DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

TEMA: DISEÑO E IMPLEMENTACIÓN DE UNA PROPUESTA DE ARQUITECTURA DE SISTEMA DE GESTIÓN DE CONSUMO ELÉCTRICO EN EL HOGAR APLICANDO UNA PLATAFORMA DE CLOUD COMPUTING Y UN ALGORITMO DE MACHINE LEARNING

AUTOR: ZAMBRANO MOYA, RICHARD ALEXANDER

DIRECTOR: ING. ALULEMA FLORES, DARWIN OMAR

SANGOLQUÍ

2019

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICADO

Certifico que el trabajo de titulación, "DISEÑO E IMPLEMENTACIÓN DE UNA PROPUESTA DE ARQUITECTURA DE SISTEMA DE GESTIÓN DE CONSUMO ELÉCTRICO EN EL HOGAR APLICANDO UNA PLATAFORMA DE CLOUD COMPUTING Y UN ALGORITMO DE MACHINE LEARNING" fue realizado por el señor ZAMBRANO MOYA, RICHARD ALEXANDER el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 03 de Julio de 2019

.....

Ing. Darwin Omar Alulema Flores

C.C.: 1002493334

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORÍA DE RESPONSABILIDAD

Yo, ZAMBRANO MOYA, RICHARD ALEXANDER, declaro que el contenido, ideas y criterios del trabajo de titulación: DISEÑO E IMPLEMENTACIÓN DE UNA PROPUESTA DE ARQUITECTURA DE SISTEMA DE GESTIÓN DE CONSUMO ELÉCTRICO EN EL HOGAR APLICANDO UNA PLATAFORMA DE CLOUD COMPUTING Y UN ALGORITMO DE MACHINE LEARNING es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, 03 de Julio de 2019

.....

Richard Alexander Zambrano Moya

C.C.: 1723026769

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

Yo, ZAMBRANO MOYA, RICHARD ALEXANDER autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: DISEÑO E IMPLEMENTACIÓN DE UNA PROPUESTA DE ARQUITECTURA DE SISTEMA DE GESTIÓN DE CONSUMO ELÉCTRICO EN EL HOGAR APLICANDO UNA PLATAFORMA DE CLOUD COMPUTING Y UN ALGORITMO DE MACHINE LEARNING en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 03 de Julio de 2019

.....

Richard Alexander Zambrano Moya

C.C.: 1723026769

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

DEDICATORIA

El logro se lo dedico a mi familia, a mi madre Martha, a mi padre Héctor, a mis hermanas Gisel y Maryuri, además, abuelo, abuela, tíos, y tías quienes me apoyaron constantemente de una u otra forma durante el transcurso de toda mi carrera y en la elaboración de este proyecto. Sus consejos y sabiduría fueron fundamentales para guiarme desde la niñez hasta el presente, con el objetivo de verme profesional.

En general a mis tíos y tías que han sido mi fuente de inspiración, fuerza que me impulsa a seguir adelante, ejemplo de respeto, honestidad y responsabilidad, valores que me han guiado por caminos de rectitud.

Por todos esto les dedico este importante logro, que sin su apoyo no hubiese podido lograr.

Richard Alexander Zambrano Moya

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA TELECOMUNICACIONES

AGRADECIMIENTO

Este trabajo de investigación es el fruto y el esfuerzo de muchas personas que tal vez me falte papel para expresarles a todos mis más grandes y sinceros agradecimientos, aquí quiero dejar constancia de el aprecio y la gratitud que siento por todas las personas.

A mis padres, abuelo, abuela, tíos, y tías por todo el esfuerzo que han hecho para formarme como una persona de bien, por su comprensión, por impulsarme a seguir siempre adelante, por el inmenso amor que me brindan. Entre ellos a Nelly y Cristóbal por permitirme implementar el escenario de prueba en su hogar.

A mis amigos Sergio y Julio por ser soporte durante la carrera, y en especial a Julio con quien desarrollamos varios proyectos, y fue pieza clave al final del presente trabajo de investigación.

A la Universidad de las Fuerzas Armadas – ESPE, al departamento de DEEE por desarrollarme como profesionalmente y seguir cultivando mis valores, y de manera especial a mi tutor de tesis, por haberme guiado en este trabajo de titulación.

Richard Alexander Zambrano Moya

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Tecnologías como 5G, sistemas de comunicación, plataformas de cloud computing, dispositivos IoT (Internet-of-Things), entre otros, han evolucionado considerablemente durante los últimos años. Sin embargo, en particular los dispositivos IoT muchas veces son fabricados por empresas privadas, y su uso y control se limita a las libertadas dadas por los fabricantes. Por tal motivo muchos programadores invierten tiempo en el desarrollo de librerías capaces de controlar y obtener información relevante de estos dispositivos. El presente trabajo de investigación hace uso de enchufes inteligentes TP-LINK-HS-110, para monitorizar el consumo eléctrico de los dispositivos conectados en el hogar (televisor, computador, y microonda). Para el diseño del backend se ha empleado una API-REST abierta que permite el envío de peticiones HTTP al dispositivo IoT. Los datos enviados emplean el formato JSON con información sobre los parámetros de consumo e información general del dispositivo como MAC, IP, alias, entre otros. Además, se ha empleado una arquitectura basada en servicios con el fin de integrar servicios AWS. La propuesta es un sistema de gestión de consumo eléctrico para el hogar aplicando la plataforma AWS, y un modelo de machine learning para aprender a reconocer patrones de consumo en los parámetros eléctricos como voltaje, corriente, y potencia; con la finalidad de identificar los hábitos en el hogar. Los resultados son visualizados en el frontend que proporciona información del enchufe, información y gráfica de parámetros eléctricos en tiempo real, y diagrama en barras de los hábitos de consumo. Para validar el sistema se ha desarrollado pruebas de funcionamiento, carga, y calidad de código.

PALABRAS CLAVE:

- **CLOUD COMPUTING**
- **MACHINE LEARNING**
- **MONITOREO DE ENERGÍA**
- **CONSUMO ELÉCTRICO**
- **HÁBITOS DE CONSUMO**

ABSTRACT

Technologies such as 5G, communication systems, cloud computing platforms, IoT (Internet-of-Things) devices, among others, have evolved considerably in recent years. However, in particular IoT devices are often manufactured by private companies, and their use and control is limited to the freedoms given by the manufacturers. For this reason, many programmers invest time in the development of libraries capable of controlling and obtaining relevant information from these devices. The present research work makes use of smart plugs TP-LINK-HS-110, to monitor the electrical consumption of the connected devices in the home (television, computer, and microwave). For the design of the backend, an open API-REST was used, which allows the sending of HTTP requests to the IoT device. The data sent uses the JSON format with information about the consumption parameters and general information of the device such as MAC, IP, alias, among others. In addition, a service-based architecture has been used to integrate AWS services. The proposal is an electrical consumption management system for the home using the AWS platform, and a machine learning model to learn to recognize consumption patterns in electrical parameters such as voltage, current, and power; with the purpose of identifying the habits in the home. The results are visualized in the frontend that provides plug information, information and graph of electrical parameters in real time, and bar diagram of consumption habits. In order to validate the system, functional tests, loading, and code quality have been developed.

KEYWORDS:

- **CLOUD COMPUTING**
- **MACHINE LEARNING**
- **MONITORING OF ENERGY**
- **ELECTRICAL CONSUMPTION**
- **CONSUMPTION HABITS**

CAPÍTULO 1

1. INTRODUCCIÓN

1.1. Antecedentes

Es conocido que la dinámica del consumo eléctrico de un país se constituye en una buena aproximación, en la intención de cuantificar el ritmo de crecimiento y desarrollo de una economía, aún más, se considera la evolución de la demanda de energía eléctrica como un indicador de éste y muchos la definen como "el motor del desarrollo", pues tiene una relación directa con todos los sectores de la economía, según la Agencia de Regulación y Control de Electricidad (ARCONEL, 2015).

En la actualidad, Ecuador consume aproximadamente 3 veces más energía eléctrica que hace 17 años; la demanda eléctrica total pasó de 7730 MWh en 2000 a 20204 MWh en 2017, a su vez en el año 2017 el sector residencial consumió cerca del 36.12% del total de la energía eléctrica del país. Mientras que, el número de clientes en el período 2010-2017 se incrementó de 3.951.991 a 5.071.690, es decir, un incremento del 28,33%, lo que significa un aumento promedio anual del 3,54% según la ARCONEL (2015) (2018).

Dentro del sector residencial, según el Instituto Nacional de Estadísticas y Censos (INEC, 2018), el tamaño de hogar promedio, en las ciudades de Quito, Guayaquil, Cuenca, Machala, Ambato, Santo Domingo, Manta, Esmeraldas y Loja en 2011-2012, es de 3.9 personas por hogar, es decir, en cada vivienda viven 4 personas. La estratificación del nivel socioeconómico en hogares urbanos de Quito, Guayaquil, Cuenca, Ambato y Machala en 2011 agrupó los hogares en 5 grupos socioeconómicos de acuerdo con características homogéneas con el fin de segmentarlos para fines comerciales o sociales, entre ellas la instrucción educativa del jefe del hogar. El grupo C- con 49.3 % es el grupo mayoritario, como se puede observar en la ***Figura 1***. A pesar de ello, debido a la

falta de estudios de afines y considerando que el ingreso familiar promedio aumento de alrededor de $500 a $720, y la cobertura del presupuesto familiar paso del 88% al 101% durante 2011-2018. Se toma como referencia al grupo C+, cuyos hogares en promedio tienen 2 televisores a color, el 62% tiene una computadora de escritorio, y más del 67% de los hogares tiene microonda, cocina con horno, lavadora, equipo de sonido y/o minicomponente.

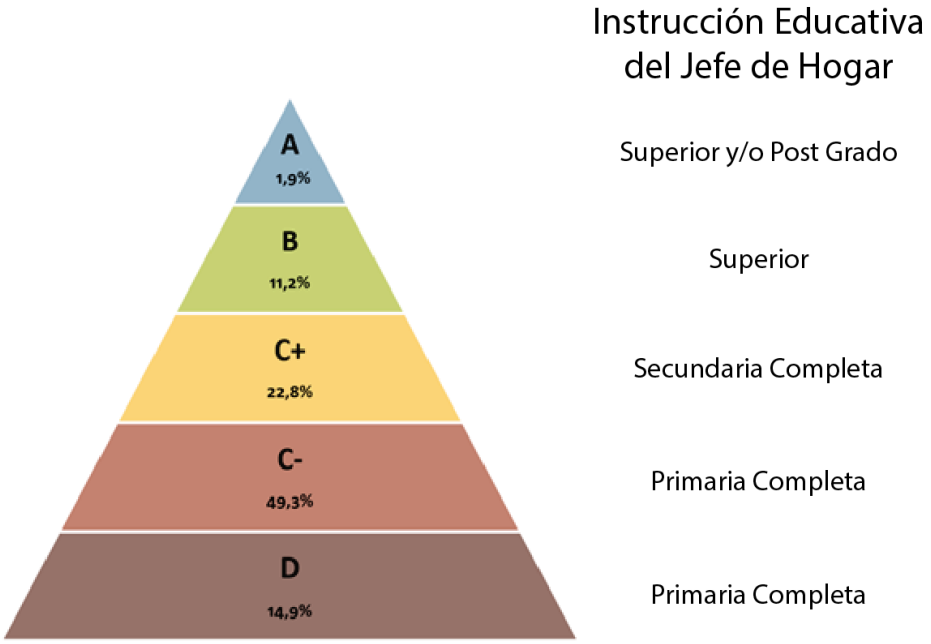


Figura 1. Estratificación del nivel socioeconómico en hogares urbanos de Quito, Guayaquil, Cuenca, Ambato y Machala en 2011.
Fuente: (INEC, 2018)

A pesar de que los electrodomésticos son cada vez más eficientes, el consumo energético global no disminuye proporcionalmente a esta mejora, al contrario de lo que cabría esperar, sino que incluso puede aumentar según el efecto rebote (Linares, 2009). El consumo eléctrico en los hogares crece debido a la adquisición de nuevos productos electrónicos que hacen la vida más fácil y práctica, es así como (Herrera V. , 2013) sugiere que se debe apagar los aparatos eléctricos y

desconectar los que no tienen interruptor cuando no se estén utilizando, incluyendo los reguladores de voltaje.

