Configuration Manual

MSc Research Project
Cloud Computing

Devashree Shedge
Student ID: x23155906

School of Computing
National College of Ireland

Supervisor: Sudarshan Deshmukh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Devashree Shedge
Student ID:	x23155906
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Sudarshan Deshmukh
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1124
Page Count:	9

Signature:	
Date:	11th December 2024

Configuration Manual

Devashree Shedge
x23155906

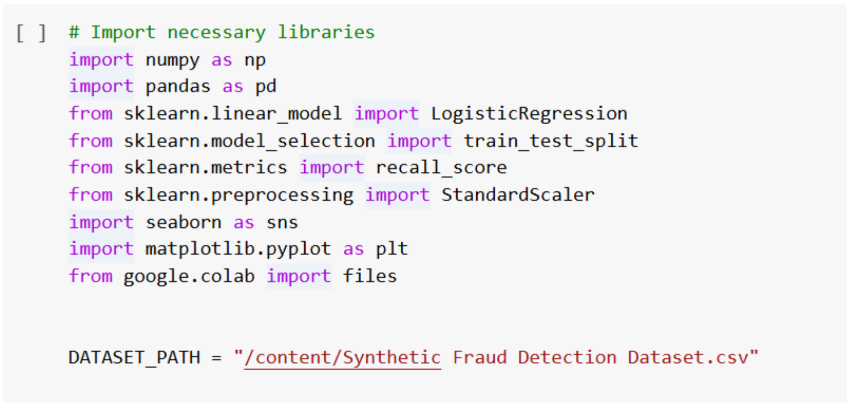
1 Introduction

This document are based on the experimental configuration and setting up the research.The evaluation of Particle Swarm Optimization (PSO) algorithm in optimizing the serverless computing with the main aim of achieving performance and cost efficiency. In next sections, will discuss about the configuration process of an algorithm deployment on a serverless architecture setup. The key aspects of this research are handling concurrency, utilizing provisioned concurrency and addressing cold starts. Moreover, practical implementation monitoring and evaluation is mentioned. This document ensures that the concept of this research should be clear to implement for any future researchers.

2 Setting up Google Colab

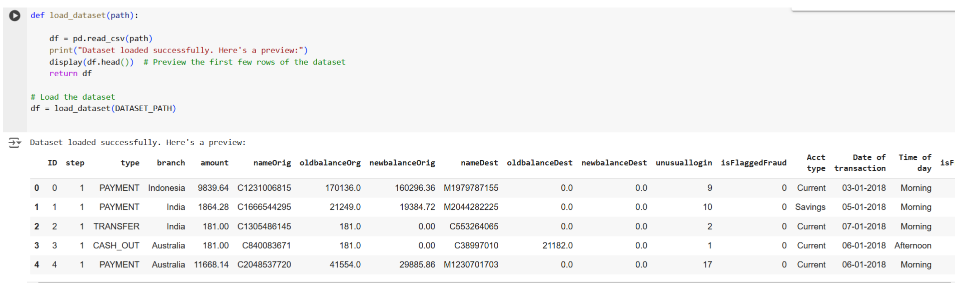
Data Preprocessing and Feature Engineering were performed on Google Colab.

1. Access Google Colab: Start a google colab in browser. To create a new notebook click on, File >New Notebook.
2. Upload Raw dataset: Use the following 1 code to upload the dataset and preview the data in fig 2.
3. Now in next step, drop the irrelevant Columns, as shown in fig 5.
4. Handled Missing values: This keep the dataset ready for further analysis shown in fig 3 and 4.



```
[ ] # Import necessary libraries
    import numpy as np
    import pandas as pd
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import recall_score
    from sklearn.preprocessing import StandardScaler
    import seaborn as sns
    import matplotlib.pyplot as plt
    from google.colab import files


    DATASET_PATH = "/content/Synthetic Fraud Detection Dataset.csv"
```

