DaaS Design using FastAPI Serverless Architecture

Summary	In this codelab, we will learn how to create Synthetic Data using Snowflake
URL	https://colab.research.google.com/drive/1XBDLOG5zPvggb2kHRQIXXFXZ7-DsNRmb?usp=sharing
Category	Web
Status	Published
Author	Nikhil Kohli

Data as a Service Pipeline - Team 3

Data as a Service using Serverless Computing

"Data as a service (DaaS) is a data management strategy that uses the cloud to deliver data storage, integration, processing, and/or analytics services via a network connection."

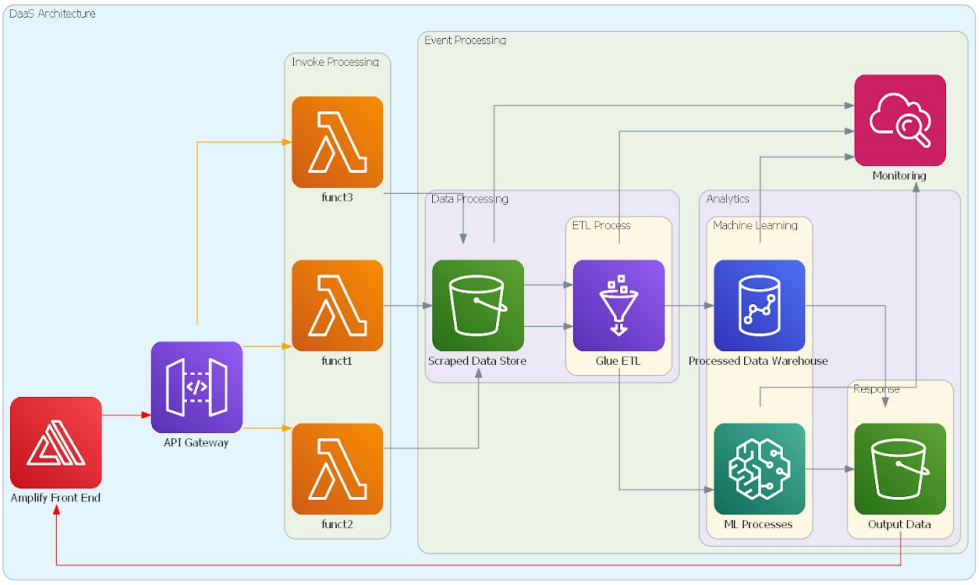
DaaS is similar to software as a service, or SaaS, a cloud computing strategy that involves delivering applications to end-users over the network, rather than having them run applications locally on their devices. Just as SaaS removes the need to install and manage software locally, DaaS outsources most data storage, integration, and processing operations to the cloud.

Moody’s Daas Design

Data-as-a-Service Architecture built in python using Diagrams

The architecture assumes that Moody’s be giving programmatic access to the API as well as a

front end access through AWS Amplify which checks the API Authentication using either a token based auth or any other means.



../DaaS Design

This is a clustered Event processing architecture, where few components like Sagemaker for

Machine learning can be used for Sentiment Analysis and Credit rating prediction using custom Sagemaker models.

The major processing steps happen in the Data processing stage using Glue ETL where all the transformations are applied to the data and then stored in Redshift Data warehouse for further processing and Analysis. Moody's might be using some other service as well for transformations to their data. Most of the requests are GET and PUT to fetch and update the data.

Data Ingestion

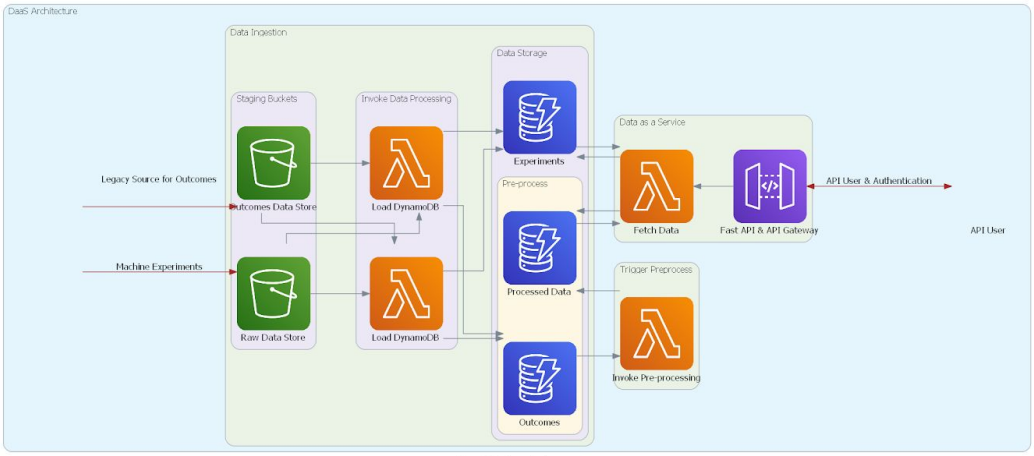
Data Ingestion

We staged the data in S3 buckets and used Serverless lambda architecture to invoke the process of storing this data into DynamoDB

