



# **VerBo-Dega**

*Proyecto: Fase 3*  
*Ricardo Menéndez*  
*201602916*



# Indice

<b>Visión</b>	<b>4</b>
<b>Misión</b>	<b>4</b>
<b>¿Que es?</b>	<b>5</b>
<b>Arquitectura</b>	<b>5</b>
Amazon Web Service	5
IoT Core	5
Capa Gratuita:	5
Lambda	5
Capa Gratuita:	5
APIGateway	6
Capa Gratuita:	6
DynamoDB	6
Capa Gratuita:	6
Cliente	6
ReactJS	6
Flujo de la Arquitectura	7
<b>Hardware</b>	<b>8</b>
Procesador	8
ESP32 - Modulo WIFI y Bluetooth	8
Modulos	8
MQ-2 (Sensor de Gases)	8
DHT-11 (Sensor de Humedad y Temperatura)	8
<b>Software</b>	<b>9</b>
Dispositivo	9
Cliente	13
Obtención de datos:	14
Graficas:	14
Iteración de un Arreglo:	15
<b>IoT Core</b>	<b>17</b>
¿Que es?	17
¿Como Funciona?	17
¿Como se usó en el proyecto?	24

# Visión

Volverse la primera opción para las empresas que utilizan almacenaje, siendo una solución para tener un fácil control de sus diferentes bodegas. De esta forma ser una vía para evitar pérdidas económicas por accidentes o deterioro del material almacenado.

# Misión

Ofrecer una solución integral para el control de bodegas o espacios de almacenamiento. Estas implementadas con diferentes dispositivos de IoT, que obtienen la información, y mostrar esta información en una aplicación web ergonómica y amigable.

# VerBo-Dega

## ¿Que es?

VerBo-Dega es una solución de IoT donde se ofrece uno o varios dispositivos que miden la humedad, temperatura, presencia de CO2, Gas Propano o Humo en una habitación. Estos dispositivos se comunican con varios servicios de AWS para poder guardar esta información, y poderla visualizar en un cliente web.

## Arquitectura

### Amazon Web Service

#### IoT Core

IoT Core es un servicio de AWS enfocado a la facil creación de dispositivos de IoT. Esto lo hace mediante la creación de “cosas” (así lo denomina el servicio), entonces a algún dispositivo le podemos asignar que sea esa cosa, y del mismo modo, asignar acciones, protocolos de comunicación y manejo de datos para esa “cosa”. Es la herramienta principal de este proyecto, por lo cual se explicara mas a detalle al final.

#### Capa Gratuita:

250,000 mensajes enviados desde IoT por mes, por 12 meses.

#### Lambda

Lambda es la solución de funciones Serverless que ofrece AWS, se utiliza esta en el proyecto para poder usar la información mandada desde IoT Core a la base de datos de DynamoDB, y para obtener información de la base de datos para dirigirla al cliente web.

En el proyecto tenemos una función que ingresa a la base de datos, y otra que regresa datos de esta misma.

#### Capa Gratuita:

1 millon de solicitudes gratuitas al mes. Por siempre. Todo esto restringido a Hasta 3,2 millones de segundos de tiempo de informática por mes.

## APIGateway

Este servicio sirve para crear métodos para crear APIs, y de una forma fácil, poderlos enlazar a otras funciones Https, Lambda, u otro protocolo. Luego nos permite hacer deploy esta colección de APIs.

Capa Gratuita:

1 millón de llamadas al mes, por los primeros 12 meses.

## DynamoDB

DynamoDB es la base de datos NoSQL que ofrece AWS. Se crea una llave primaria para la tabla, y luego ya puede recibir cualquier colección de datos asociadas a una llave.

Capa Gratuita:

25GB de almacenamiento, gratis para siempre!

