

**ANNA UNIVERSITY REGIONAL CAMPUS**

**COIMBATORE-641046**

**NAAN MUDHALVAN COURSE**

**PHASE 5**

|  |  |
| --- | --- |
| NAME | SUBASH S |
| NAAN MUDHALVAN REGISTER NO. | au710021106035 |
| DOMAIN | CLOUD COMPUTING |
| PROJECT | SERVERLESS IOT DATA PROCESSING |

**INTRODUCTION**

This documentation provides a comprehensive guide to creating a serverless data processing system using IoT with IBM Cloud. The system's primary goal is to efficiently receive, process, and store IoT data, making it suitable for various IoT applications.

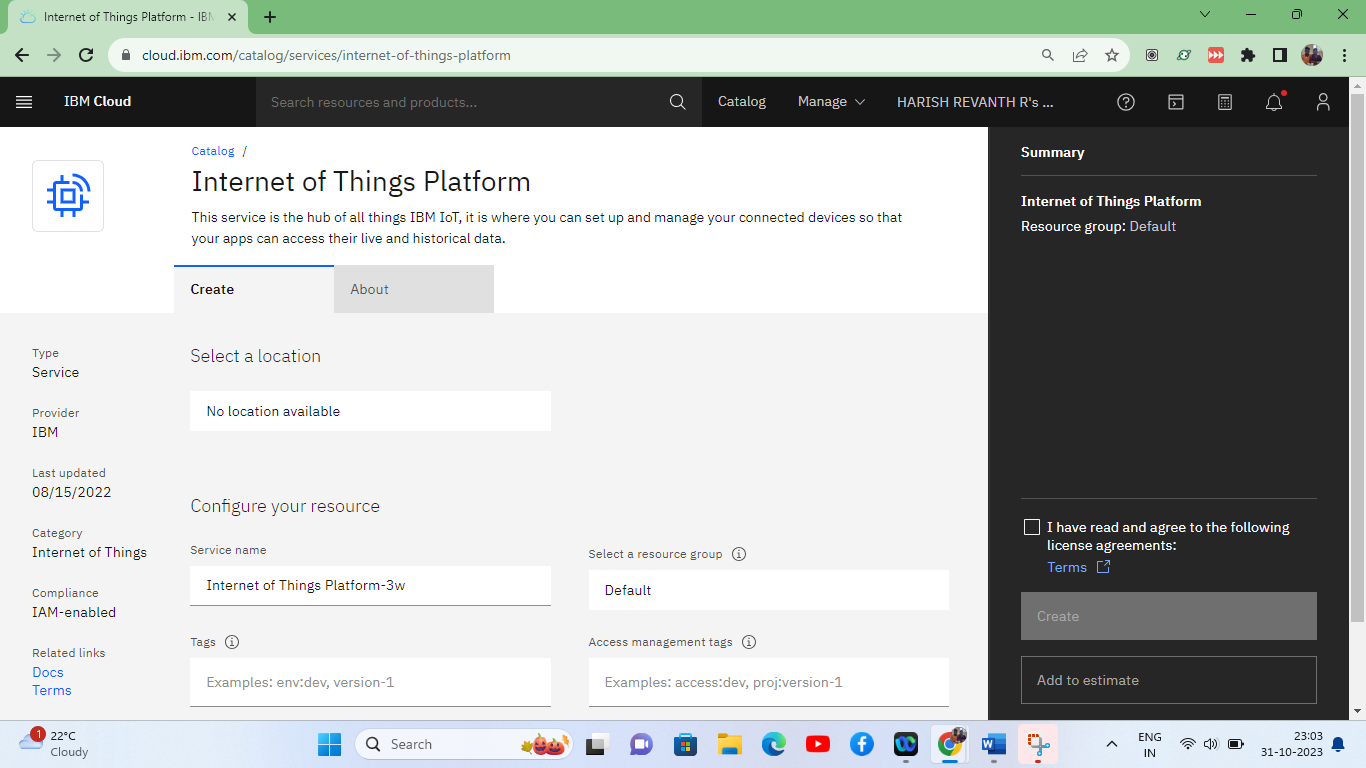
**PROJECT OVERVIEW**

This project consists of several key steps:

1. Set up IBM Cloud services to create the foundation for your IoT data processing system.
2. Create a serverless function that will receive, process, and respond to incoming IoT data.
3. Connect an IoT platform or device simulator to send data to your serverless function.
4. Set up an object storage service to store processed data.
5. Test the end-to-end workflow to ensure that data is received, processed, and stored correctly.

