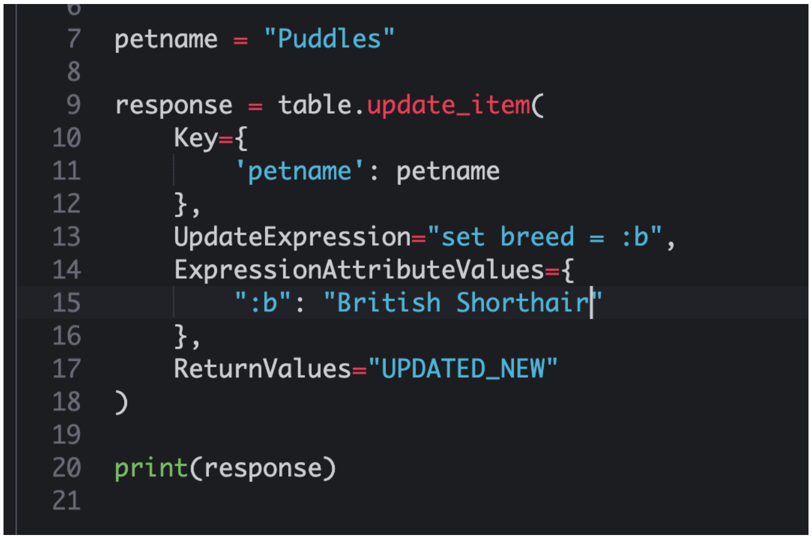
Task 3: Update an item in a DynamoDB table

1- You added data to your database, but you discover that `Puddles` is not a `Russian Blue`. He is a British Shorthair. You could go into the Amazon DynamoDB console and edit this entry, but instead you decide to write some code to update the item.

2- Refer to the following link for documentation about the method to use to add items to the table:
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html#DynamoDB.Client.update_item

3- In Cloud9, create a new file called `updateitem.py.`

4- Write the following code to update the item. Make sure to add code to define the DynamoDB client.



```
petname = "Puddles"

response = table.update_item(
    Key={
        'petname': petname
    },
    UpdateExpression="set breed = :b",
    ExpressionAttributeValues={
        ":b": "British Shorthair"
    },
    ReturnValues="UPDATED_NEW"
)

print(response)

```

5- Run `updateitem.py`

6- Confirm that your code works by going to DynamoDB console and checking on Items in `pets` table.

Lab screenshot #11: go to DynamoDB console and take a screenshot of your table showing the updated pet.

Task 4: Populate the table with data from a json file

1- Download the file `pets_data.json` from D2L and upload the file to your Cloud9 directory.

2- Create a file called `uploaddata.py`.

3- Write code similar to Exercise 1 to read all lines from `pets_data.json` and print the `petname` attribute on the screen. Run `uploaddata.py`

Lab screenshot #12: take a screenshot of your code and the corresponding output.

4- Change the code you wrote in step #3 by adding a line inside your for loop to call `put_item` for each line you read to insert the pet to the pets table. Run `uploaddata.py`

Lab screenshot #13: take a screenshot of your python code.

Lab screenshot #14: go to DynamoDB console and take a screenshot of your table showing the uploaded data.

Task 5: Create a GSI on a DynamoDB table

1- In this task, you will create a Global Secondary Index (GSI) on the `breed` attribute of the `pets` table to be able to issue queries on breeds.

2- Create a file on Cloud9 called `addindex.py` and add the following code to create an index on the `breed` attribute.

```
import boto3

dynamodb_client = boto3.client("dynamodb", region_name="us-east-1")

res = dynamodb_client.update_table(
    AttributeDefinitions=[
        {
                "AttributeName": "breed",
                "AttributeType": "S"
        },
    ],
    TableName="pets",
    GlobalSecondaryIndexUpdates=[
        {
            'Create': {
                'IndexName': 'breed_index',
                'KeySchema': [
                    {
                        'AttributeName': 'breed',
                        'KeyType': 'HASH'
                    }
                ],
                'Projection': {
                    'ProjectionType': 'ALL'
                },
                'ProvisionedThroughput': {
                    'ReadCapacityUnits': 1,
                    'WriteCapacityUnits': 1
                }
            }
        }
    ],
)
print(res)
```

3- Go to your table's dashboard in DynamoDB console. Click on **Indexes** tab.

Lab screenshot #15: take a screenshot showing that `breed_index` is successfully created.

4- Go to **Items** tab. Click on the arrow next to table name to 'Expand to query or scan items'. Click on the drop-down menu for **Table or index** and choose `breed_index`. Click on Run. If you get an error about 'Cannot read from backfilling global secondary index', wait for few minutes and try again because creating the index may take few minutes.