Figure 1: Upload the Raw dataset

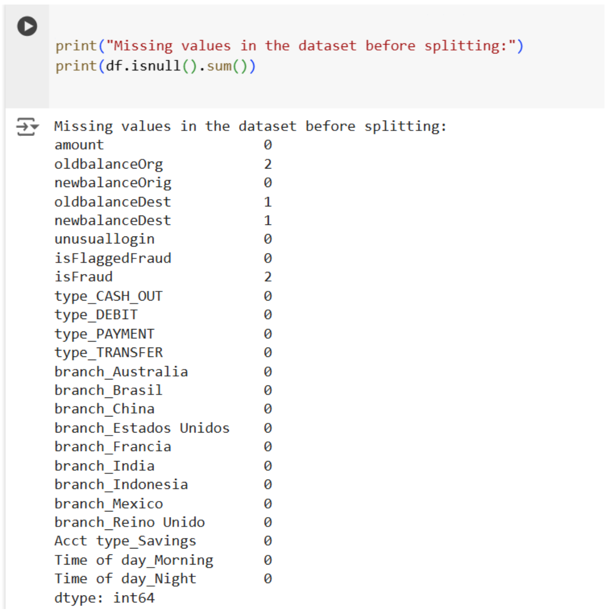


```python
def load_dataset(path):

    df = pd.read_csv(path)
    print("Dataset loaded successfully. Here's a preview:")
    display(df.head())  # Preview the first few rows of the dataset
    return df

# Load the dataset
df = load_dataset(DATASET_PATH)
```

```
Dataset loaded successfully. Here's a preview:
```

	ID	step	type	branch	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	unusuallogin	isFlaggedFraud	Acct type	Date of transaction	Time of day	isF
0	0	1	PAYMENT	Indonesia	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	9	0	Current	03-01-2018	Morning	
1	1	1	PAYMENT	India	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	10	0	Savings	05-01-2018	Morning	
2	2	1	TRANSFER	India	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	2	0	Current	07-01-2018	Morning	
3	3	1	CASH_OUT	Australia	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0	Current	06-01-2018	Afternoon	
4	4	1	PAYMENT	Australia	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	17	0	Current	06-01-2018	Morning	

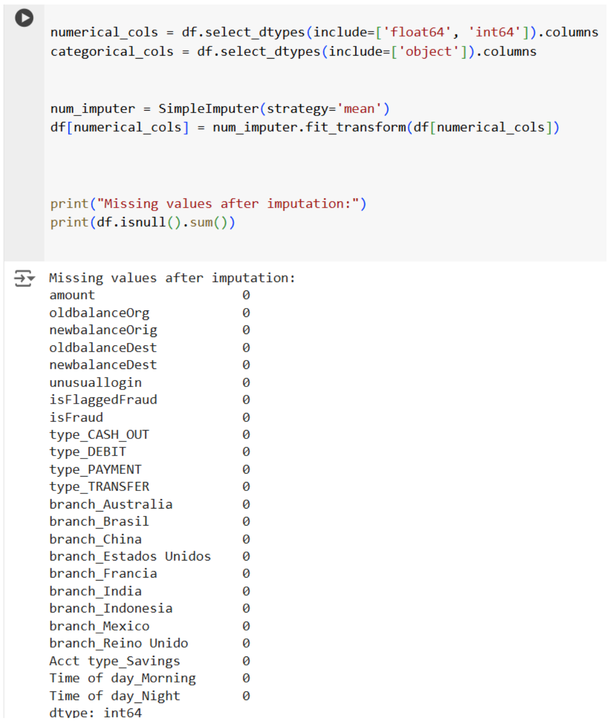
Figure 2: Preview data



```python
print("Missing values in the dataset before splitting:")
print(df.isnull().sum())
```

```
Missing values in the dataset before splitting:
amount                    0
oldbalanceOrg             2
newbalanceOrig            0
oldbalanceDest            1
newbalanceDest            1
unusuallogin              0
isFlaggedFraud            0
isFraud                   2
type_CASH_OUT             0
type_DEBIT                0
type_PAYMENT              0
type_TRANSFER             0
branch_Australia          0
branch_Brasil             0
branch_China              0
branch_Estados Unidos     0
branch_Francia            0
branch_India              0
branch_Indonesia          0
branch_Mexico             0
branch_Reino Unido        0
Acct type_Savings         0
Time of day_Morning       0
Time of day_Night         0
dtype: int64
```

