**SMART BUILDING ENERGY OPTIMIZATION WITH AWS IOT**

**Step 1: Set Up IoT Devices in AWS IoT Core**

**1. Register Devices**

1. Go to the [AWS Management Console](https://aws.amazon.com/console/).
2. Navigate to **IoT Core**.
3. Under **Manage > Things**, click **Create Thing**:
   * Choose "Create a single thing."
   * Provide a **name** (e.g., Building\_TempSensor).
   * **Download certificates** (X.509) for secure communication.
4. Save the certificates securely; you’ll need them later to connect your devices.

**2. Configure MQTT Topics**

1. Go to **Test > MQTT Test Client** in IoT Core.
2. Define topics for device communication (e.g., building/temperature and building/humidity).

**3. Create and Attach Policies**

1. Navigate to **Secure > Policies**, create a new policy:

json

Copy code

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "iot:Publish",

"Resource": "arn:aws:iot:<region>:<account\_id>:topic/building/\*"

},

{

"Effect": "Allow",

"Action": "iot:Connect",

"Resource": "\*"

}

]

}

1. Attach this policy to the registered device.

**Step 2: Process IoT Data Using AWS Lambda**

**1. Create a Lambda Function**

1. Navigate to **AWS Lambda** and click **Create Function**:
   * Choose "Author from scratch."
   * Provide a **name** (e.g., TempControlFunction).
   * Select **Python 3.x** as the runtime.

**2. Write Function Code**

This Lambda function processes incoming temperature data and triggers actions based on conditions:

python

Copy code

import json

import boto3

def lambda\_handler(event, context):

# Log incoming data

print("Received event: ", event)

# Extract temperature data

temperature = event['temperature']

device\_id = event['device\_id']

# Check temperature threshold

if temperature > 30:

# Example action: Notify via SNS

sns\_client = boto3.client('sns')

sns\_client.publish(

TopicArn="arn:aws:sns:<region>:<account\_id>:HighTempAlerts",

Message=f"Temperature Alert! Device {device\_id} reported {temperature}°C.",

Subject="High Temperature Alert"

)

return {"message": "Alert sent!"}

else:

return {"message": "Temperature is normal."}

**3. Attach IoT Rule to Lambda**

1. In **IoT Core > Act > Rules**, create a new rule:
   * Use the SQL query:

sql

Copy code

SELECT \* FROM 'building/temperature' WHERE temperature > 30

* + Add an **action** to invoke the Lambda function.

**Step 3: Store and Analyze Data**

**1. Create an S3 Bucket**

1. Navigate to **Amazon S3** and create a bucket (e.g., building-energy-data).
2. Set bucket permissions to allow access from Lambda or IoT.

**2. Set Up Amazon Redshift**

1. In **Redshift**, create a new cluster.
2. Use **AWS Query Editor** to create a table for storing data:

sql

Copy code

CREATE TABLE energy\_data (

device\_id VARCHAR(50),

timestamp TIMESTAMP,

temperature FLOAT,

humidity FLOAT

);

**3. Load Data into Redshift**

Use the **COPY** command to load data from S3 into Redshift:

sql

Copy code

COPY energy\_data

FROM 's3://building-energy-data/device\_data.csv'

CREDENTIALS 'aws\_access\_key\_id=<key>;aws\_secret\_access\_key=<secret>'

CSV;

**Step 4: Create a Web-Based Dashboard**

**1. Set Up API Gateway**

1. Navigate to **API Gateway** and create a REST API.
2. Add a resource (e.g., /energy-data) and a method (e.g., GET).
3. Integrate the GET method with a Lambda function or Redshift query.

**2. Frontend Development**

Use **React** or **Angular** to create a dashboard interface. Example for React:

jsx

Copy code

import React, { useState, useEffect } from "react";

function Dashboard() {

const [data, setData] = useState([]);

useEffect(() => {

fetch("https://<api-gateway-endpoint>/energy-data")

.then((res) => res.json())

.then((data) => setData(data));

}, []);

return (

<div>

<h1>Energy Dashboard</h1>

<ul>

{data.map((item, index) => (

<li key={index}>

{item.device\_id}: {item.temperature}°C at {item.timestamp}

</li>

))}

</ul>

</div>

);

}

export default Dashboard;

**Step 5: Monitor with AWS CloudWatch**

**1. Set Up Metrics**

1. Navigate to **CloudWatch > Metrics**.
2. Create custom metrics for IoT device performance and Lambda executions.

**2. Create Dashboards**

1. In **CloudWatch > Dashboards**, create a new dashboard.
2. Add widgets for key metrics, such as temperature trends and Lambda errors.

**Step 6: Deployment**

**1. CI/CD Pipeline**

1. Use **AWS CodePipeline** to automate deployments:
   * Source: GitHub or CodeCommit.
   * Build: AWS CodeBuild.
   * Deploy: AWS Lambda or ECS.

**2. Security**

1. Regularly review **IAM policies**.
2. Enable **CloudTrail** for auditing API activity.

**3. Backup and Scalability**

1. Set up regular backups for S3 and Redshift.
2. Use **Auto Scaling** for Lambda and Redshift to handle load variations.

By following these steps and using the provided code snippets, you can successfully implement and deploy your project on AWS. Let me know if you need help with specific configurations or debugging during deployment!