# ♥Serverless IOT Data Processing♥

➢

➢ Phase 5: submission document

❖

❖ Cloud Application Development

# Project Title: Serverless IoT Project with Amazon Voice Service Integration

**Objective Outline:**

- **Objective:** Develop a serverless IoT application that integrates with Amazon Voice Service (AVS).
- **Key Results:**
- IoT devices can understand and respond to voice commands.
- AVS integration enables natural language processing and voice recognition.
- **Data Ingestion and Processing:**
- **Objective:** Implement data ingestion from IoT devices and processing to prepare data for AVS.
- **Key Results:**
- IoT data is collected securely and reliably.
- Data processing converts IoT device data into a format suitable for AVS interaction

## ➢ AVS Integration:

- ➢ **Objective:** Integrate with AVS for voice recognition and natural language understanding.

- ➢ **Key Results:**
  - ➢ IoT devices can send voice commands to AVS.
  - ➢ AVS provides accurate voice recognition and understanding.

## ➢ Voice Commands and Responses:

- ➢ **Objective:** Enable IoT devices to respond to voice commands and provide relevant information or actions.

- ➢ **Key Results:**
  - ➢ IoT devices can receive and process voice commands.
  - ➢ Devices can provide appropriate responses or execute actions based on voice input.

## ➢ Serverless Computing

➢ **Objective:** Utilize serverless technologies for efficient and scalable data processing and AVS interactions.

➢ **Key Results:**

➢ Serverless functions handle data processing.

➢ Autoscaling and cost-effective resource allocation are achieved.

## ➢ Security and Privacy:

➢ **Objective:** Ensure the security and privacy of voice data and interactions with AVS.

➢ **Key Results:**

➢ Implement encryption and access control for voice data.

➢ Comply with Amazon's AVS security guidelines and best practices.

## ➢ User Interface (Optional):

➢ **Objective:** Develop a user interface for users to interact with IoT devices using voice commands.

➢ **Key Results:**

➢ Create a user-friendly interface for voice interactions.

➢ Enable users to set up and manage IoT devices through the interface.

## ➢ Testing and Quality Assurance:

➢ **Objective:** Conduct thorough testing to ensure the project's reliability and accuracy.

➢ **Key Results:**

➢ Successful testing of voice commands and device responses.

➢ Identify and resolve any issues or discrepancies

## ➢ Documentation:

- **Objective:** Create comprehensive documentation for the project.
- **Key Results:**
- Detailed project documentation for developers, users, and administrators.
- Maintenance and troubleshooting guides.

## Deployment and Scalability:

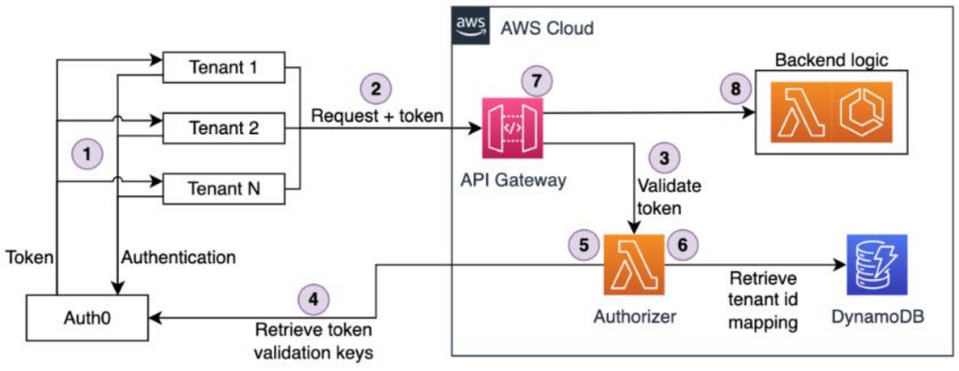
- **Objective:** Deploy the application to a production environment and ensure it can scale to handle increasing voice interactions.
- **Key Results:**
- Successful deployment to a production environment.
- Demonstrated ability to scale resources as needed.
- **Cost Management:**
- **Objective:** Monitor and optimize costs associated with AVS integration and serverless IoT operations.
- **Key Results:**
- Implement cost-saving measures without compromising functionality.