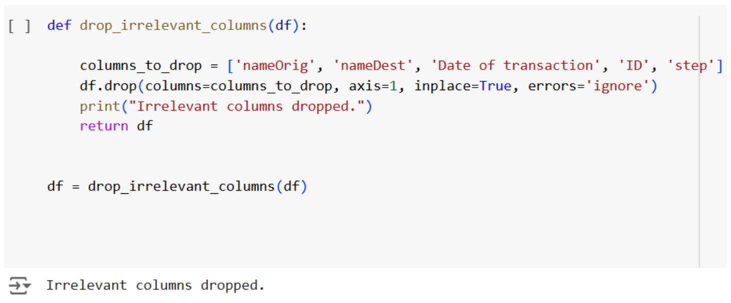
Figure 3: Missing values



```python
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns


num_imputer = SimpleImputer(strategy='mean')
df[numerical_cols] = num_imputer.fit_transform(df[numerical_cols])



print("Missing values after imputation:")
print(df.isnull().sum())
```

```
Missing values after imputation:
amount                    0
oldbalanceOrg             0
newbalanceOrig            0
oldbalanceDest            0
newbalanceDest            0
unusuallogin              0
isFlaggedFraud            0
isFraud                   0
type_CASH_OUT             0
type_DEBIT                0
type_PAYMENT              0
type_TRANSFER             0
branch_Australia          0
branch_Brasil             0
branch_China              0
branch_Estados Unidos     0
branch_Francia            0
branch_India              0
branch_Indonesia          0
branch_Mexico             0
branch_Reino Unido        0
Acct type_Savings         0
Time of day_Morning       0
Time of day_Night         0
dtype: int64
```

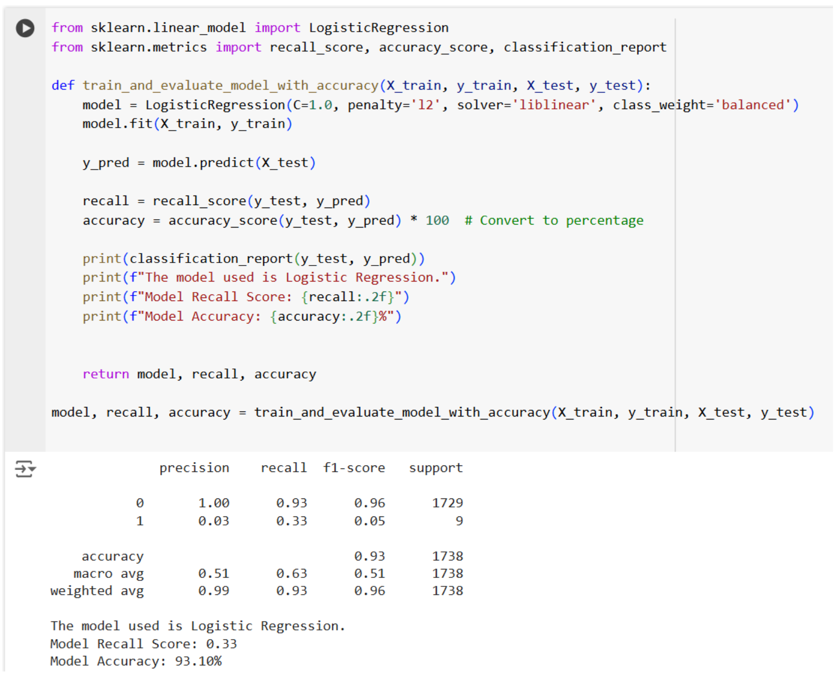
Figure 4: After handling missing values



```
[ ] def drop_irrelevant_columns(df):

        columns_to_drop = ['nameOrig', 'nameDest', 'Date of transaction', 'ID', 'step']
        df.drop(columns=columns_to_drop, axis=1, inplace=True, errors='ignore')
        print("Irrelevant columns dropped.")
        return df


    df = drop_irrelevant_columns(df)
```

```
Irrelevant columns dropped.
```

Figure 5: Drop irrelevant columns



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import recall_score, accuracy_score, classification_report

def train_and_evaluate_model_with_accuracy(X_train, y_train, X_test, y_test):
    model = LogisticRegression(C=1.0, penalty='l2', solver='liblinear', class_weight='balanced')
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    recall = recall_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred) * 100  # Convert to percentage

    print(classification_report(y_test, y_pred))
    print(f"The model used is Logistic Regression.")
    print(f"Model Recall Score: {recall:.2f}")
    print(f"Model Accuracy: {accuracy:.2f}%")


    return model, recall, accuracy

model, recall, accuracy = train_and_evaluate_model_with_accuracy(X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96      1729
           1       0.03      0.33      0.05         9

    accuracy                           0.93      1738
   macro avg       0.51      0.63      0.51      1738
weighted avg       0.99      0.93      0.96      1738