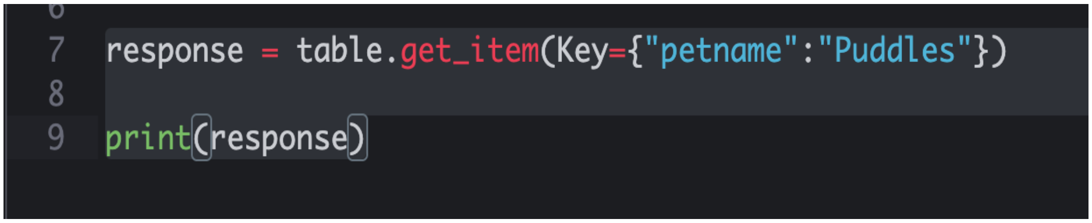
Lab screenshot #16: take a screenshot showing to show the Russian Blue pets.

Task 6: Performing CRUD operations

1- Create: we used `put_item` in Task 2 to create an item in a DynamoDB table.

2- Update: we used `update_item` in Task 3 to update an item in a DynamoDB table.

3- **R**ead: the method `get_item` is used to retrieve an item from the table given the value of partition key (i.e., `petname`). For example, the following code can be used to retrieve information about pet with `petname` "Puddles".



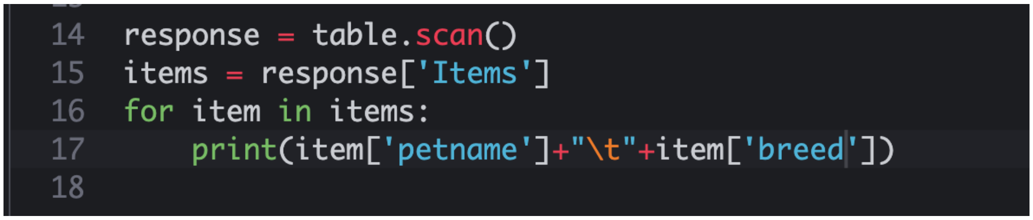
```
response = table.get_item(Key={"petname":"Puddles"})

print(response)
```

4- Create a file called `search.py` and write code to search for and print information for a pet with name "Bella". Run your code.

Lab screenshot #17: take a screenshot of your python code and the corresponding output.

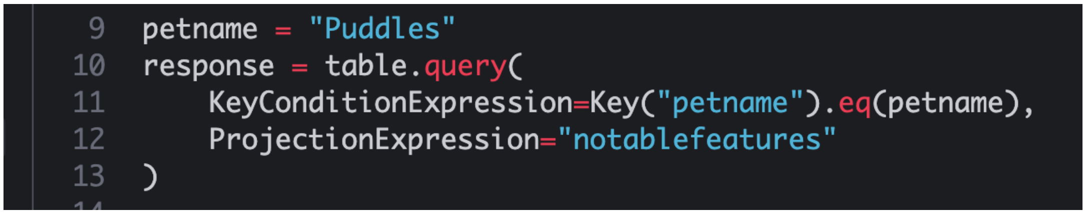
5- **Scan:** `scan` is used to return all items in the table. For example**,** the following code prints `petname` and `breed` for all items in the table. Add code to your `search.py` to scan the table.



```
response = table.scan()
items = response['Items']
for item in items:
    print(item['petname']+"\t"+item['breed'])
```

Lab screenshot #18: take a screenshot to show the output of the `scan` query.

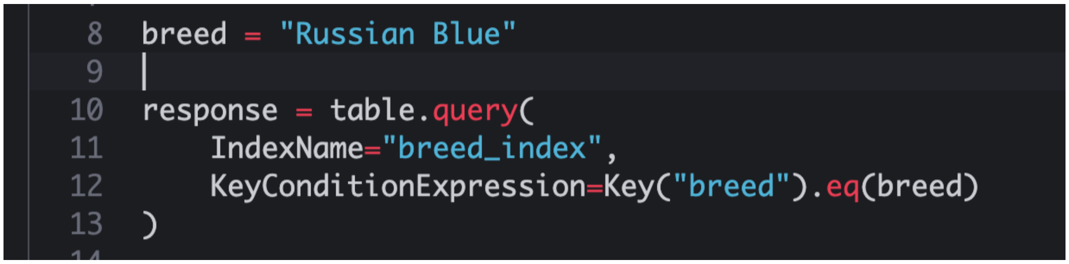
6- **Query:** the `query` method in the following code displays only `notablefetures` for a given `petname`. Note that you must specify the partition key on the query. Execute this query.



```
petname = "Puddles"
response = table.query(
    KeyConditionExpression=Key("petname").eq(petname),
    ProjectionExpression="notablefeatures"
)
```

Lab screenshot #19: take a screenshot to show the output of the `query` using a bet name other than Puddles.

7- In order to execute a query, you must use either partition key or GSI. Run following query that uses `breed_index` to retrieve all pets with “Russian Blue” breed.