- **Compliance and Regulations:**

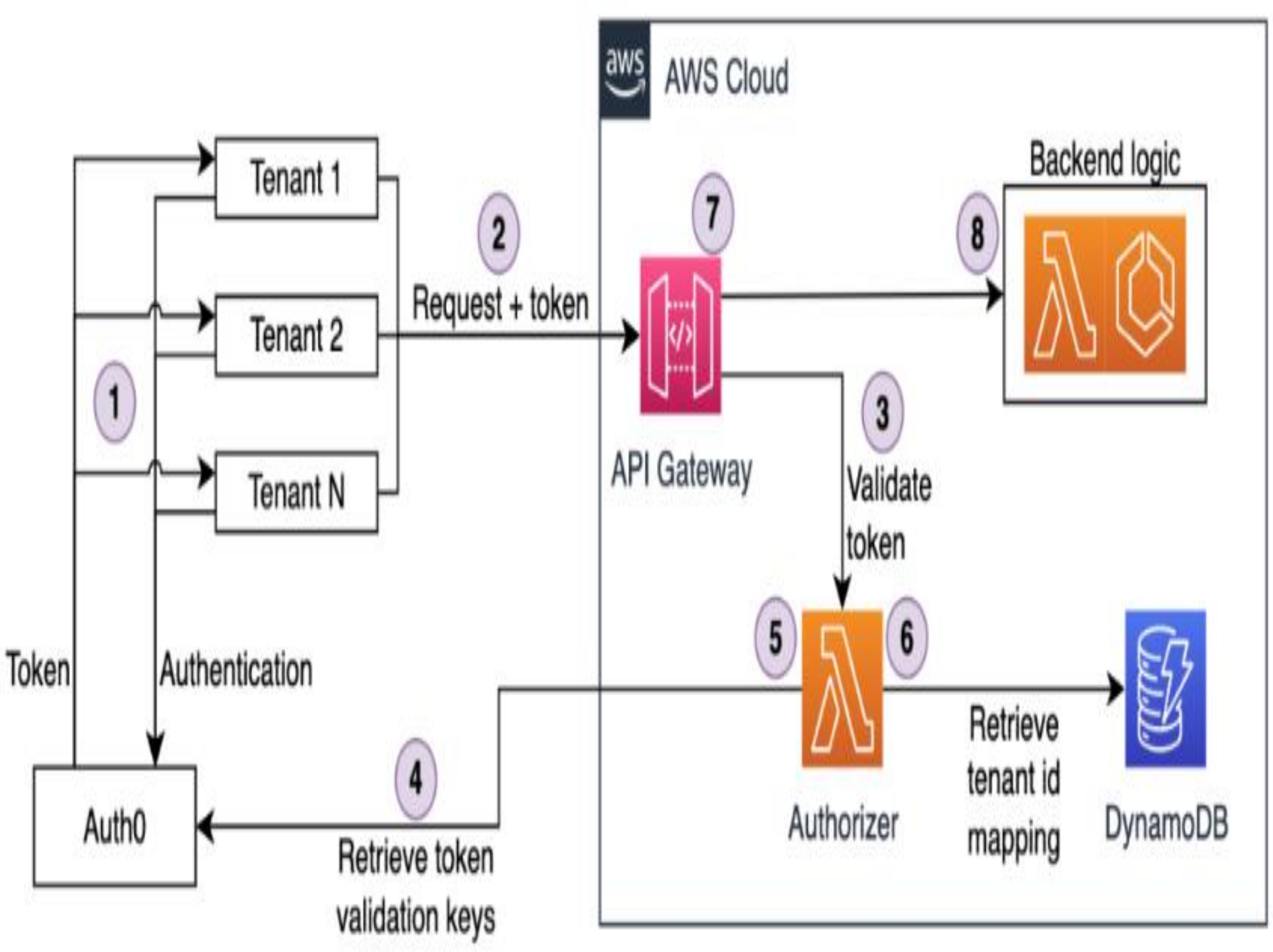
- **Objective:** Ensure the project complies with relevant regulations and data privacy laws, particularly in the context of voice data.
- **Key Results:**
- Document compliance measures and practices.
- **User Training and Support:**

- **Objective:** Provide training and support to end-users on using voice commands with IoT devices.
- **Key Results:**
- Develop user training materials and support channels.

- **Project Completion:**
- **Objective:** Successfully complete the project with all objectives met.
- **Key Results:**
- A fully functional serverless IoT application integrated with AVS.
- Project sign-off and handover to maintenance and support teams.