Ecuador ha definido la hoja de ruta REDIE (Redes Inteligentes en Ecuador), en concordancia con la Constitución Nacional de la República del Ecuador y las Políticas establecidas por el Ministerio de Electricidad y energía Renovable (2018), ha dado un paso hacia un nuevo modelo energético 2013-2017 mediante el proyecto EcoSmart Galápagos, que se enfoca en los cinco dominios de las redes inteligentes: Distribución Flexible, Hogar Eficiente, Gestión de la Demanda y Smart Generation.

Los avances tecnológicos plantean cerrar la distancia entre el mundo físico y el mundo digital en sistemas que se refuerzan y mejoran automáticamente (Universidad Tecnológica de Chile INACAP, s.f.). Entre ellos, IoT o Internet de las Cosas, que es una red que interconecta objetos físicos valiéndose de las tecnologías de internet, ya sea para informar su estado actual, recopilar datos de interés o simplemente hacer uso de recursos en la nube. Los parámetros son representados en plataformas web y/o móvil para la interacción entre el usuario y controlador.

Tomando como referencia al Hogar Eficiente, mejor conocido como Smart Home, y a los Dispositivos IoT, en la actualidad se han desarrollado dispositivos de monitoreo como el enchufe inteligente TP-LINK HS-110 que son sistemas embebidos que adquiriere datos del consumo eléctrico de equipos electrónicos (computador de escritorio, microondas, TV), cuyos valores se alojan en la plataforma cloud del fabricante, para su visualización y control. Además, plataformas de cloud computing despliegan sistemas de recursos distribuidos de manera horizontal, e introducidos como servicios virtuales de tecnologías de información (Ávila, 2011), facilitando la interoperabilidad entre plataformas. Entre los servicios virtuales está machine learning, cuyo enfoque es analizar los datos, aprender de ellos y predecir su comportamiento.

1.2. Justificación

Las REI (Redes Eléctricas Inteligentes) definen la gestión de la demanda como la planificación e implementación de medidas destinadas a influir en el comportamiento de los consumidores para disminuir su perfil de consumo diario (Herrera V. , 2013), mientras que eficiencia en Smart Home hace referencia a generar un potencial de ahorro de consumo energético de electrodomésticos, iluminación, calentamiento de agua, climatización y cocina (Herrera V. , 2013).

El término Smart Home se aplica a los modernos ambientes de vivienda acompañados de procesos automáticos, flexibilidad y comodidad (Diego, 2016). De acuerdo con Matija y Andrej (2013) un Smart Home está representado por la automatización en los siguientes sistemas:

- Iluminación
- Temperatura y control de ventilación
- Automatización de aplicaciones para el hogar
- Computadores y cargas electrónicas
- Administración de energía en el hogar
- Micro generación de energía
- Seguridad
- Operación de ventanas y puertas
- Administración y visualización

En el presente proyecto se plantea diseñar e implementar una propuesta de arquitectura de sistema de gestión de consumo eléctrico en el hogar aplicando una plataforma de cloud computing y un algoritmo de machine learning, haciendo énfasis al hogar eficiente como uno de los cinco dominios de las Redes Eléctricas Inteligentes (REI), según el Ministerio de Electricidad y Energía

Renovable; y como eje fundamental hacia las Smart Home hacia la administración de energía en el hogar

Tomando como referencia los ámbitos del IoT, que son: aplicación de dispositivos de sensorización, gestión de la comunicación y datos, y generación de conocimiento (Universidad Tecnológica de Chile INACAP, s.f.). El trabajo de investigación se ha estructurado de tal manera que coincida con las áreas mencionadas, las cuales se detallan a continuación:

El primer ámbito de IoT son los dispositivos de monitoreo. Estos son una herramienta para analizar y gestionar sistemas desde un punto central de control (Valdiosera, 2013). Es posible diseñar e implementar dispositivos de monitoreo de consumo eléctrico, con uso de sensores y elementos de comunicación, pero conlleva un gasto económico e intelectual elevado, y además en el mercado ya existen dispositivos de monitoreo de consumo eléctrico. El presente trabajo de investigación usa el Enchufe Inteligente Wi-Fi HS110 para monitorizar y gestionar el consumo eléctrico histórico y en tiempo real para los dispositivos electrónicos conectados (tp-link, s.f.). Mediante el uso de plataformas de desarrollo colaborativo como GITHUB, se obtuvo un API (Seal, 2018) que ofrece libertades para monitorizar y gestionar los dispositivos mediante el uso de un dispositivo centralizado como lo es la RASPBERRY PI.

El segundo ámbito hace referencia a la gestión de la comunicación y datos, se usa plataformas de cloud computing privadas gracias a la interoperabilidad y variedad de servicios que ofrece, y la posibilidad de incrementar su capacidad de cómputo y almacenamiento de datos en función de las necesidades y en tiempo real (Bocchio, 2013). Microsoft Azure, Google Cloud y AWS son plataformas sólidas de IaaS (Infraestructura como servicio), entre las cuales AWS se destaca debido a su alta personalización en función a las necesidades que se requiere, y al costo a pagar (CloudTech, 2016).

El tercer ámbito es generación de conocimiento, mediante el servicio de Machine Learning que ofrece AWS se puede analizar los datos, aprender de ellos y ser capaces de hacer una predicción (Rodriguez, 2017). AWS Machine Learning, ofrece en general 3 modelos de algoritmos, de los cuales se hará énfasis a un modelo binario, cuyo objetivo es aprender de los hábitos de consumo eléctrico de los electrodomésticos en el hogar, para dar sugerencias al usuario sobre la desconexión de estos para una gestión del consumo eléctrico.

Además, es necesario el desarrollo de una aplicación multiplataforma (web/móvil) para la interacción del usuario, la cual permita visualizar y gestionar el consumo eléctrico de los electrodomésticos conectados a la arquitectura.

1.3. Alcance del Proyecto

El presente proyecto se enfoca en el hogar eficiente y propone diseñar e implementar una propuesta de arquitectura de sistema de gestión de consumo eléctrico en el hogar aplicando una plataforma de cloud computing y un algoritmo de machine learning, el cual va a cumplir con los siguientes requerimientos:

- Monitoreo del consumo eléctrico de tres dispositivos electrónicos (computador de escritorio, microondas, TV) conectados a tres enchufes inteligentes TP-Link HS110.
- Uso de una Raspberry PI 3 como equipo central que permita la comunicación entre los enchufes inteligentes TP-Link HS110 y la plataforma AWS.
- Uso de una API para mantener una comunicación directa con los enchufes inteligentes TP-Link HS110 sin necesidad de recuperar los datos en la plataforma TP-Link, y permita la adquisición de datos en tiempo real y el control on/off de cada uno de los enchufes.

- Diseño de una arquitectura en microservicios en la plataforma AWS, capaz de ofrecer comunicación con la RASPBERRY PI 3, alojamiento de una página web estática, y un modelo de machine learning.
- Procesamiento de los datos antes y después de ejecutar el modelo de machine learning, para general gráficas de los hábitos de consumo de cada uno de los dispositivos electrónicos, mediante el uso de Excel como herramienta para manipular los datos.
- Diseño de una página web capaz de visualizar gráficamente parámetros de consumo eléctrico como voltaje, corriente y potencia, como también información de general (nombre, tipo de dispositivo, mac, ip, latitud y longitud) de cada uno de los enchufes inteligentes TP-Link HS110 en tiempo real. Además, ofrecer un control ON/OFF de los dispositivos electrónicos conectados, y mostrar los hábitos de consumo obtenidos a partir imágenes obtenidas de Excel.

1.4. Objetivos

1.4.1. General

Diseñar e implementar una propuesta de arquitectura de sistema de gestión de consumo eléctrico en el hogar aplicando una plataforma de cloud computing y un algoritmo de machine learning.

1.4.2. Específico

- Establecer el estado del arte de sistemas de monitoreo eléctrico, cloud computing y machine learning.
- Caracterizar al dispositivo de monitoreo de consumo eléctrico comercial.
- Determinar los requisitos del sistema (hardware y software) para visualización y control de consumo eléctrico de electrodomésticos (computador de escritorio, microondas, TV) vinculado con una plataforma cloud.

- Diseñar e implementar los servicios en una plataforma cloud y un modelo de machine learning.
- Diseñar una aplicación multiplataforma (web-móvil) para que consulte los servicios disponibles en la plataforma cloud.
- Implementar un caso de prueba para validar el funcionamiento de la arquitectura en un entorno experimental.
- Analizar los resultados.

1.5. Estado del Arte

En la actualidad la eficiencia energética en una residencia no solo beneficia al medio ambiente, sino también ayuda a disminuir el valor de la factura de luz eléctrica. En el presente trabajo de investigación, se ha establecido al Smart Home como eje fundamental para su desarrollo, pese a ello se requiere la búsqueda de información relevante que permitan conocer el desarrollo de investigación que al momento se han desarrollado para alcanzar este objetivo.

Plataformas y sitios web, como repositorio de universidades de Ecuador, y centro de formación de AWS, son ejemplos de sitios donde se comienza con la búsqueda, sin embargo, para mejorar el contenido obtenido se ha dividido en función de los siguientes palabras o temas clave:

- Sistemas de control y monitoreo de energía
- Aplicaciones de cloud computing
- Aplicaciones de machine learning.

Cada uno de los temas clave albergan una gran cantidad de resultados, en el caso de los repositorios a la universidad ESPE y EPN se pueden encontrar alrededor de 294 resultados con sistemas de control y monitoreo de energía, 36 resultados con cloud compuing, y 81 resultados con machine learning. Debido a la gran cantidad de resultados obtenidos, se ha optado por seleccionar

los siguientes 14 trabajos o temas de investigación porque en los Sistemas de Control y Monitoreo hacen uso de lenguaje Javascript e integran dispositivos IoT mediante el uso de APIs junto al proceso de configuración. Las aplicaciones de cloud computing muestran el proceso de diseño y configuración de varios servicios en la plataforma AWS como AWS RDS, AWS S3 o AWS SDK para migrar servicios empresariales y gestión de datos a la nube. Las aplicaciones de Machine Learning hacen uso de estructuras los datos clave para implementar modelos de aprendizaje binarios, y a su vez hacen uso de AWS Machine Learning, lo que permite conocer el proceso de configuración y flujo de la información desde datos primitivos hasta llegar a ser información útil para el usuario.

1.5.1. Sistemas de control y monitoreo de energía

Trabajos de investigación tales como (Chamba, 2015; Estévez, 2005; Flores, 2017; Freire, 2008; Herrera y Salguero, 2010; Morales, 2011), se orientan a las REI (Red Eléctrica Inteligente), y se enfocan en monitorizar, adquirir, procesar y almacenar parámetros de consumo eléctrico tales como voltaje, corriente y/o potencia; provenientes de sistemas embebidos con tecnologías de comunicación que permiten el monitoreo en tiempo real, ya sea local o remoto. Pese a ello, son sistemas implementados localmente y son susceptibles a fallas, sin contar con la inversión y costes que implica la adquisición de nuevo hardware y software y el mantenimiento de la infraestructura.