## Cliente

### ReactJS

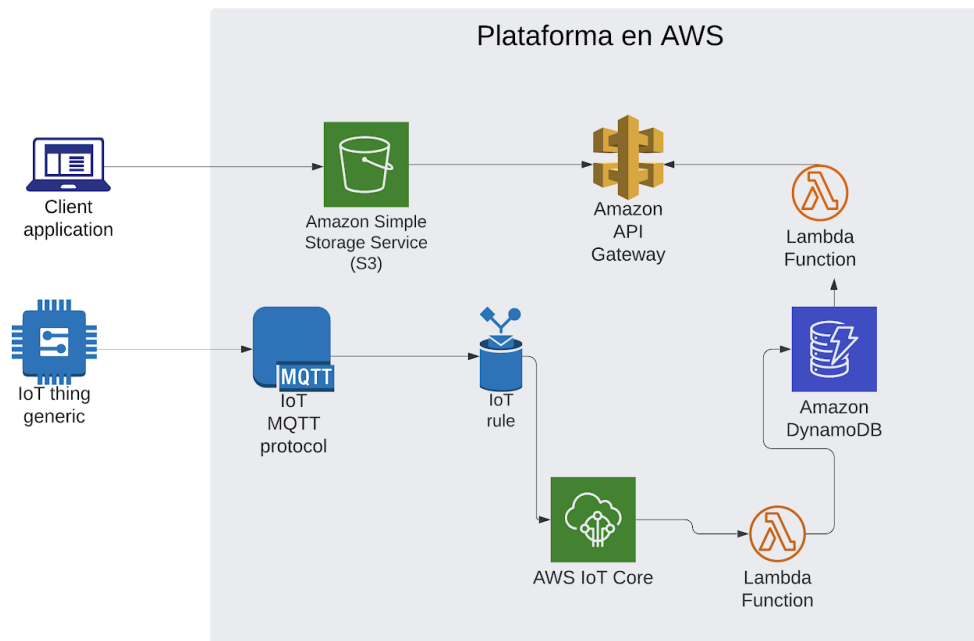
Se realizó una aplicación para mostrar los datos capturados desde el dispositivo IoT. Estos datos son:

- Temperatura
- Humedad
- Nivel de CO2
- Nivel de gas propano (LGP)
- Presencia de Humo

La temperatura y la humedad se ven representadas en gráficas a través del tiempo, mientras que las demás, solo se ve el último dato capturado.

Pero se pueden visualizar todas las entradas enviadas desde los dispositivos en una tabla debajo de los datos anteriormente mencionados.

## Flujo de la Arquitectura



La aplicación funciona de modo que cada 5 minutos, el dispositivo VerBodega envía un dato en formato JSON con las condiciones de la bodega. Esta por un protocolo MQTT llega a IoT Core, de modo que con una Regla de este mismo, se enlaza con una función Lambda. Esta función Lambda ingresa a DynamoDB la información del JSON.

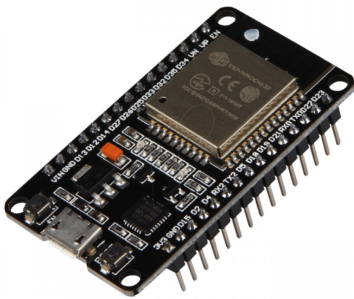
Por el otro lado, el cliente se conecta a una API la cual tiene enlazada una función Lambda conectada a la base de datos, que devuelve todos los registros de la base de datos.

# Hardware

## Procesador

### ESP32 - Modulo WIFI y Bluetooth

Tarjeta de desarrollo con Wi-Fi y Bluetooth LE , compatible con Arduino. Ideal para aplicaciones de IOT debido a su versatilidad y facilidad para programar.



**Microprocesador :** LX6 Tensilica (32 bits y núcleo dual)

**Voltaje de operación :** 2.2 a 3.6 V

**Corriente de Deep Sleep :** 2.5  $\mu$ A

**Memoria Flash :** 4 MB

**Frecuencia de Reloj :** hasta 240 MHz (en IDE solo hasta 80mhz)

**Cifrado con Hardware Acelerado :** (AES, SHA2, ECC, RSA-4096)

## Modulos

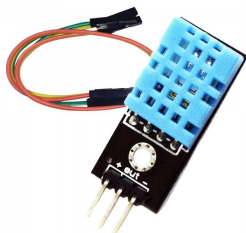
### MQ-2 (Sensor de Gases)



Este sensor es adecuado para detectar GLP, propano, metano, alcohol, hidrógeno, humo. Siendo más sensible al GLP y propano.

