

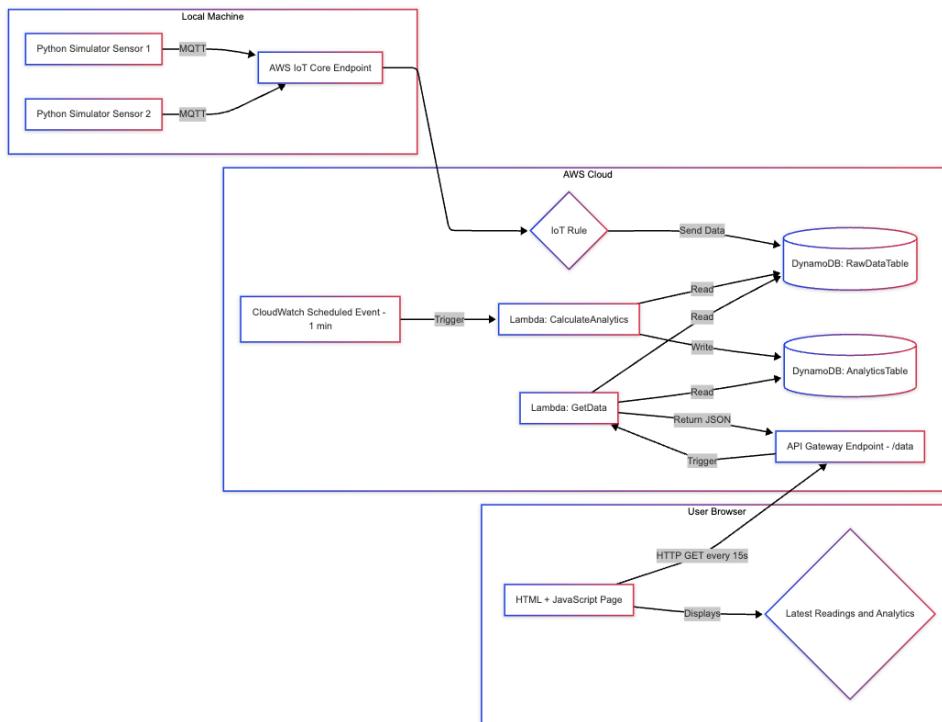
Cloud Application Workshop- AWS IOT DASHBOARD

Team Name: Nano Clouds

Team Members: Apoorva, Junie, Rinku, Ananya

Date: 05/01/2025

Architecture diagram :



Resources used:

1. **AWS IoT Core:** To securely connect simulated devices and ingest MQTT data.
2. **AWS Lambda (x2):**
 - One for basic analytics (triggered periodically).
 - One to serve data to the dashboard via API Gateway.
3. **Amazon DynamoDB (x2 tables):**
 - One to store raw sensor data.
 - One to store calculated analytics.
4. **Amazon CloudWatch Events (EventBridge):** To trigger the analytics Lambda periodically.
5. **Amazon API Gateway:** To create a simple HTTP endpoint for the dashboard to fetch data.
6. **(Local) Python Script:** To simulate the two sensors.
7. **(Local) Basic HTML/JavaScript:** For the dashboard visualization.

Phase 1: Setup AWS resources

1. Create DynamoDB tables

The screenshot shows the AWS DynamoDB Tables page. On the left, there's a sidebar with links like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a DAX section with Clusters, Subnet groups, Parameter groups, and Events. The main area is titled "Tables (2) Info". It has a search bar and filters for "Any tag key" and "Any tag value". A table lists two tables: "RawSensorData" and "SensorAnalytics". Both are active, with "RawSensorData" having a partition key "deviceID (S)" and a sort key "timestamp (N)", and "SensorAnalytics" having a partition key "sensorType (S)". Both have 0 indexes, 0 replication regions, and deletion protection set to "Off".

2. Setup AWS IoT core:

The screenshot shows the AWS IoT Policies page. On the left, there's a sidebar with Monitor, Connect (Connect one device, Connect many devices, Domain configurations), Test (Device Advisor, MQTT test client, Device Location, Query connectivity status), and a bottom section with three dots. The main area is titled "AWS IoT policies (1) Info". It has a success message: "Successfully created policy MyDevicePolicy." A "View policy" button is available. The table lists one policy: "MyDevicePolicy". A note states: "AWS IoT policies allow you to control access to the AWS IoT Core data plane operations. AWS IoT policies are separate and different from IAM policies. AWS IoT policies apply only to AWS IoT data plane operations." At the bottom right, there's a "Talking:" indicator.

