

Uploading IoT Sensor Data to DynamoDB using Python and Design Pattern

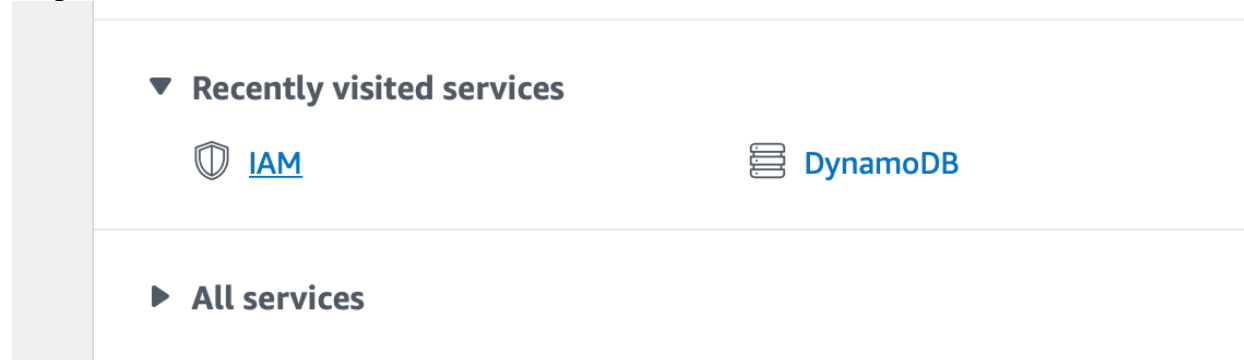


Amazon DynamoDB

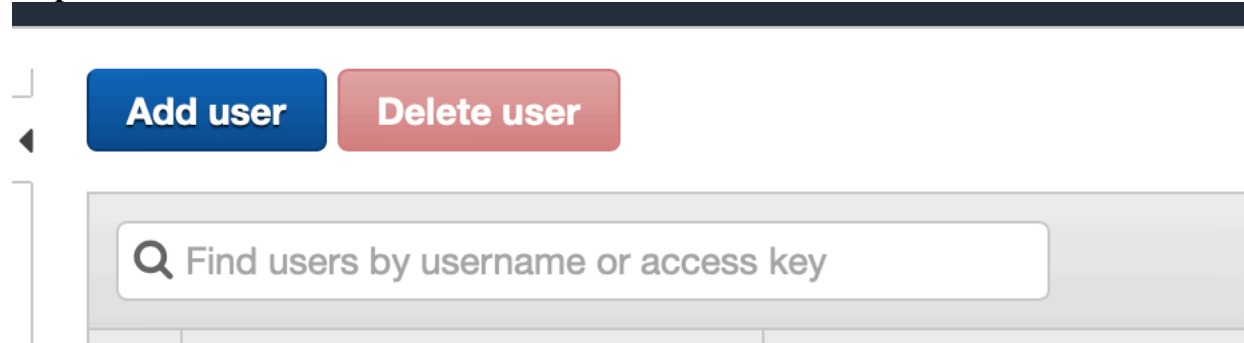
Aim: Upload IoT Sensor Data to AWS DynamoDB

Objective: In this Lab we will learn how to upload sensor data to DynamoDB using Python Boto3 and Facade Design Pattern

Step 1: Create a Free Tier Account on AWS and. Create a User



Step 2: Create a New User



Step 3: Add username as Python and make sure access type is as Programmatic

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type* ☒ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

Step 4: Create a Policy Make sure to Give Admin Access

Set permissions

Add user to group

Copy permissions from existing user

Attach existing policies directly

Create policy

Filter policies

Search

Showing 469 results

	Policy name	Type	Used as	Description
<input type="checkbox"/>	AdministratorAccess	Job function	Permissions policy (2)	Provides full access to AWS services and re...
<input type="checkbox"/>	AlexaForBusinessD...	AWS managed	None	Provide device setup access to AlexaForBu...
<input type="checkbox"/>	AlexaForBusinessF...	AWS managed	None	Grants full access to AlexaForBusiness reso...
<input type="checkbox"/>	AlexaForBusinessG...	AWS managed	None	Provide gateway execution access to Alexa...
<input type="checkbox"/>	AlexaForBusinessR...	AWS managed	None	Provide read only access to AlexaForBusine...
<input type="checkbox"/>	AmazonAPIGatewa...	AWS managed	None	Provides full access to create/edit/delete A...
<input type="checkbox"/>	AmazonAPIGatewa...	AWS managed	None	Provides full access to invoke APIs in Amaz...
<input type="checkbox"/>	AmazonAPIGatewa...	AWS managed	None	Allows API Gateway to push logs to user's ...