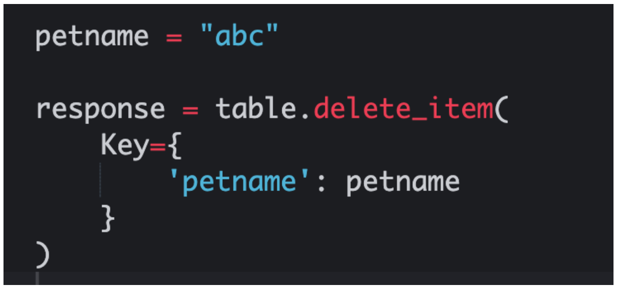


```
breed = "Russian Blue"

response = table.query(
    IndexName="breed_index",
    KeyConditionExpression=Key("breed").eq(breed)
)
```

Lab screenshot #20: take a screenshot to show the output of the `query` using `breed_index`. Note that there could be more than one item with the same breed and hence you need to print the output of this query using a `for` loop.

8- **D**elete: the `delete` method is used to delete a specific item from the table given the partition name. Execute the delete method to delete one item from your table.



```
petname = "abc"

response = table.delete_item(
    Key={
        'petname': petname
    }
)
```

Lab screenshot #21: take a screenshot of your DynamoDB table showing that the item you chose is deleted.

Task 7: Clean up

Delete all the DynamoDB tables created in the exercises.

Exercise 3: Google Firestore Document Database

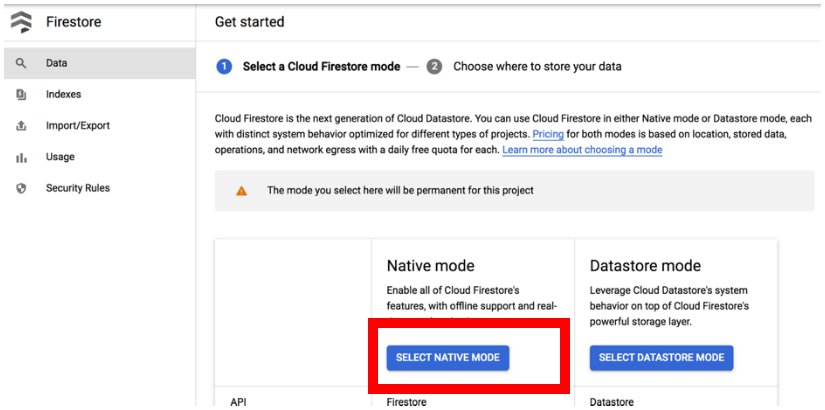
In this part of the lab, we will practice using the GPC's Firestore document database by creating a Pets collection and populating it with data from the file `pets_data.json`. Make sure to distinguish between Firestore which is a NoSQL database and Filestore which is a NFS file system.

You will complete this exercise in the following three tasks.

1. Create a Firestore table and upload data to the table using the console.
2. Use Python to add items to and query the Firestore table.
3. Clean up.

Task 1: Creating a table and inserting data using Firestore Console

1. From GCP main menu, go to **DATABASES → Firestore → Data**.
2. Click on **SELECT NATIVE MODE**.



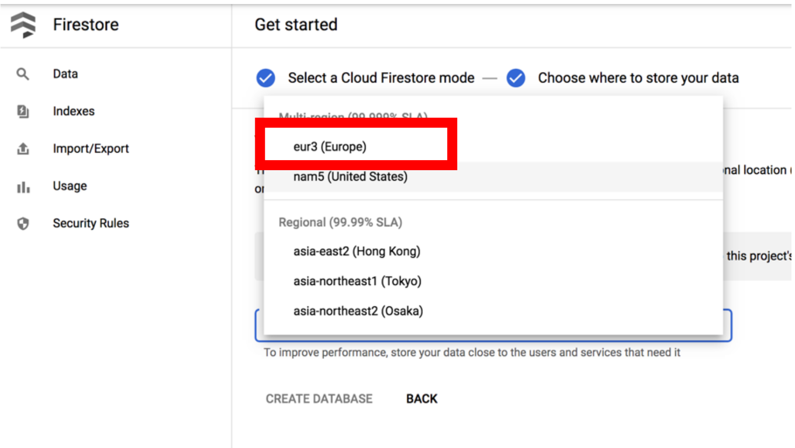
Cloud Firestore is the next generation of Cloud Datastore. You can use Cloud Firestore in either Native mode or Datastore mode, each with distinct system behavior optimized for different types of projects. Pricing for both modes is based on location, stored data, operations, and network egress with a daily free quota for each. Learn more about choosing a mode

The mode you select here will be permanent for this project

Native mode
Enable all of Cloud Firestore's features, with offline support and real-