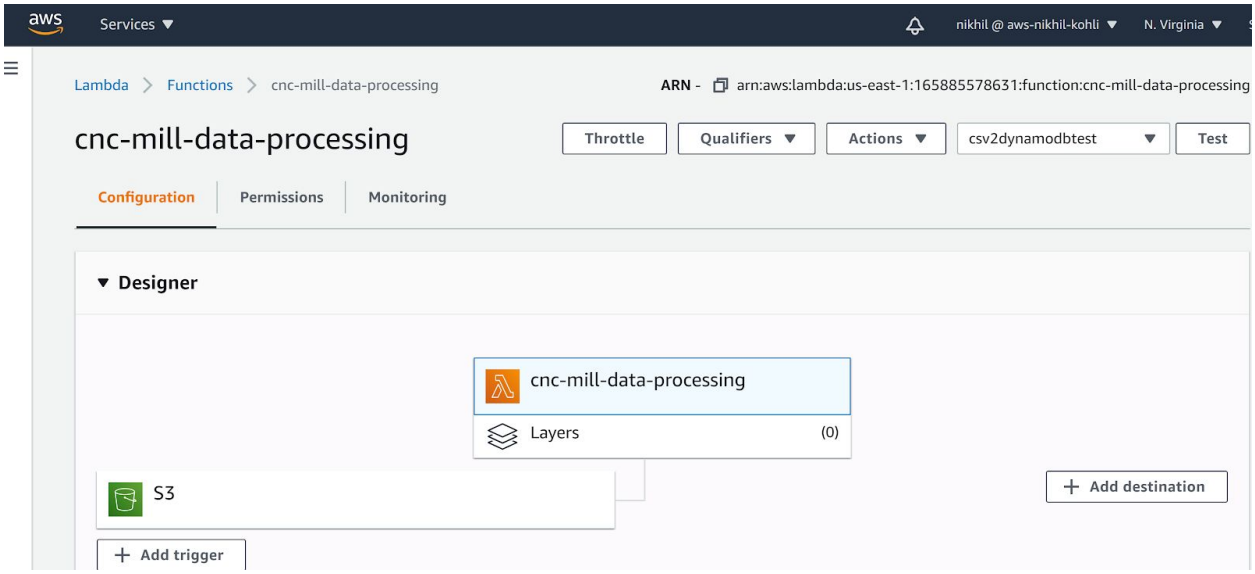
The overall architecture flow originates from the trigger event on file upload in two different S3 buckets. Both the buckets have data uploaded, which further invocates two different Lambda functions executing concurrently. The functions are mainly responsible for importing data from both the buckets into two different DynamoDB tables. Since one of the DynamoDB tables requires pre-processing for cleaning the underlying data another invocation of a different Lambda function happens to perform necessary transformations and data cleaning. This data is further stored in a new dynamodb table.

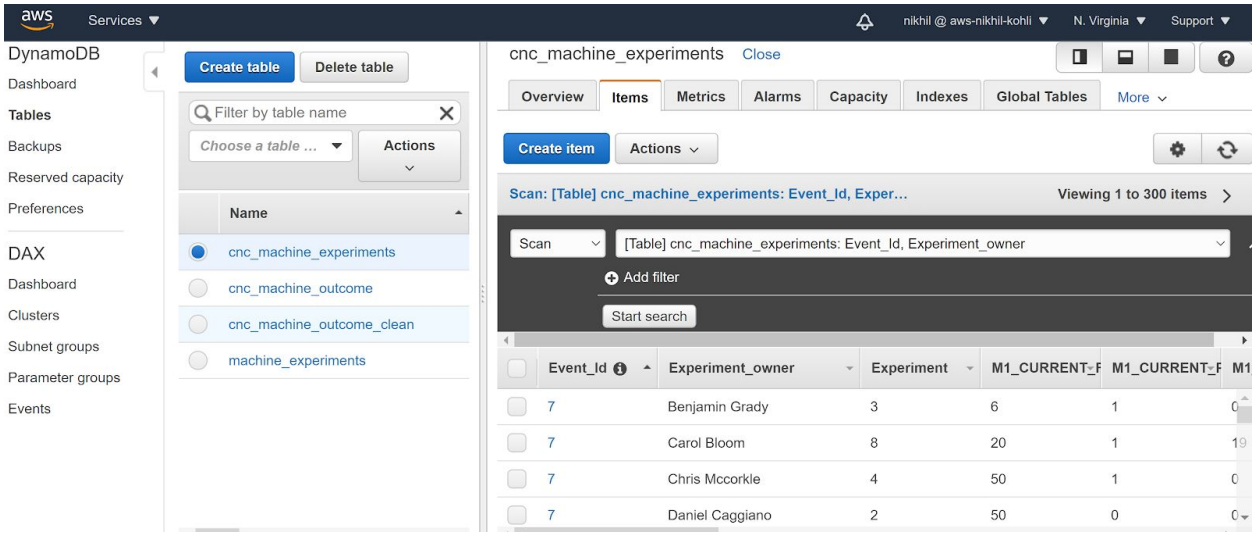
Addition to that a FastAPI application hosted locally connects to DynamoDB to fetch data from tables. Furthermore, to deploy the application in a cloud environment, the application is wrapped up and packaged using Mangum and Docker, in turn to be deployed in the AWS Cloud ecosystem mainly using a Lambda function and an API Gateway. API Gateway helps in deploying the application to be accessible anywhere.