Click on Administrative Access

	Policy name	Type
<input checked="" type="checkbox"/>	AdministratorAccess	Job function
<input type="checkbox"/>	AmazonAPIGatewa...	AWS managed

Step 5: Create the user

Cancel

Previous

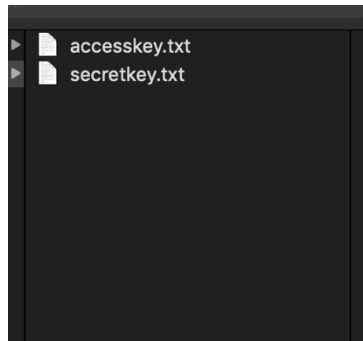
Create user

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy

Step 7: Copy the Secret Key and Access Key

User	Access key ID	Secret access key
python	AKIAYQ5FSEDYFFM7KA72	***** Show

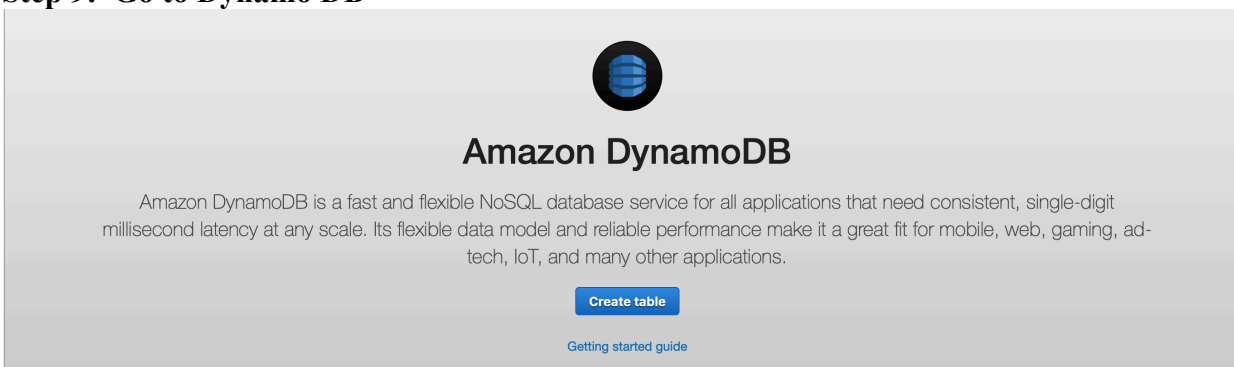
Step 7: Create a folder and store the access key and secret key in that folder



Step 8: Configure AWS CLI if you haven't installed pip install aws-cli or go the the documentation to download AWS CLI once downloaded we need to configure type the following command on mac or command Shell aws configure and paste your credentials

```
KEYS — -bash — 80x24
(base) Soumils-MacBook-Air-8:KEYS soumilshah$ aws configure
AWS Access Key ID [*****KA72]:
AWS Secret Access Key [*****1UY1]:
Default region name [us-east-1]:
Default output format [json]:
(base) Soumils-MacBook-Air-8:KEYS soumilshah$
```

Step 9: Go to Dynamo DB



Step 10: Create a table with following name

Create DynamoDB table

Tutorial

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

String ⓘ

☒ Add sort key

String ⓘ

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☒ Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".
- Encryption at Rest with DEFAULT encryption type.

ⓘ You do not have the required role to enable Auto Scaling by default.
Please refer to [documentation](#).

+ Add tags **NEW!**

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel **Create**

Step 11: Click on create button to create a table

Create table Delete table

Filter by table name

Choose a table ... Actions

Name

DHT

DHT Close

Overview Items Metrics Alarms Capacity Indexes Global Tables Backups Triggers Access control Tags

Table is being created

Recent alerts

No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled No

View type -

Latest stream ARN -

Manage Stream

Table details

Table name DHT

Primary partition key date (String)

Primary sort key time (String)

Point-in-time recovery -

Encryption Type DEFAULT

KMS Master Key ARN Not Applicable

Step 12: Add the Entry Manually to see if everything is working properly

Create item

Tree ▾

Item {3}

- date String : 10-9-2019
- time String : 21:31:43
- Temperature String : 22

Append ▾

- String
- Binary
- Number
- StringSet
- NumberSet
- BinarySet
- Map
- List
- Boolean
- ...