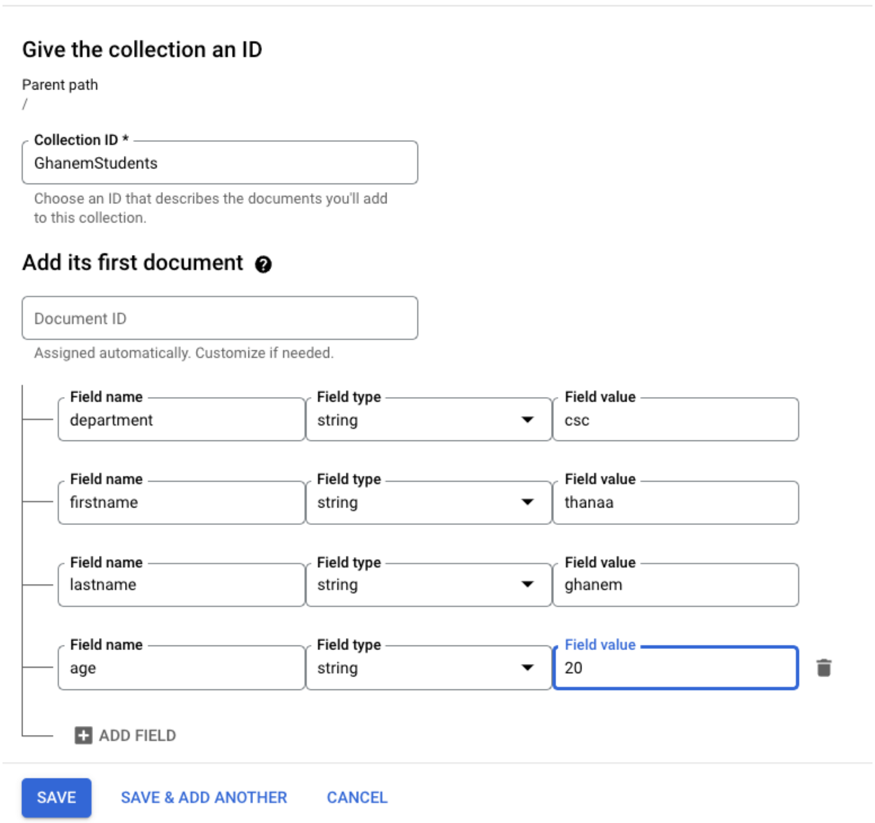
Datastore mode
Leverage Cloud Datastore's system behavior on top of Cloud Firestore's powerful storage layer.

3. Select location as **nam5 (United States)** then click on **CREATE DATABASE.**



To improve performance, store your data close to the users and services that need it

4. Click on **START COLEECTION** then create a collection (i.e., table) called <your-last-name>Students and insert the first document in the collection as shown in the following figure but replace my information with values of your choice. Click on **(+ ADD FIELD)** to add fields to the document. After inserting the four fields, click on **SAVE**.



Give the collection an ID

Parent path
/

Collection ID *
GhanemStudents

Choose an ID that describes the documents you'll add to this collection.

Add its first document

Document ID

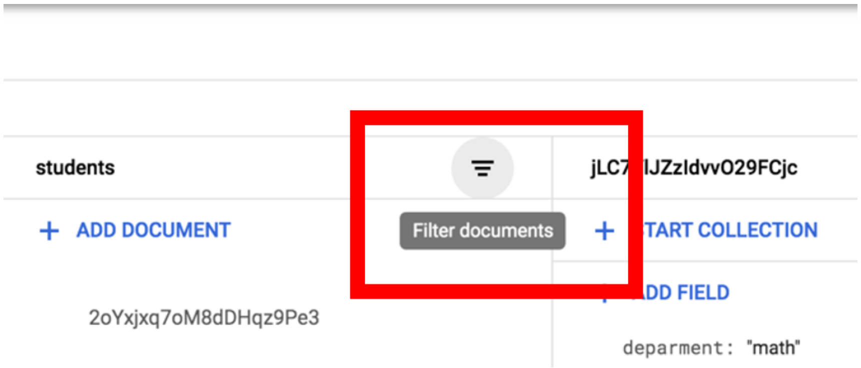
Assigned automatically. Customize if needed.

Field name	Field type	Field value
department	string	csc
firstname	string	thanaa
lastname	string	ghanem
age	string	20

5. Click on ADD DOCUMENT to add another student. Add three more students where two students are from csc department and two from math department.

Lab screenshot #22: take a screenshot to show the name of your collection and the four items.