**IBM CLOUD FUNCTIONS SETUP**

In this section, the steps for setting up IBM Cloud Functions, which provide the necessary computational power for data processing, are outlined. IBM Cloud was accessed through a web browser, and if an IBM Cloud account wasn't available, the option to sign up for one was utilized. Subsequently, an IBM Cloud Functions instance was created by navigating to the "Catalog" section in the IBM Cloud dashboard. The chosen plan, often the free "Lite" plan, was selected to meet specific needs. The prompts provided were then followed to create the IBM Cloud Functions instance. Once the instance was created, access to the IBM Cloud Functions dashboard was enabled to manage serverless functions.



**IBM CLOUD OBJECT STORAGE SETUP**

This section covers the creation of an instance of IBM Cloud Object Storage, ensuring data durability and availability for storing processed IoT data. The IBM Cloud dashboard was initially accessed through a web browser. Following this, an IBM Cloud Object Storage instance was created by selecting "Object Storage" from the catalog. The choice of the appropriate plan, such as the "Lite" plan, which is free of charge, was made. On-screen instructions were then followed to create the IBM Cloud Object Storage instance. Subsequently, access to the IBM Cloud Object Storage dashboard was gained, enabling the management of data storage and related configurations.

**IOT PLATFORM OR DEVICE SIMULATOR CONFIGURATION**

This section focuses on the configuration of an IoT platform or device simulator to send temperature data to the serverless function. To begin, access to the IoT platform, such as the IBM Watson IoT Platform or a chosen platform, was enabled through a web browser. Access involved logging in with IBM Cloud credentials or platform-specific login details. Within the IoT platform, an IoT device was created or a simulator was set up. The device or simulator was configured to send temperature data to the serverless function endpoint established in IBM Cloud Functions. Finally, the data transmission was validated by instructing the IoT device or simulator to send temperature data to the serverless function.

**CREATING A SERVERLESS FUNCTION**

**Develop a Serverless Function**: Create a serverless function that receives IoT data and processes it based on your requirements. This function should be designed to work with the data sent by the IoT platform.

**PROGRAM:**

import json

def process\_iot\_data(iot\_data):

try:

# Parse the incoming JSON data

data = json.loads(iot\_data)

# Check if the required data fields are present

if "temperature" in data:

temperature = data["temperature"]

# Your data processing logic here

if temperature > 30:

message = "Temperature is too high!"

else:

message = "Temperature is within the acceptable range."

return {"result": message}

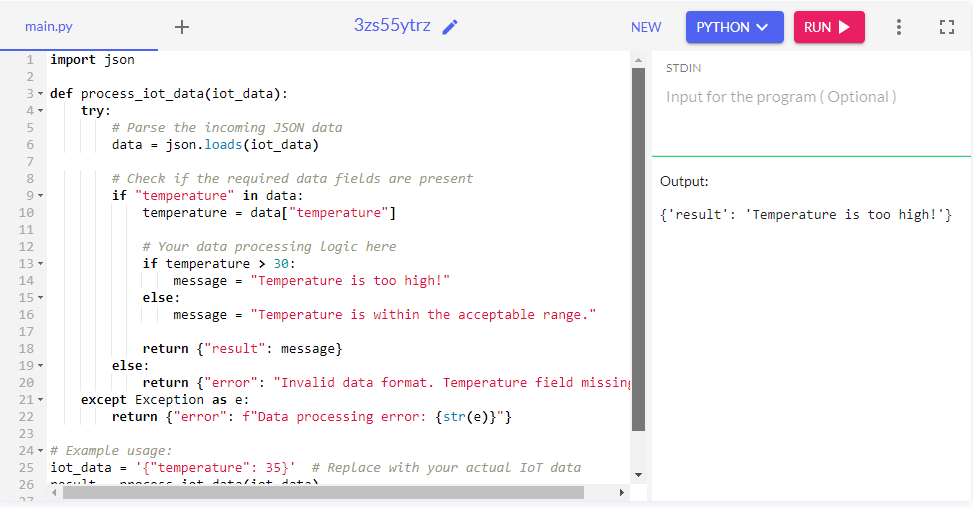
else:

