***Python code to integrate smart devices with IoT:***

import paho.mqtt.client as mqtt

import jsonimport time

# MQTT settings

mqtt\_broker = "mqtt.example.com“

mqtt\_port = 1883

mqtt\_topic = "smart\_home/devices“

# Device settings

devices ={

"light\_1": {"state": False},

"light\_2": {"state": True},

"thermostat": {"temp": 22}

}

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

print("Connected with result code "+str(rc))

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

print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect(mqtt\_broker, mqtt\_port)

while True:

for device, state in devices.items():

payload = json.dumps(state)

client.publish(mqtt\_topic+"/"+device, payload)

time.sleep(5)

client.loop\_forever()

***EXPLANATION :*** This code creates a MQTT client to connect to a MQTT broker. It has a devices dictionary containing the state of some sample smart devices. In the main loop, it publishes the state of each device to a MQTT topic periodically. The topic is in the format "smart\_home/devices/<device\_id>".The MQTT broker can then distribute these messages to other clients like a home automation controller to take action based on the device state. Additional devices can be added to the devices dictionary.

***Some key points:***

* - paho.mqtt.client library is used for MQTT messaging.
* - json is used to serialize the device state into JSON payload.
* - Devices publish state messages periodically to broker using MQTT topics.
* - Broker can distribute messages to interested clients like a home automation system.

***Python code to collect data from the integrated IoT devices using MQTT:***

import paho.mqtt.client as mqtt

import json

# MQTT settings

mqtt\_broker ="mqtt.example.com“

mqtt\_port = 1883

mqtt\_topic ="smart\_home/devices“

# Callback for when a PUBLISH message

is received from the server.

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

# Decode JSON payload

data =json.loads(msg.payload.decode())

device\_id = msg.topic.split("/")[-1]

# Print device data

print(f"Received data from {device\_id}: {data}"))

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect(mqtt\_broker, mqtt\_port)

while True:

for device, state in devices.items():

payload = json.dumps(state)

client.publish(mqtt\_topic+"/"+device, payload)

time.sleep(5)

client.loop\_forever()

***EXPLANATION :***

* This code creates a MQTT client and subscribes to all topics under "smart\_home/devices". When a new message is received, the on\_message callback parses the JSON payload, extracts the device id from topic and processes the data.The data can be stored to a database, file or processed further. Additional logic can be added to the on\_message callback for specific data handling required for different devices. This allows collecting data published by the variety of IoT devices in an automated way for further analytics and visualization.

***DATA STORAGE :***

Python

import ibm\_boto3

from ibm\_botocore.client import Config

# Initialize SDK credentials

ibm\_api\_key\_id = "<ibm\_api\_key>"

service\_instance\_id = "<service\_instance\_id>“

# Create resource for accessing

IBM Cloud Object Storage

cos = ibm\_boto3.resource("s3",

ibm\_api\_key\_id=ibm\_api\_key\_id,

ibm\_service\_instance\_id=

service\_instance\_id,

config=Config(signature\_version="oauth"),

endpoint\_url='https://s3.private.us-south.cloud-object-storage.appdomain.cloud’)

# Get reference to bucket

bucket = cos.Bucket("iot-data-bucket")

#Sample IoT data

iot\_data = {

"timestamp": "1608150940",

"device\_id": "sensor1",

"temperature": 29,

"humidity" : 80

}

# Upload data as object to COS

object = bucket.Object("iot-data.json")

response = object.put(

Body=json.dumps(iot\_data)

)

print(response)

***EXPLANATION :***

* This saves the IoT data dictionary as a JSON object in IBM Cloud Object Storage. The credentials, service instance ID, bucket name etc. would need to be configured as per your environment. The data can then be processed and analyzed from COS.

***DATA PROCESSING :***

Python

import json

import ibm\_boto3from ibm\_botocore.client

import Config

# SDK credentials

api\_key = "<ibm\_api\_key>“

service\_instance\_id = "<service\_instance\_id>“

# COS access

cos = ibm\_boto3.resource("s3",

ibm\_api\_key\_id=api\_key,

ibm\_service\_instance\_id=

service\_instance\_id,

config=Config(signature\_version="oauth"),

endpoint\_url='https://s3.private.us

cloud-object-storage.appdomain.cloud’)

bucket = cos.Bucket("iot-data-bucket")

# Read object containing IoT data

obj = bucket.Object("iot-data.json“)

file\_content = obj.get()

['Body'].read()

# Deserialize JSON data

iot\_data = json.loads(file\_content)

# Process data

device\_id =iot\_data['device\_id’]

temperature = iot\_data['temperature'] humidity = iot\_data['humidity’]

# Print processed results

print(f"Device {device\_id}

recorded temperature {temperature} and humidity {humidity}")

***EXPLANATION :***

* This loads the IoT data JSON from COS, deserializes it and extracts the device ID, temperature and humidity. These values are printed, but additional processing and analysis can be done on the extracted data based on your requirements.The same approach of reading data from COS and processing it can be implemented in a serverless function as well.