1.5.2. Aplicaciones de cloud computing

Cloud computing ofrece servicios de computación a través de Internet, esto ha llevado al diseño y desarrollo de sistemas mediante arquitecturas en microservicios en la nube. Algunos trabajos de investigación se basan en los microservicios para: ofrecer sistemas de gestión de calidad (Fernandez & Guerrero, 2016), o desplegar servicios empresariales específicos en la nube (Egas, 2014; Chandi y Roldán, 2015), o migración de infraestructura a la nube (Sánchez, 2014).

Cabe destacar que uno de los proveedores de cloud computing con mayor infraestructura es Amazon Web Services (AWS), en su infraestructura global se ha migrado servicios empresariales de empresas como Kellogg's, Philips, Time, Vodafone, entre otros. Por ejemplo, Siemens ha creado una Plataforma segura, compatible con HIPAA y escalable en AWS, mediante su sistema "Siemens Healthcare Diagnostics", que ayuda a mejorar la salud humana a través de la innovación. Al utilizar servicios como Elastic Load Balancing, Amazon Simple Queue Service (Amazon SQS), Amazon Elastic Cloud Computing (Amazon EC2), Amazon Relational Database Service (Amazon RDS), Auto Scaling, Amazon ElastiCache y AWS CloudTrail (Amazon Web Services , 2018),

1.5.3. Aplicaciones de machine learning

Desde su inicio en el año 1959, nuevos modelos y algoritmos de machine learning se han desarrollado, y diversas aplicaciones se han dado en diferentes áreas, una de ellas son los videojuegos. Pese a ello, diversos trabajos hacen uso del machine learning para fines específicos como: (Godoy, 2017) que desarrollo un guante inteligente que utiliza machine learning para el reconocimiento de lenguaje de señas; (Morejón, 2015) para clasificación de eventos vulcanólogos; o (Ruiz & Tello, 2017) para asignar turnos de emergencia en hospitales.

AWS dentro de su abanico de servicios, cuenta con modelos de machine learning. Debido a que es una de las plataformas privadas con mayor experiencia en el sector, empresas como Kía, Siemens, Netflix e incluso General Electric, entre otras; confían en su trabajo y optan para mejorar la calidad de la atención, combatir el tráfico de personas, brindar un mejor servicio al cliente y protegerlo contra el fraude (Amazon Web Services , 2018).

1.5.4. Conclusión

Durante la búsqueda efectuada se encontró que varios temas de investigación se enfocan a monitorizar el consumo eléctrico en tiempo real e integrando de manera inteligente las acciones de

todos los usuarios conectados a ella (Chamba, 2015; Flores, 2017; Morales, 2011), con la finalidad de administrar de manera eficiente, sin embargo, existen muy pocas que hacen referencia a plataformas de cloud computing, y menos aún a modelos o algoritmos de machine learning. Por este motivo se ha optado por tomar como principal referencia las arquitecturas, librerías, y ejemplos proporcionados por la plataforma AWS (Amazon Web Services , 2018) para el desarrollo de la arquitectura en la nube.

CAPÍTULO 2

2. FUNDAMENTO TEÓRICO Y CONCEPTUAL

2.1. Internet de las Cosas (IoT)

La frase aparentemente comenzó a partir de la presentación titulada “Internet of Things” en Procter & Gamble (P&G) en 1999, vinculada a la idea implementar la tecnología RFID en la cadena de suministro de P&G, según (Ashton, 2009). Los datos creados por seres humanos pueden ser capturados por computadoras o dispositivos al escribir, presionar un botón de grabación, tomar una foto digital o escanear un código de barras. Es así como Kevin Ashton concibe el IoT como una red en la que los dispositivos hacen uso del internet de forma independiente para informar su estado en todo momento y hacer uso de recursos en la nube (Gonzalez & Verdugo, 2018).

Por su parte CISCO define al IoT como un punto en la historia donde más cosas que personas están conectado a la red (CISCO, s.f.).

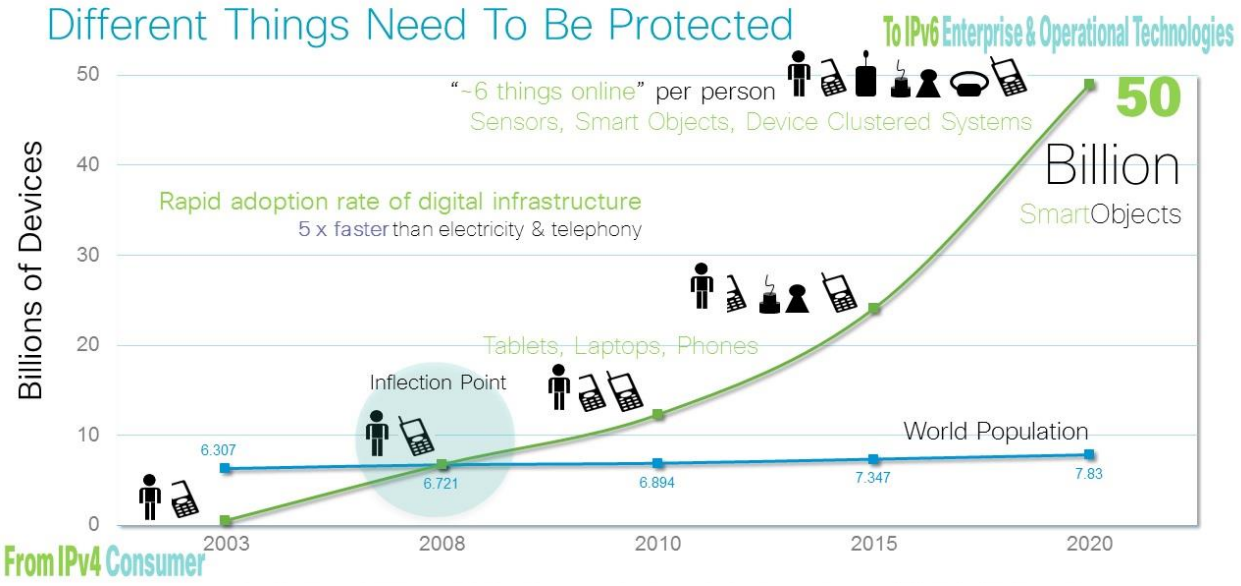


Figura 2. Relación entre número dispositivos inteligentes y personas por año.
Fuente: (CISCO, s.f.)

2.2. Raspberry PI

Raspberry Pi es una computadora de tamaño de tarjeta de crédito creada a comienzos de 2006 en el Reino Unidos por la Fundación Raspberry Pi con el propósito de promover y fomentar la enseñanza de informática básica. El 19 de febrero de 2012 se anunciaron dos modelos: Modelos A y B (PiAustralia, s.f.). A continuación, se presentan algunas características de la Raspberry PI:

- Raspberry Pi tienen puertos USB (uso teclado, ratón entre otros), entrada HDMI (conexión a monitor o TV), puerto DSI, entrada a audio, 40 pines GPIO, Bluetooth incorporado, WIFI, entre otras entradas.

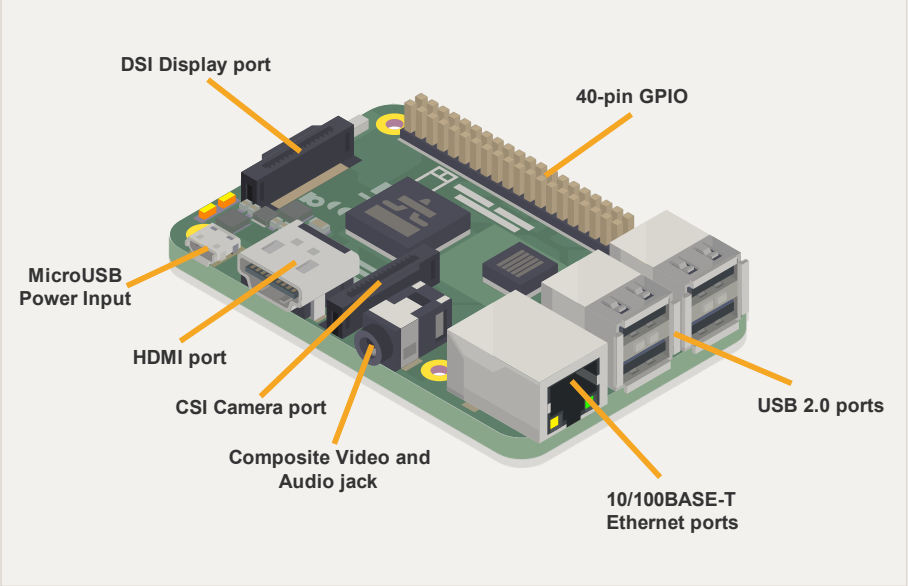


Figura 3. Detalle de principales conectores de la placa Raspberry PI 3.
Fuente: (PiAustralia, s.f.)

- Tiene un sistema Broadcom (conjunto de sistemas integrados) en un chip, que incluye unidad de procesamiento central (CPU) compatible con ARM y unidad de procesamiento gráfico.
- Intervalos de velocidad de la CPU son de 700 MHz a 1,2 GHz.
- Memoria RAM integrada entre 256 MB a 1 GB.

- El sistema operativo propietario se llama Raspbian, una derivación de Debian, y soporta varios lenguajes de programación como C++, Python, SQL, Java, JS, PHP, entre otros.

2.3. TP-Link

Es un proveedor de dispositivos y accesorios redes de origen chino. Esta marca ofrece todo tipo de productos como routers, switches, cámaras IP, PLC (PowerLine), servidores de impresión y también equipos para VDSL; cuyos equipos se caracterizan por su excelente calidad-precio (REDES ZONE, s.f.).

TP-Link ha evolucionado más allá de la tecnología inalámbrica, ofreciendo nuevos productos en las categorías de hogar inteligente, teléfono inteligente y accesorios inteligentes. Entre los productos hacia el hogar inteligente se puede encontrar cámaras cloud, bombillas inteligentes, y enchufes inteligentes. Cabe recalcar que en presente trabajo de investigación se hará uso de tres enchufes en inteligentes modelo HS110, descritos a continuación (TP-Link, s.f.).

Figura 4. Enchufe inteligente HS110.
Fuente: (TP-Link, s.f.)

2.3.1. Enchufe Inteligente HS110

Es un dispositivo enfocado al hogar inteligente que permite controlar y monitorizar la energía en los dispositivos conectados, mediante el uso de la aplicación gratuita llamada Kasa en el smartphone. Además, admite programar de forma manual el encendido y apagado de los dispositivos electrónicos que requiera, ya sea dentro o fuera del hogar. El enchufe inteligente es compatible con diferentes plataformas en la nube como Amazon Eco y Google Home para control mediante voz (TP-Link, s.f.). A continuación, se presentan algunas características relevantes del dispositivo:

Tabla 1

Especificaciones generales del enchufe inteligente HS110.

RED	
Protocolo	IEEE 802.11b/g/n
Tipo de Wi-Fi	2.4GHz, 1T1R
GENERAL	
Certificación	RoHS, EAC, CE
Ambiente	• Temperatura de Funcionamiento: 0 ºC ~ 40 ºC (32°F ~ 104°F) • Humedad de Funcionamiento: 5%~90%RH, Sin condensación
Dimensiones	100.3 x 66.3 x 77 mm. (3.9 x 2.6 x 3 in)
Peso	131.8g
ESTADO DE TRABAJO	
Voltaje de Entrada	100 - 240VAC
Voltaje de Salida	100-240VAC
Carga Máxima	13A
Alimentación Máxima	3.68KW

Fuente: (TP-Link, s.f.)

