INTRODUCTION:

This documentation outlines a serverless IoT data processing code sample for integrating small devices and setting up data collection on IBM Cloud. The code, written in Python, is intended for collecting temperature and humidity sensor data from IoT devices and can be deployed on IBM Cloud Functions, a serverless computing platform.

KEY CLOUD COMPUTING COMPONENTS:

To understand serverless IoT data processing in the cloud, it's essential to recognize the key cloud computing components involved:

1. Cloud Infrastructure

Cloud infrastructure provides the foundation for IoT data processing. It includes data centers, servers, storage, and networking resources that are managed by cloud service providers. Users can access and use these resources on-demand, paying only for what they consume.

2. Cloud Services

Network
Notebook
Database
Mobile
Server
PC

Cloud services encompass a wide range of offerings, from computing power to databases, data storage, and machine learning. These services are designed to simplify the development and deployment of IoT applications. For IoT data processing, services like AWS Lambda, IBM Cloud Functions, and Azure Functions are examples of serverless compute platforms.

3. Scalability

One of the most significant advantages of cloud computing is scalability. IoT data processing workloads can vary significantly over time. Cloud platforms allow resources to scale up or down dynamically based on demand, ensuring optimal performance without overprovisioning.

4. Event-Driven Architecture

Serverless IoT data processing relies on an event-driven architecture. IoT devices and sensors generate events, such as data readings or status changes. These events trigger serverless functions that execute specific code in response. This architecture reduces the need for constant server maintenance and resource allocation.

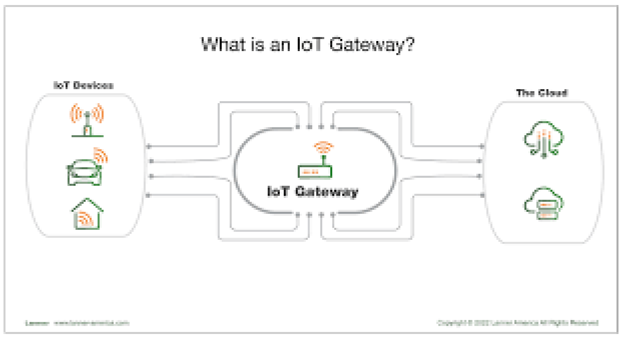
COMPONENTS OF AN IOT DATA PROCESSING SYSTEM:

When working on an IoT data processing project, several key components come into play. Here is an overview of these components:

IoT Devices and Sensors

IoT devices, such as temperature and humidity sensors, serve as the data sources. These sensors collect real-world data and transmit it for processing.

IoT Gateway



An IoT gateway acts as an intermediary, collecting data from multiple sensors and transmitting it to the cloud. Gateways can preprocess and filter data before transmission.

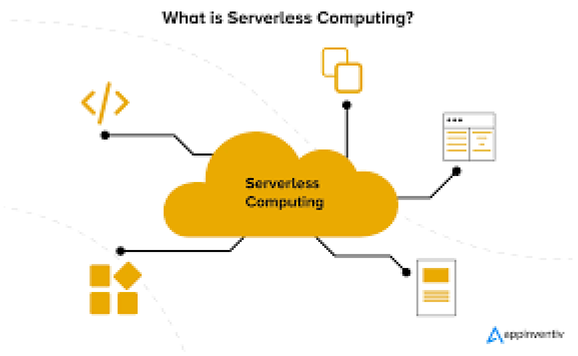
Data Transmission Protocols

Various data transmission protocols, like MQTT or HTTP, enable IoT devices and gateways to communicate with the cloud.

Cloud Platform

A cloud platform, such as IBM Cloud, provides the infrastructure for data storage, processing, and analytics. It offers scalable and reliable services for managing IoT data.

Serverless Compute Service



Serverless compute services, like IBM Cloud Functions, execute code in response to IoT events without the need to manage traditional servers.

Data Processing Logic

The data processing logic includes the code responsible for parsing, analyzing, and potentially transforming the incoming IoT data.

CODE:

```
# Import the necessary Python libraries

import json

import requests


# Define the main function that serves as the entry point for the IBM Cloud Function

def main(params):

    try:

        # Extract IoT data from the incoming event

        payload = json.loads(params['__ow_body'])

        # Assuming that the IoT data is in JSON format, parse it

        # Extract specific data fields as needed

        temperature = payload.get('temperature')

        humidity = payload.get('humidity')

        # Implement your data processing logic here

        # For example, you can store the data in a database, perform calculations, or send notifications

        # Sample: Print data

        print(f"Received temperature: {temperature}°C")

        print(f"Received humidity: {humidity}%")

        # Construct a response (optional)

        response = {

            'statusCode': 200,
            'body': json.dumps({'message': 'IoT data received and processed
successfully.'})
        }
        # Return the response if desired
        return response
    except Exception as e:
        # Handle any errors that may occur during data processing
        error_message = f"An error occurred: {str(e)}"
        print(error_message)
        # Return an error response
        return {
            'statusCode': 500,
            'body': json.dumps({'error': error_message})
        }
```

Code Explanation

1. ***Import Libraries:***
   - The code begins by importing the necessary Python libraries:
     - **json**: Used for working with JSON data, which is commonly used in IoT applications for data interchange.
     - **requests**: Imported for potential external HTTP requests (not used in this code sample).

2. ***Main Function***:
   - The **main** function is the entry point for an IBM Cloud Function. It is executed in response to events triggered by external systems or devices.