The screenshot shows the AWS IoT Things management interface. On the left, a sidebar lists various management options like Test, Device Advisor, MQTT test client, Device Location, and Query connectivity status. The main area displays a success message: "You successfully created certificate 5af8a099a8f14bdb80627ca18763f4ec5ffcfb7c1829e53a0ebfefb19eb8500." Below this, a "Things (1) Info" section shows a single thing named "MySimulatedDevice".

iot endpoint :

The screenshot shows the AWS IoT Domain configurations page. It lists a single domain configuration named "iot:Data-ATS" with the domain name "a1lh83zoeq7hyw-ats.iot.us-east-1.amazonaws.com" and a status of "Enabled".

3. Create IoT rule to store data

The screenshot shows the AWS IoT Rules page. A success message at the top states "Successfully created rule StoreSensorDataRule." The main area displays a "Rules (1) Info" section with one rule named "StoreSensorDataRule" which is active and has a rule topic of "sensors/data".

Phase 2: Create Lambda functions

1. Create IAM role for lambdas

The screenshot shows the AWS IAM Roles page with the LambdaDynamoDBAccessRole selected. The role summary indicates it allows Lambda functions to call AWS services on behalf of the user. The Permissions tab is active, showing two attached managed policies: AmazonDynamoDBFullAccess and AWSLambdaBasicExecutionRole. The ARN of the role is listed as arn:aws:iam::992382397944:role/LambdaDynamoDBAccessRole.

2. Create Analytics lambda function and deployed it

The screenshot shows the AWS Lambda Functions page with the CalculateSensorAnalytics function selected. A success message at the top states "Successfully updated the function CalculateSensorAnalytics." The Code source tab is active, displaying the function's code in a code editor. The code is written in Python and uses the AWS Lambda handler interface to interact with a DynamoDB table. The code includes logic to calculate statistics for data since a specified timestamp and handle potential pagination if data volume grows.

```
def lambda_handler(event, context):
    cutoff_time_ms = now_ms - TIME_WINDOW_MS
    print(f"Calculating stats for data since timestamp: {cutoff_time_ms}")

    # Use Scan - less efficient for large tables, but simpler for this demo
    # In production, consider Query with a GSI on timestamp if needed
    response = raw_table.scan(
        FilterExpression=boto3.dynamodb.conditions.Attr('timestamp').gt(Decimal(cutoff_time_ms))
    )

    items = response.get('Items', [])
    # Handle potential pagination if data volume grows
    while 'LastEvaluatedKey' in response:
        response = raw_table.scan(
            FilterExpression=boto3.dynamodb.conditions.Attr('timestamp').gt(Decimal(cutoff_time_ms))
            ExclusiveStartKey=response['LastEvaluatedKey']
        )
```

3. Create Dashboard lambda function

The screenshot shows the AWS Lambda console with the function `GetSensorDataForDashboard`. A green success message at the top says "Successfully updated the function GetSensorDataForDashboard." The function overview section shows the function name `GetSensorDataForDashboard`, a description field (empty), and a last modified time of "2 minutes ago". The function ARN is listed as `arn:aws:lambda:us-east-1:992382397944:function:GetSensorDataForDashboard`. The function URL is also listed. Below the overview, there are tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The "Code" tab is selected.

Phase 3: Connect Components

1. Schedule the analytics lambda

The screenshot shows the AWS EventBridge console with the schedule `RunAnalyticsEveryMinute`. A green success message at the top says "Your schedule RunAnalyticsEveryMinute is being created." The schedule detail section shows the following configuration:

Schedule name	Status	Schedule start time	Flexible time window
<code>RunAnalyticsEveryMinute</code>	Enabled	-	-
Description	Schedule ARN	Schedule end time	Created date
Triggers the CalculateSensorAnalytics Lambda every minute.	<code>arn:aws:scheduler:us-east-1:992382397944:schedule/default/RunAnalyticsEveryMinute</code>	-	May 01, 2025, 19:01:54 (UTC-07:00)
Schedule group name	Action after completion	Execution time zone	Last modified date
default	NONE	America/Los_Angeles	May 01, 2025, 19:01:54 (UTC-07:00)

Below the schedule detail, there are tabs for Schedule, Target, Retry policy, Dead-letter queue, and Encryption. The "Schedule" tab is selected.