2.4. Cloud Computing

Cloud Computing es conocida como una nueva tecnología que permite ofrecer servicios de computación a través de Internet, es decir, la entrega a pedido de capacidades de cómputo, almacenamiento de bases de datos, aplicaciones y otros recursos de tecnología de la información (TI, más conocida por IT por su significado en inglés, Information Technology) a través de una plataforma de servicios en la nube a través de Internet con precios de pago por uso, y brinda acceso rápido a recursos de IT flexibles y de bajo costo. (Amazon Web Services , 2018).

2.4.1. Ventajas

- Pagar solo cuando consume recursos informáticos y la cantidad que consume, en lugar de tener que invertir mucho en centros de datos y servidores antes de saber cómo los va a utilizar. (Amazon Web Services , 2018)
- Beneficio de las economías de escala masivas, puede lograr un costo variable más bajo del que puede obtener por su cuenta. (Amazon Web Services , 2018)
- Puede acceder a tanta o tan poca capacidad como necesite, y ampliar y reducir según sea necesario, en lugar de adivinar sobre las necesidades de capacidad de la infraestructura. (Amazon Web Services , 2018)
- Los recursos de TI están a solo un clic de distancia, lo que significa que reduce el tiempo para hacer que esos recursos estén disponibles para sus desarrolladores de semanas a minutos, el costo y el tiempo que se tarda en experimentar y desarrollar es significativamente menor. (Amazon Web Services , 2018)

- Evita el gasto dinero en la ejecución y el mantenimiento de los centros de datos, lo que permite centrarse en los proyectos que diferencian su negocio, no la infraestructura. (Amazon Web Services , 2018)
- Facilidad de implementar aplicaciones en múltiples regiones del mundo con solo unos pocos clics. Esto significa que puede proporcionar una menor latencia y una mejor experiencia para sus clientes a un costo mínimo. (Amazon Web Services , 2018)

2.4.2. Tipos de Cloud Computing

Los servicios ofrecidos por la nube se distribuyen entre todas las capas tradicionales de un sistema informático, desde la capa de hardware hasta la capa aplicación software propiamente dicha (Relica, 2014). En la práctica, los proveedores de cloud computing tienden a ofrecer servicios que pueden ser agrupados en tres categorías, tal como se puede observar en la ***Figura 5***:

- IaaS (Infraestructura como servicio).
- PaaS (Plataforma como servicio).
- SaaS (Software como servicio).

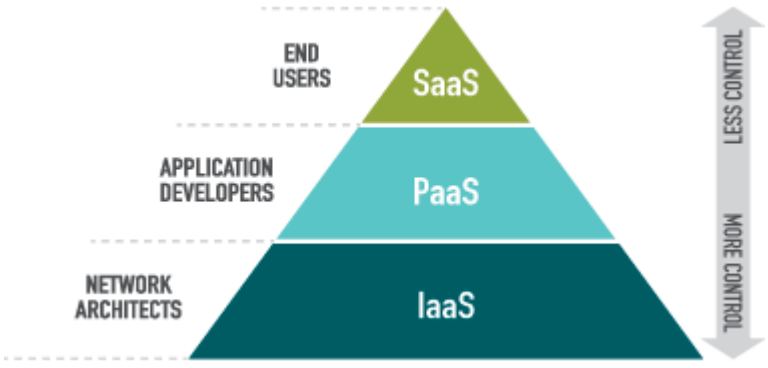


Figura 5. Tipos de Cloud Computing.
Fuente: (Redcentric, s.f.)

2.4.2.1. IaaS (Infraestructura como servicio)

IaaS, o Infraestructura como Servicio, describe los sistemas hardware que se le ofrecen a un usuario en forma de servicios para que éste pueda situar en ellos sus aplicaciones informáticas o su información. Los servicios que se ofrecen habitualmente abarcan desde capacidad de procesamiento o de almacenamiento ya sea alquilando servidores, discos duros, procesamiento en un CPD, etc. Hasta la capacidad de transmisión de información en forma de redes de comunicaciones de alta capacidad. (Relica, 2014)

2.4.2.2. PaaS (Plataforma como servicio)

PaaS, o Plataforma como Servicio, es un modelo orientado a equipos de trabajo que realicen proyectos de manera compartida, para lo cual proporciona un conjunto de herramientas y funcionalidades de software como sistemas operativos y servicios asociados a los mismos para el desarrollo conjunto de software y aplicaciones, situados en una red de máquinas de Cloud Computing y accesibles a través de Internet. Esto elimina la necesidad de que la organización administre la infraestructura subyacente (generalmente hardware y sistemas operativos) y le permite concentrarse en la implementación y administración de sus aplicaciones. (Amazon Web Services , 2018) (Relica, 2014)

2.4.2.3. SaaS (Software como servicio)

SaaS, o Software como Servicio le proporciona un producto completo que es ejecutado y administrado por el proveedor del servicio. En la mayoría de los casos, las personas que se refieren al Software como un Servicio se refieren a las aplicaciones del usuario final. Con una oferta de SaaS, no tiene que pensar en cómo se mantiene el servicio ni en cómo se gestiona la infraestructura subyacente; solo tiene que pensar en cómo va a utilizar ese software en particular. (Amazon Web Services , 2018)

SaaS a la final permite que el usuario del servicio no necesite instalar o actualizar la aplicación en sus equipos, sin necesidad de realizar una gran inversión inicial en adquisición de licencias o sistemas informáticos. De hecho, la inversión se realiza únicamente en función del uso de los servicios SaaS, cuyo costo a corto plazo suele ser bastante reducido. (Relica, 2014)

2.4.3. Modelos de Implementación de Cloud Computing

2.4.3.1. Nube pública

Hace referencia al modelo estándar de Cloud Computing, en el que el prestador de servicios pone a disposición de cualquier usuario en Internet su infraestructura (software o hardware) de forma gratuita o mediante el abono de cierta cantidad relacionada con el volumen o tiempo de uso de estos (Relica, 2014).

Las aplicaciones en la nube se han creado en la nube o se han migrado de una infraestructura existente para aprovechar los beneficios del cloud computing. Las aplicaciones basadas en la nube pueden construirse en piezas de infraestructura de bajo nivel o pueden usar servicios de nivel superior que brindan abstracción de los requisitos de administración, arquitectura y escalamiento de la infraestructura central (Amazon Web Services , 2018).

Tabla 2
Beneficios y riesgos de implementar una nube pública.

BENEFICIOS	RIESGOS
• Poca inversión. • Pago por uso • Buen entorno de pruebas y preproducción. • Fácil escalado de aplicaciones sobre múltiples servidores.	• Seguridad • Pérdida de control sobre el data center o sus datos.

Fuente: (Routing the World, 2012)

2.4.3.2. Nube híbrida

Las nubes híbridas consisten en combinar las aplicaciones propias de la empresa con las consumidas a través de la nube pública, entendiéndose también como la incorporación de servicios de Cloud Computing a las aplicaciones privadas de la organización (Relica, 2014).

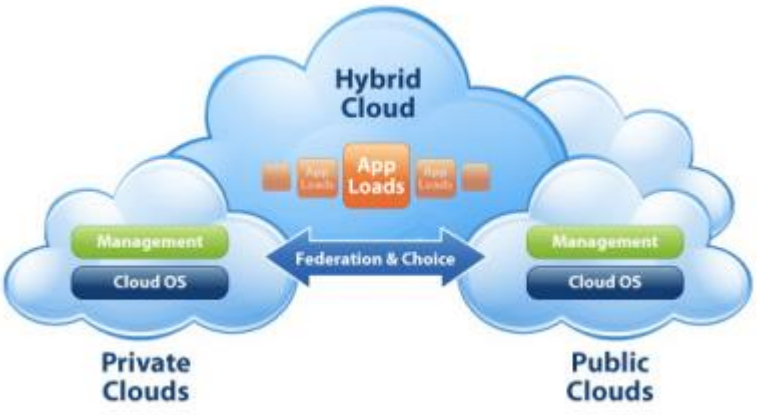


Figura 6. Modelo de implementación de la nube híbrida.
Fuente: (Routing the World, 2012)

El método más común de implementación híbrido es entre la nube y la infraestructura para ampliar y hacer crecer la infraestructura de una organización en la nube mientras se conectan los recursos de la nube al sistema interno (Amazon Web Services , 2018).

Tabla 3
Beneficios y riesgos de implementar una nube híbrida.

BENEFICIOS	RIESGOS
• Flexibilidad operativa • Entornos de desarrollo, test y producción no críticos pasan a la nube privada. • Escalabilidad	• Control de la seguridad entre la nube privada y la nube pública.

Fuente: (Routing the World, 2012)

2.4.3.3. Nube privada

Hacen referencia a redes o centros de procesamiento de datos propietarios que utilizan tecnologías y características de Cloud Computing, tales como la virtualización y la administración de recursos. Así, parten de los principios del Cloud Computing tradicional y ofrecen los mismos servicios dentro en la propia estructura de la compañía en busca de proporcionar recursos dedicados (Relica, 2014).

En la mayoría de los casos, este modelo de implementación es el mismo que la infraestructura de TI heredada, mientras que se utilizan las tecnologías de virtualización y la administración de aplicaciones para intentar aumentar la utilización de los recursos (Amazon Web Services , 2018).

Tabla 4
Beneficios y riesgos de implementar una nube privada.

BENEFICIOS	RIESGOS
• Menos cuestiones de seguridad ya que los elementos activos de la nube se encuentran "en casa". • La organización y sus departamentos IT mantienen el control sobre el data center.	• Alta inversión y costes de implantación, implica la adquisición de nuevo hardware y software y el mantenimiento de la infraestructura en el último escalón de la tecnología. • Son necesarios nuevos procesos operativos, la gestión tradicional de los activos IT no es aplicable a los entornos en la nube.

Fuente: (Routing the World, 2012)

2.5. Amazon Web Service (AWS)

Cloud Computing proporciona una forma sencilla de acceder a servidores, almacenamiento, bases de datos y un amplio conjunto de servicios de aplicaciones a través de Internet. Una plataforma de servicios en la nube, como Amazon Web Services, posee y mantiene el hardware conectado a la red requerido para estos servicios de aplicaciones, mientras que el usuario aprovisiona y utiliza lo que necesita a través de una aplicación web (Amazon Web Services , 2018).

AWS Cloud proporciona un amplio conjunto de servicios de infraestructura, como potencia de cómputo, opciones de almacenamiento, redes y bases de datos, que se ofrecen como una utilidad: bajo demanda, disponibles en cuestión de segundos y pagando solo por lo que utiliza (Amazon Web Services, 2018).

2.5.1. Infraestructura Global

AWS atiende a más de un millón de clientes activos en más de 190 países, ampliando constantemente la infraestructura para lograr una menor latencia y un mayor rendimiento, y para garantizar que los datos residan solo en la región de AWS que especifican. A medida que los clientes hacen crecer sus negocios, AWS continúa proporcionando infraestructura que cumpla con sus requisitos establecidos (Amazon Web Services , 2018).

Figura 7. Infraestructura global de AWS
Fuente: (Amazon Web Services, 2018)

La infraestructura AWS Cloud se basa en las regiones y zonas de disponibilidad de AWS. Una región de AWS es una ubicación física en el mundo que alberga múltiples zonas de disponibilidad. Las zonas de disponibilidad constan de uno o más centros de datos discretos, cada uno con alimentación redundante, redes y conectividad, alojados en instalaciones separadas, y ofrecen capacidad de operar aplicaciones de producción y bases de datos con mayor disponibilidad, tolerancia a fallos y escalables. AWS Cloud opera 57 zonas de disponibilidad en 19 regiones geográficas de todo el mundo (Amazon Web Services , 2018).