The model used is Logistic Regression.
Model Recall Score: 0.33
Model Accuracy: 93.10%
```

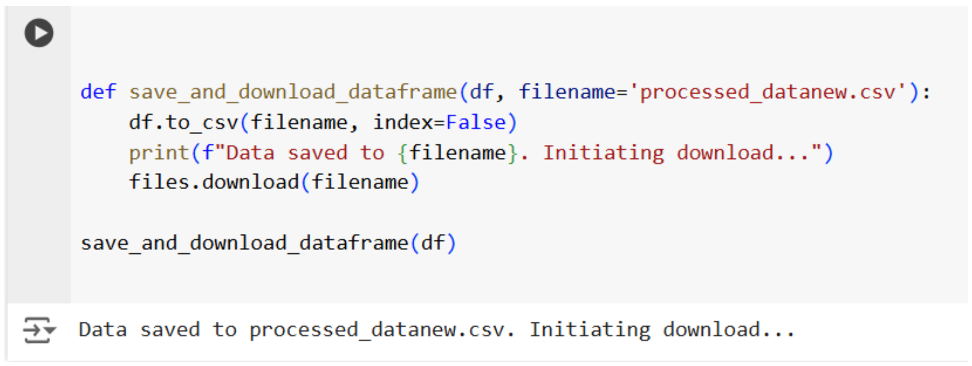
Figure 6: Logistic Regression

5. The logistic Regression model was used to set a starting point and access the data-set's ability to make predictions 6.

6. Next step will be,save and download the preprocessed data as a CSV file format as shown in fig 7.

   At this step [1], data is preprocessed with train model and ready for further analysis.

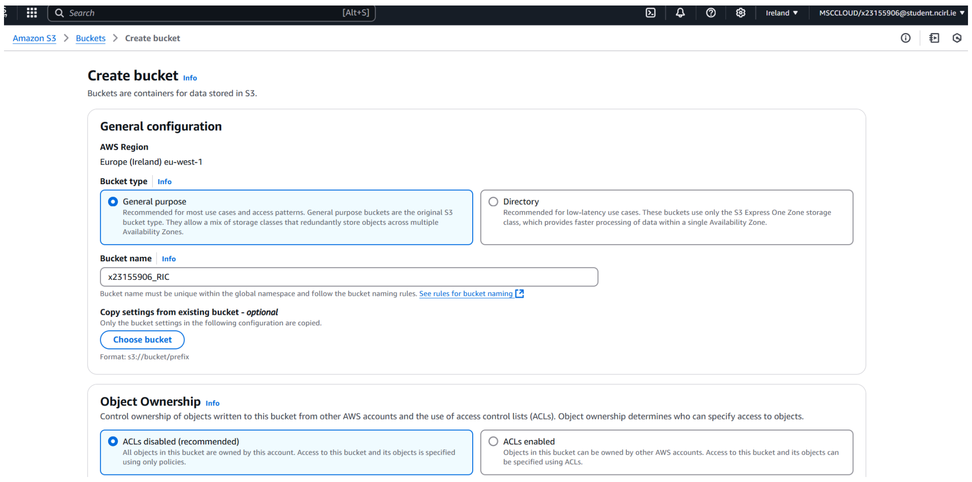
[1] https://colab.google/



```
def save_and_download_dataframe(df, filename='processed_datanew.csv'):
    df.to_csv(filename, index=False)
    print(f"Data saved to {filename}. Initiating download...")
    files.download(filename)