We have implemented GET, PUT request types in FastAPI which Queries, Updates, Deletes the data if the user has the required authentication access key



../DaaS Design Assignment





FAST API Design

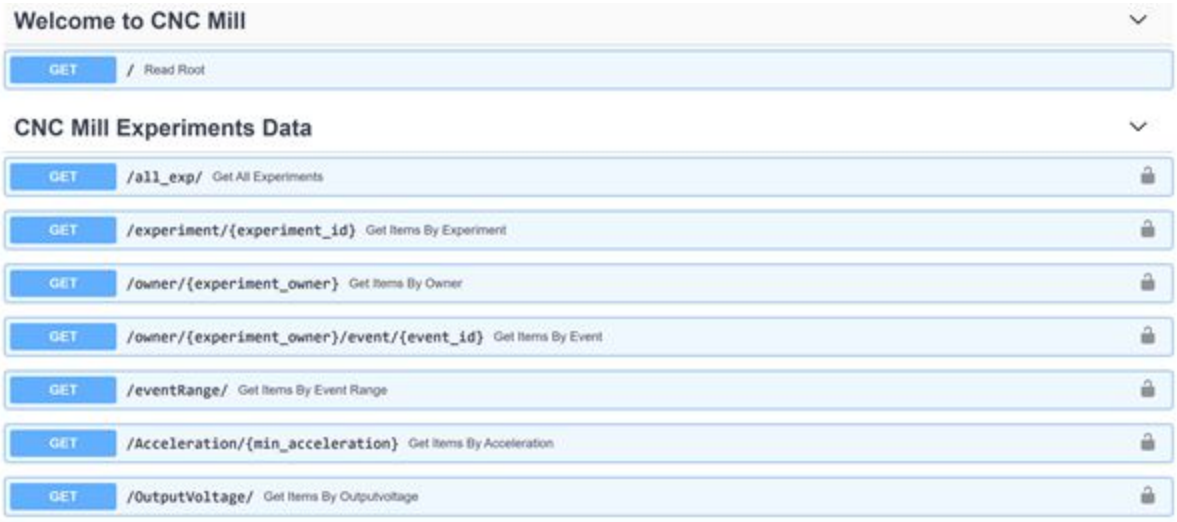
FAST API Design

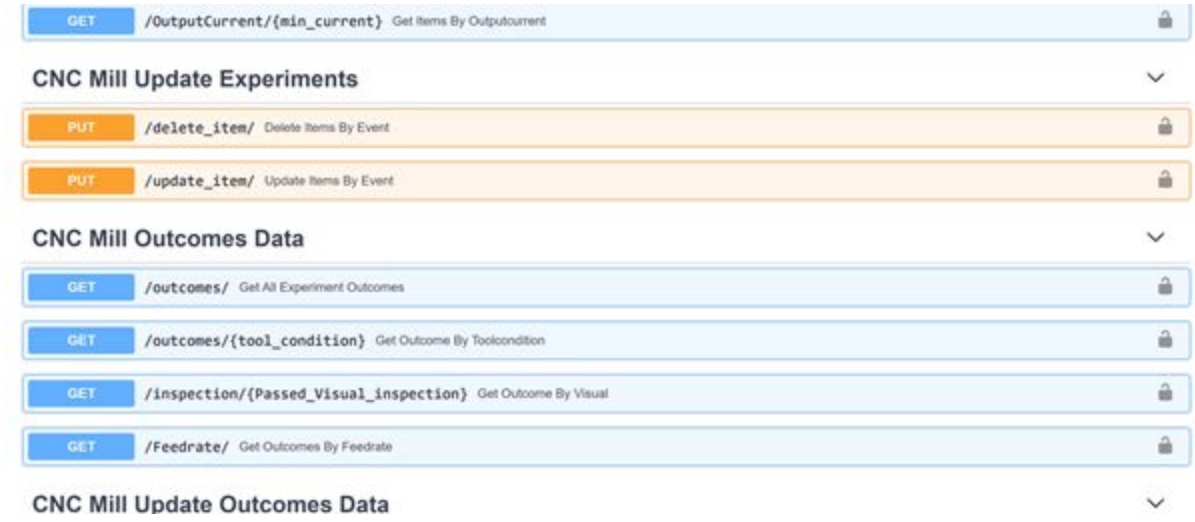
Creating the SQL statements by hand is labor-intensive, but fortunately you can automate the task. All you need is a specification that defines table and column names, data types, and additional statistical information.