2.5.2. Seguridad y Cumplimiento

2.5.2.1. Seguridad

La seguridad en la nube es muy parecida a la seguridad en los datacenter locales, pero sin los costos de mantenimiento de las instalaciones y el hardware. En su lugar, utiliza herramientas de seguridad basadas en software para monitorizar y proteger el flujo de información dentro y fuera de la nube (Amazon Web Services , 2018).

Una ventaja de la nube de AWS es que le permite escalar e innovar, a la vez que mantiene un entorno seguro y paga solo por los servicios que utiliza. Esto significa que puede tener la seguridad que necesita a un costo menor que en un entorno local. A pesar de ello, el cliente es responsable de la seguridad en la nube. Esto significa que conserva el control de la seguridad que elige implementar para proteger su propio contenido, plataforma, aplicaciones, sistemas y redes de manera diferente a como lo haría en un datacenter local (Amazon Web Services , 2018).

2.5.2.1.1. Beneficios de la seguridad de AWS:

- Todos los datos se almacenan en varios datacenter de AWS altamente seguros.
- AWS administra docenas de programas de cumplimiento en su infraestructura.

- Reducción de los costos y cumplimiento del más alto estándar de seguridad sin tener que administrar instalaciones privadas.
- La seguridad se amplía con su uso de AWS Cloud.

2.5.2.2. Certificaciones y acreditaciones

La infraestructura AWS está diseñada y administrada de acuerdo con las mejores prácticas de seguridad y una variedad de estándares de seguridad de TI. La siguiente es una lista de los principales programas de protección con los que cumple AWS:

- ISO 9001, ISO 27001, ISO 27017, ISO 27018
- SOC 1 / ISAE 3402, SOC 2, SOC 3
- FISMA, DIACAP y FedRAMP
- PCI DSS Nivel 1

2.5.3. Arquitectura

Hace referencia a la forma en que se diseñan las aplicaciones. En lugar de ser monolitos, las aplicaciones se descomponen en servicios menores y descentralizados. Estos servicios se comunican a través de API o mediante el uso de eventos o de mensajería asincrónica. Las aplicaciones se escalan horizontalmente, agregando nuevas instancias, tal y como exigen las necesidades. Las operaciones se realizan en paralelo y de forma asincrónica. El sistema como un todo debe ser resistente cuando se producen errores (Microsoft Azure, s.f.).

AWS tiene un estilo de arquitectura de microservicios, la cual se ha empezado a hacer más popular mediante el cloud computing, debido a que no requieren el uso de tecnologías concretas para lograr satisfacer las exigencias de los usuarios.

2.5.3.1. Arquitectura de Microservicios

Durante los últimos años, los microservicios han sido una tendencia importante en la arquitectura de TI, debido a que consta de una colección de servicios autónomos, pequeños e independientes entre sí, destinados a cumplir una tarea específica. Los microservicios son la evolución natural de las arquitecturas orientadas a servicios, pero hay algunas diferencias (Microsoft Azure, s.f.).

A continuación, una estructura general de la arquitectura de microservicios:

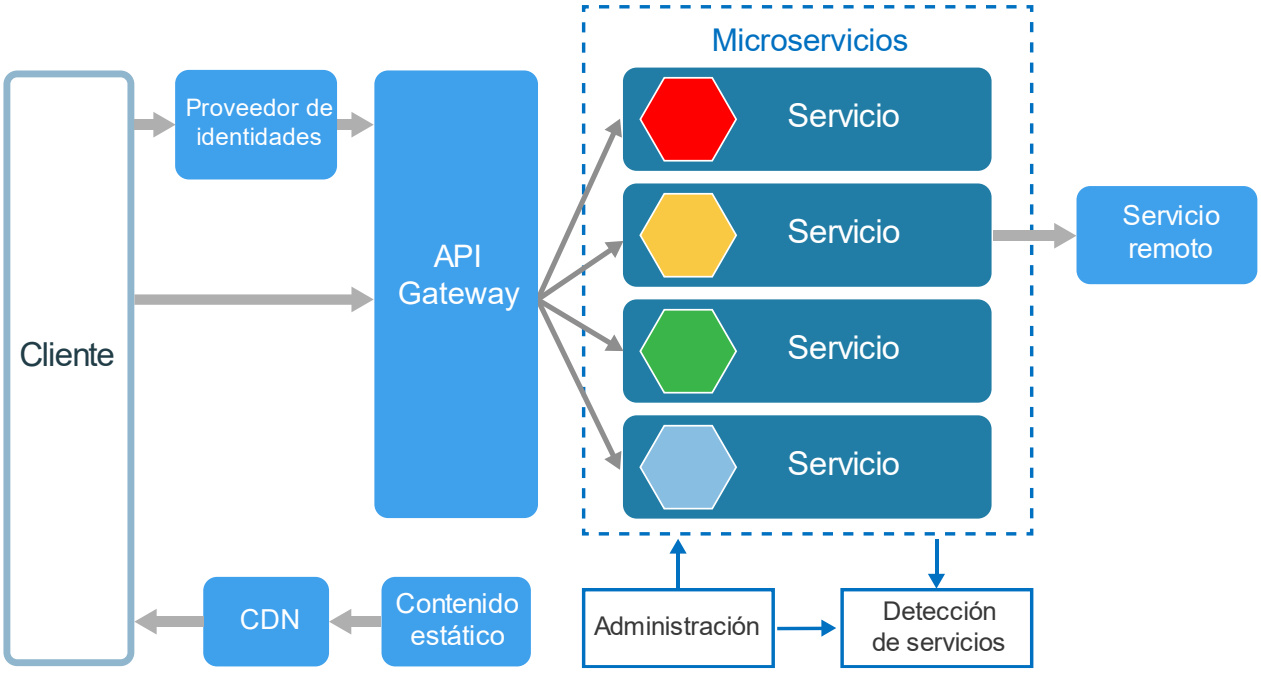


Figura 8. Estructura general de una arquitectura de microservicios.
Fuente: (Microsoft Azure, s.f.)

Estas son algunas de las características que definen un microservicio:

- Los servicios son pequeños e independientes y están acoplados de forma flexible.
- Cada servicio es un código base independiente, que puede gestionarse por un equipo de desarrollo pequeño.

- Cada equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación.
- Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.
- Los servicios se comunican entre sí mediante API bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios.
- No es necesario que los servicios compartan la misma pila de tecnología, las bibliotecas o los marcos de trabajo.

2.5.4. Plataforma AWS Cloud

AWS consta de muchos servicios en la nube que se puede utilizar en combinaciones adaptadas a las necesidades del cliente. Para acceder a los servicios, puede usar: la consola de administración de AWS, la interfaz de línea de comandos o los kits de desarrollo de software (SDK) (Amazon Web Services , 2018).

2.5.4.1. Consola de administración

Permite la administración de los servicios de AWS mediante una interfaz de usuario sencilla e intuitiva. También puede utilizar la aplicación móvil de la consola de AWS para ver rápidamente los recursos en curso (Amazon Web Services , 2018).

2.5.4.2. Kits de desarrollo de software (SDK)

Es una herramienta que simplifican el uso de los servicios de AWS en sus aplicaciones con una interfaz de programa de aplicación (API) adaptada a su plataforma o lenguaje de programación (Amazon Web Services , 2018). Entre las plataformas que soporta se encuentran las siguientes:

2.5.4.2.1. AWS SDK para JavaScript

Proporciona una API de JavaScript para los servicios de AWS que puede utilizar para crear aplicaciones para Node.js o el navegador. La API de JavaScript permite a los desarrolladores crear bibliotecas o aplicaciones que hacen uso de los servicios de AWS (Amazon Web Services, 2018).

Figura 9. Modelo gráfico de funcionamiento de AWS SDK para JavaScript.
Fuente: (Amazon Web Services, 2018)

2.5.4.1. Interfaz de línea de comandos (CLI)

Es una herramienta unificada para administrar los servicios de AWS. Con solo descargar y configurar, es posible controlar múltiples servicios de AWS desde la línea de comandos y automatizarlos a través de scripts (Amazon Web Services , 2018).

2.5.5. Productos de AWS

AWS ofrece un amplio conjunto de servicios de computación, almacenamiento, bases de datos, análisis, aplicaciones e implementación que permite a las organizaciones moverse con mayor

rapidez y reducir los costos de TI. Las arquitecturas que no usen ese conjunto de servicios (por ejemplo, si solo usan Amazon EC2) puede que no aprovechen al máximo la informática en la nube y pierdan la oportunidad de incrementar la productividad y la eficiencia operativa de los desarrolladores (Amazon Web Services, 2016).

AWS ofrece un amplio conjunto de productos globales basados en la nube, incluidas:

- Aplicaciones para cómputo
- Almacenamiento
- Bases de datos
- Análisis
- Redes
- Dispositivos móviles
- Herramientas para desarrolladores
- Herramientas de administración
- IoT
- Seguridad
- Aplicaciones empresariales.
- Machine learning

Estos servicios ayudan a clientes a avanzar con mayor rapidez, reducir los costos de TI y escalar. A continuación, se describe cada uno de los servicios que son parte del modelo de arquitectura propuesto.

2.5.5.1. Aplicaciones para cómputo

2.5.5.1.1. Amazon Elastic Compute Cloud (Amazon EC2)

Es un servicio web que proporciona capacidad de cálculo segura y de tamaño variable. La interfaz de Amazon EC2 proporciona un control completo de los recursos informáticos, reduciendo el tiempo requerido para obtener e iniciar nuevas instancias del servidor (llamadas instancias de Amazon EC2) en minutos, lo que le permite aumentar rápidamente la capacidad, a medida que cambian los requisitos informáticos. Amazon EC2 proporciona a los desarrolladores y

administradores de sistemas las herramientas para crear aplicaciones resistentes a fallas y aislarse de los escenarios de falla comunes (Amazon Web Services , 2018).

EC2 ofrece varios beneficios como:

- Integración con la mayor parte de servicios AWS, como Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), y Amazon Virtual Private Cloud (Amazon VPC) para proveer soluciones completas de seguridad de computo, procesamiento de consultas, y almacenamiento en la nube a través de un amplio rango de aplicaciones.
- Proporciona seguridad y una red robusta para los recursos informáticos en conjunto con Amazon VPC.
- Conexión mediante red privada virtual (VPN) IPsec cifradas entre la infraestructura de TI existente y los recursos en VPC.
- Suministro de recursos de Amazon EC2 como instancias dedicadas que se ejecutan en hardware dedicado a un solo cliente para un aislamiento adicional.
- Suministro de recursos de Amazon EC2 en Hosts Dedicados, que son servidores físicos con capacidad de instancia de EC2 totalmente dedicada a su uso.

2.5.5.2. Almacenamiento

2.5.5.2.1. Amazon Simple Storage Service (Amazon S3)

Es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento líderes en la industria. Esto significa que es posible usar para almacenar y proteger cualquier cantidad de datos para una gama de casos de uso, como sitios web, aplicaciones móviles, copia de seguridad y restauración. Amazon S3 está diseñado para un 99.999999999% (11

9) de durabilidad, y almacena datos para millones de aplicaciones para empresas de todo el mundo (Amazon Web Services , 2018).

2.5.5.3. Bases de datos

2.5.5.3.1. Amazon Relational Database Service (Amazon RDS)

Proporciona una capacidad rentable y redimensionable a la vez que automatiza las tareas de administración que requieren mucho tiempo, como el aprovisionamiento de hardware, la configuración de la base de datos, la aplicación de parches y las copias de seguridad. Amazon RDS está disponible en varios tipos de instancia de base de datos (optimizado para memoria, rendimiento o E / S) y le brinda seis motores de base de datos familiares para elegir, incluidos Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database y SQL Server (Amazon Web Services , 2018).