2. Create Api Gateway endpoint

The screenshot shows two views of the AWS API Gateway console.

Top View: The "APIs" page displays a single API named "SensorDataAPI". The details are as follows:

Name	Description	ID	Protocol	API endpoint type	Created
SensorDataAPI		nbwco816l2	REST	Regional	2025-05-02

Bottom View: The "Resources" page for the "SensorDataAPI" shows a single resource path: "/data". Under this path, there is an "OPTIONS" method defined.

Method type	Integration type	Authorization	API key
OPTIONS	Mock	None	Not required

The screenshot shows the AWS API Gateway interface. On the left, the navigation pane is open with the path: API Gateway > APIs > SensorDataAPI (nbwco816l2) > Stages. The main content area displays a green success message: "Successfully created deployment for SensorDataAPI. This deployment is active for prod." Below this message, there's a "Notifications" section with various icons and a "Stage actions" dropdown. To the right, a large button labeled "Create stage" is visible. The "prod" stage is selected, and its details are shown in the "Stage details" panel. The stage name is "prod", and it has a rate limit of 10000. The "Web ACL" is set to "-". The "Burst" limit is 5000. Under "Default method-level caching", it says "Inactive". The "Invoke URL" is listed as <https://nbwco816l2.execute-api.us-east-1.amazonaws.com/prod>. There are also sections for "Cache cluster" (inactive) and "Client certificate" (-).

Invoke URL : <https://nbwco816l2.execute-api.us-east-1.amazonaws.com/prod>

Phase 4: Simulate Sensors

1. Create sensor_simulator.py

The screenshot shows the AWS IoT Core AWS Toolkit for VS Code. The code editor window displays the file `sensor_simulator.py` with the following content:

```

1 import time
2 import json
3 import random
4 from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
5 import datetime
6
7 # --- AWS IoT Core Configuration ---
8 # Replace with your actual values
9 AWS_IOT_ENDPOINT = "YOUR_AWS_IOT_ENDPOINT" # e.g., "xxxxx-ats.iot.us-east-1.amazonaws.com"
10 CLIENT_ID = "MySimulatedDevice" # Must match the client ID allowed in your policy
11 PATH_TO_CERT = "./YOUR-certificate.pem.crt" # e.g., "./xxx-certificate.pem.crt"
12 PATH_TO_KEY = "./YOUR-private.pem.key" # e.g., "./xxx-private.pem.key"
13 PATH_TO_ROOT_CA = "./AmazonRootCA1.pem" # Path to the downloaded Amazon Root CA 1
14 TOPIC = "sensors/data"
15
16 # --- Sensor Simulation Parameters ---
17 SENSOR_TYPES = ["temperature", "humidity"] # Our two distinct sensors
18 DEVICE_ID = CLIENT_ID # Use the client ID as the device ID for simplicity
19 SEND_INTERVAL_SECONDS = 10 # Send data every 10 seconds (meets >= 15s requirement per sensor effectively)
20
21 # --- Initialize MQTT Client ---
22 myMQTTClient = AWSIoTMQTTClient(CLIENT_ID)
23 myMQTTClient.configureEndpoint(AWS_IOT_ENDPOINT, 8883)
24 myMQTTClient.configureCredentials(PATH_TO_ROOT_CA, PATH_TO_KEY, PATH_TO_CERT)
25
26 # --- AWSIoTMQTTClient connection configuration ---
27 myMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
28 myMQTTClient.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing
29 myMQTTClient.configureDrainingFrequency(2) # Draining for QoS1 only
30 myMQTTClient.configureConnectDisconnectTimeout(10) # 10s
31 myMQTTClient.configureMQTTOperationTimeout(5) # 5s