Este sensor no proporciona valores absolutos, sino que simplemente proporciona una salida analógica que debe ser monitoreada y comparada con los valores de umbral.

### DHT-11 (Sensor de Humedad y Temperatura)



El DHT11 es un sensor de temperatura y humedad digital de bajo costo. Utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de

datos (no hay pines de entrada analógica). Es bastante simple de usar, pero requiere sincronización cuidadosa para tomar datos.

## Software

### Dispositivo

La tarjeta de desarrollo ESP-32 se programa desde el IDE de Arduino. Esta de por sí tiene propiedades para su fácil conexión desde el **setup()** ingresando solo las credenciales de la red a la que queremos conectarnos.

Pero lo que deseamos es convertir este dispositivo en un dispositivo de IoT que se conecte con AWS IoT Core. Para esto necesitamos una librería, “**AWS\_IOT**”, esta es proporcionada desde la web de AWS IoT Core a la hora de crear un “cosa”, las credenciales de esta misma “cosa”, que las otorga AWS IoT Core al crear el dispositivo, y las librerías para utilizar los módulos MQ-2 y DHT-11.

Se deben definir un nombre para el dispositivo, y otro para la dirección de la “cosa” a la que nos vamos a conectar. Esta se da al momento de crear la cosa en IoT Core, y es una url “secreta”

HTTPS

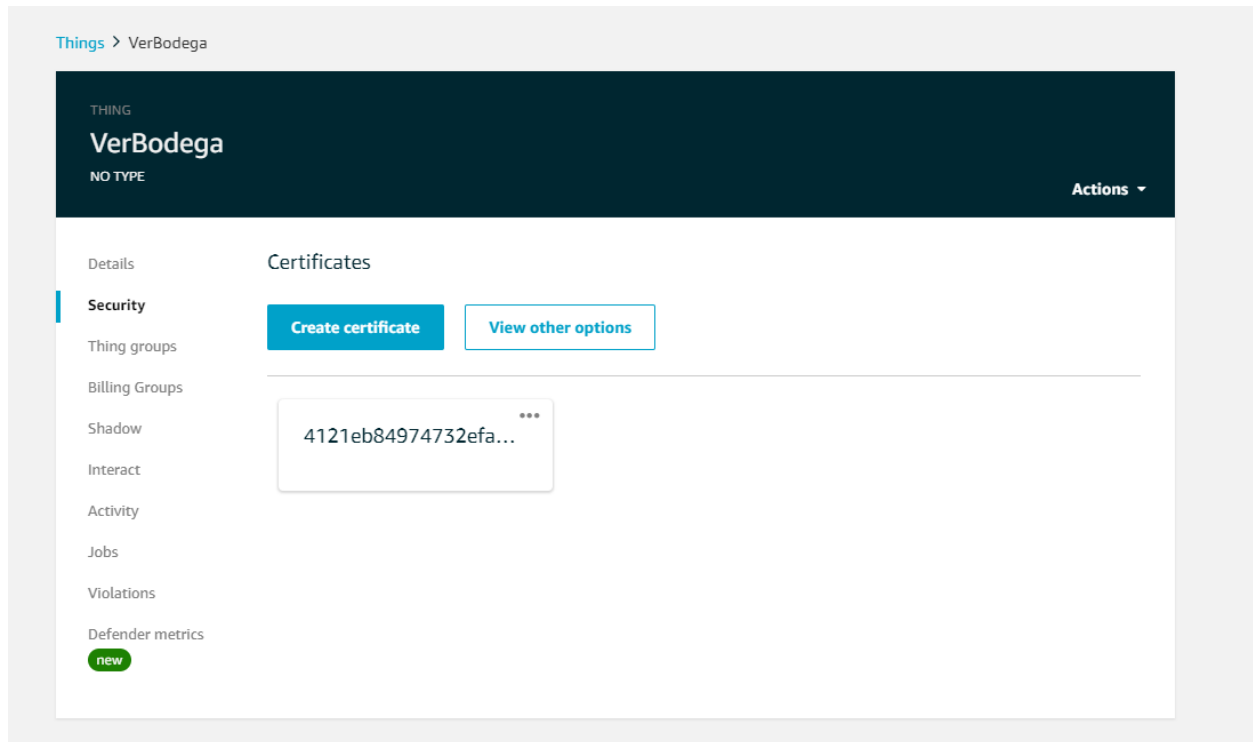
Update your Thing Shadow using this Rest API Endpoint. [Learn more](#)