2.5.5.3.2. Amazon DynamoDB

Es una base de datos de documentos y valores clave que ofrece un rendimiento de milisegundos a cualquier escala. Se trata de una base de datos de múltiples maestros, multirregión y totalmente administrada, con seguridad, copia de seguridad y restauración incorporadas, y almacenamiento en caché en memoria para aplicaciones de escala de Internet. DynamoDB puede manejar más de 10 billones de solicitudes por día y soportar picos de más de 20 millones de solicitudes por segundo, destinada para aplicaciones que necesitan acceso a datos de baja latencia en cualquier escala (Amazon Web Services , 2018).

2.5.5.4. Análisis

2.5.5.4.1. AWS Data Pipeline

Es un servicio web para procesar y mover datos de manera confiable entre diferentes servicios de almacenamiento y cálculo de AWS, así como fuentes de datos locales, a intervalos específicos. Con AWS Data Pipeline, puede acceder regularmente a sus datos donde se almacenan,

transformarlos y procesarlos a escala, y transferir eficientemente los resultados a servicios de AWS como Amazon S3, Amazon RDS, Amazon DynamoDB y Amazon EMR. AWS Data Pipeline también le permite mover y procesar datos que previamente estaban encerrados en almacenes de datos locales (Amazon Web Services , 2018).

2.5.5.5. Redes y Entrega de Contenido

2.5.5.5.1. Amazon Virtual Private Cloud (Amazon VPC)

Permite suministrar una sección aislada lógicamente de AWS Cloud donde puede iniciar los recursos de AWS en una red virtual, con ello se obtiene un control completo sobre el entorno de la red virtual, incluida la selección del rango de direcciones IPv4 como IPv6, la creación de subredes y la configuración de tablas de rutas y puertas de enlace de red. Amazon VPN crea subredes públicas para los servidores web que tiene acceso a Internet y coloca los sistemas de backend, como bases de datos o servidores de aplicaciones, en una subred privada sin acceso a Internet. Además, puede aprovechar varias capas de seguridad (grupos de seguridad y las listas de control de acceso a la red) para ayudar a controlar el acceso a las instancias de EC2 en cada subred. Entre sus principales funciones está la creación de conexiones de red privada virtual (VPN) de hardware entre el datacenter corporativo y la VPC, y aprovechar AWS Cloud como una extensión del datacenter corporativo (Amazon Web Services , 2018).

2.5.5.6. Dispositivos móviles

2.5.5.6.1. AWS Amplify

Facilita la creación, configuración e implementación de aplicaciones móviles escalables desarrolladas por AWS, mediante la administración del backend móvil e integración con las interfaces de iOS, Android, Web y React Native. Amplify también automatiza el proceso de

lanzamiento de la aplicación del frontend y backend, lo que le permite entregar funciones más rápido (Amazon Web Services , 2018).

2.5.5.6.2. AWS AppSync

Es un backend sin servidor para facilitar la creación de aplicaciones móviles y web controladas por datos al manejar de forma segura todas las tareas de administración de datos de la aplicación, como el acceso de datos en línea y fuera de línea, la sincronización de datos y la manipulación de datos en múltiples fuentes de datos. AWS AppSync utiliza GraphQL, un lenguaje de consulta de API diseñado para crear aplicaciones cliente al proporcionar una sintaxis intuitiva y flexible para describir sus requisitos de datos (Amazon Web Services , 2018).

2.5.5.7. Seguridad, identidad y cumplimiento

2.5.5.7.1. AWS Identity and Access Management (IAM)

Permite controlar de forma segura el acceso a los servicios y recursos de AWS para sus usuarios. Con IAM, puede hacer lo siguiente:

- Crear usuarios, asignarles credenciales de seguridad individuales (claves de acceso, contraseñas y dispositivos de autenticación de múltiples factores) o solicitar credenciales de seguridad temporales para proporcionar a los usuarios acceso a los servicios y recursos de AWS.
- Crear roles y administrar permisos para controlar qué operaciones puede realizar la entidad, o el servicio de AWS, que asume el rol.
- Habilitar la federación de identidades para permitir que las identidades existentes (usuarios, grupos y roles) en su empresa accedan a la Consola de administración de AWS, llamen a las API de AWS y accedan a los recursos, sin la necesidad de crear un usuario IAM.

2.5.5.7.2. AWS Key Management Service (KMS)

Permite la creación y administración de claves y el control del uso del cifrado en una amplia gama de servicios de AWS y en sus aplicaciones. AWS KMS es un servicio seguro y resistente que utiliza módulos de seguridad de hardware validados por FIPS 140-2 para proteger las claves. AWS KMS está integrado con AWS CloudTrail para proporcionar registros de todas las claves para satisfacer las normativas y su cumplimiento (Amazon Web Services , 2018).

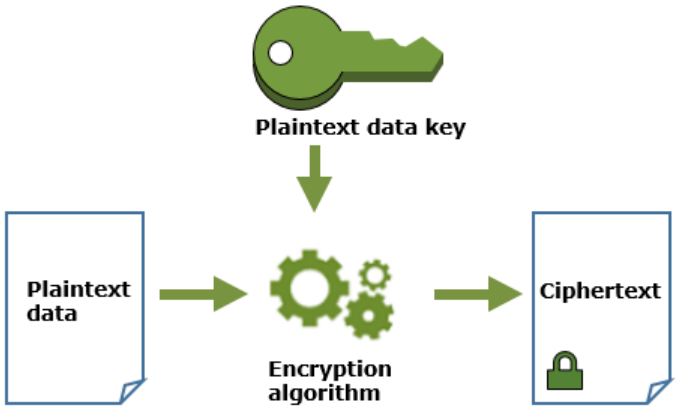


Figura 10. Estructura de cifrado con una clave de datos.
Fuente: (Amazon Web Services, 2018)

2.5.5.7.3. Amazon Cognito

Permite agregar el registro de usuario, el inicio de sesión y el control de acceso a la web y aplicaciones móviles de forma rápida y sencilla, también tiene la opción de autenticar usuarios a través de proveedores de identidad social como Facebook, Twitter o Amazon, con soluciones de identidad SAML, o mediante el uso de su propio sistema de identidad. Además, Amazon Cognito permite guardar datos localmente en los dispositivos de los usuarios, lo que permite que sus aplicaciones funcionen incluso cuando los dispositivos están apagados (Amazon Web Services , 2018).

2.5.5.7.4. AWS Secrets Manager

Ayuda a proteger los secretos necesarios para acceder a las aplicaciones, servicios y recursos de TI. El servicio le permite rotar, administrar y recuperar fácilmente las credenciales de la base de datos, las claves API y otros secretos a lo largo de su ciclo de vida. Los usuarios y las aplicaciones recuperan secretos con una llamada a la API de Secrets Manager, eliminando la necesidad de codificar información confidencial en texto plano. Secrets Manager ofrece una rotación secreta con integración integrada para Amazon RDS para MySQL, PostgreSQL y Amazon Aurora. Además, Secrets Manager le permite controlar el acceso a los secretos mediante el uso de permisos específicos y auditar la rotación secreta de los recursos en la nube de AWS, servicios de terceros y locales (Amazon Web Services , 2018).

2.5.5.8. Machine learning

2.5.5.8.1. Amazon Machine Learning (ML)

Es un servicio robusto basado en la nube que permite a los desarrolladores el uso de la tecnología de aprendizaje automático. Amazon ML proporciona herramientas de visualización y asistentes para la creación de modelos de aprendizaje automático, y facilita la obtención de predicciones para las aplicaciones mediante el uso de API, sin tener que implementar un código de generación de predicción personalizado ni administrar ninguna infraestructura (Amazon Web Services, 2016).

Algunos conceptos se describen a continuación:

- **Fuentes de datos:** contienen metadatos asociados con las entradas de datos a Amazon ML
- **Modelo ML:** genera predicciones utilizando los patrones extraídos de los datos de entrada
- **Evaluación:** mide la calidad de los modelos ML

- **Predicción por lotes:** genera predicciones de forma asíncrona para múltiples observaciones de datos de entrada
- **Predicción en tiempo real**: genera sincrónicamente predicciones para datos individuales

2.5.5.8.1.1. Modelo ML

Un modelo ML es un modelo matemático que genera predicciones al encontrar patrones en sus datos. Amazon ML admite tres tipos de modelos ML: clasificación binaria, clasificación multiclase y regresión.

La siguiente tabla define los tipos de modelos de ML:

Tabla 5
Tipos de Modelo de Amazon Machine Learning

TIPO DE MODELO	DEFINICIÓN
Regresión	• El objetivo de entrenar un modelo ML de regresión es predecir un valor numérico.
Multiclases	• El objetivo de entrenar un modelo ML multiclase es predecir los valores que pertenecen a un conjunto limitado y predefinido de valores permisibles.
Binario	• El objetivo de entrenar un modelo ML binario es predecir valores que solo pueden tener uno de dos estados, como verdadero o falso.

Fuente: (Amazon Web Services, 2016)

2.6. Tipos de Pruebas de Software

La prueba de software es la verificación dinámica del comportamiento de un programa contra el comportamiento esperado, usando un conjunto finito de casos de prueba, seleccionados de manera adecuada (Mera, 2016)

2.6.1. Pruebas de Funcionamiento

Se basan en funciones, prestaciones y la interoperabilidad con sistemas específicos, y pueden llevarse a cabo en todos los niveles de prueba. Es importante mencionar que se orientan en el comportamiento externo de un producto o aplicativo software. (Mera, 2016)

La aplicación es sometida en un escenario de prueba controlado para comprobar que todos los componentes funcionan correctamente. El resultado permite definir requisitos del usuario, requisitos del sistema, casos de uso, etc. Al final se enfocan en la aceptación o cumplimiento de los objetivos definidos al inicio. (Hernández, 2016)

2.6.2. Pruebas de Calidad de Software

Las pruebas o comprobaciones de calidad del código son test realizados sobre la marcha y permiten a los desarrolladores arreglar los problemas antes de que se vuelvan difíciles y más caros de arreglar. El análisis estático del código fuente del proyecto se logra mediante el cumplimiento de estándares en función del tipo de lenguaje de programación. (Hernández, 2016)

Estas pruebas tienen por objeto localizar defectos y comprobar el funcionamiento de módulos software, programas, objetos, clases, etc. Que puedan probarse por separado; es decir, se pueden realizar de manera independiente al resto del sistema en función del contexto. (Mera, 2016)

Sonarqube es el software más popular para realizar análisis estático de código. Es un programa open source, y requiere hacer la instalación en la máquina, y mantenerlo actualizado. A pesar de ello, mantiene una capa gratuita, y otra de pago. Entre las aplicaciones similares en internet se puede encontrar: Codacy, Code Climate, Landscape (únicamente Python), Teamscale, entre otros.

2.6.3. Pruebas de Carga

Este tipo de prueba evalúa el rendimiento del sistema con una carga predefinida. La prueba de carga mide cuánto se tarda un sistema para realizar diversas tareas y funciones del programa bajo condiciones normales o predefinidas. Debido a que el objetivo de las pruebas de carga es determinar si el rendimiento del sistema satisface los requisitos no funcionales de carga, es pertinente determinar, antes de comenzar las pruebas, la configuración mínima y máxima y los niveles de actividad. (Zapata & Cardona, 2011)

Esta prueba da una idea al propietario de la aplicación como actuara su sistema bajo una carga “normal” cuando esté en producción, es decir, comprobar escenarios “reales” para saber si todas las respuestas están dentro de los estándares aceptados (Mera, 2016). Entre los programas se puede encontrar Apache JMeter y HP LoadRunner los cuales se utilizan para generar volumen de carga predefinidos que nos permita analizar y medir el rendimiento del programa.