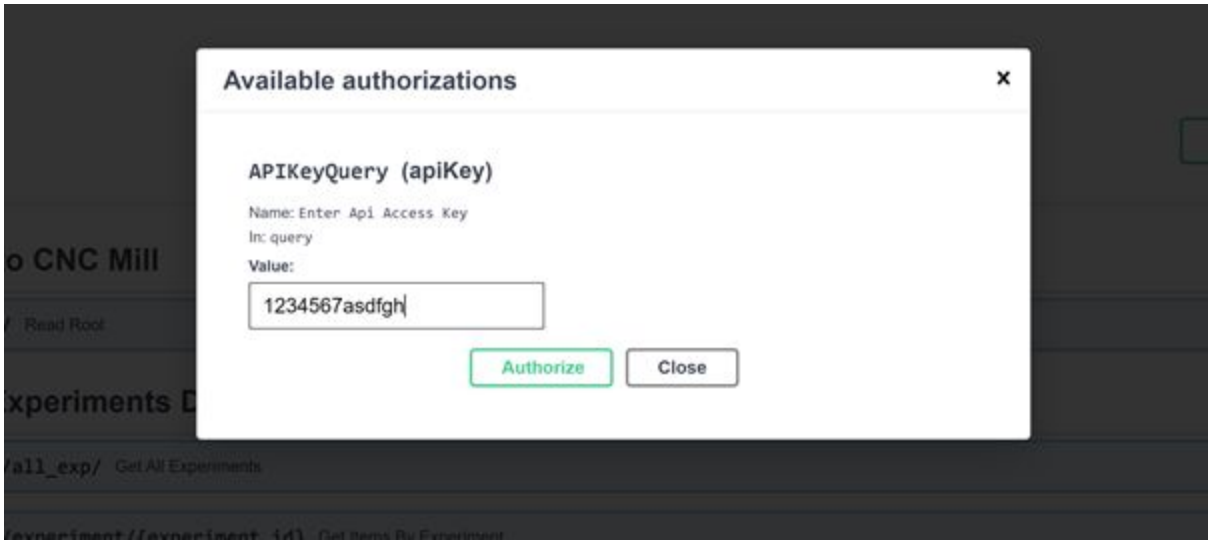
We have designed the FAST API using GET and PUT requests to query and analyse various aspects of our dataset as well as update and delete any records with the proper API Authentication.

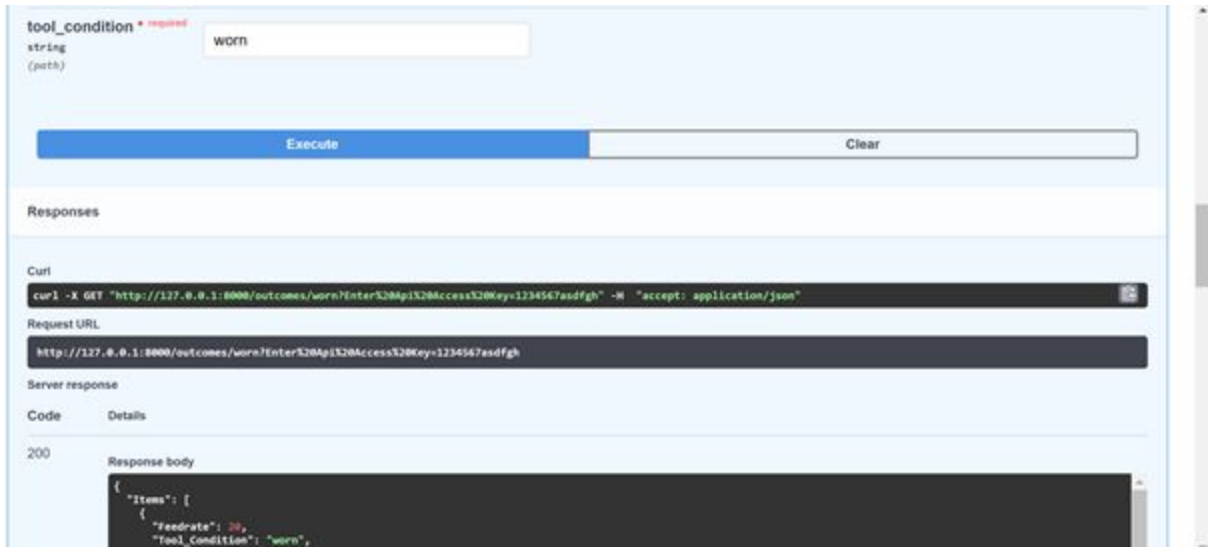
SCREENSHOTS

```python
region = 'us-east-1'
dynamodb_client = boto3.client('dynamodb', region_name = region)

TableName = "cnc_machine_experiments"
TableName_ = "cnc_machine_outcome"
dynamodb = boto3.resource('dynamodb', region_name=region)
table = dynamodb.Table(TableName)
table_ = dynamodb.Table(TableName_)


@app.get("/",tags=["Welcome to CNC Mill"])
async def read_root():
    return {"Hello": "World"}

@app.get("/all_exp/",tags=["CNC Mill Experiments Data"])
async def get_all_experiments(api_key: APIKey = Depends(get_api_key)):
    if APIKey:
        response = table.scan()
    else:
        response = JSONResponse(
        get_openapi(title="FastAPI security test", version=1, routes=app.routes)
        )
    return response
```