## Design of AWS cloud Serverless



**1. IoT Devices:**

- AVS-enabled IoT devices like smart speakers, voice-controlled gadgets, etc., collect voice and other sensor data.

**2. Data Ingestion:**

- Set up an IoT data ingestion service that can receive and securely store AVS data streams. AWS IoT Core, Azure IoT Hub, or Google Cloud IoT Core are popular choices. These services can handle device authentication, data routing, and provide MQTT or HTTP interfaces for data ingestion.

**3. Data Transformation:**

- Use AWS Lambda, Azure Functions, or Google Cloud Functions to process and transform incoming AVS data streams. This can include data parsing, validation, and initial processing.

**4. Data Storage:**

- Store your processed AVS data in a scalable and cost-effective data store. Options include:
- Amazon S3 for raw voice recordings.
- A NoSQL database like Amazon DynamoDB, Azure Cosmos DB, or Google Cloud Firestore for metadata, device information, and processed data.

- Data warehousing solutions like Amazon Redshift, Azure Synapse Analytics, or Google BigQuery for analytical queries and reporting.

**5. Real-time Data Processing:**

- Implement real-time processing of AVS data using serverless compute services like AWS Lambda, Azure Functions, or Google Cloud Functions. These functions can process and analyze data as it arrives for real-time insights or to trigger immediate actions.

**6. Batch Data Processing:**

- Perform batch processing of AVS data for historical analysis, reporting, and long-term storage. You can use AWS Glue, Azure Data Factory, or Google Cloud Dataflow to process large volumes of data periodically.

**7. Voice Transcription and NLP:**

- If required, use services like AWS Transcribe, Azure Speech to Text, or Google Cloud Speech-to-Text to transcribe voice data into text. Apply Natural Language Processing (NLP) services to extract insights, entities, and sentiments from the transcribed text.

**8. Security and Compliance:**

- Implement robust security measures to ensure the confidentiality, integrity, and availability of AVS data. Use IAM roles, encryption, and access control policies to protect data. Ensure compliance with privacy regulations such as GDPR and HIPAA if applicable.

**9. Monitoring and Alerting:**

- Set up monitoring and alerting using services like AWS CloudWatch, Azure Monitor, or Google Cloud Monitoring to keep an eye on the health and performance of your serverless functions and data processing pipeline.

**10. Reporting and Visualization:**

- Use tools like Amazon QuickSight, Power BI, or Data Studio to create dashboards and reports for visualizing insights and analytics derived from AVS data.

**11. Continuous Integration/Continuous Deployment (CI/CD):**

- Implement CI/CD pipelines to automate deployment, testing, and scaling of your serverless functions and data processing workflows.

**12. Scaling:**

- Serverless platforms automatically scale based on the load, but you should configure appropriate scaling triggers and limits to ensure efficient resource utilization.

## Architecture benefits

The architecture implemented by SeatGeek provides several benefits:

- Centralized authorization: Using Auth0 with API Gateway and Lambda authorizer allows for standardization the API authentication and removes the burden of individual applications having to implement authorization.