CAPÍTULO 3

3. ANÁLISIS Y DISEÑO

3.1. Análisis

Para el análisis, diseño, desarrollo e implementación del sistema, se requiere definir y establecer la comunicación entre los enchufes inteligentes HS110 y la plataforma AWS, tal como se observa en la ***Figura 11***.

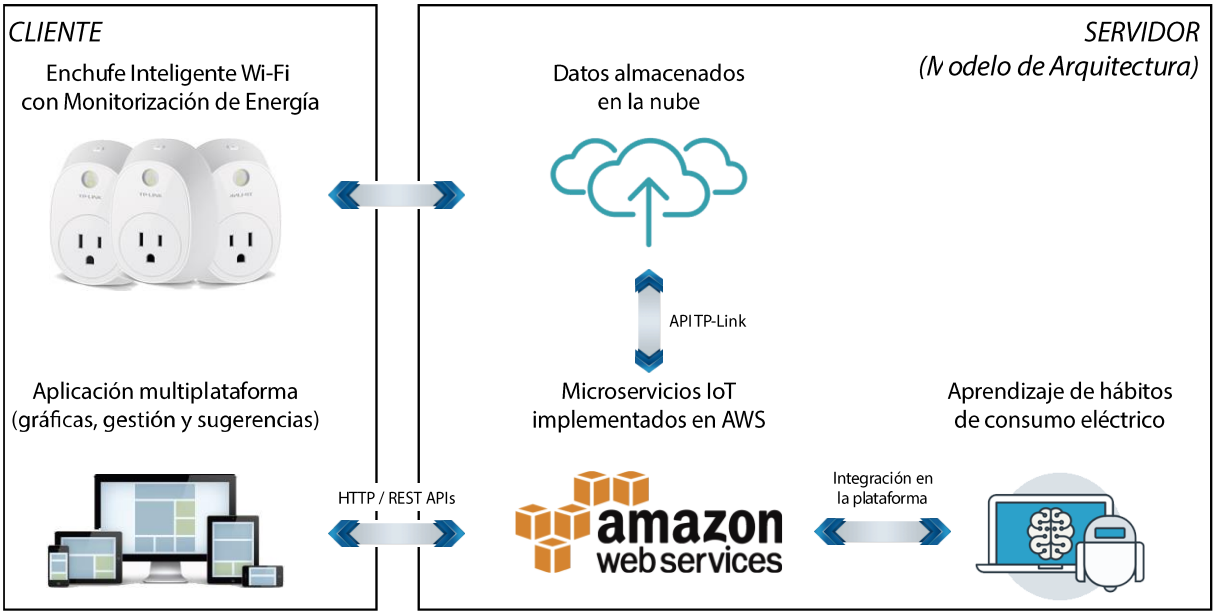


Figura 11. Esquema inicial de la arquitectura de sistema de gestión de energía

Sin embargo, es simplemente un esquema inicial del trabajo de investigación. Para encaminar hacia una propuesta final de arquitectura de sistema de gestión de energía se requiere establecer algunas pautas generales y requerimientos funcionales y no funcionales del sistema descritos a continuación:

3.1.1. Análisis del Enchufe Inteligente HS110

Los enchufes inteligentes son dispositivos IoT que permiten la adquisición de datos de consumo eléctrico y control de los dispositivos electrónicos conectados (computadora de escritorio, microondas y TV). Para ello se requiere el uso del aplicativo móvil llamado Kasa que permite la comunicación con varios dispositivos TP-Link destinados al hogar inteligente. Pese a ello es necesario establecer formas de comunicación con plataformas de cloud computing, entre estas están:

- Comunicación mediante RESTful
- Comunicación mediante API

3.1.1.1. Comunicación mediante RESTful

RESTful es un servicio web que implementa una arquitectura REST, y permite realizar llamadas o peticiones a un servidor web, y de forma inmediata ser respondido mediante JSON. Es decir, se utilizan métodos como GET, POST, entre otros; para establecer comunicación con la plataforma de TP-Link. Cabe destacar que se requiere un análisis a profundidad para conocer específicamente como acceder a los recursos que brinda TP-Link, y por tal motivo realizar una comunicación a bajo nivel.

Mediante el uso de la aplicación "Insomnia" disponible para Linux, MAC y Window, se establece un cliente REST para mantener comunicación con la plataforma TP-Link cuya URL es https://wap.tplinkcloud.com. Para ello se requiere tener activo un cliente en el servidor, y tener registrado al menos uno de los enchufes inteligentes mediante la app Kasa. A continuación, en la ***Figura 12*** se muestra un ejemplo de acceso a la plataforma de TP-Link y en la ***Figura 13*** se muestra un ejemplo para obtener los parámetros como voltaje, corriente o potencia.

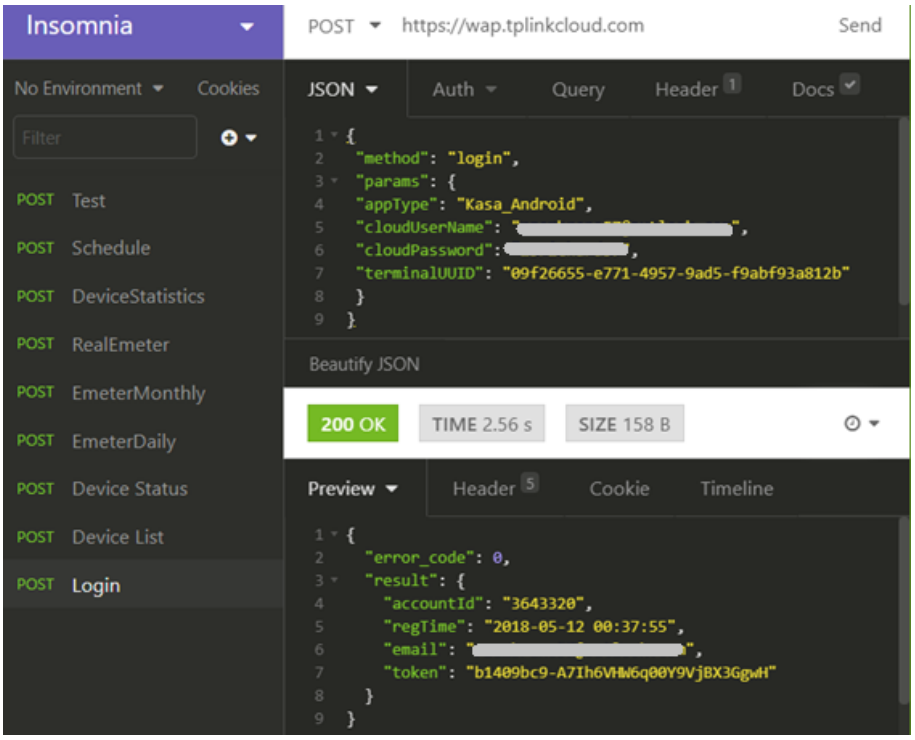


Figura 12. Acceso a la plataforma de TP-Link mediante Insomnia.

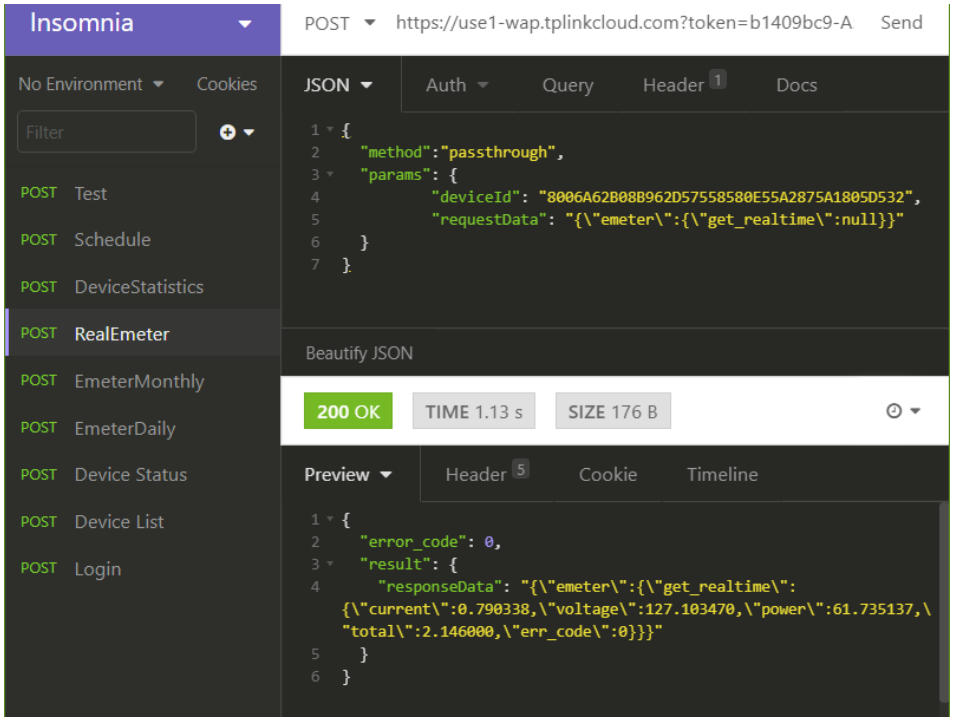


Figura 13. Obtención de parámetros de consumo eléctricos de un enchufe inteligente mediante Insomnia.

Sin embargo, el proceso requiere tener claro cuales recursos solicitar a la plataforma y como realizarlo. Por lo tanto es un proceso complejo y dependiente al 100% de cambio y disponibilidad de la plataforma de TP-Link, además para el acceso a la plataforma se requiere especificar el nombre y la contraseña de la cuenta de TP-Link, disminuyendo la seguridad de los enchufes inteligentes. Por tal motivo, no es una opción vible para el diseño de la arquitectura.

3.1.1.2. Comunicación mediante API

La interfaz de programación de aplicaciones conocida como API permite establecer comunicación entre diferentes servicios. Mediante su búsqueda en la plataforma de desarrollo colaborativo Github se ha encontrado algunas API desarrolladas en diferentes lenguajes de programación, sin embargo, cada una de ellas ofrece ciertas libertades y limitaciones, es así como se ha definido ciertas pautas o requerimientos necesarios para lograr el objetivo, las cuales se enlistan a continuación:

- Capacidad de descubrir los enchufes inteligentes TP-Link.
- Facilidades de comunicación
- Información de dispositivos en tiempo real.
- Información del estado actual.
- Control de encendido y apagado.
- Lenguaje de programación compatible con el kit de desarrollo de software de AWS conocido como AWS SDK, entre los lenguajes que soporta se encuentra: JS, Python, PHP, .NET, Ruby, Java, Go, Node.js, C++.

En función de las pautas o requerimientos establecidos, se han encontrado 2 API que satisfacen las condiciones y cuya comparación general se encuentra en la **Tabla 6**.