6. To run a query on the collection, add on Filter documents.



students

jLC7 lJZzIdvvO29FCjc

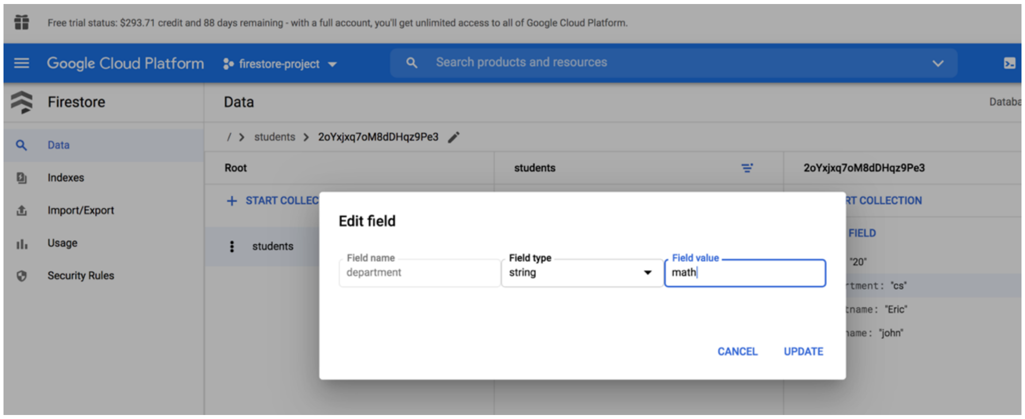
2oYxjxq7oM8dDHqz9Pe3

deparment: "math"

7. To list all students in the csc department, enter a filtering document as shown in the following figure. You have to specify a field to filter by (department), operation (==), value (csc), then click **APPLY**. Copy the query code that is automatically generated based on your filter and paste it in your lab report.

Lab screenshot #23: take a screenshot of the output of the filter query.

8. You can update the fields in any document by hovering over the field and inserting the new value. For example, to update the department, click on department, click on the pencil symbol, the value to 'math' and then click **UPDATE**.



Free trial status: $293.71 credit and 88 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform.

/ > students > 2oYxjxq7oM8dDHqz9Pe3

"20"
rtment: "cs"
tname: "Eric"
name: "john"

Edit field

Field name: department
Field type: string
Field value: math

CANCEL UPDATE

9. After changing the department, this document will be removed from the output of the csc filtering query.

Lab screenshot #24: take a screenshot of the output of the filter query after updating one document in the collection.

Task 2: Using Python to read from and write to a Firestore table

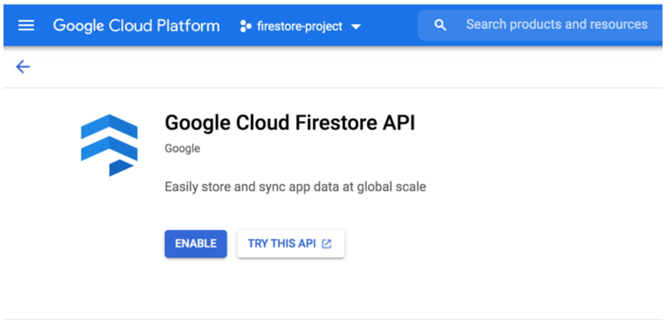
1. Open Cloud Shell.
2. Configure the command shell to the current project where <project ID> is the id of your firestore project. If prompted, click **Authorize**.

```
gcloud config set project <project_ID>
```

3. Run the following command to download the required libraries.

```
pip3 install firebase-admin google-cloud-firestore
```

4. Enable Cloud Firestore API by from GCP main menu → APIs & Services → Library. Search for Firestore. Click on the output of the search and then and click on **Enable**.