GET /OutputCurrent/{min_current} Get Items By Outputcurrent
CNC Mill Update Experiments
PUT /delete_item/ Delete Items By Event
PUT /update_item/ Update Items By Event
CNC Mill Outcomes Data
GET /outcomes/ Get All Experiment Outcomes
GET /outcomes/{tool_condition} Get Outcome By Toolcondition
GET /inspection/{Passed_Visual_inspection} Get Outcome By Visual
GET /Feedrate/ Get Outcomes By Feedrate
CNC Mill Update Outcomes Data

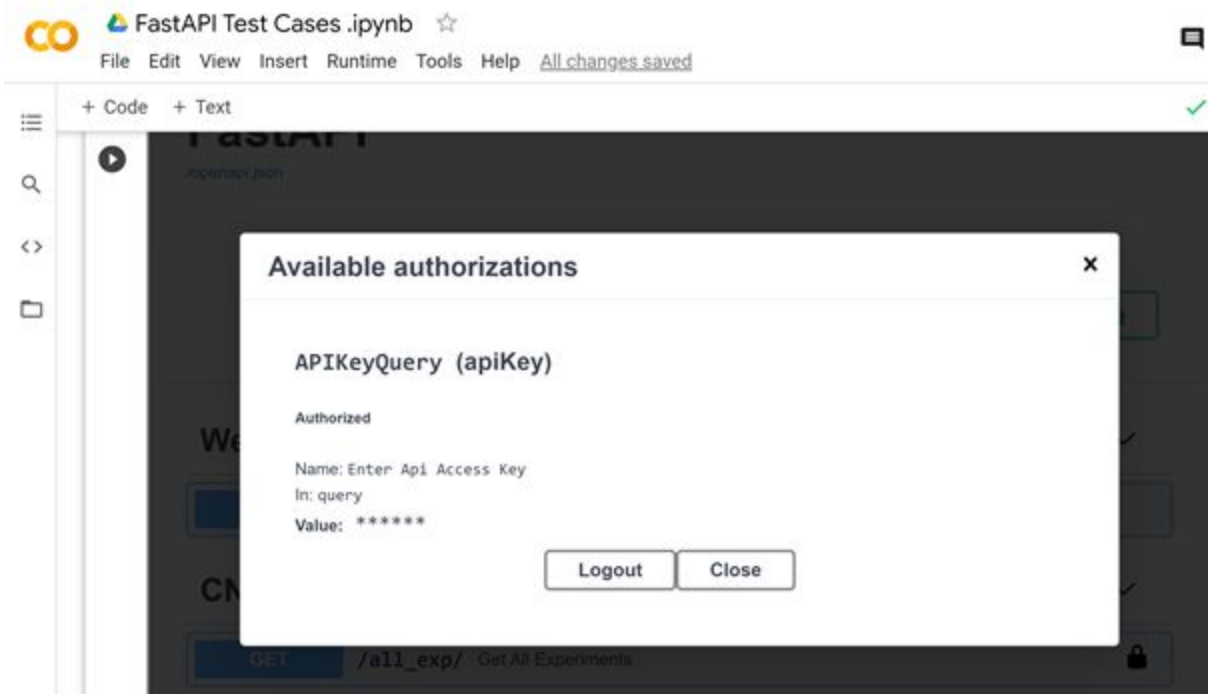

Available authorizations
APIKeyQuery (apiKey)
Name: Enter Api Access Key
In: query
Value:
1234567asdfgh
Authorize
Close