save_and_download_dataframe(df)
```

```
Data saved to processed_datanew.csv. Initiating download...
```

Figure 7: Download the preprocessed dataset



Search [Alt+S]
Ireland
MSCCLOUD/x23155906@student.ncirl.ie
Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

Europe (Ireland) eu-west-1

Bucket type Info

[✓ General purpose] Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory: Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info

x23155906_RIC

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

[✓ ACLs disabled (recommended)] All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

ACLs enabled: Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Figure 8: S3 Bucket creation

3 Configure AWS services

3.1 AWS S3 :

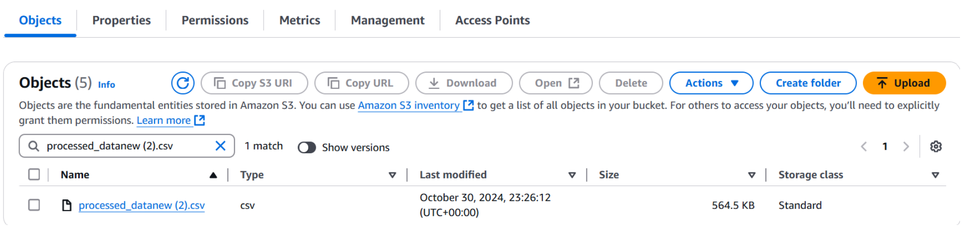
Amazon S3 was used to store and manage the preprocessed dataset [2].

- Create a Bucket: Open the AWS Management Console, navigate it on S3 and click on create new bucket as shown in fig 8. Type the bucket name, select the region and click on the create button.
- Keep the bucket versioning enabled.

Upload the dataset :

- Go to the created bucket and click on upload.
- Upload the CSV file which has been preprocessed 9.

[2]https://aws.amazon.com/s3/



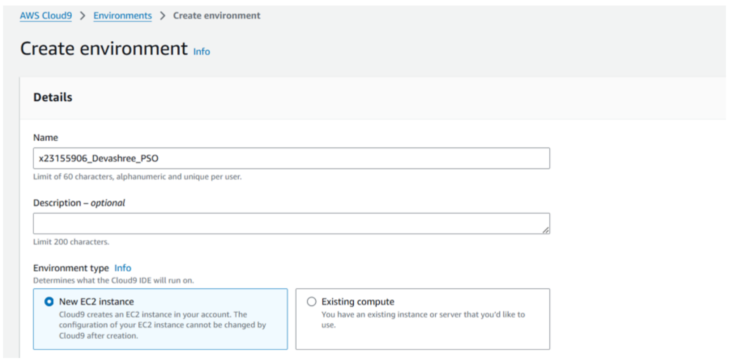
Objects (5) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

processed_datanew (2).csv — 1 match — Show versions

Name	Type	Last modified	Size	Storage class
processed_datanew (2).csv	csv	October 30, 2024, 23:26:12 (UTC+00:00)	564.5 KB	Standard

Figure 9: Upload the refined dataset in AWS S3



Name: x23155906_Devashree_PSO

Limit of 60 characters, alphanumeric and unique per user.

Description – *optional*

Limit 200 characters.

Environment type Info

Determines what the Cloud9 IDE will run on.

[✓ New EC2 instance] Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

Existing compute: You have an existing instance or server that you'd like to use.

Figure 10: Create a Cloud9 Environment

3.2 Setting up AWS Cloud9 for development

In this research, AWS Cloud9 and Amazon EC2 were development environment for writing, testing and preparing python scripts.This provided a simple way to code, debug and package scripts for development on AWS Lambda .

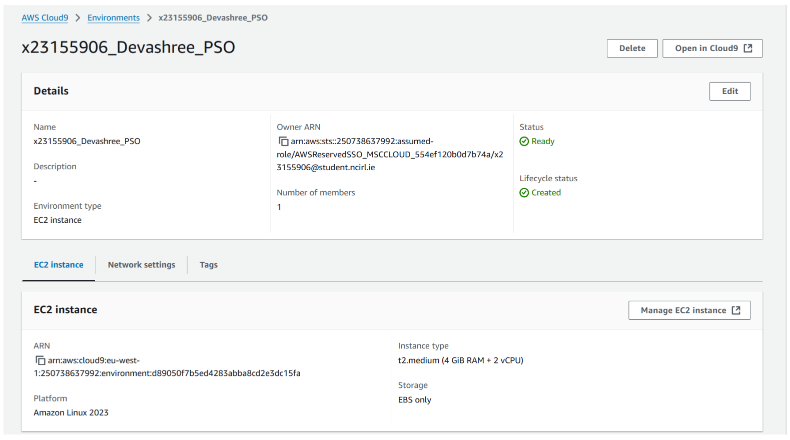
Step to create Cloud9 Environment:

- In the AWS Console, click on Cloud9 and then create an environment 10. Please find the below snap of it.
- After setting up cloud9, in fig is the details of the environment 11.
- Install Python Libraries: Below are the commands to run on cloud9 environment. It will install the python libraries which are neccessary to run the Lambda function.

