**DOMAIN NAME: CLOUD APPLICATION DEVELOPMENT**

**PROJECT NAME: TRANSFORM YOUR HOME INTO A SMART**

**LIVING SPACE USING IBM CLOUD FUNCTION FOR IOT DATA PROCESSING.**

**Phase 3 Submission Document**

# BASIC COMPONENTS USED IN HOME AUTOMATION:

# Smart lightning.

# 

# Smart thermostat.

# Smart lock.

# Smart security cameras.

# Speaker and voice assistant.

# Smart irrigation system.

# Smart garage door openers.

# Smart water leak detectors.

# Smart Garden system.

# Smart entertainment system.

# Smart Appliances.

# ARCHITECTURE OF HOME AUTOMATION:

# 

# An application was designed for the management and control of smart objects. The application is developed using AWS Lambda and the MQTT protocol to provide flexible interaction between the users and the extractor fan. Amazon Alexa is used to implement a vocal interface for the control of the smart object. State updates are also sent by the fan using the custom commands protocol and are processed by a dedicated Lambda function on the cloud to keep track of the state of each device.

# The application was developed on AWS, and the serverless platform of choice was Amazon Lambda. Two Lambda functions were implemented in NodeJS (JavaScript) using the AWS SDK: one to forward user commands to devices, either by an application or Alexa skill, and the other to handle state updates from the devices. JavaScript, being a high-level interpreted language, lends itself quite naturally to the development of serverless or FaaS applications and, alongside Python, is one of the most used languages for the development of serverless applications [3]. Lambda functions comprised one or more handlers that are used to manage triggers, which can be received by various outside services, including Amazon Alexa or AWS IoT core. Each handler was defined on the basis of the custom command protocol and of the intents that were implemented in the Alexa skill. This way, users are able to control and monitor all the admissible states of the devices.

# The project uses two AWS storage services: Amazon S3 and Amazon DynamoDB. DynamoDB is used to store devices states and devices and user identifiers. Similarly to the approach used in [61] to store Lambda code, in this project, Amazon S3 is used to store the device firmware. This way, each device can download any firmware version at any moment, greatly simplifying software updates and the deployment of new code. A software update can be initiated by sending the url of the S3 bucket containing the new firmware inside an MQTT command. The device will then download the file from the S3 bucket and execute the firmware update in autonomy.

# An Alexa skill has been developed to interact with the devices. The endpoint of the skill is Amazon Lambda. A Lambda function is triggered when a command is invoked by the user. The Alexa skill takes care of translating input vocal commands into text and then translates text into JSON inputs that can be parsed by Lambda functions. Each JSON input produced by the Alexa skill also contains a user identifier that is used by Lambda to retrieve (from DynamoDB) the device ID associated with the user that invoked the skill. This mechanism guarantees that Lambda functions are only able to control the fans corresponding to the user that invoked the Alexa skill from their personal account. Custom slots are used to define a series of extra identifiers to distinguish between different fans when a user owns or controls more than a single fan. Figure 3 illustrates the message flow of the application. Green boxes indicate vocal commands and responses, yellow boxes are messages in JSON format generated by the Alexa skill, and the pink box indicates the payload of the MQTT message generated by the AWS application and forwarded to the device.