return {"error": "Invalid data format. Temperature field missing."}

except Exception as e:

return {"error": f"Data processing error: {str(e)}"}

**OUTPUT:**



**CONNECTING IOT PLATFORM OR DEVICE SIMULATOR**

**Configure IoT Platform**:

Set up the IoT platform or device simulator to trigger your serverless function when new data is available. Ensure that it can communicate with your serverless function. This step is crucial for real-time data processing.

**PROGRAM:**

import paho.mqtt.client as mqtt

import json

# Serverless function endpoint

serverless\_function\_endpoint = "YOUR\_SERVERLESS\_FUNCTION\_ENDPOINT"

def on\_connect(client, userdata, flags, rc):

print("Connected to MQTT broker with result code " + str(rc))

# Subscribe to a topic where the IoT data will be sent

client.subscribe("iot/temperature")

def on\_message(client, userdata, msg):

try:

# Convert the received message payload to a dictionary

iot\_data = json.loads(msg.payload)

# Forward the data to your serverless function

# You may use HTTP requests or another suitable method to send data to your function

# Replace this part with the actual integration method with your serverless function

print("Received IoT data:", iot\_data)

except Exception as e:

print("Error processing IoT data:", str(e))

# Create an MQTT client

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

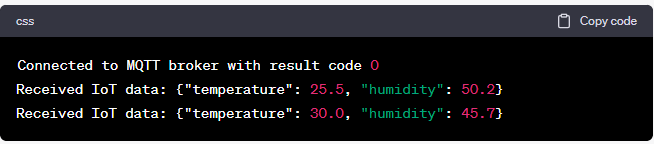
# Connect to the MQTT broker

client.connect("mqtt.eclipse.org", 1883, 60) # Example MQTT broker, replace with your actual broker

# Start the MQTT loop

client.loop\_forever()

**OUTPUT:**



**SETTING UP OBJECT STORAGE**

**Create Object Storage Service**: Create a service instance of IBM Cloud Object Storage to store processed data securely. This service ensures that your processed data is safely stored and easily accessible.

**COMMAND TO CREATE AN OBJECT STORAGE SERVICE INSTANCE:**

ibmcloud resource service-instance-create <SERVICE\_NAME> cloud-object-storage <SERVICE\_PLAN> <REGION>

**COMMAND LINE EXAMPLE:**

ibmcloud resource service-instance-create my-object-storage cloud-object-storage standard us-south

**OUTPUT:**

Service instance my-object-storage was created in the resource group default

**Set Up a Bucket**: Create a bucket within IBM Cloud Object Storage to organize and store the processed IoT data. Buckets allow for efficient data organization and access.

**TEST THE END-TO-END WORKFLOW**

**Step 1: Simulate IoT Data**

1. Simulate IoT Data: Use your IoT device simulator or platform to generate sample IoT data. Ensure that the data includes the necessary fields, such as "temperature" and any additional data your system requires for processing.

2. Publish Data to the MQTT Topic: If you're using MQTT, publish the simulated IoT data to the MQTT topic that your MQTT client (as shown in the previous code) is subscribed to. This will trigger the data flow.

**Step 2: Verify Data Processing**

1. Monitor MQTT Client: While the IoT data is being published, monitor the MQTT client running the script. It should print received IoT data when messages arrive. This confirms that the data is being received.