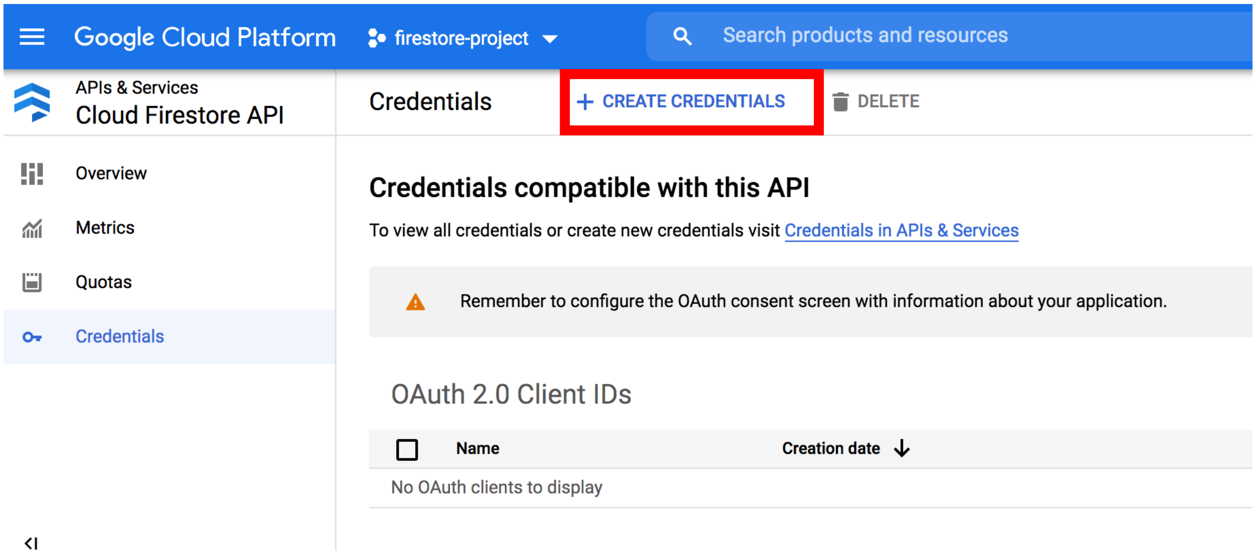
Google Cloud Firestore API

Google

Easily store and sync app data at global scale

5- To connect to Firestore from Python, you need to create a certificate with appropriate credentials same as we created SSL certificates to connect to Cloud SQL. To create such certificate, you will create a Service Account and grant it access to Firestore.

6- After the API is enabled, click on Manage. From the left menu bar click on **Credentials** → **CREATE CREDENTIALS** then click on **Help me choose.**



Credentials compatible with this API

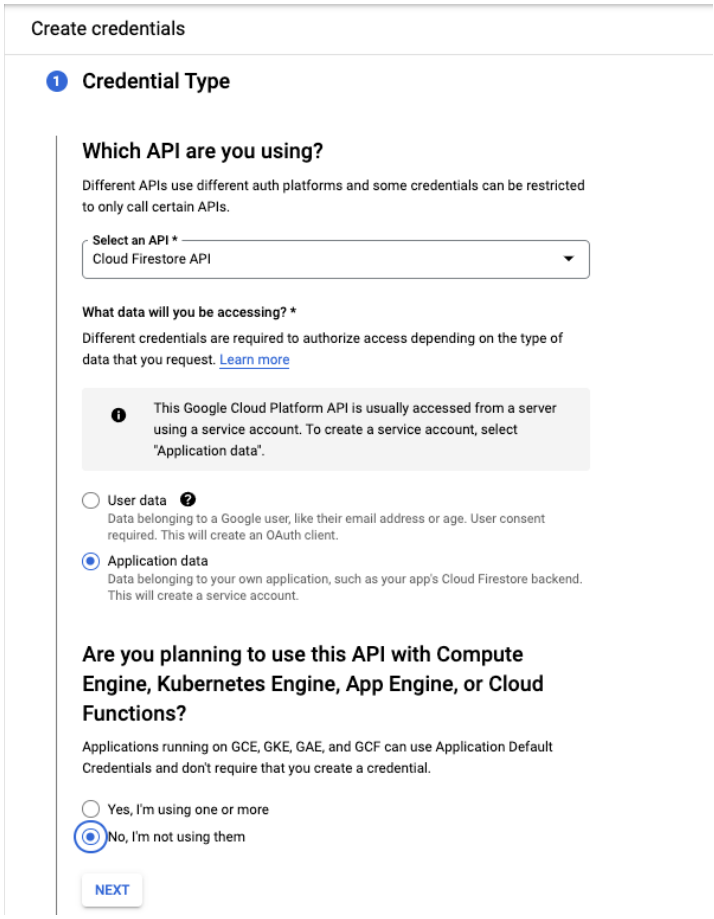
To view all credentials or create new credentials visit Credentials in APIs & Services

Remember to configure the OAuth consent screen with information about your application.

OAuth 2.0 Client IDs

□	Name	Creation date ↓
No OAuth clients to display		

7- In the Credential Type page, choose 'Application data' and choose 'No, I am using them'. Click on Next.



Create credentials

1 Credential Type

Which API are you using?

Different APIs use different auth platforms and some credentials can be restricted to only call certain APIs.

Select an API * Cloud Firestore API

What data will you be accessing? *

Different credentials are required to authorize access depending on the type of data that you request. Learn more

This Google Cloud Platform API is usually accessed from a server using a service account. To create a service account, select "Application data".

○ User data
Data belonging to a Google user, like their email address or age. User consent required. This will create an OAuth client.

● Application data
Data belonging to your own application, such as your app's Cloud Firestore backend. This will create a service account.

Are you planning to use this API with Compute Engine, Kubernetes Engine, App Engine, or Cloud Functions?

Applications running on GCE, GKE, GAE, and GCF can use Application Default Credentials and don't require that you create a credential.