# 

# 

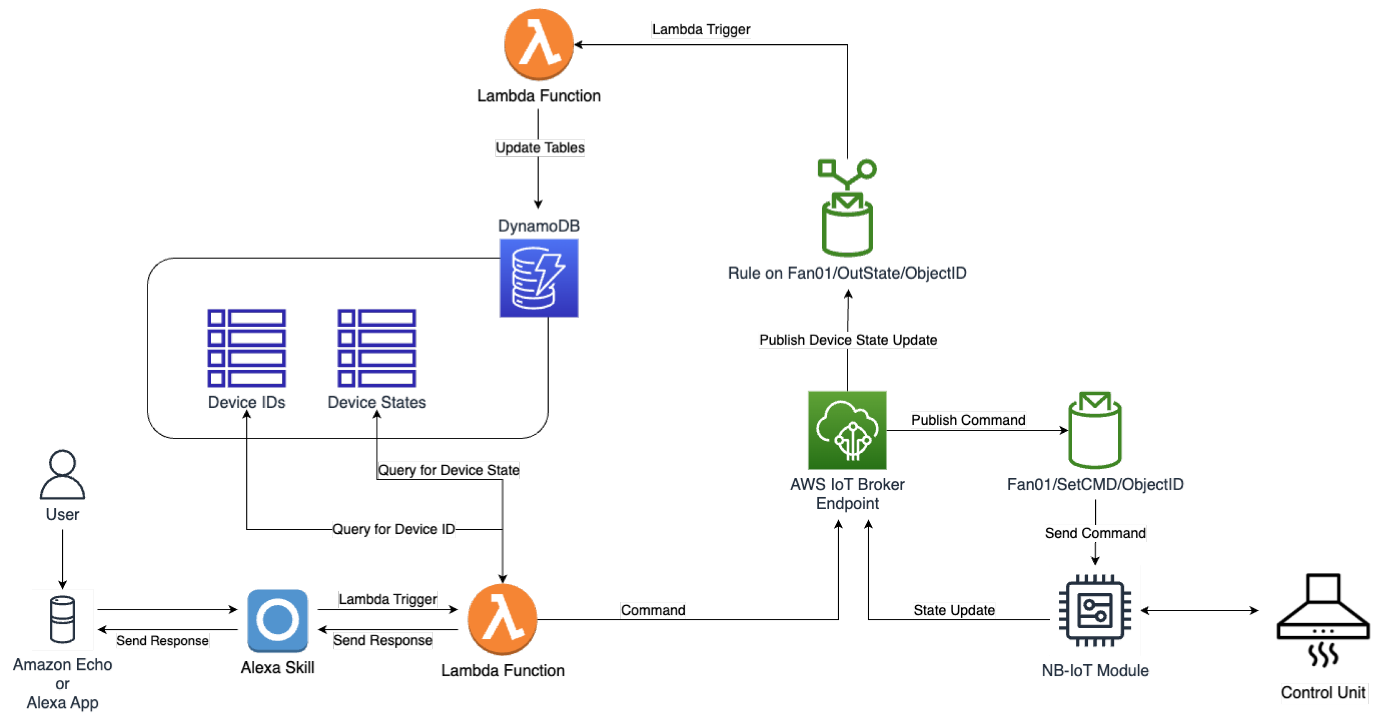
# 

**IoT Devices and Sensors:**

These are the physical devices and sensors installed throughout your home to collect data. Examples include motion detectors, temperature sensors, light sensors, smart locks, and more.

**Cloud IoT Platform:**

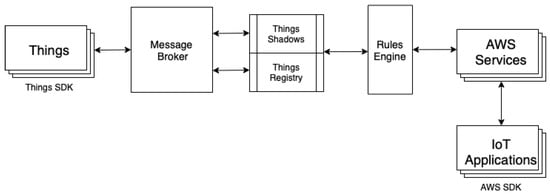
A cloud-based IoT platform like AWS IoT Core, Google Cloud IoT, or Azure IoT Hub can be used to manage and securely connect your IoT devices to the cloud. This platform handles device registration, authentication, and message routing.



**Amazon Web Services:**

Amazon Web Services is one of the most famous and oldest cloud platforms, offering services ranging from databases and data analytics to internet of things and mobile computing, besides many services dedicated to enterprises [57]. The smart appliance system proposed in this paper makes use of AWS IoT core, AWS Lambda, and two data storage services, AWS S3 and DynamoDB. Furthermore, Amazon Cloudwatch, a service used to monitor AWS resources and applications, is used to keep track of the performance and cost of each deployed service.