Field DynamoDB type "String". This is a String with quotes around it

Step 13:

Scan ▾ [Table] DHT: date, time ▾ ^

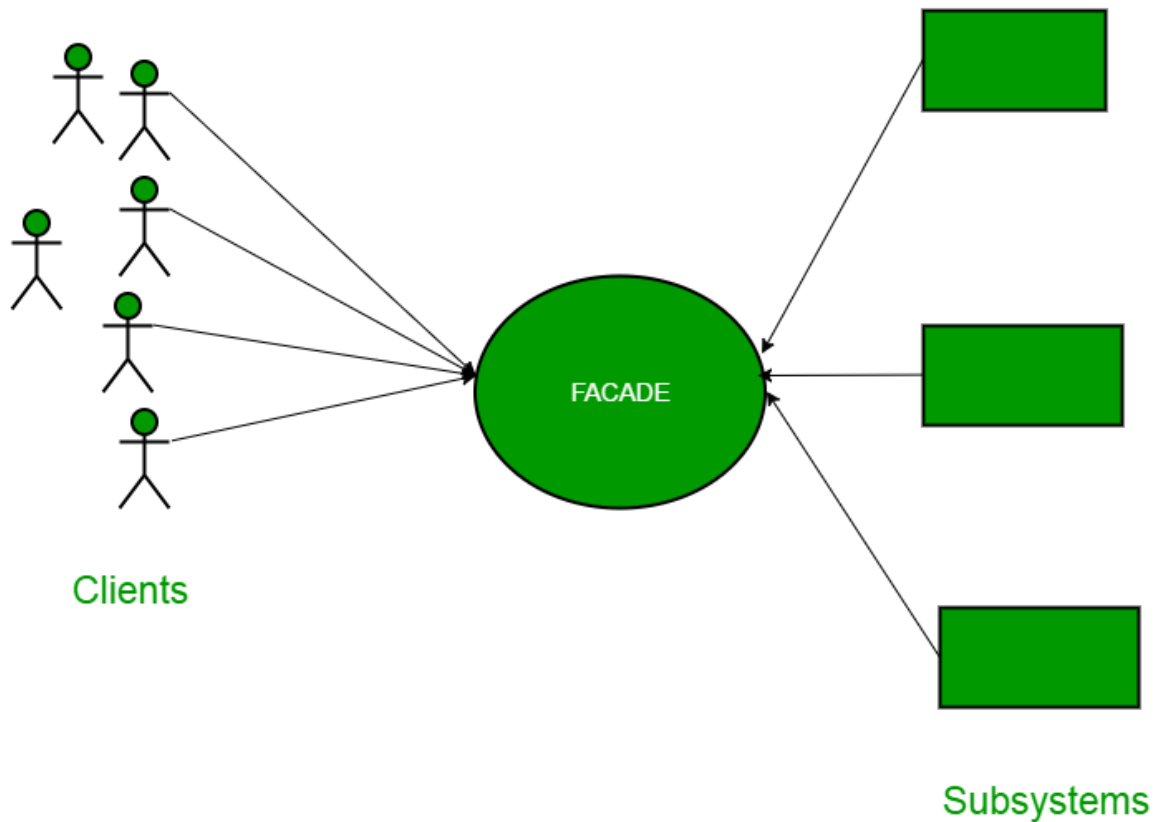
+ Add filter

Start search

	date ⓘ	time	Humidity	Temperature
	10-9-2019	21:31:43	44	22

Congrats you have just added data on AWS Dynamo DB

Let's us use Façade and Singleton Design Pattern in Python to Upload the Data on Dynamo DB