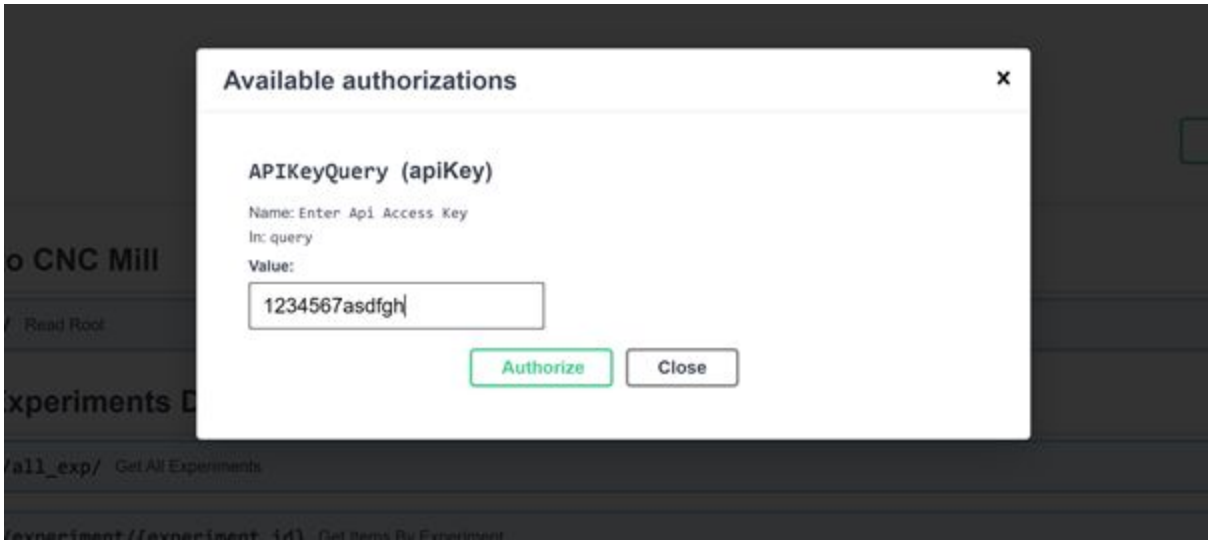

tool_condition * required
string
(path)
worn
Execute
Clear
Responses
Curl
curl -X GET "http://127.0.0.1:8000/outcomes/worn?Enter%20Api%20Access%20Key=1234567asdfgh" -H "accept: application/json"
Request URL
http://127.0.0.1:8000/outcomes/worn?Enter%20Api%20Access%20Key=1234567asdfgh
Server response
Code
Details
200
Response body
{
"Items": [
{
"Feedrate": 20,
"Tool_Condition": "worn",

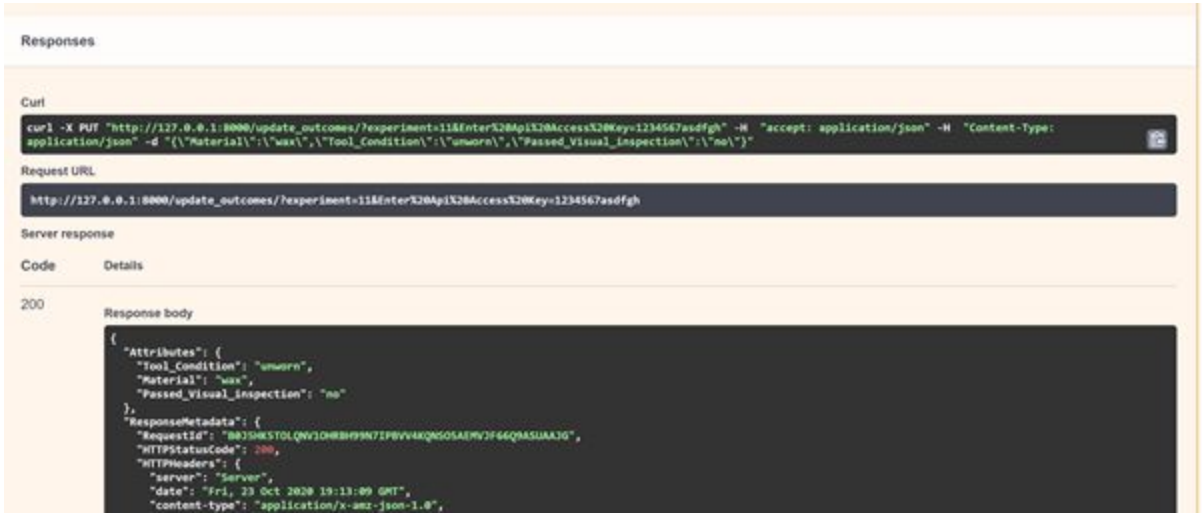
Enabling Key Based Authentication

Enabling Key Based Authentication

In our FastAPI application we have implemented API key based authentication for secure access for different endpoints. We have also established constraints in which the correct API key input will yield results or else the front end displays the error message.


FastAPI Test Cases .ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Available authorizations
APIKeyQuery (apiKey)
Authorized
Name: Enter Api Access Key
In: query
Value: ******
Logout
Close
/all_exp/ Get All Experiments


Available authorizations
APIKeyQuery (apiKey)
Name: Enter Api Access Key
In: query
Value:
1234567asdfgh
Authorize
Close