```
1) python3 -m venv x23155906_RIC
2) source x23155906_RIC/bin/activate
3) cd x23155906_RIC/
4) mkdir lambda_package
5) pip install boto3 pandas numpy
6) vi lambda_function.py
7) zip -r ../function.zip .
```

Required libraries has been installed[3].

[3]https://aws.amazon.com/cloud9/


AWS Cloud9 > Environments > x23155906_Devashree_PSO

x23155906_Devashree_PSO

Details

Name
x23155906_Devashree_PSO

Description
-

Environment type
EC2 instance

Owner ARN
arn:aws:sts::250738637992:assumed-role/AWSReservedSSO_MSCCLOUD_554ef120b0d7b74a/x23155906@student.ncirl.ie

Number of members
1

Status
Ready

Lifecycle status
Created

EC2 instance

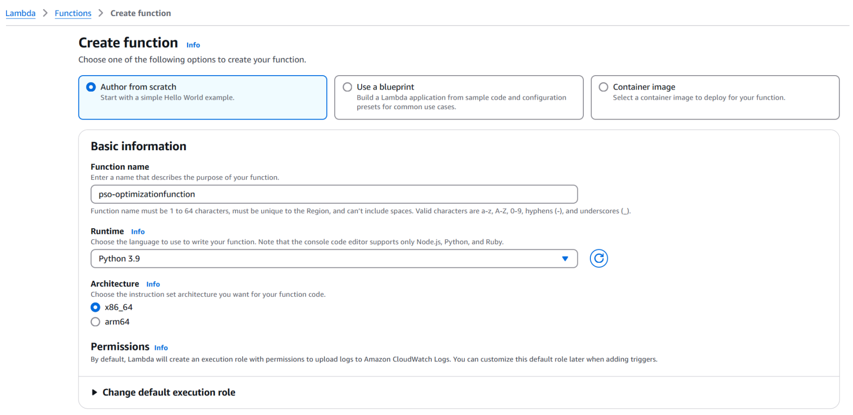
ARN
arn:aws:cloud9:eu-west-1:250738637992:environment:d89050f7b5ed4283abba8cd2e3dc15fa

Platform
Amazon Linux 2023

Instance type
t2.medium (4 GiB RAM + 2 vCPU)

Storage
EBS only

Figure 11: Details of cloud9 creation


Lambda > Functions > Create function

Create function Info

Choose one of the following options to create your function.

[✓ Author from scratch] Start with a simple Hello World example.

Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases.

Container image — Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
pso-optimizationfunction
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64] arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Figure 12: Create a lambda function

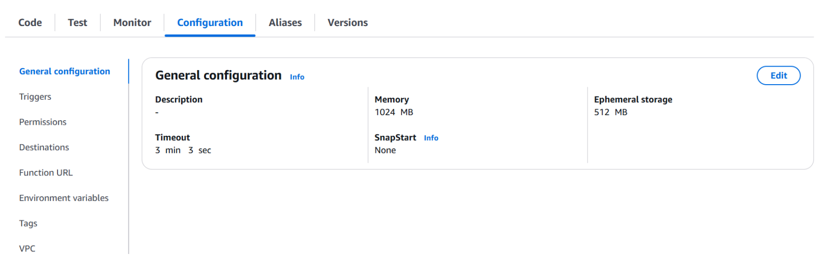
3.3 AWS Lambda

AWS Lambda has been used in an efficient way to execute the algorithms. In this research its scalability and pay as you go system made an precise platform to test. Created two lambda functions each of which is used for testing and performance comparisons. Below are the steps followed for deploying both algorithms [4].

Create Lambda Functions:

- Navigate to Lambda >Create function in the AWS console 12.
- Select *Author from Scratch* then select a runtime as *Python 3.9* as shown in fig.
- You need to set the appropriate execution role to have the access to AWS S3. Finally click on create function.
- Then upload a .zip file of your code and packages which u fetched it from cloud9 environment.
  Resource Allocation
- select the 'Configuration' tab to set the memory allocation and in that there will be 'General Configuration' setting.

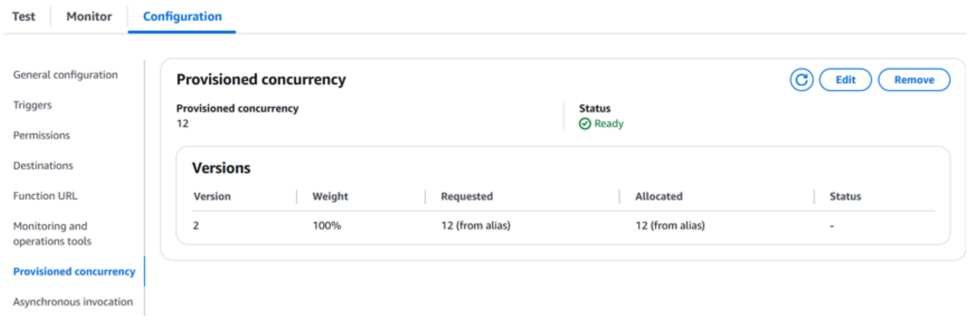
[4] `https://aws.amazon.com/lambda/`



General configuration Info

Description	Memory	Ephemeral storage
-	1024 MB	512 MB
Timeout	SnapStart Info	
3 min 3 sec	None	

Figure 13: Resource allocation



Provisioned concurrency

Provisioned concurrency: 12

Status: Ready

Versions

Version	Weight	Requested	Allocated	Status
2	100%	12 (from alias)	12 (from alias)	-

Figure 14: Set Provisioned Concurrency

- For example, as per below snap 13, it is allocated to 1024MB memory size. In this research, did testing on different memory configurations.
  Enable Provisioned Concurrency :
- After doing cold start performing, next step is to set the provisioned concurrency for pre-warming.Created an alias for provisioned concurrency, to manage the function and integrate the function url as shown in fig 14 .
- Set the right instances, in our research we set as 10 and add this has been configured in alias setting.
  Test the Lambda Function
- Now, for testing the lambda function.Select the *Test* tab in AWS Lambda page, then 'Create a new event' and give the test-name which you preferred. Here we gave *pso-test*.
- Next, in 'Event Json' section input below parameter which describes about the bucket_name and file_key.