~~https://ats.iot.us-east-2.amazonaws.com~~

```
#define CLIENT_ID "VerBodega_Demo"
#define MQTT_TOPIC "outTopic"
#define AWS_HOST "ats.iot.us-east-2.amazonaws.com"
```

Una vez configurado esto, tenemos que definir las credenciales del dispositivo. Esto se consigue creandolo en el momento de la configuración de la cosa. Con esto conseguiremos 3 archivos con keys de los certificados del dispositivo, que tendremos que guardar con mucho cuidado, ya que no las podremos volver a descargar. Estas 3 keys, deben ingresarse en la librería de AWS-IOT, la cual podemos descargar desde el mismo enlace donde obtenemos las keys.





De modo de configurar las siguientes variables, y los siguientes includes.

```
#include <WiFi.h>
#include <DHT.h>
#include <AWS_IOT.h>
#include <MQ2.h>

#define DHT_PIN 33
#define DHT_TYPE DHT11

#define WIFI_SSID "CLAROL 2021"
#define WIFI_PASS "88451231"

#define CLIENT_ID "VerBodega_Demo"
#define MQTT_TOPIC "outTopic"
#define AWS_HOST "a121eb84974732efa-ats.iot.us-east-2.amazonaws.com"

DHT dht(DHT_PIN, DHT_TYPE);
MQ2 mq2(32);
AWS_IOT aws;

int lpg, co, smoke;
```

En la carpeta de Arduino>Libraries>AWS\_IOT>SRC, debemos ingresar al archivo "aws\_iot\_certificates.c", en este debemos sustituir las 3 keys con las descargadas. Siguiendo la misma sintaxis (cada línea terminada con un \n).

```
const char aws_root_ca_pem[] = {"-----BEGIN CERTIFICATE-----\n\
MIIEKjCCA3qAwIBAgITBn+USionzfp6wq4rAfKI7rnExjANBgkqhkiG9w0BAQsF\n\
ADCBmDELMAkGA1UEBhMCVVMxEDAOBgNVBAGTB0FyaXpvcjExMC5kaW4uY291\n\
b3R0c2RhbGUxJTAjBgNVBAoTTFh0YXJmaWVzZCB1Zm9udm9sb2dpZC51eSIE\n\
luYy4xLjE1OTYxMjM1MDUyNTYyMDAwMFoXDTEzMTIzMTAxMDAwMFowOTEL\n\
dGhvcm10eSAtIEcyMB4XDTE1MDUyNTYyMDAwMFoXDTEzMTIzMTAxMDAwMFow\n\
OTEL\n\
-----END CERTIFICATE-----"};
```

```
const char certificate_pem_crt[] = {"-----BEGIN CERTIFICATE-----\n\nMIIDWTCCAKGgAwIBAgIUWCHj6S38PSH185N0tLwYGu5y7KAwDQYJKoZIhvcNAQEL\nBQAwTTFLEkGA1UECwwxQW1hem9uIFdlYiBTZXJ2ZWVlcyBPPUFTYXpvbi5jb20g\nSW5jLiBMPVNlYXR0bGUgU1Q9V2FzaGluz3RvbiBDPVVtMB4XDTEwMDQwNDIxMjM1\nMVYwXDTEwMTI=NTT=NTk1OVRvLjE1fCMBAGAAUFAA+TCQYDTERlZGVzY3Rvbi5jb20g\n-----END CERTIFICATE-----"};
```

```
const char private_pem_key[] = {"-----BEGIN RSA PRIVATE KEY-----\n\
MIIEpAIBAAKCAQEAA9bZyo1eZYV0+5q+z2oS93JYL+CJDGwdqcBPVBLgVpALSqD26\n\
riv9Tt1t+IX3cVMDYI0fB1ihNlsYvi6g6cavmhGSWgvir93njYktT1qZG44kwMgm\n\
Izwb5oNkzLAZWynJlws8iAhElgJzmEDjSAoopQpfA3K1Bd5nKahYh/EMRPMYbF1B\n\
j1H7d3ldojKbfnCs2uKePH5A9DsXRFGuwtcUmpQJogB8XiNa6BSAaj7KE+GCiwbY\n\
j0316D11-0185163ilUW0: 00F0-M-T-XP3-A-NF-HB5(WK-C)\n\
-----END RSA PRIVATE KEY-----"};
```