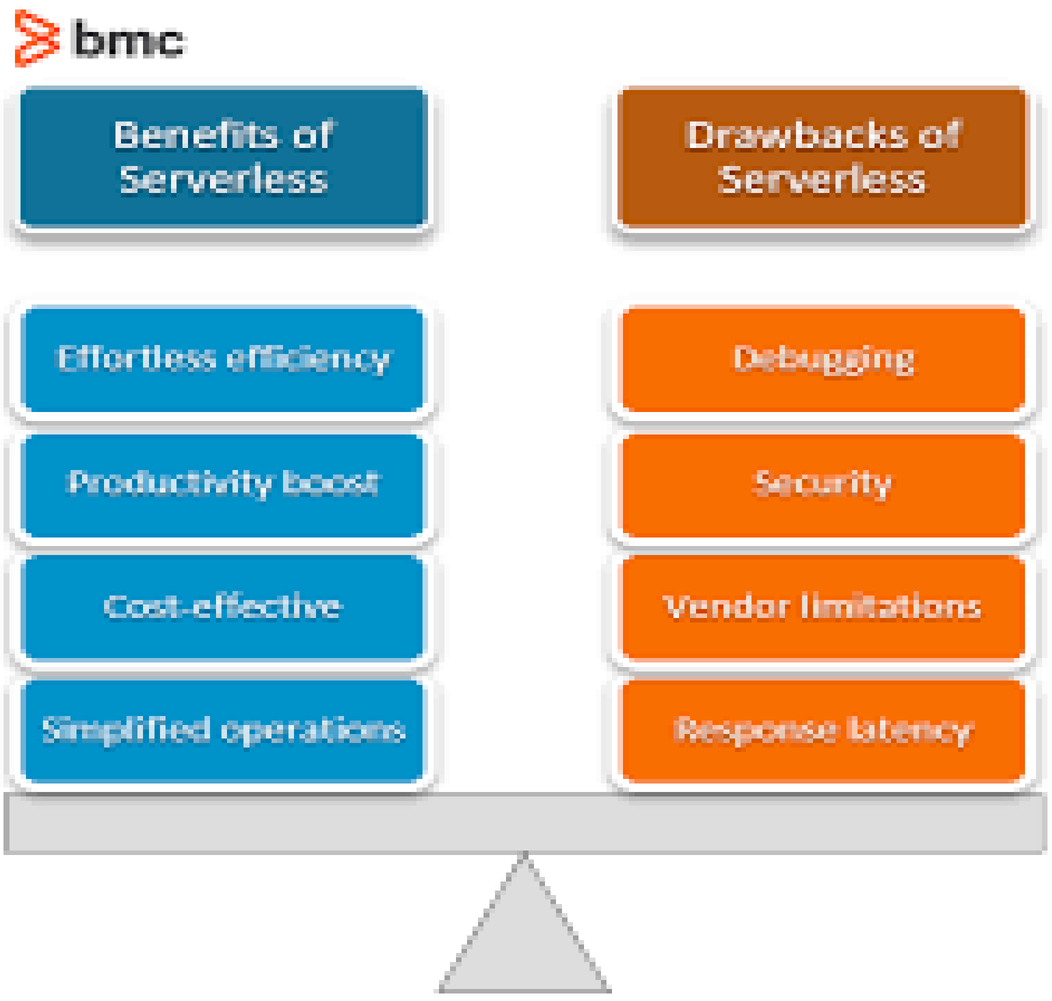
- Multiple levels of caching: Each Lambda authorizer launch environment caches token validation keys in memory to validate tokens locally. This reduces token validation time and helps to avoid excessive traffic to Auth0. In addition, API Gateway can be configured with up to 5 minutes of caching for Lambda authorizer response, so the same token will not be revalidated in that timespan. This reduces overall cost and load on Lambda authorizer and DynamoDB.

- Noisy neighbor prevention: Usage plans and rate limits prevent any particular tenant from monopolizing the shared resources and causing a negative performance impact for other tenants.

- Simple management and reduced total cost of ownership: Using AWS serverless services removed the infrastructure maintenance overhead and allowed SeatGeek to deliver business value faster. It also ensured they didn’t pay for over-provisioned capacity, and their environment could scale up and down automatically and on demand.

➢

➢



## Platform Features

- **Scalability**: Serverless platforms, such as AWS Lambda, Azure Functions, and Google Cloud Functions, can automatically scale to handle varying workloads. This is crucial for IoT applications where data volumes can fluctuate significantly.
- **Cost-Efficiency**: With serverless computing, you only pay for the compute resources you use. This "pay-as-you-go" model can be more cost-effective for IoT data processing, especially when dealing with sporadic or bursty data streams.

- **Event-Driven**: Serverless functions are triggered by events, making them ideal for processing IoT data generated by sensors, devices, or applications. You can set up triggers based on specific events, such as new data arriving, and process that data in real-time.
- **Automatic Scaling**: Serverless platforms automatically provision and manage resources, eliminating the need for manual scaling. This ensures that your IoT data processing remains responsive without manual intervention.
- **Quick Deployment**: Serverless functions can be deployed quickly and easily, allowing you to respond rapidly to changes in your IoT data sources or processing requirements.
- **Zero Server Management**: In a serverless architecture, you don't need to worry about server maintenance or infrastructure management. This reduces operational overhead and allows you to focus on developing your IoT data processing logic.
- **Support for Various IoT Data Sources**: Serverless platforms can integrate with a wide range of IoT data sources, including MQTT, HTTP endpoints, message queues, and more. This flexibility enables you to ingest and process data from diverse IoT devices.
- **Pre-built Integrations**: Leading cloud providers offer pre-built integrations with IoT services, databases, analytics tools, and more. This simplifies the development of end-to-end IoT data processing pipelines.
- **Stateless and Isolated**: Serverless functions are typically stateless and isolated from one another. This isolation helps prevent issues caused by one function affecting others, making your IoT data processing more resilient.
- **Fine-Grained Security**: Serverless platforms provide fine-grained security controls, allowing you to restrict access to your IoT data and functions. This is essential for protecting sensitive information and ensuring data privacy and compliance.