```

The terminal at the bottom shows the command `12.0` and a list of installed packages:

```

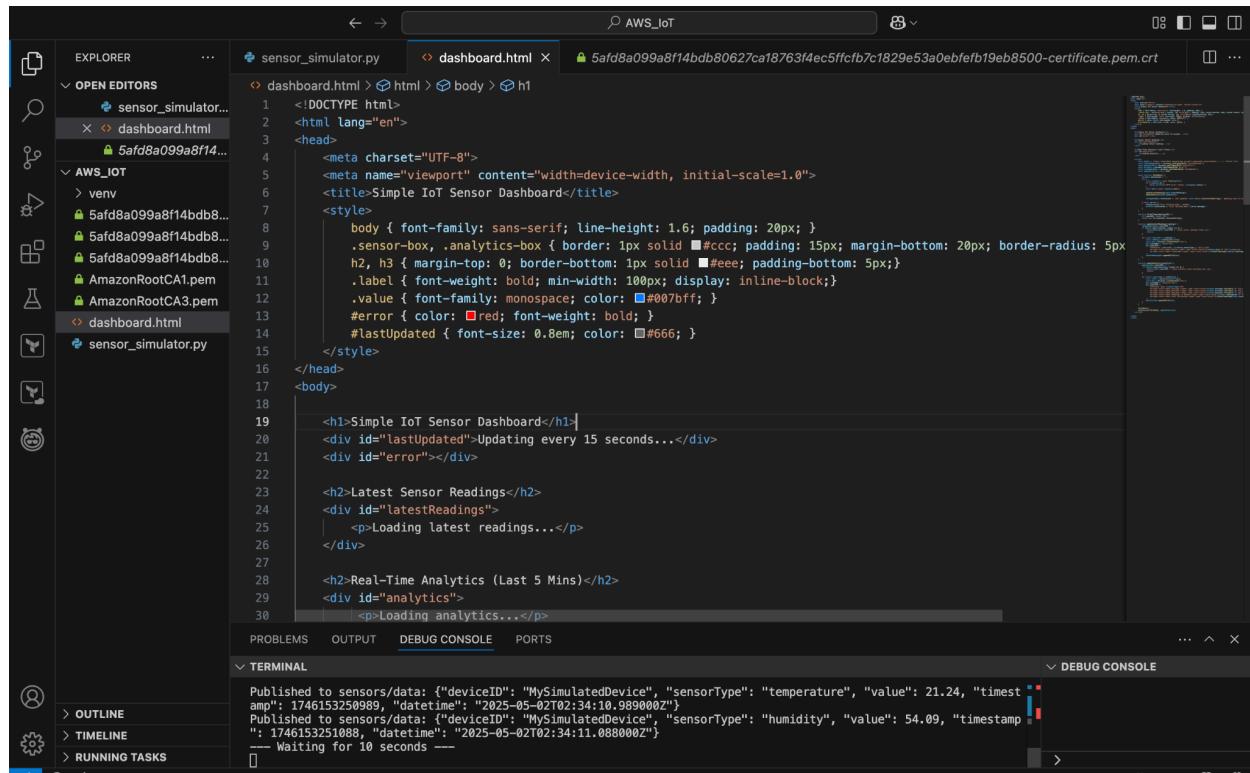
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in ./venv/lib/python3.13/site-packages (from botocore<1.39.0,>=1.38.7->botocore) (2.9.0.post0)
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in ./venv/lib/python3.13/site-packages (from botocore<1.39.0,>=1.38.7->botocore) (2.4.1)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.13/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.39.0,>=1.38.7->botocore) (1.17.0)