Con esto sustituido, tenemos ya acceso a nuestro dispositivo.

El setup, configura la conexión con el WIFI, y luego trata de conectarse con la “cosa” en AWS.

```

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.print("\n Inicializando AWS_THING VerBodega_Test \n");

  Serial.print("\n Conectando WIFI a:");
  Serial.print(WIFI_SSID);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print(" ");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.print("\n Dispositivo Conectado!!!\n");
  Serial.println("Incializando DH11");
  dht.begin();
  mq2.begin();
  Serial.println("DH11 Inicializado!!!");

  Serial.println("Conectando Dispositivo a AWS! o:");
  if (aws.connect(AWS_HOST, CLIENT_ID) == 0) {
    Serial.println("Conectado a AWS! :) ");
  } else {
    Serial.println("Fallo la conexión a AWS! :)");
  }
}

```

Para la ejecución del programa es facil, solo leemos los datos de los sensores, y creamos una cadena JSON, luego la enviamos por la función `aws.publish` de la librería de `aws-iot`.

```

void loop() {
  // put your main code here, to run repeatedly:
  float temperatura = dht.readTemperature();
  float humedad = dht.readHumidity();
  float* values = mq2.read(true);
  lpg = mq2.readLPG();
  co = mq2.readCO();
  smoke = mq2.readSmoke();

  if (temperatura == NAN || humedad == NAN) {
    Serial.println("Fallo la lectura de datos");
  } else {
    String mess = "{ \"Temperatura\": \";
    mess += String((int)temperatura);
    mess += "\", \"Humedad\": \";
    mess += String((int)humedad);
    mess += "\", \"CO2\": \";
    mess += String(values[1]);
    mess += "\", \"Smoke\": \";
    mess += String(values[2]);
    mess += "\", \"LPG\": \";
    mess += String(values[0]);
    mess += "\", \"Dispositivo\": \";
    mess += "VerBodega_1";
    mess += "\"}";

    char payload[140];
    mess.toCharArray(payload, 140);
    Serial.println("Publicando Mensaje!!! ~ ");
    Serial.println(payload);
    if (aws.publish(MQTT_TOPIC, payload) == 0) {
      Serial.println("Mensaje enviado exitosamente!!");
    } else {
      Serial.println("Se fallo enviando el mensaje :)");
    }
  }

  delay(120000);
}

```

## Cliente

El cliente desarrollado en React consiste de 1 modulo principal, el cual contiene, 3 graficas, la información del ultimo registro, y luego una tabla con todos los registros. Para su estilo se utilizo el CDN de Bootstrap 4.

Obtención de datos:

Para la lectura de datos de API Gateway se utilizó "Axios", esta librería de Node.js nos permite hacer llamadas a APIs de una forma muy sencilla. Instanceando esta librería podemos usar el método `get` que nos devuelve los valores del dispositivo de IoT.

```
function getDat() {  
  axios  
    .get(  
      "https://gys3v1rzp9.execute-api.us-east-1.amazonaws.com/Test//getBodega"  
    )  
    .then((res) => {  
      return res.data.Items.sort(function (a, b) {  
        return a.id - b.id;  
      });  
    });  
}
```

Graficas:

Para las graficas se utilizó la librería de ReactJS de "ReCharts", esta tiene una amplia gama de gráficos para hacer display de nuestros datos. Se consideró que una gráfica de línea era la mejor forma de representar los datos que nos interesaban a través del tiempo, los cuales eran la temperatura y la humedad.