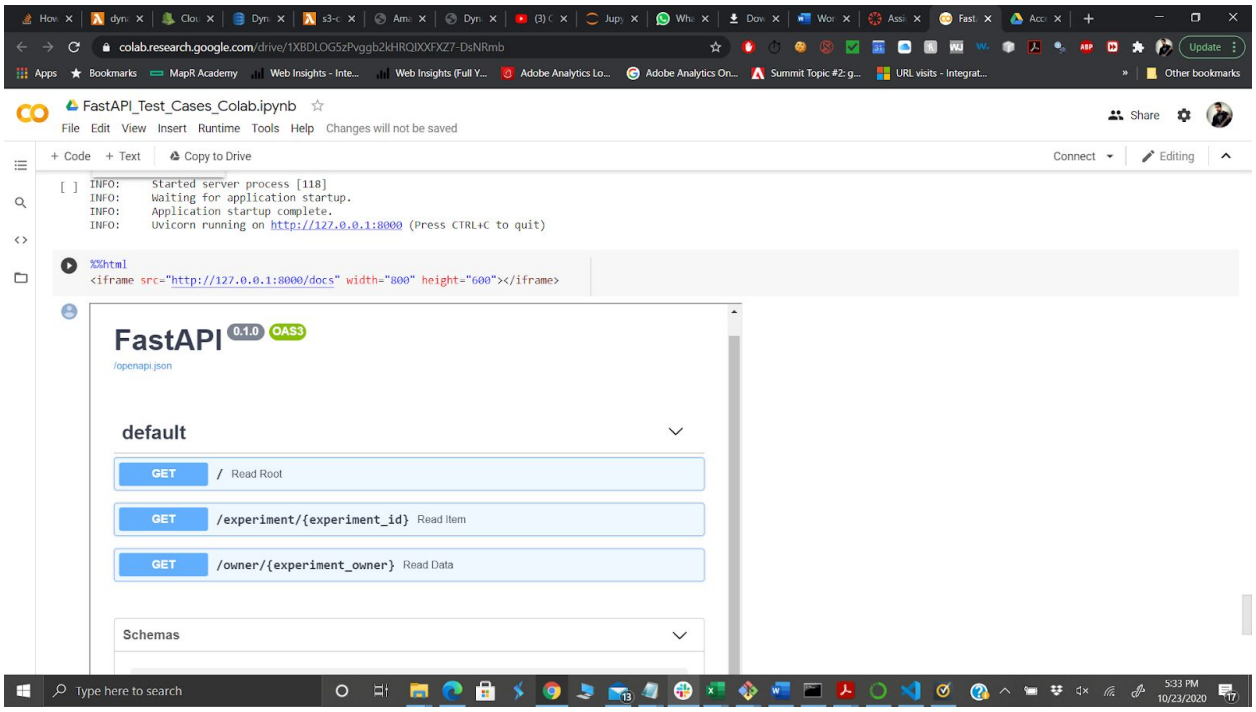

Responses
Curl
curl -X PUT "http://127.0.0.1:8000/update_outcomes/?experiment=11&Enter%20Api%20Access%20Key=1234567asdfgh" -H "accept: application/json" -H "Content-Type: application/json" -d "{\"Material\":\"wax\",\"Tool_Condition\":\"unworn\",\"Passed_Visual_Inspection\":\"no\"}"
Request URL
http://127.0.0.1:8000/update_outcomes/?experiment=11&Enter%20Api%20Access%20Key=1234567asdfgh
Server response
Code
Details
200
Response body
{
"Attributes": {
"Tool_Condition": "unworn",
"Material": "wax",
"Passed_Visual_Inspection": "no"
},
"ResponseMetadata": {
"RequestId": "B0JSHKSTOLQNV1OHRBH99N7IPBVV4KQNSO5AEMVJF66Q9ASUAAJG",
"HTTPStatusCode": 200,
"HTTPHeaders": {
"server": "Server",
"date": "Fri, 23 Oct 2020 19:13:09 GMT",
"content-type": "application/x-amz-json-1.0",


Responses
Curl
curl -X GET "http://127.0.0.1:8000/Feedrate/?min_feedrate=2&max_feedrate=5" -H "accept: application/json"
Request URL
http://127.0.0.1:8000/Feedrate/?min_feedrate=2&max_feedrate=5
Server response
Code
Details
403
Undocumented
Error: Forbidden
Response body
{
"detail": "Could not validate credentials"
}
Download
Response headers
content-length: 43