- **Serverless Ecosystem**: A rich serverless ecosystem exists, which includes tools and libraries for common IoT data processing tasks, as well as monitoring, logging, and debugging capabilities.
- **Automatic Load Balancing**: Serverless platforms handle load balancing automatically, distributing incoming requests across multiple instances to ensure high availability and responsiveness.
- **Reduced Development Time**: Serverless simplifies the development process by abstracting infrastructure management. This can lead to faster development cycles for IoT data processing applications.
- **Logging and Monitoring**: Serverless platforms offer built-in logging and monitoring tools, making it easier to track the performance of your IoT data processing functions and troubleshoot issues.
- **Global Reach**: Leading cloud providers have serverless offerings available in multiple regions, allowing you to deploy IoT data processing functions close to the data source or target audience, improving latency and data locality.

## Exaple of Sample

```
/* ESP32 AWS IoT

*

* Simplest possible example (that I could come up with) of using an ESP32 with AWS IoT.

*

* Author: Anthony Elder

* License: Apache License v2

* Sketch Modified by Stephen Borsay for www.udemy.com

* https://github.com/sborsay

* Add in Char buffer utilizing sprintf to dispatch JSON data to AWS IoT Core

* Use and replace your own SID, PW, AWS Account Endpoint, Client cert, private cert, x.509 CA root Cert

*/

#include <WiFiClientSecure.h>

#include <PubSubClient.h> // install with Library Manager, I used v2.6.0
```

```
const char* ssid = "<YOUR-SSID>";

const char* password = "<YOUR-PASSWORD>";


const char* awsEndpoint = "YOURACCOUNT-
ats.iot.yourregion.amazonaws.com";


// Update the two certificate strings below. Paste in the text of your AWS

// device certificate and private key. Add a quote character at the start

// of each line and a backslash, n, quote, space, backslash at the end

// of each line:


// xxxxxxxxxx-certificate.pem.crt

const char* certificate_pem_crt = \


"-----BEGIN CERTIFICATE-----\n" \
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxs+8i8TUiMA0GCSqGSIb3DQEB\n" \
"CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t\n" \
"IEluYy4gTD1TZWF0dGxlIFNUPVdhc2hpbmd0b24gQz1VUzAeFw0xOTExMDYxODI1\n" \
"NDFaFw00OTEyMzEyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBJb1QgQ2VydGlmaWNh\n" \
"dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDRSWA/2xSg/OYOOM6d\n" \
"2smJIEn3VIgmqCEmrg1+6vdoKBLxgMh194z2Ns83siRG9GPf7+v5oWk0Bu9kDKNl\n" \
"vquAnBO3+eW1Sgg3JE9SP5utExYGDXud1im6dlG/YbnN8gWCG3W0Abx1vfsiJMP7\n" \
"yhez8Xp4lb+/fSDH/vbi5IqYLsCsUgSB7hrg0a8zqXa16lq5FjTgUk9CxFDb5V+z\n" \
"ipXLfcXboisLIAjJMhNXnef+CpM4rQJulf1eZxCV3P9Du8eFpGKx1VFm8/D1pwrh\n" \
"GJ1N9kFuSWfQHhj+gA383OL7andGE9h2097O4KaqXW9coCZVKv3AjX2WWqczL3uV\n" \
"6AOdAgMBAAGjYDBeMB8GA1UdIwQYMBaAFOwXufzDBIb6LGYzX9hsuuTkGBggMB0G\n" \