A este componente se le crea una clase .js que va a representar el componente, y luego necesita de un arreglo de datos para el eje x, y otro para el eje y los cuales pasaremos como "props" del componente.

```
import {  
  LineChart,  
  Line,  
  XAxis,  
  YAxis,  
  CartesianGrid,  
  Tooltip,  
  Legend,  
} from "recharts";
```

Luego mandamos en esta sección de la grafica estos parametros para mandar a imprimir los mismos.

```

export default class Demo extends PureComponent {
  static jsfiddleUrl = "https://jsfiddle.net/alidingling/xqjtetw0/";

  render() {
    this.query = this.props.data;
    return (
      <LineChart
        width={500}
        height={300}
        data={this.props.data}
        margin={{
          top: 5,
          right: 30,
          left: 20,
          bottom: 5,
        }}
      >
        <CartesianGrid strokeDasharray="3 3" />
        <XAxis dataKey="date" />
        <YAxis />
        <Tooltip />
        <Legend />
        <Line type="monotone" dataKey="temperatura" stroke="#8884d8" />
      </LineChart>
    );
  }
}

```

Para invocar el componente lo hacemos de esta forma:

```

<div className="card-body">
  <Demo2 data={datos} />
</div>

```

Iteración de un Arreglo:

Para iterar un arreglo (de los datos que recibimos) se debe usar la función nativa de javascript de ForEach, la cual podemos usar en react para que por cada objeto, nos retorne un objeto JSX (lenguaje de react que nos permite usar javascript + html), por lo cual, llenamos la tabla de datos, con esta función.

```
const content = () => {  
  data.forEach((element) => {  
    console.log(element);  
    elements.push(<Registro object={element} />);  
  });  
  return elements;  
};
```

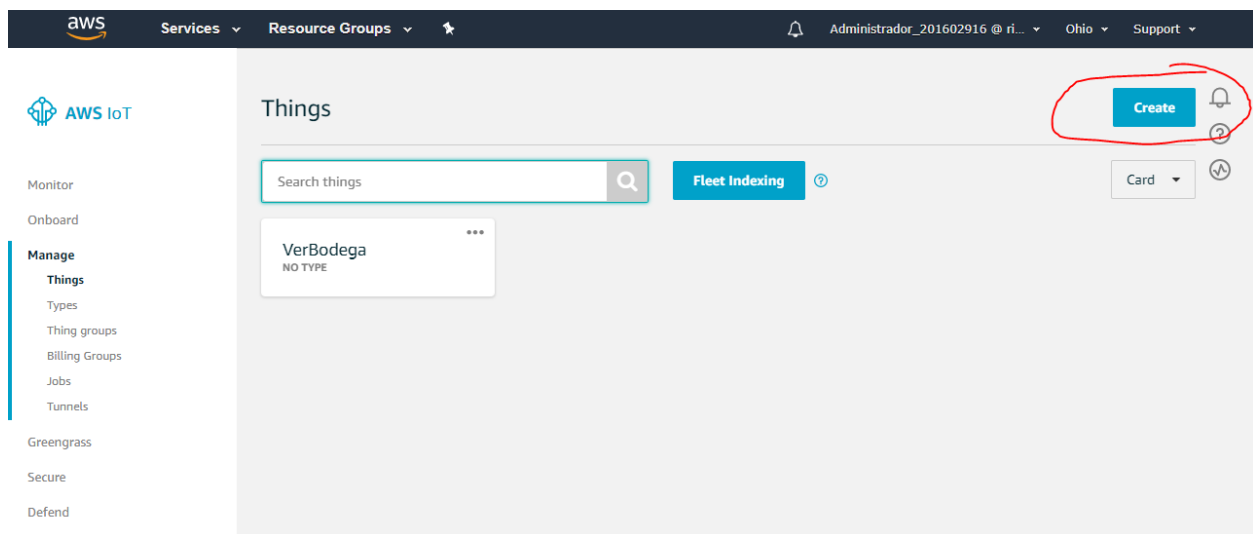
# IoT Core

## ¿Que es?

Fdsafsa

## ¿Como Funciona?

Para crear una **cosa**, debemos entrar a AWS IoT e ir a Manage -> Things. Ahí podemos ver nuestras **cosas** o **crear una**.