Testing the API

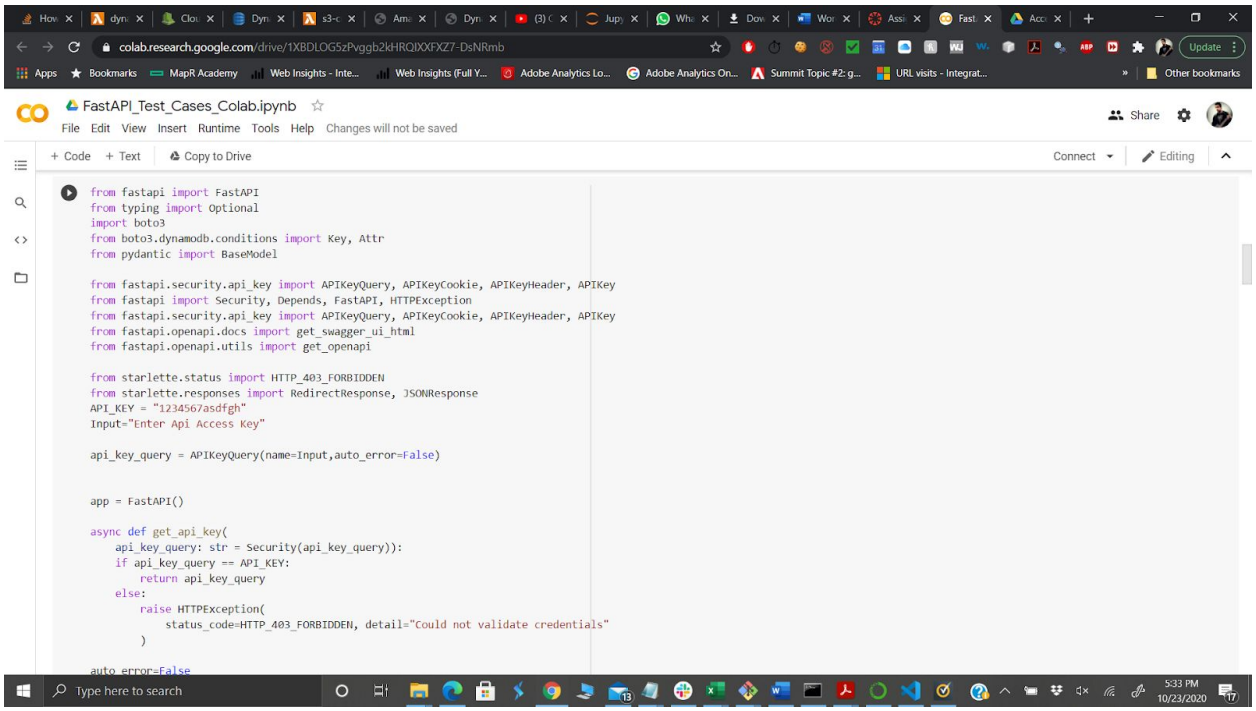
Testing the API

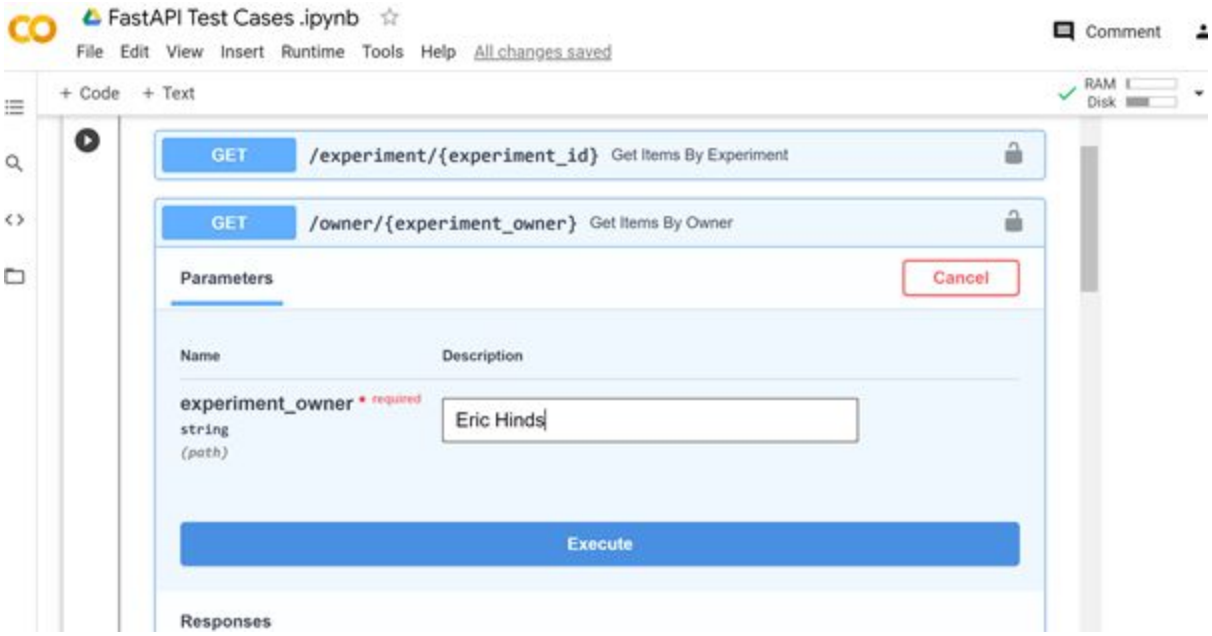
After establishing the connection between FastAPI application and DynamoDB tables.We used Google Colab to deploy test cases for our API and tested it on certain aspects of the dataset. Here are the screenshots for the same



[18]:
```
from boto3.dynamodb.conditions import Key

response = table.query(
            KeyConditionExpression= Key('Index').eq('1004'))

response['Items']
```

Out[18]:
```
[{'Y1_CommandAcceleration': '6',
  'X1_OutputCurrent': '326',
  'Z1_CurrentFeedback': '0',
  'X1_ActualAcceleration': '-62.5',
  'Y1_CommandPosition': '73.8',
  'Z1_OutputVoltage': '0',
  'S1_OutputCurrent': '317',
  'S1_OutputPower': '0.189',
  'S1_SystemInertia': '12',
  'Z1_ActualVelocity': '0',
  'X1_CommandPosition': '162',
  'S1_ActualAcceleration': '4.19',
  'Z1_OutputCurrent': '0',
  'Z1_CommandAcceleration': '0',
  'Y1_ActualPosition': '73.9',
  'M1 sequence number': '126',
```




FastAPI Test Cases .ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
+ Code + Text
RAM
Disk
GET /experiment/{experiment_id} Get Items By Experiment
GET /owner/{experiment_owner} Get Items By Owner
Parameters
Cancel
Name
Description
experiment_owner * required
string
(path)
Eric Hinds
Execute
Responses