"A1UdDgQWBBQRfgU2R8TDTFz6lgUra5M/9pFn6jAMBgNVHRMBAf8EAjAAMA4GA1Ud\n" \
"DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAB4tmHWDNBu74BDIn+f32c1ED\n" \
"r+HJjZAVWBwG9v7ubFu/uA+TaHYT+KBaAfB4NfAlOmTpMZN6egLkKfRNNDIA+tAH\n" \
"PV/QBDZWjCR3YS5sAUIrnSSRzFGRtAFtscYJhJ73Ahhob8H6zXyH2XQRft4663dI\n" \
"fqMlXsofXS7QH/mhy1zv13su8EaS/UNJRMvB/+ESyEwkQ2BhzpE8TjaBXTkyZuKw\n" \
"wSBoHO71UqJGhmLkDLp99hKc90KWh08v9jmNBfbTZvlUcCsIz6EhcbfUAg8oBSE0\n" \
"yuzWG1P4ZwUk70SDcHhDty8GZ9oajrCsLTcv7zrw4aUtfuIFOqXLybqVOCaGOw==\n" \
"-----END CERTIFICATE-----\n";

// xxxxxxxxxx-private.pem.key
const char* private_pem_key = \

"-----BEGIN RSA PRIVATE KEY-----\n" \
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxJ91SIJqghJq4Nfur3aCgS8YDI\
n" \
"dfeM9jbPN7IkRvRj3+/r+aFpNAbvZAyjZb6rgJwTt/nltUoINyRPUj+brRM
WBg17\n" \
"ndYpunZRv2G5zfIFght1tAG8db37IiTD+8oXs/F6eJW/v30gx/724uSKmC
7ArFIE\n" \
"ge4a4NGvM6l2tepauRY04FJPQsRQ2+Vfs4qVy33F26IrCyAIyTITV53n/g
qTOK0C\n" \
"bpX9XmcQldz/Q7vHhaRisdVRZvPw9acK4RidTfZBbkln0B4Y/oAN/Nzi+2
p3RhPY\n" \
"dtPezuCmql1vXKAmVSr9wI19llqnMy97legDnQIDAQABAoIBAAKo/Rkyrq
txK3do\n" \
"z2+ANWmRyH7lSym6n1k/gxpm4CgpwjvhmCqvr9H9Vlrt37orJw3Undo5a/3mTKqn\n" \
"4nfLmaBz22hOO9Y3D62rv3pbJp5njLMc/sI905/0GwvksRjmB53kvXVBRGT2ujdv\n" \
"4jHUGsMFwbnApYRIkd8Y5YhywTQ39gQeC30suTikIlqSbXnT5KJkKcKvsfnAbsx0\n" \
"bp+7grEYZjkmki7f7KJgJpVWOQBg8JhbRHBFMiIsMQORPDI+NfaHtgUKxBKykCju\n" \
"gkkf7ljslfMrfLJvB5gGJetoc/ilLS+19O6VZVN3bChpL4OKE+WnsOdjMLPzmfWL\n" \
"fIoYiAECgYEA9TpqOwUbWuFjAXIU+lTjn/jJCmnkvX3S0tJ9AXxkZBzfN0BlDlMC\n" \
"XkEfaARLqj05YbvwfikmnXbER9hE3TjHh8uq+Pv8ZzK432X9/kTLuNKksan3sDgP\n" \
"e94sQmKrNm5cTw2K9fMsk04W9UdkQGG2duZvOINW8iPqtfNy0jj6rZ0CgYEA2nrM\n" \
"jmoC4MTuhtEDaishocVH5Q3MVCrZWh1QELgu+d7XUFuvpByT1Dw68PAdlcUjQa8t\n" \
"+1jopKHGEaZ3781dSkFL6fCcpOn4bGBvNKzURt6avj9r4WoK6tzLqymZbr+32eNc\n" \
"vXgui3fz6V7fZDtE03ixw62qEgLJF+08yhFNzgECgYB7WW+30kDJPNe1EXI13N9G\n" \
"RziwsUUqf5C9FL1mMvC4XsF0pEJwqxZk3LL0ejypG/SyEXvNqdtPlz7xuHojIH1U\n" \
"9ABDD1UNf7j4PfA9ptMmW2YWK514GSrIrp9qoQDn9ykdZn2Aa1n/mmb353oo3D2Q\n" \
"nyZQsdfZInHcJeGalqiZDQKBgQDZ6N0ZtlbEhOc7hEEZpYdX6IL6zLZdxJchMFdp\n" \
"Nat1AXRT6/7VesNfTeuj4HpBpWyy2NzN8zGm8suxUw4RGg3QQCxNWvWB7vMedVi1\n" \
"eyQGw4Qn+O3K2I+nDS3+u/ES6xmesw0O1U3nQW7/9uZs/Eh7e8gh2Zble+3CdZbK\n" \
"HFQ0AQKBgQCQWrO4PaQQIZ/ZPi9/lKZLRHAb30gx1Qpgpv2pbpxVyIKRfOdDvpHg\n" \
"Fp+icoIMvHcPoJPz13QYujADzmKpLviKYXDDojJ6sCrPmieV0wAEsF19zbOkMZgr\n" \
"guGf0uCCv7CpEoRvaVUAX12/Kp9iA03b5hc3hTqBTMSKirdRlouugA==\
n" \
"-----END RSA PRIVATE KEY-----\n";

/* root CA can be downloaded in:

https://www.symantec.com/content/en/us/enterprise/verisign/roots/Ve
riSign-Class%203-Public-Primary-Certification-Authority-G5.pem
*/
const char* rootCA = \
"-----BEGIN CERTIFICATE-----\n" \
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxyjANBgkqhkiG9w0BAQsF
\n" \
"ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkw
FwYDVQQDExBBbWF6\n" \
"b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExN
zAwMDAwMFowOTEL\n" \
"MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