**AWS IoT Core:**

AWS IoT core is the AWS platform for the internet of things. It allows devices to connect to the cloud and other AWS services. The architecture of AWS IoT core is described in Figure 1. The central component of the architecture is the message broker, which dispatches messages to and from things (i.e., IoT devices) and other AWS services and user applications. AWS IoT core supports MQTT and MQTT over Websockets, but clients can also publish messages on the broker using HTTP. A rules engine is available to automatically process messages and to generate triggers. Each IoT device has an associated “shadow” to keep track of its status, and shadows are stored in a corresponding registry. Various security and identity mechanisms are employed to guarantee secure communication between the broker and devices, including TLS data encr 

**Architecture of AWS IoT core**

**AWS Lambda:**

AWS Lambda is a serverless computing platform that lets users deploy applications and run code without managing servers, delegating to Lambda all the server and operating system maintenance, including capacity provisioning and automatic scaling of resources, while leaving the code as the sole responsibility of the developer. Lambda can be used to build IoT backends and countless other applications. Lambda functions are event-driven, meaning that they are instantiated to manage events. Each function is usually linked to an event source that generates triggers and will contain a series of handlers to manage those triggers. Various AWS services are event sources and can be configured to trigger Lambda functions, including Amazon Alexa skills and AWS IoT core rules.

**Amazon Alexa:**

Amazon Alexa skills employ ASR and natural language understanding technology [60] for the development of custom applications that take vocal commands as input, convert them into text, interpret them to execute various actions, and possibly return a vocal response to the user that invokes the application. Alexa skills can be configured to send messages to various endpoints, including Lambda functions, essentially turning vocal commands into Lambda triggers.

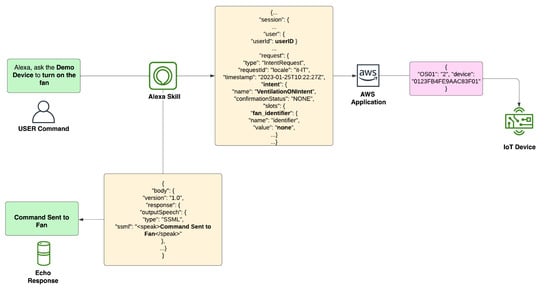
The components of an Alexa skill invocation are the following:

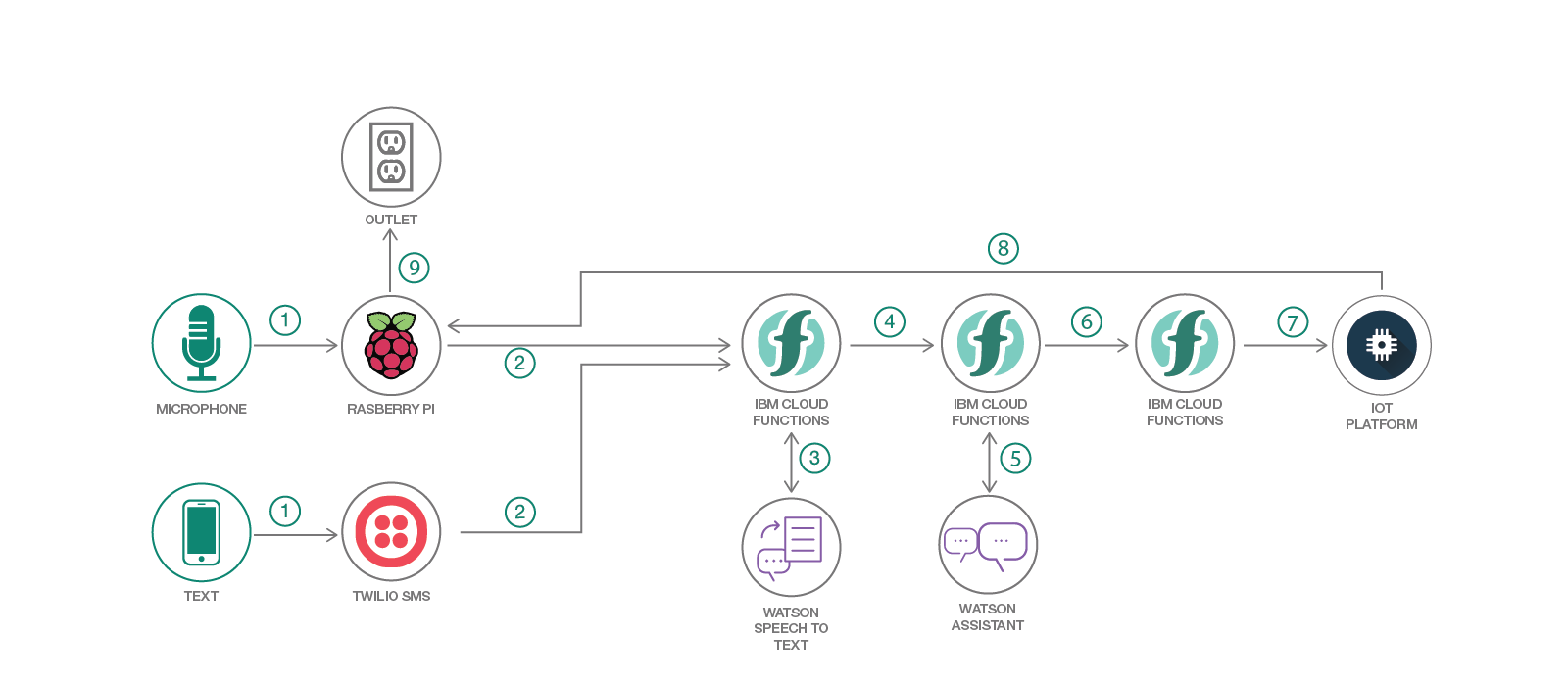
A wake word, such as “Alexa”, to turn on the device;