***DATA VISUALIZATION :***

Python

import json

import ibm\_boto3from ibm\_botocore.client

import Config

import matplotlib.pyplot as plt

# SDK credentials

api\_key = "<ibm\_api\_key>“

service\_instance\_id = "<service\_instance\_id>"

# COS access

cos = ibm\_boto3.resource("s3",

ibm\_api\_key\_id=api\_key,

ibm\_service\_instance\_id=

service\_instance\_id,

config=Config(signature\_

version="oauth"),

endpoint\_url='https://s3.private.us.cloud-object-storage.appdomain.cloud')bucket = cos.Bucket("iot-data-bucket")

# Read JSON object from COS

obj = bucket.Object("iot-data.json")

file\_content = obj.get()

['Body'].read()

# Deserialize JSON

iot\_data = json.loads(file\_content)

# Extract data

timestamps = []

temps = []

for d in iot\_data:

timestamps.append(d['timestamp'])

temps.append(d['temperature’])

# Plot visualization

plt.plot(timestamps, temps)

plt.title("Temperature Sensor Data")

plt.xlabel("Time")

plt.ylabel("Temperature (C)")

plt.show()

***EXPLANATION :***

* This loads the IoT JSON data from COS, extracts the timestamp and temperature values into lists, and uses matplotlib to plot a graph with time on x-axis and temperature on y-axis.Similar visualizations can be created using IBM Cloud services like Watson Studio notebooks leveraging various Python libraries like matplotlib, seaborn, plotly etc. based on the specific visualization needs.

***AUTOMATION :***

Python

import json

from ibm\_cloud\_sdk\_core

.authenticators import

IAMAuthenticator

from ibm\_funcs.runtime\_client

import RuntimeClient

# IBM Functions credentials

ibm\_api\_key = '<ibm-api-key>’

function\_namespace = '<function-namespace>’

authenticator = IAMAuthenticator(ibm\_api\_key)

client = RuntimeClient(authenticator=authenticator)

# Cloud Function to process IoT data

@client.function.invoke

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

# Extract sensor readings

device\_id = iot\_data['device\_id']

temperature = iot\_data['temperature']

# Check temperature threshold

if temperature > 30:

send\_alert(device\_id)

# Cloud Function to send alert

@client.function.invoke

def send\_alert(device\_id):

print(f'Alert! High temperature on device

{device\_id}')

# Sample IoT data

iot\_data = { "device\_id": "sensor-1", "temperature": 35 }

***EXPLANATION :***

* This code defines two IBM Cloud Functions - one to process IoT data and another to send alerts. The process\_iot\_data function checks the temperature reading and invokes send\_alert if it exceeds the threshold. The functions can be triggered through events, webhooks or directly with data as shown. This demonstrates how serverless functions can automate actions based on IoT data analysis.

***SECURITY AND COMPLIANCE :***

Python

import json

from ibm\_cloud\_sdk\_core.security

import BaseAuthenticator,

IAMTokenAuthenticator

from ibm\_boto3 import client

# Authentication using IAM API

Key

iam\_api\_key = '<iam\_api\_key>’

authenticator = IAMAuthenticator(iam\_api\_key)

# Encryption using IBM Hyper Protect Crypto

from hpcs\_python\_sdk import

HPCSClient

crypto\_client = HPCSClient('<service-URL>’)

# Encrypt data

encrypted\_data = crypto\_client

.encrypt('<plaintext-data>’

)# IoT platform service client

service\_client = client('iot’,

authenticator=authenticator)

# Securely send data

response =

service\_client.publish\_event(

typeId='sensor-data',

deviceType='raspberry-pi’,

deviceId='pi01',

eventId='event01',

data=json.dumps({'encrypted\_data’:

encrypted\_data}),

qualityOfService=0)

print(response)

***EXPLANATION :***

* Using IAM authentication instead of API key for service access- Encrypting IoT data before sending to platform - Publishing events securely over HTTPS- Storing encryption keys in secure vault like Key ProtectOther measures like VPC, firewall rules, VPN, role-based access, audit logs etc. can be implemented in Infrastructure-as-Code templates.This demonstrates a secure approach for building IoT applications on IBM Cloud.