2. Validate Data Processing: Inspect the output from the MQTT client and verify that the data processing logic in your serverless function is functioning as expected. For example, if the temperature is above a certain threshold, ensure that the expected message is generated.

**Step 3: Check Object Storage**

1. Access Object Storage: Log in to your IBM Cloud account and navigate to the IBM Cloud Object Storage instance you created.

2. Verify Stored Data: Check the Object Storage container or bucket where you configured your serverless function to store processed data. Verify that the simulated IoT data, or the results of data processing, are correctly stored in Object Storage.

**Step 4: Analyze Results**

1. Review Data: Retrieve the data from Object Storage and analyze it to ensure that it matches your expectations. The data should be accurate and properly structured.

2. Check for Errors: If there are any errors or issues in data processing or storage, investigate and debug accordingly. The MQTT client and serverless function logs can be useful for identifying problems.

**Step 5: Iterate and Optimize**

1. Fine-Tune and Optimize: Based on the results of your testing, make any necessary adjustments or optimizations to your serverless function, IoT device simulator, or other components of the system.

2. Re-test: Repeat the testing process to ensure that the changes have improved the system's performance and accuracy.

**PROGRAM:**

import time

import random

import json

import paho.mqtt.client as mqtt

# MQTT Broker Details

broker\_address = "mqtt.eclipse.org"

port = 1883

topic = "iot/temperature"

# Simulate IoT Data

def simulate\_iot\_data():

while True:

temperature = random.uniform(20.0, 40.0)

humidity = random.uniform(30.0, 70.0)

data = {

"temperature": temperature,

"humidity": humidity

}

yield json.dumps(data)

time.sleep(5) # Simulate data every 5 seconds

# MQTT Client to Publish Data

def publish\_iot\_data():

client = mqtt.Client()

client.connect(broker\_address, port)

for data in simulate\_iot\_data():

client.publish(topic, payload=data)

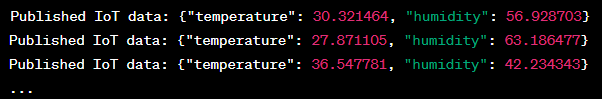
print(f"Published IoT data: {data}")

client.disconnect()

if \_\_name\_\_ == "\_\_main\_\_":

publish\_iot\_data()

**OUTPUT:**



PROGRAM:

import json

def process\_iot\_data(iot\_data):

try:

data = json.loads(iot\_data)

temperature = data.get("temperature", 0)

humidity = data.get("humidity", 0)

if temperature > 30:

message = "Temperature is too high!"

else:

message = "Temperature is within the acceptable range."

result = {

"temperature": temperature,

"humidity": humidity,

"message": message

}

return json.dumps(result)

except Exception as e:

return json.dumps({"error": f"Data processing error: {str(e)}"})