An invocation name, the de facto name of the Alexa skill that is being invoked;

An utterance, telling the Alexa skill what action to execute, or, in Alexa skill jargon, which “intent” is to be called;

Optionally, a series of custom “slots”, which are additional variables that are being passed to the intent.

Intents produce a message in JSON format that is then forwarded to a Lambda endpoint. The Lambda function will parse the JSON message content, call a dedicated handler based on the intent, execute the handler’s routine, and lastly produce a JSON response message that is sent back to the Alexa ski 



**CODING:**

#include <WiFiClientSecure.h>

#include <PubSubClient.h>

#include <DHT.h>

// Define your Wi-Fi credentials

const char\* ssid = "YourWiFiSSID";

const char\* password = "YourWiFiPassword";

// Define AWS IoT credentials

const char\* awsEndpoint = "your-endpoint.iot.us-west-2.amazonaws.com";

const char\* clientId = "SmartHomeDevice";

const char\* awsCertCA = "your-ca.pem.crt";

const char\* awsCertClient = "your-cert.pem.crt";

const char\* awsCertPrivateKey = "your-private.pem.key";

// Define AWS IoT topics

const char\* sensorTopic = "home/sensor";

const char\* actuatorTopic = "home/actuator";

// Define your sensor (e.g., DHT22)

DHT dht(D5, DHT22);

WiFiClientSecure net;

PubSubClient client(awsEndpoint, 8883, net);

void setup() {

Serial.begin(115200);

dht.begin();

setupWiFi();

connectAWS();

// Subscribe to the actuator topic to receive control commands

client.subscribe(actuatorTopic);

}

void loop() {

if (!client.connected()) {

connectAWS();

}

client.loop();

// Read sensor data

float temperature = dht.readTemperature();

float humidity = dht.readHumidity();

// Create a JSON payload with sensor data

String sensorData = "{\"temperature\":" + String(temperature) + ", \"humidity\":" + String(humidity) + "}";

// Publish sensor data to the AWS IoT topic

client.publish(sensorTopic, sensorData.c\_str());

delay(5000); // Publish sensor data every 5 seconds

}

void setupWiFi() {

Serial.println("Connecting to Wi-Fi");

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to Wi-Fi");

}

void connectAWS() {

Serial.println("Connecting to AWS IoT");

net.setCACert(awsCertCA);

net.setCertificate(awsCertClient);

net.setPrivateKey(awsCertPrivateKey);

while (!client.connect(clientId)) {

delay(1000);

Serial.println("AWS IoT Connection Failed");

}

Serial.println("AWS IoT Connected");

}

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

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

Serial.println(topic);

// Handle actuator control based on received payload

// Example: Parse JSON payload and control an actuator

// You can add your logic here

}