```

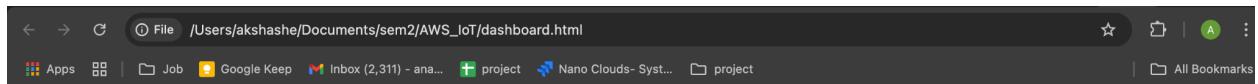
2. Run the simulator

```
0SError: ./YOUR-AWS-IoT.pem.crt: No such file or directory
○ (venv) akshashe@Akshayes-MacBook-Air AWS_IoT % python sensor_simulator.py
Connecting to AWS IoT...
Connected.
/Users/akshashe/Documents/sem2/AWS_IoT/sensor_simulator.py:76: DeprecationWarning: datetime.datetime.utcnowfromtimes
tamp() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datet
ime in UTC: datetime.datetime.fromtimedelta(timestamp, datetime.UTC).
    "datetime": datetime.datetime.utcnowfromtimestamp(timestamp_ms / 1000).isoformat() + "Z"
Published to sensors/data: {"deviceID": "MySimulatedDevice", "sensorType": "temperature", "value": 19.91, "timest
amp": 1746152771176, "datetime": "2025-05-02T02:26:11.176000Z"}
Published to sensors/data: {"deviceID": "MySimulatedDevice", "sensorType": "humidity", "value": 51.1, "timestam
p": 1746152771270, "datetime": "2025-05-02T02:26:11.270000Z"}
--- Waiting for 10 seconds ---
Published to sensors/data: {"deviceID": "MySimulatedDevice", "sensorType": "temperature", "value": 20.22, "timest
amp": 1746152781377, "datetime": "2025-05-02T02:26:21.377000Z"}
Published to sensors/data: {"deviceID": "MySimulatedDevice", "sensorType": "humidity", "value": 50.79, "timestam
p": 1746152781480, "datetime": "2025-05-02T02:26:21.480000Z"}
--- Waiting for 10 seconds ---
Published to sensors/data: {"deviceID": "MySimulatedDevice", "sensorType": "temperature", "value": 20.18, "timest
amp": 1746152791588, "datetime": "2025-05-02T02:26:31.588000Z"}
Published to sensors/data: {"deviceID": "MySimulatedDevice", "sensorType": "humidity", "value": 52.38, "timestam
p": 1746152791695, "datetime": "2025-05-02T02:26:31.695000Z"}
--- Waiting for 10 seconds ---
```

3. Create dashboard.html



Phase 5: Call dashboard.html in browser to see analytics



Simple IoT Sensor Dashboard

Last updated: 7:32:48 PM. Updating every 15 seconds...

Latest Sensor Readings

Device: MySimulatedDevice (humidity)

Value: 50.53

Timestamp: 5/1/2025, 7:30:16 PM

Real-Time Analytics (Last 5 Mins)

Sensor Type: temperature

Average: 20.42

Minimum: 19.59

Maximum: 21.16

Readings in Window: 30

Real-Time Analytics (Last 5 Mins)

Sensor Type: temperature

Average: 20.55

Minimum: 19.59

Maximum: 21.41

Readings in Window: 30

Last Calculated: 5/1/2025, 7:33:00 PM

Sensor Type: humidity

Average: 51.49

Minimum: 48.27

Maximum: 55.77

Readings in Window: 30

Last Calculated: 5/1/2025, 7:33:00 PM

Logs in cloudwatch:

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar includes sections for CloudWatch, Favorites and recents, AI Operations (Preview), Alarms, Logs, and Log groups (New). The main area displays log events for the Lambda function 'CalculateSensorAnalytics' on May 2, 2025. The log entries show various requests and responses related to sensor data processing.

Timestamp	Message
2025-05-02T02:34:00.179Z	Updating analytics for humidity: Avg=52.76, Min=48.35, Max=57.17
2025-05-02T02:34:00.238Z	END RequestId: ad68142f-12cf-48e9-94b5-f427a244dc68
2025-05-02T02:34:00.238Z	REPORT RequestId: ad68142f-12cf-48e9-94b5-f427a244dc68 Duration: 142.32 ms Billed Du...
2025-05-02T02:35:00.091Z	START RequestId: ad68142f-4ecf-48e9-94b5-f427a244dc68 Version: \$LATEST
2025-05-02T02:35:00.092Z	Calculating stats for data since timestamp: 1746153000092
2025-05-02T02:35:00.120Z	Found 58 items in the time window.
2025-05-02T02:35:00.120Z	Updating analytics for temperature: Avg=20.81, Min=19.59, Max=22.48
2025-05-02T02:35:00.179Z	Updating analytics for humidity: Avg=53.79, Min=48.66, Max=57.41
2025-05-02T02:35:00.220Z	END RequestId: ad68142f-4ecf-48e9-94b5-f427a244dc68
2025-05-02T02:35:00.220Z	REPORT RequestId: ad68142f-4ecf-48e9-94b5-f427a244dc68 Duration: 128.83 ms Billed Du...

Github URL:

<https://github.com/ananya101001/aws-iot-dashboard>