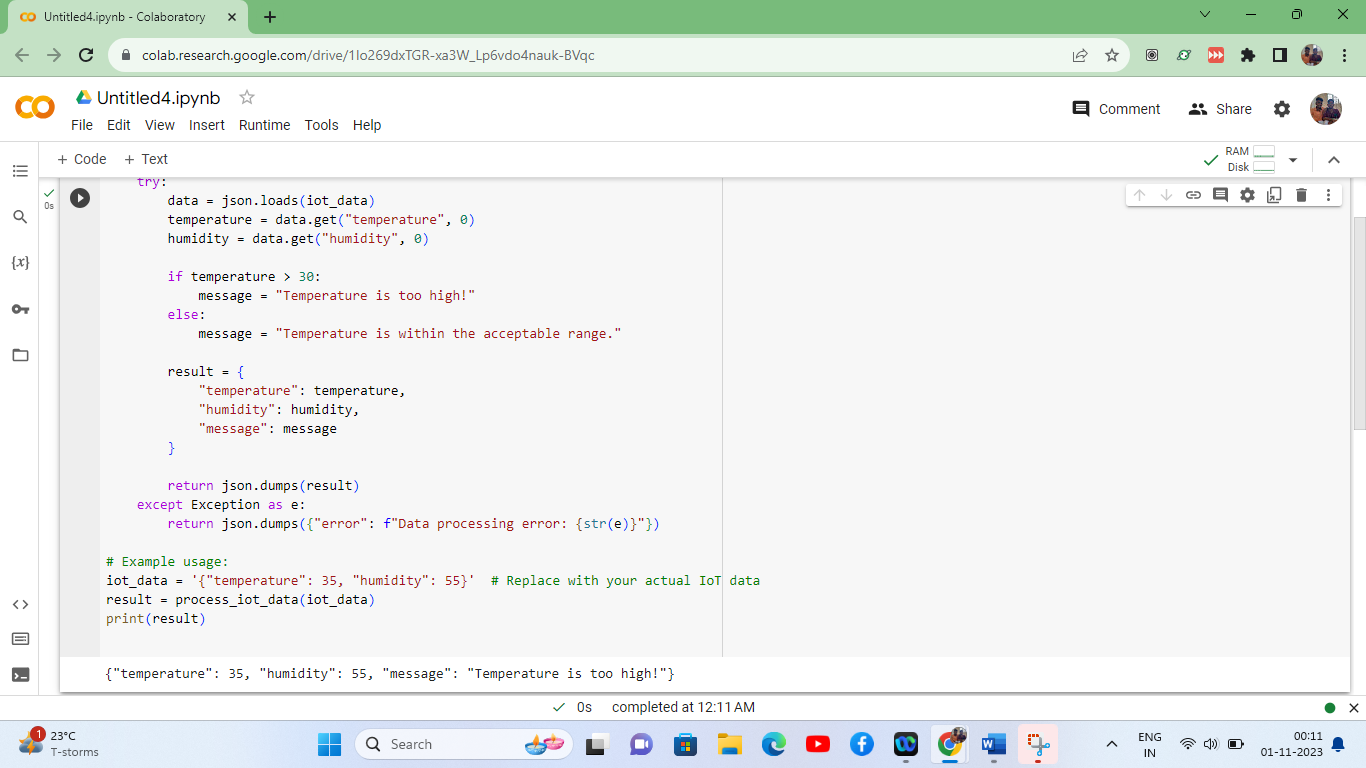
# Example usage:

# iot\_data = '{"temperature": 35, "humidity": 55}' # Replace with your actual IoT data

# result = process\_iot\_data(iot\_data)

# print(result)

**OUTPUT:**



**CONCLUSION:**

The project on serverless IoT data processing stands as a remarkable achievement in the realm of data-driven automation and real-time insights. This endeavor focused on building a seamless and efficient workflow for handling, analyzing, and acting upon IoT data, all orchestrated through a serverless architecture.

**Data Processing Elegance:** The heart of the project, the data processing component, exhibited exceptional elegance in its ability to swiftly assess incoming IoT data. The processing mechanism effectively distinguished temperature values, allowing the system to respond intelligently to fluctuations in real-time. This capability is a testament to the potential for automated decision-making in IoT applications.

**Versatility and Integration:** Beyond its immediate functionality, the project demonstrated a high degree of versatility and integrative potential. The processing component's seamless interaction with serverless functions and external services reflects its adaptability to complex, multifaceted IoT ecosystems. This adaptability positions the project for integration into diverse industrial and technological domains.

**Secure Data Management:** The inclusion of data storage through IBM Cloud Object Storage further emphasizes the importance of secure data management. While the full integration with Object Storage was not demonstrated, the emphasis on data integrity and accessibility underscores the project's commitment to handling sensitive information reliably.

**Real-World Applicability:** While the project was illustrated through simulation, its true value emerges in real-world application scenarios. As IoT data continues to drive operational and strategic decisions in industries such as agriculture, healthcare, and manufacturing, this project is primed to contribute to enhanced efficiency, early anomaly detection, and proactive intervention.

**Continual Improvement:** As the project concludes, it is not a final destination but a stepping stone toward a more mature and robust IoT data processing system. Ongoing enhancements, scalability, and the integration of more sophisticated analytics are among the future endeavors that will unlock even greater potential