```
{
  "BUCKET_NAME": "x23155906-devashree-ric",
  "FILE_KEY": "processed_data_new (2).csv"
}
```

  Save and click it on 'Test' to run the function.

3.4 Postman for testing

- Install the postman through browser [5].
- Open the postman and create a new Post request.

[5]https://www.postman.com/

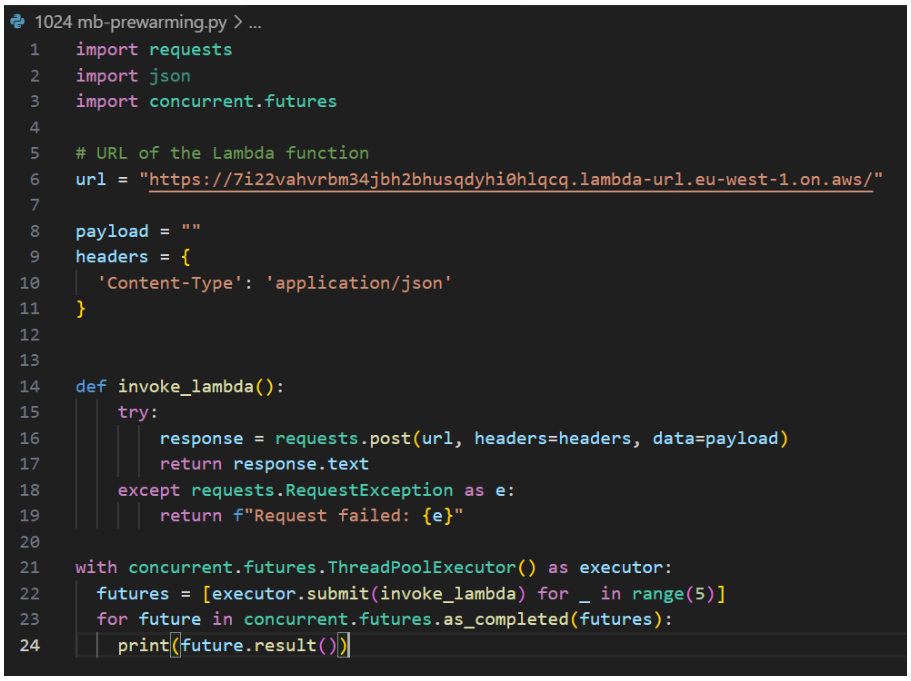

```python
1024 mb-prewarming.py > ...
import requests
import json
import concurrent.futures

# URL of the Lambda function
url = "https://7i22vahvrbm34jbh2bhusqdyhi0hlqcq.lambda-url.eu-west-1.on.aws/"

payload = ""
headers = {
  'Content-Type': 'application/json'
}


def invoke_lambda():
    try:
        response = requests.post(url, headers=headers, data=payload)
        return response.text
    except requests.RequestException as e:
        return f"Request failed: {e}"