Luego creamos una nueva **AWS IoT thing**.

En este punto, debemos hacer las configuraciones iniciales de la **cosa**. Ponerle un nombre, ponerlo en un grupo, o asignarle una categoría para tenerlo identificado o agrupado para asignarle acciones en el futuro. También podemos ponerle atributos para búsquedas en el registro.



This step creates an entry in the thing registry and a thing shadow for your device.

Name

cosaEjemplo

### Apply a type to this thing

Using a thing type simplifies device management by providing consistent registry data for things that share a type. Types provide things with a common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type

No type selected

Create a type

### Add this thing to a group

Adding your thing to a group allows you to manage devices remotely using jobs.

Thing Group

Groups /

Create group Change

### Set searchable thing attributes (optional)

Enter a value for one or more of these attributes so that you can search for your things in the registry.

Attribute key

Provide an attribute key, e.g. Manufacturer

Value

Provide an attribute value, e.g. Acme-Corporation

Clear

Add another

Show thing shadow ▼

Ahora nos toca generar los certificados y las llaves de acceso de la **cosa**, para tener poder acceder a esta luego.

CREATE A THING

STEP 2/3

Add a certificate for your thing

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)  
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

Create certificate

Create with CSR  
Upload your own certificate signing request (CSR) based on a private key you own.

Create with CSR

Use my certificate  
Register your CA certificate and use your own certificates for one or many devices.

Get started

Skip certificate and create thing  
You will need to add a certificate to your thing later before your device can connect to AWS IoT.

Create thing without certificate

Descargamos las llaves y certificados, y por último root CA de AWS IoT, y le damos a activar.

Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

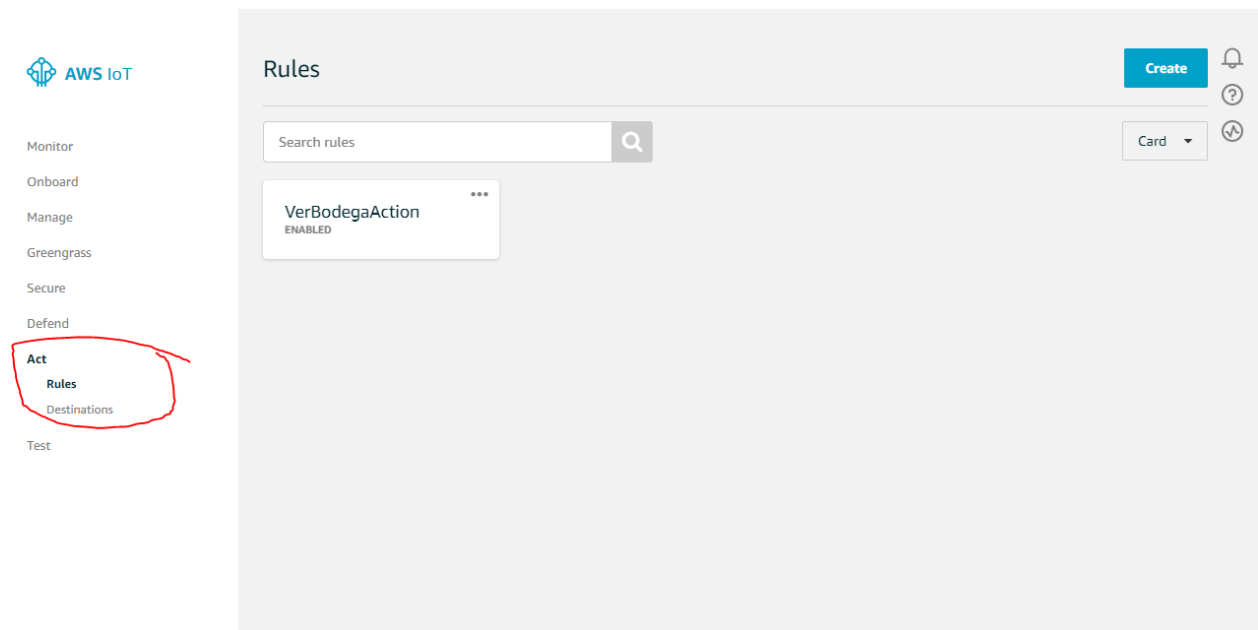
A certificate for this thing	bf396c9a62.cert.pem	Download
A public key	bf396c9a62.public.key	Download
A private key	bf396c9a62.private.key	Download

You also need to download a root CA for AWS IoT:  
A root CA for AWS IoT [Download](#)

Activate

Con esto tenemos creada la cosa. Para poder programar la, podemos ir a la sección anterior de Arduino.