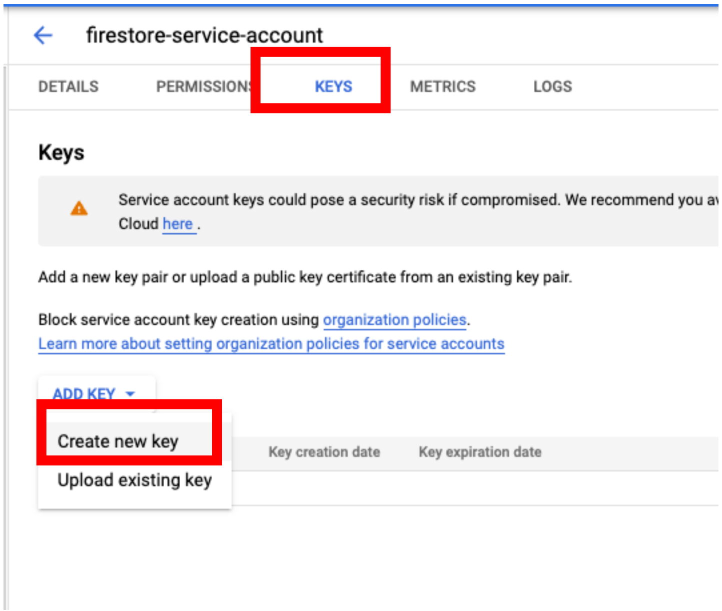
○ Yes, I'm using one or more

● No, I'm not using them

8- In the Service account details page, enter firestore-service-account as the account name. Click on **CREATE AND CONTINUE**.

9- In the **Grant this service account access to project** page, click on the dropdown menu for ‘Select a role’ and select **Firebase Rules System** as the role. Click on **CONTINUE**. Click on **DONE**.

10- A list of all service account will be displayed. Click on the service account you created. Then click on KEYS → Create new key.



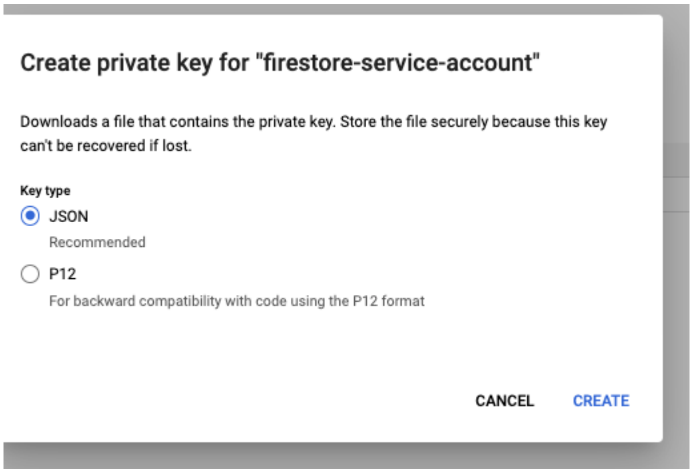
Keys

Service account keys could pose a security risk if compromised. We recommend you a
Cloud here .

Add a new key pair or upload a public key certificate from an existing key pair.

Block service account key creation using organization policies.
Learn more about setting organization policies for service accounts

11- Choose key type **json** and then click **CREATE**. A json file is downloaded to your computer.



Create private key for "firestore-service-account"

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

● JSON
Recommended

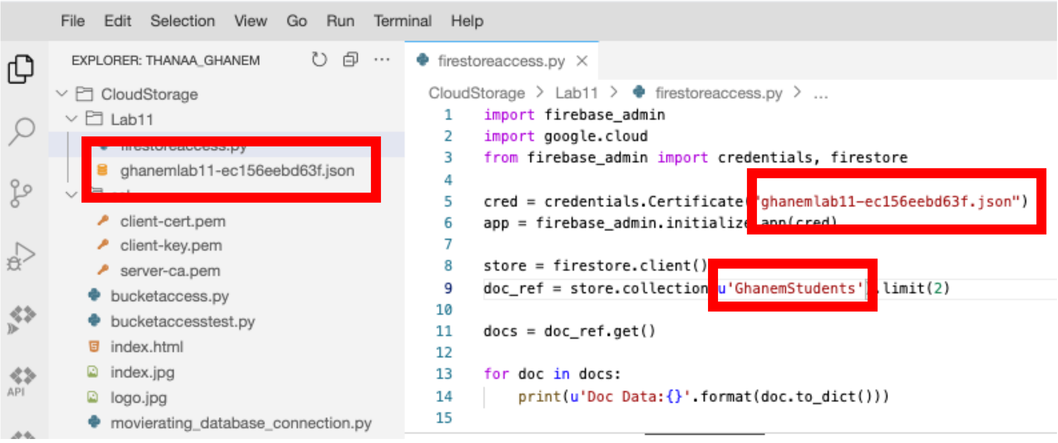
○ P12
For backward compatibility with code using the P12 format

12- Go to cloud shell, open Editor, create a folder called <your-last-name>Lab11. Upload the key json file from your computer to this folder.

Lab screenshot #25: take a screenshot of the folder on GCP editor and show the credentials json file is successfully uploaded.

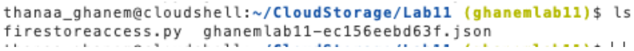
13- Inside Lab 11 folder, create a file called `firestoreaccess.py`.