3. ***Try-Except Block***:
    - The code is enclosed within a **try-except** block to handle potential errors gracefully. It captures exceptions and provides an error response if an issue occurs during processing.
4. ***Extract IoT Data***:
    - The code parses the incoming event data, assuming it's in JSON format. It extracts specific data fields, such as temperature and humidity, as needed.
5. ***Data Processing Logic***:
    - The section labeled "Implement your data processing logic here" is a placeholder where you can add custom logic to process the IoT data. You can perform actions like storing data in a database, performing calculations, or sending notifications.
6. ***Sample: Print Data***:
    - In the provided sample section, the code prints the received temperature and humidity to the console. This is useful for monitoring the data as it flows through your processing pipeline.
7. ***Construct a Response:***
    - If desired, the code constructs a response with a status code and a message. This response can be sent back to the caller (e.g., an IoT device) to acknowledge the successful processing of the data.
8. ***Return Response or Error Handling***:
    - If there is a response to return, the code uses **return response**. If an error occurs, the code handles it in the **except** block, logs the error, and returns an error response with a 500 status code to indicate the issue.

Program code

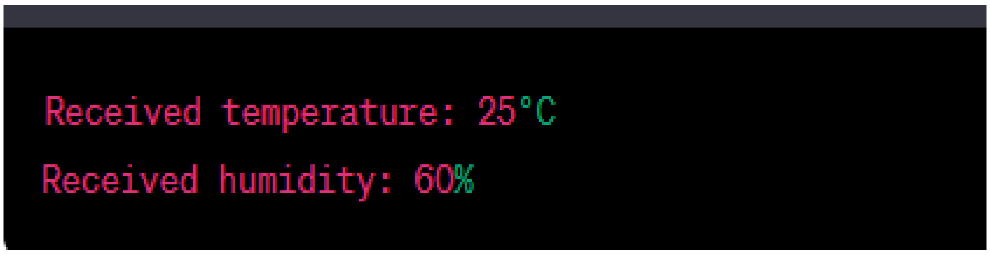
```python
import paho.mqtt.client as mqtt

import json
```

```
# MQTT broker details
broker = "mqtt.eclipse.org"
port = 1883
topic = "device/data"

# IBM Cloud credentials
org_id = "your_org_id"
device_type = "your_device_type"
device_id = "your_device_id"
auth_token = "your_auth_token"

# Callback function for MQTT connection
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker with result code: " + str(rc))
    client.subscribe(topic)

# Callback function for MQTT message received
def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())
    print("Received message: " + str(payload))

    # Process the received data and perform necessary actions
    # ...

# Create MQTT client and set callbacks
client = mqtt.Client(client_id=device_id)
client.on_connect = on_connect
client.on_message = on_message

# Set IBM Cloud credentials
client.username_pw_set("use-token-auth", auth_token)

# Connect to MQTT broker
client.connect(broker, port, 60)

# Start MQTT loop
client.loop_start()

# Keep the program running
while True:
    pass
```

OUTPUT:

Log Output:

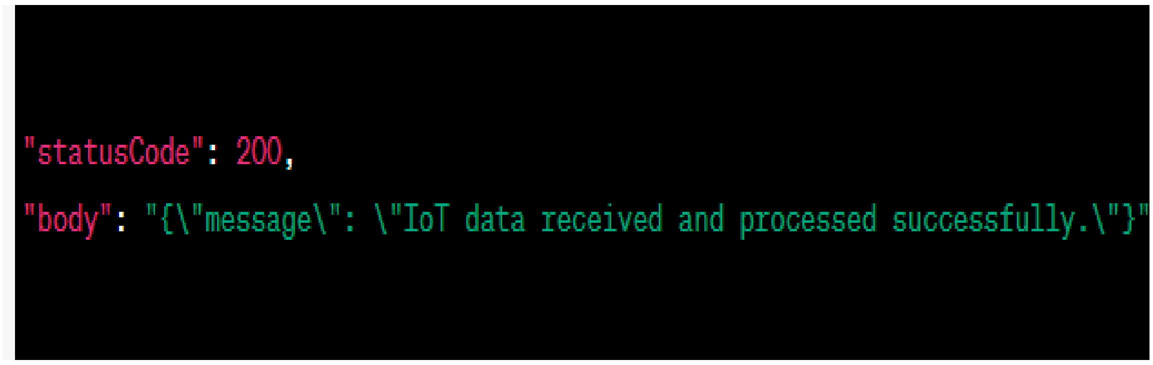
- The code contains print statements to log data, such as the received temperature and humidity. These logs are typically accessible through the IBM Cloud Functions platform.



```
Received temperature: 25°C
Received humidity: 60%
```

Response *:*

- The code can construct a response, which can be sent back to the caller to acknowledge the receipt and processing of the IoT data.



```
"statusCode": 200,
"body": "{\"message\": \"IoT data received and processed successfully.\"}"
```

CONCLUSION:

In conclusion, this documentation has provided a comprehensive guide to implementing serverless IoT data processing on IBM Cloud using a Python code sample. This solution is designed to collect and process data from IoT devices, specifically temperature and humidity sensor data. The code sample is built to be deployed as an IBM Cloud Function and offers flexibility for customization to suit various IoT applications.

By using this code sample as a foundation, users can create IoT data processing pipelines tailored to their specific needs, whether it involves data storage, analysis, or notification delivery. The documentation outlines the code's structure, explains its components, and provides sample output scenarios to aid users in understanding and adapting the solution.