"b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
"ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
"9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
"IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
"VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
"93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
"jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
"AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
"A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX
9rbxenDI\n" \
"U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ
1wUs\n" \
"N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWl
JbYK8U90vv\n" \
"o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZ
Fo6IQ6XU\n" \
"5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQde
XeGADbkpy\n" \
"rqXRfboQnoZsG4q5WTP468SQvvG5\n" \
"-----END CERTIFICATE-----\n";

WiFiClientSecure wiFiClient;
void msgReceived(char* topic, byte* payload, unsigned int len);
PubSubClient pubSubClient(awsEndpoint, 8883, msgReceived, wiFiClient);
```

```
void setup() {

Serial.begin(115200); delay(50); Serial.println();

Serial.println("ESP32 AWS IoT Example");

Serial.printf("SDK version: %s\n", ESP.getSdkVersion());


Serial.print("Connecting to "); Serial.print(ssid);

WiFi.begin(ssid, password);

WiFi.waitForConnectResult();

Serial.print(", WiFi connected, IP address: "); Serial.println(WiFi.localIP());


wiFiClient.setCACert(rootCA);

wiFiClient.setCertificate(certificate_pem_crt);

wiFiClient.setPrivateKey(private_pem_key);

}


unsigned long lastPublish;
```

```
int msgCount;

void loop() {

pubSubCheckConnect();

//If you need to increase buffer size you need to change
MQTT_MAX_PACKET_SIZE in PubSubClient.h

char fakeData[128];

float var1 =  random(55,77); //fake number range, adjust as you like

float var2 =  random(77,99);

sprintf(fakeData,
"{\"uptime\":%lu,\"temperature\":%f,\"humidity\":%f}", millis() / 1000,
var1, var2);
```

```
if (millis() - lastPublish > 10000) {

//String msg = String("Hello from ESP32: ") + ++msgCount;

// boolean rc = pubSubClient.publish("outTopic", msg.c_str());

boolean rc = pubSubClient.publish("outTopic", fakeData);

Serial.print("Published, rc="); Serial.print( (rc ? "OK: " : "FAILED: ") );

Serial.println(fakeData);

lastPublish = millis();




}

}



void msgReceived(char* topic, byte* payload, unsigned int length) {

Serial.print("Message received on "); Serial.print(topic); Serial.print(": ");

for (int i = 0; i < length; i++) {

Serial.print((char)payload[i]);
}

Serial.println();

}


void pubSubCheckConnect() {

if ( ! pubSubClient.connected()) {

Serial.print("PubSubClient connecting to: "); Serial.print(awsEndpoint);

while ( ! pubSubClient.connected()) {

Serial.print(".");

pubSubClient.connect("ESPthingXXXX");

delay(1000);

}

Serial.println(" connected");

pubSubClient.subscribe("inTopic");

}

pubSubClient.loop();
```

```
}
```

THANK YOU