Code :

```
"""
We will use Fascade Design Pattern and Singleton Design Pattern to Upload the Sensor Data
"""
try:
    import os
    import sys
    import boto3
    import datetime
    import random
except Exception as e:
    print("Some Modules are Missings ")

class Metaclass(type):
    """ Implementing Singleton Design Pattern """
    _instance = {}

    def __call__(cls, *args, **kwargs):
```

```

        if cls not in cls._instance:
            cls._instance[cls] = super(Metaclass, cls).__call__(*args, **kwargs)
            return cls._instance[cls]

class Sensor(object):

    def __init__(self):
        """ Constructor """
        pass

    def get(self):

        """ Get the Sensor Data """

        Temperature = random.randint(0,70)
        Humidity = random.randint(0,70)
        d = datetime.datetime.now()
        _Date = "{}-{}-{}".format(d.month, d.day, d.year)
        _Time = "{}: {}: {}".format(d.hour, d.minute, d.second)
        return Temperature, Humidity, _Date, _Time

class DynamoDb(object):

    __slots__ = ["Table_Name", "db", "table", "client"]

    def __init__(self, Table_Name='DHT'):
        self.Table_Name=Table_Name
        self.db = boto3.resource('dynamodb')
        self.table = self.db.Table(Table_Name)
        self.client = boto3.client('dynamodb')

    def put(self, date='', time='', Temperature='', Humidity=''):
        self.table.put_item(
            Item={
                'date':date,
                'time':time,
                'Temperature':Temperature,
                'Humidity' :Humidity
            }
        )

class Facade(metaclass=Metaclass):

    def __init__(self):
        self._sensor = Sensor()
        self._database = DynamoDb()

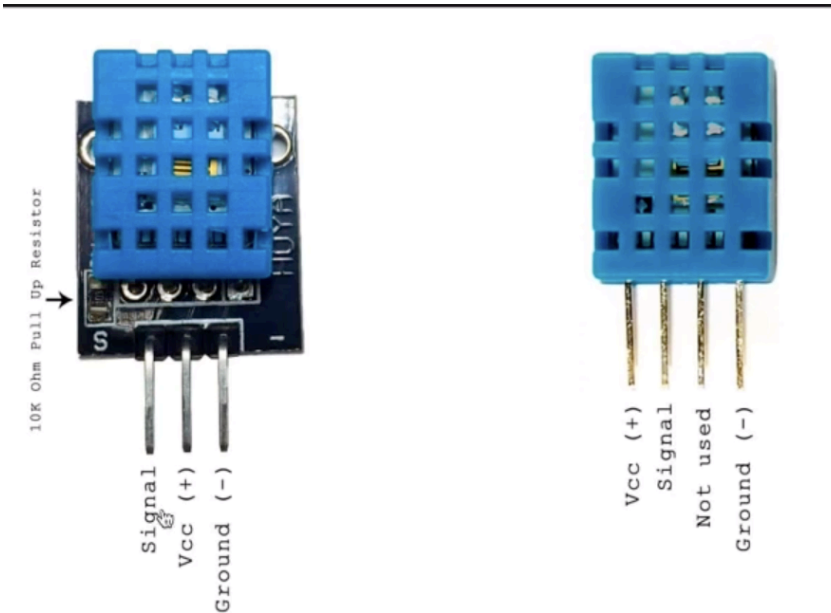
    def upload(self):
        Temperature, Humidity, _Date, _Time = self._sensor.get()
        self._database.put(_Date, _Time, str(Temperature), str(Humidity))
        data = {
            "Date":_Date,
            "Time":_Time,
            "Temperature":Temperature,
            "_humidity":Humidity
        }
        print(data)

def main():
    facade = Facade()
    facade.upload()

if __name__ == "__main__":
    main()

```


Connecting Actual Sensor and Logging Value to Dynamo DB



Shows the Pinout of DHT-11 Sensor

Connect the Data Pin to GPIO 23 on raspberry pi and VCC and GND to respective pins. Once hooked up configure the AWS CLI on Raspberry pi and run the Following Python Code which will upload the sensor data to DynamoDB

```
"""
We will use Facade Design Pattern and Singleton Design Pattern to Upload the Sensor Data
"""
try:
    import os
    import sys
    import boto3
    import datetime
    import random
    import Adafruit_DHT
except Exception as e:
```

```

print("Some Modules are Missings ")

class Metaclass(type):

    """ Implementing Singleton Design Pattern """

    _instance = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instance:
            cls._instance[cls] = super(Metaclass, cls).__call__(*args, **kwargs)
            return cls._instance[cls]

class Sensor(object):

    def __init__(self):
        """ Constructor """
        pass

    def get(self):

        """ Get the Sensor Data """
        pin = 23
        sensor = Adafruit_DHT.DHT11
        Humidity, Temperature = Adafruit_DHT.read_retry(sensor, pin)

        # Temperature = random.randint(0,70)
        # Humidity = random.randint(0,70)

        d = datetime.datetime.now()
        _Date = "{}-{}-{}".format(d.month, d.day, d.year)
        _Time = "{}: {}: {}".format(d.hour, d.minute, d.second)

        return Temperature, Humidity, _Date, _Time

class DynamoDb(object):

    __slots__ = ["Table_Name", "db", "table", "client"]

    def __init__(self, Table_Name='DHT'):
        self.Table_Name=Table_Name
        self.db = boto3.resource('dynamodb')
        self.table = self.db.Table(Table_Name)
        self.client = boto3.client('dynamodb')

    def put(self, date='', time='', Temperature='', Humidity=''):
        self.table.put_item(
            Item={
                'date':date,
                'time':time,
                'Temperature':Temperature,
                'Humidity' :Humidity
            }
        )

class Facade(metaclass=Metaclass):

    def __init__(self):
        self._sensor = Sensor()
        self._database = DynamoDb()

    def upload(self):
        Temperature, Humidity, _Date, _Time = self._sensor.get()
        self._database.put(_Date, _Time, str(Temperature), str(Humidity))
        data = {

```

```
        "Date":_Date,  
        "Time":_Time,  
        "Temperature":Temperature,  
        "_humidity":Humidity  
    }  
    print(data)  
  
def main():  
    facade = Facade()  
    facade.upload()  
  
if __name__ == "__main__":  
    main()
```