Ahora para asignarle una acción. Debemos ir a **Act.**



Aca creamos una nueva Rule, la cual tendra un nombre, le podremos poner descripción, manipular la data que recibe, y por ultimo, asignarle una acción.

## Create a rule

Create a rule to evaluate messages sent by your things and specify what to do when a message is received (for example, write data to a DynamoDB table or invoke a Lambda function).

Name

Description

### Rule query statement

Indicate the source of the messages you want to process with this rule.

Using SQL version

2016-03-23 ▼

### Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```
1 SELECT * FROM 'iot/topic'
```

```
1 SELECT * FROM 'iot/topic'
```

### Set one or more actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (\*.required)

Add action

### Error action

Optionally set an action that will be executed when something goes wrong with processing your rule.

Add action

### Tags

Apply tags to your resources to help organize and identify them. A tag consists of a case-sensitive key-value pair. [Learn more](#) about tagging your AWS resources.

Tag name

Provide a tag name, e.g. Manufacturer

Value

Provide a tag value, e.g. Acme-Corporation









Clear












Add another

Entre las acciones que se pueden realizar, hay una amplia gamma de opciones a realizar, entre estas estan:

## Select an action

Select an action.

<input type="radio"/>	 DYNAMODB	Insert a message into a DynamoDB table
<input type="radio"/>	 DYNAMODB V2	Split message into multiple columns of a DynamoDB table (DynamoDBv2)
<input type="radio"/>	 LAMBDA	Send a message to a Lambda function
<input type="radio"/>	 SNS	Send a message as an SNS push notification
<input type="radio"/>	 SQS	Send a message to an SQS queue
<input type="radio"/>	 AMAZON KINESIS	Send a message to an Amazon Kinesis Stream
<input type="radio"/>	 AWS IOT REPUBLISH	Republish a message to an AWS IoT topic
<input type="radio"/>	 S3	Store a message in an Amazon S3 bucket

<input type="radio"/>	 Store a message in an Amazon S3 bucket S3
<input type="radio"/>	 Send a message to an Amazon Kinesis Firehose stream AMAZON KINESIS FIREHOSE
<input type="radio"/>	 Send message data to CloudWatch metric CLOUDWATCH METRICS
<input type="radio"/>	 Change the state of a CloudWatch alarm CLOUDWATCH ALARMS
<input type="radio"/>	 Send message data to CloudWatch logs CLOUDWATCH LOGS
<input type="radio"/>	 Send a message to the Amazon Elasticsearch Service AMAZON ELASTICSEARCH
<input type="radio"/>	 Send a message to a Salesforce IoT Input Stream SALESFORCE IOT
<input type="radio"/>	 Send a message to IoT Analytics IOT ANALYTICS
<input type="radio"/>	 Send a message to an IoT Events Input IOT EVENTS
<input type="radio"/>	 Start a Step Functions state machine execution STEP FUNCTIONS
<input type="radio"/>	 Send a message to a downstream HTTPS endpoint HTTPS

Para acceder a esta Regla, solo debemos conectarnos a ella desde el dispositivo de IoT como se explicó en la sección de arduino.

## ¿Como se usó en el proyecto?

En el caso del proyecto, se creo una **cosa** que representa a la placa. Y a esta se le asoció una acción que se enlazaba con una función lambda, esto se pudo haber saltado ingresando directamente a la base de datos, pero quería implementar el uso de funciones lambda.

## Actions

---

Actions are what happens when a rule is triggered. [Learn more](#)



Send a message to a Lambda function  
postVerbodega

Remove

Edit ▶

Add action