14- Write the following code to retrieve two (2) documents from your students collection. Make sure to put the correct json file name on line 5. Make sure also to use your collection name instead of mine in line 9.



```
1   import firebase_admin
2   import google.cloud
3   from firebase_admin import credentials, firestore
4
5   cred = credentials.Certificate("ghanemlab11-ec156eebd63f.json")
6   app = firebase_admin.initialize_app(cred)
7
8   store = firestore.client()
9   doc_ref = store.collection(u'GhanemStudents').limit(2)
10
11  docs = doc_ref.get()
12
13  for doc in docs:
14      print(u'Doc Data:{}'.format(doc.to_dict()))
15
```

15- In the editor's command window, use `cd` to move inside your Lab11 folder. When you type `ls` you should see both the key json file and `firestoreaccess.py`.



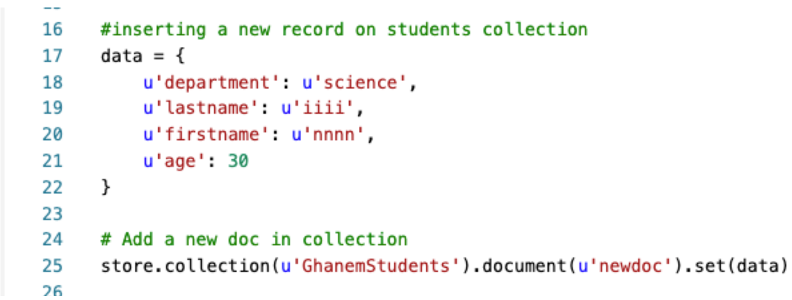
```
thanaa_ghanem@cloudshell:~/CloudStorage/Lab11 (ghanemlab11)$ ls
firestoreaccess.py  ghanemlab11-ec156eebd63f.json
```

16- Run the code using the following command:

```
python3 firestoreaccess.py
```

Lab screenshot #26: take a screenshot of your python code and the corresponding output.

17- Add the following code to your python file to add a new document to the students collection. Change the values of the attributes to values of your choice.



```
#inserting a new record on students collection
data = {
    u'department': u'science',
    u'lastname': u'iiii',
    u'firstname': u'nnnn',
    u'age': 30
}

# Add a new doc in collection
store.collection(u'GhanemStudents').document(u'newdoc').set(data)
```

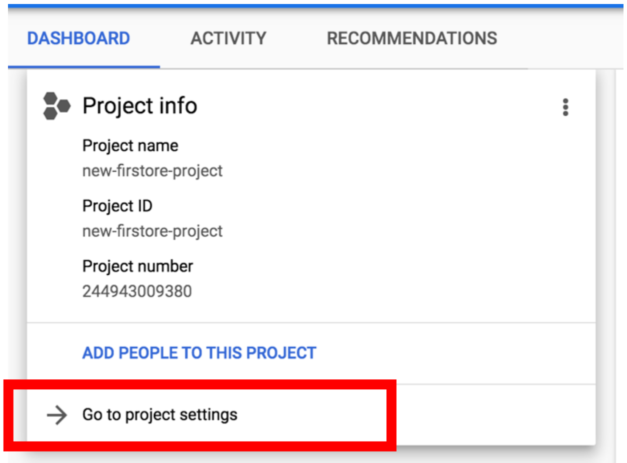
Lab screenshot #27: go to Firestore dashboard and show the contents of the students collection showing the newly added document.

18- In step 7 above, you copied the code to issue a query to return all document with csc department. Use this code in your python file to run the query.

Lab screenshot #28: take a screenshot of you python code and the output of running the query.

Task 3: Clean up

1. Shutdown the project that you created in this lab. To shout down a project, go to Project Settings.



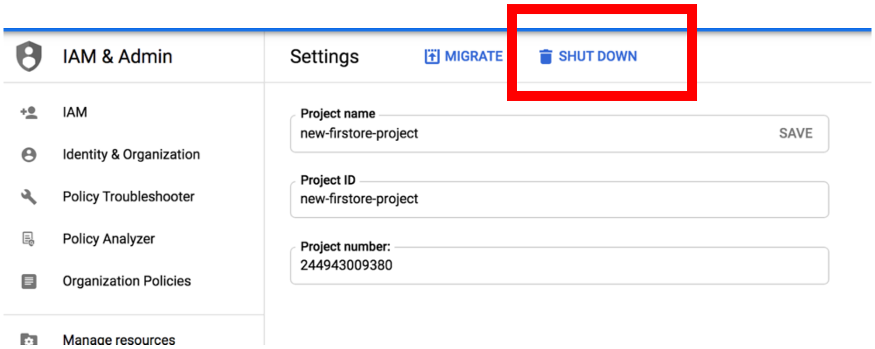
Project info

Project name
new-firstore-project

Project ID
new-firstore-project

Project number
244943009380

2. Click on shutdown.



Project name
new-firstore-project

Project ID
new-firstore-project

Project number:
244943009380