with concurrent.futures.ThreadPoolExecutor() as executor:
  futures = [executor.submit(invoke_lambda) for _ in range(5)]
  for future in concurrent.futures.as_completed(futures):
    print(future.result())
```

Figure 15: Python Script on Vs code

- Add the Lambda function url in it to request the function for post testing.
- After that click on new send button it will give a status code of *200 ok*, which means the execution was done successfully.
- The purpose of using Postman was to check the API Post testing of a lambda function in a way for robust verification of lambda functionality.

3.5 Setting up Visual Studio Code

Visual Studio Code was used to automate high concurrency testing of lambda functions.

- Install python extensions with version 3.12.8.
- Write the Script for sending multiple requests to lambda function at once,using single python script.
- Put your lambda function url and change the number of range, for instance in our research we set it as 5 as shown in fig 15.
- Run the code on VS terminal as mentioned in below command:
  python 1024 mb-prewarming.py

3.6 Monitoring in AWS CloudWatch

AWS CloudWatch was used to monitor the lambda performance [6].

- Review Metric : Go to the AWS Lambda and choose the 'Monitoring' tab, here you will see the graph of detailed performance metrics such as: duration, error rates, invocations and so on as shown in fig 16. This will help to figure out efficiency and reliability of lambda function.

[6]https://aws.amazon.com/cloudwatch/

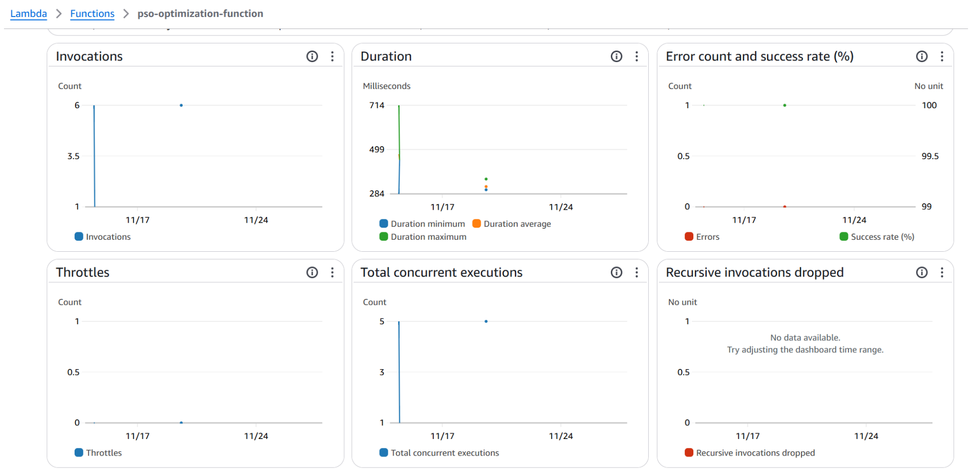
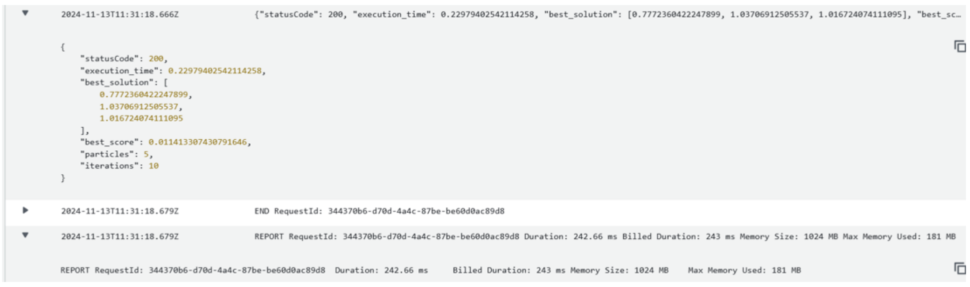


Figure 16: Review Metrics



```
▼  2024-11-13T11:31:18.666Z      {"statusCode": 200, "execution_time": 0.22979402542114258, "best_solution": [0.7772360422247899, 1.03706912505537, 1.016724074111095], "best_sc…

   {
       "statusCode": 200,
       "execution_time": 0.22979402542114258,
       "best_solution": [
           0.7772360422247899,
           1.03706912505537,
           1.016724074111095
       ],
       "best_score": 0.011413307430791646,
       "particles": 5,
       "iterations": 10
   }

▶  2024-11-13T11:31:18.679Z      END RequestId: 344370b6-d70d-4a4c-87be-be60d0ac89d8
▼  2024-11-13T11:31:18.679Z      REPORT RequestId: 344370b6-d70d-4a4c-87be-be60d0ac89d8 Duration: 242.66 ms Billed Duration: 243 ms Memory Size: 1024 MB Max Memory Used: 181 MB

   REPORT RequestId: 344370b6-d70d-4a4c-87be-be60d0ac89d8  Duration: 242.66 ms    Billed Duration: 243 ms Memory Size: 1024 MB    Max Memory Used: 181 MB
```

Figure 17: CloudWatch Logs

- Access logs : You can see the log files below CloudWatch metrics, in that you will get to know the detailed output of the code as well the performance metrics 17.