Tabla 6

Comparación entre las principales API para comunicación con los enchufes inteligentes HS110

NOMBRE	tplink-cloud-api	tplink-smarthome-api
AUTOR	Alexandre Dumond	Patrick Seal
LICENSIA	GPL-3.0	MIT
LENGUAJE DE PROGRAMACIÓN	TypeScript	JavaScript
DISPOSITIVOS COMPATIBLES	smartplugs (HS100, HS110), smartswitches (HS200), and smartbulbs (LB100, LB110, LB120, LB130)	smartplugs (HS100, HS105, HS110), smartswitches (HS200), and smartbulbs (LB100, LB110, LB120, LB130, LB200, LB230)
CONEXIÓN CON PLATAFORMA DE TPLINK	Si	No
DESCUBRIMIENTO DE DISPOSITIVOS	No	Si
INFORMACIÓN EN TIEMPO REAL	Si	Si
INFORMACIÓN DE ESTADO ACTUAL	Si	Si
TIPO DE CONTROL	Local/Remoto	Local
EVENTOS	No	Si
OTRAS OPCIONES	• Requiere acceso a plataforma TP-Link • Nodejs > v7.7x	• Permite uso de CLI • Permite realizar operaciones a bajo nivel como enviar peticiones RESTful • Detecta nuevos dispositivos conectados a la red • Permite definir temporizadores y reglas de uso • Permite definir eventos de uso, conexión y datos en tiempo real del dispositivo. • Nodejs > v7.7x
SEGUIDORES EN GITHUB	42	471

Fuente: (Dumont, 2018) y (Seal, 2018)

En función a los resultados obtenidos en la **Tabla 6**, ambas API cumplen con la mayoría de los requisitos estipulados, sin embargo, la API desarrollada por (Seal, 2018) contiene mayores libertades en cuanto a funcionalidades y capacidad de soportar eventos (ver todas las funciones y eventos en el Anexo 1). Además, se comunica directamente con los enchufes inteligentes sin necesidad de depender de la plataforma de TP-Link para su funcionamiento, logrando ser totalmente independiente.

A pesar de todo esto, se requiere de un módulo central para establecer la comunicación. Entre las opciones para lograrlo se encuentra el uso de una RASPBERRY PI o un computador, siendo la primera la opción más viable para debido a su bajo costo, tamaño reducido, y suficientes capacidades para la posterior implementación. A partir de esto, en la ***Figura 14*** se ha representado un esquema simplificado de la arquitectura.

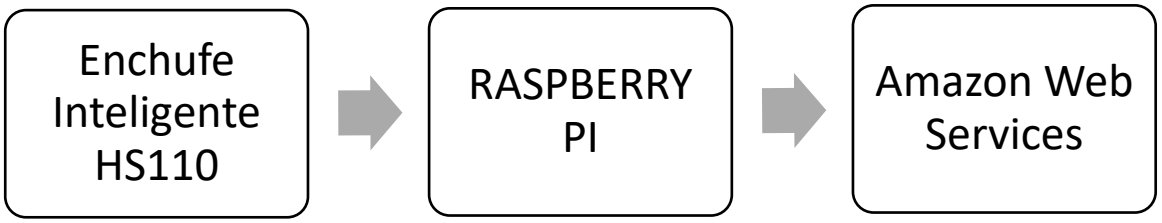


Figura 14. Esquema simplificado de la arquitectura de sistema de gestión de energía.

3.1.2. Análisis de la Raspberry PI

Raspberry PI es un ordenador destinado para programación desarrollado por la Fundación Raspberry PI. Hasta en la actualidad se han lanzado alrededor de 11 modelos, de las cual la Raspberry PI 3 Modelo B+ y la Raspberry PI 3 Modelo B son las que ofrecen mayores beneficios, debido a su velocidad de procesamiento, memoria RAM y conectividad, algunas especificaciones se muestran en la **Tabla 7**.

Tabla 7
Especificaciones de 2 modelos de Raspberry PI con mayores prestaciones

	Raspberry Pi 3 Modelo B	**Raspberry Pi 3 Modelo B+**
SoC	Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + Puerto USB)	
CPU	1.2GHz 64-bit quad-core ARMv8	1.4GHz 64-bit quad-core ARMv8
GPU	• Broadcom VideoCore IV, • OpenGL ES 2.0, • MPEG-2 y VC-1 (con licencia), • 1080p30 H.264/MPEG-4 AVC	
Memoria (SDRAM)	1 GB (compartidos con la GPU)	
Puertos USB 2.0	4	
Conectividad de red	• 10/100 Ethernet (RJ-45) vía hub USB • Wifi 802.11n • Bluetooth 4.1	• 10/100/1000 Ethernet (RJ-45) vía hub USB Max 300Mbits/s • Wifi 802.11n/ac • Bluetooth 4.2 BLE
Consumo energético	800 mA, (4.0 W)	
Fuente de alimentación	5 V vía Micro USB o GPIO header	
Sistemas operativos soportados	• GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, SUSE Linux Enterprise Server for ARM. • RISC OS2	

Fuente: (PiAustralia, s.f.)

Ambos modelos podrían ser utilizados en el presente trabajo de investigación, pese a ello se ha optado por la Raspberry Pi 3 Modelo B como servidor local, debido a su menor coste y mayor facilidad de adquirir en el mercado. A pesar de que el objetivo de la Raspberry Pi es mantener comunicación entre los enchufes inteligentes y la plataforma de AWS, se utilizarán los recursos para implementar una base de datos de backup para el almacenamiento del histórico de los datos generados por los enchufes, y para la programación del cliente y servidor respectivo. Para lo cual se requiere definir librerías npm para el funcionamiento, los cuales se listan a continuación:

- Instalación de API – comunicación con enchufes inteligentes
- Instalación de LAMP (apache, mysql y php) – despliegue de base de datos y centro de desarrollo de página web
- Instalación de phpmyadmin – gestión y visualización de base de datos
- Instalación de AWS SDK – comunicación con servicios de AWS
- Instalación de AWS Amplify – desarrollo de aplicativo móvil
- Instalación de varias dependencias de las librerías descritas previamente

Cabe destacar que la Raspberry Pi es el servidor que proporciona a la plataforma AWS la información requerida de consumo eléctrico, y es así como su entorno de ejecución será Node.js, ya que permite la ejecución de aplicaciones basado en el lenguaje de programación ECMAScript, es decir, JavaScript.

3.1.3. Análisis de Plataforma de AWS

En la plataforma de AWS se encuentran una variedad de servicios, varios de los cuales interactúan entre sí para lograr un objetivo funcional. Previo a ello, se requiere registrar a la plataforma para acceder como Root a todos los servicios disponibles. Mediante el nuevo usuario, es posible acceder a los servicios de AWS a través de la consola, AWS SDK, o AWS CLI.

Cabe recalcar que la plataforma de AWS es un medio por el cual se procesará los datos para ofrecer información en tiempo real de los parámetros de consumo eléctrico, y aplicar un modelo de machine learning para conocer los hábitos de consumo, que posteriormente serán visualizados en una página web. Sin embargo, hay que especificar formas de comunicación tanto con el servidor como con el cliente.

3.1.3.1. Comunicación de la Raspberry Pi

Para mantener comunicación con el servidor local, que anteriormente mencionamos es la Raspberry Pi 3, se debe tener en consideración la compatibilidad con la API que permite la comunicación con los enchufes inteligentes. La API es desarrollada en JavaScript por lo tanto se hará uso de librerías que permitan la comunicación con la plataforma de AWS como: AWS SDK para JavaScript y AWS Amplify. Para ello, por norma de seguridad es necesario crear y acceder a la cuenta de AWS mediante un usuario con funcionalidades reducidas como:

- Acceso mediante programación
- Acceso a la consola de administración de AWS

3.1.3.2. Comunicación entre servicios AWS

La comunicación se realiza de forma transparente para el cliente, sin embargo, protocolos ligeros como HTML son los utilizados para transmitir la información entre microservicios. Para el presente trabajo de investigación se ha establecido 3 principales servicios. Cada uno es un conjunto de microservicios que trabajan entre sí para cumplir el objetivo final. Si bien, todos parten de un servidor local y finalizan en una página web, estos siguen o utilizan varios microservicios de la plataforma de AWS para desarrollar la arquitectura. Los servicios descritos son los siguientes:

- Visualización de parámetros en tiempo real.
- Control de encendido y apagado de los enchufes inteligentes.
- Aprendizaje de hábitos de consumo

3.1.3.2.1. Visualización de parámetros en tiempo real

La API permite la definición de eventos, los cuales se activan cada vez que exista un cambio en los parámetros de consumo eléctrico de cada enchufe inteligente, pese a ello, la visualización de

parámetros en tiempo real requiere de microservicios capaces de soportar gran cantidad de solicitudes de subida y bajada, como también baja latencia.

Para ello es necesario microservicios que brinden:

- Baja latencia
- Capacidad de soportar al menos 180 solicitudes por minuto
- Integración a plataforma web
- Integración con node js
- Carga y descarga selectiva de datos
- Registro a eventos asíncronos (cambio en datos)

3.1.3.2.2. Control de encendido y apagado de los enchufes inteligentes

El proceso de encender y apagar los enchufes inteligentes permite conectar y desconectar cada uno de los dispositivos electrónicos conectados (computadora de escritorio, microondas y TV) a merced del usuario. Desconectar los dispositivos electrónicos beneficia tanto al usuario como al dispositivo. Al usuario le permite disminuir el consumo eléctrico cuando los dispositivos se encuentran apagados, mientras que al dispositivo le protege contra descargas eléctricas que puedan afectar los circuitos electrónicos internos, y con ello daños internos de los dispositivos.

Si bien la visualización de parámetros en tiempo real requiere de un flujo de datos desde el servidor hacia el cliente, el control de encendido y apagado de los enchufes inteligentes requiere de un flujo desde el cliente hacia el servidor. Este último además debe proporcionar a la API los recursos necesarios para lograr el objetivo de encender o apagar un dispositivo en específico. Por lo tanto, los microservicios utilizados en este servicio son los mismos que para el servicio anterior, con la diferencia del sentido del flujo de los datos.

3.1.3.2.3. Aprendizaje de hábitos de consumo

Otro servicio fundamental es el aprendizaje de hábitos de consumo, como su nombre lo dice, se requiere del aprendizaje de los datos obtenidos de los parámetros de consumo eléctrico para generar un modelo que permita visualizar franjas horarias donde los usuarios hacen uso de los dispositivos electrónicos, y con ello conocer los hábitos de consumo.

El microservicio requerido es machine learning, debido a su función base de aprender, y ser capaz de predecir a partir de los datos proporcionados. Previo a ello se requiere extraer, transformar y cargar los datos proporcionados al microservicio.

Para ello el servicio requiere seguir el siguiente procedimiento:

- Definición de datos necesarios para el aprendizaje
- Almacenamiento de los datos históricos en una base de datos
    - Definición de característica de la base de datos
        - Capacidad rentable y de tamaño ajustable
        - Compatibilidad el motor de base de datos de backup
        - Compatibilidad con machine learning
    - Acceso a bases de datos mediante el servidor local
- Extracción de datos históricos
- Transformación de datos para el aprendizaje
- Carga de los datos en un microservicio
    - Archivo almacenado en formato compatible con AWS Machine Learning
    - Archivo almacenado en un microservicio compatible
- Creación del modelo en AWS Machine Learning

- Definición del tipo de modelo a utilizar
- Establecer porcentaje para crear y evaluar el modelo

- Generación de gráficas de hábitos de consumo
  - Segmentación de los datos por hora de cada día de la semana
  - Visualización en la página web

3.1.3.3. Comunicación del Cliente

En cuanto al cliente, debido a que el aplicativo web utiliza las librerías de AWS como AWS SDK y AWS Amplify para el desarrollo de aplicaciones móviles, y en su mayoría utilizan JavaScript como lenguaje de programación, se utiliza HTML y JavaScript para el desarrollo de la página web.

Pese a ello se debe establecer algunas consideraciones:

- Creación de usuarios y acceso a la plataforma
- Capacidad de visualizar información de cada uno de los enchufes inteligentes
- Capacidad de visualizar la información generada por cada servicio
- Capacidad de visualizar información general de la cuenta

3.2. Diseño

Luego del análisis realizado, el siguiente paso corresponde al diseño de la arquitectura que comienza desde la adquisición de los datos en el hogar, la comunicación y ejecución de microservicios en la plataforma AWS, hasta proporcionar información de los dispositivos en tiempo real al cliente mediante una página web.

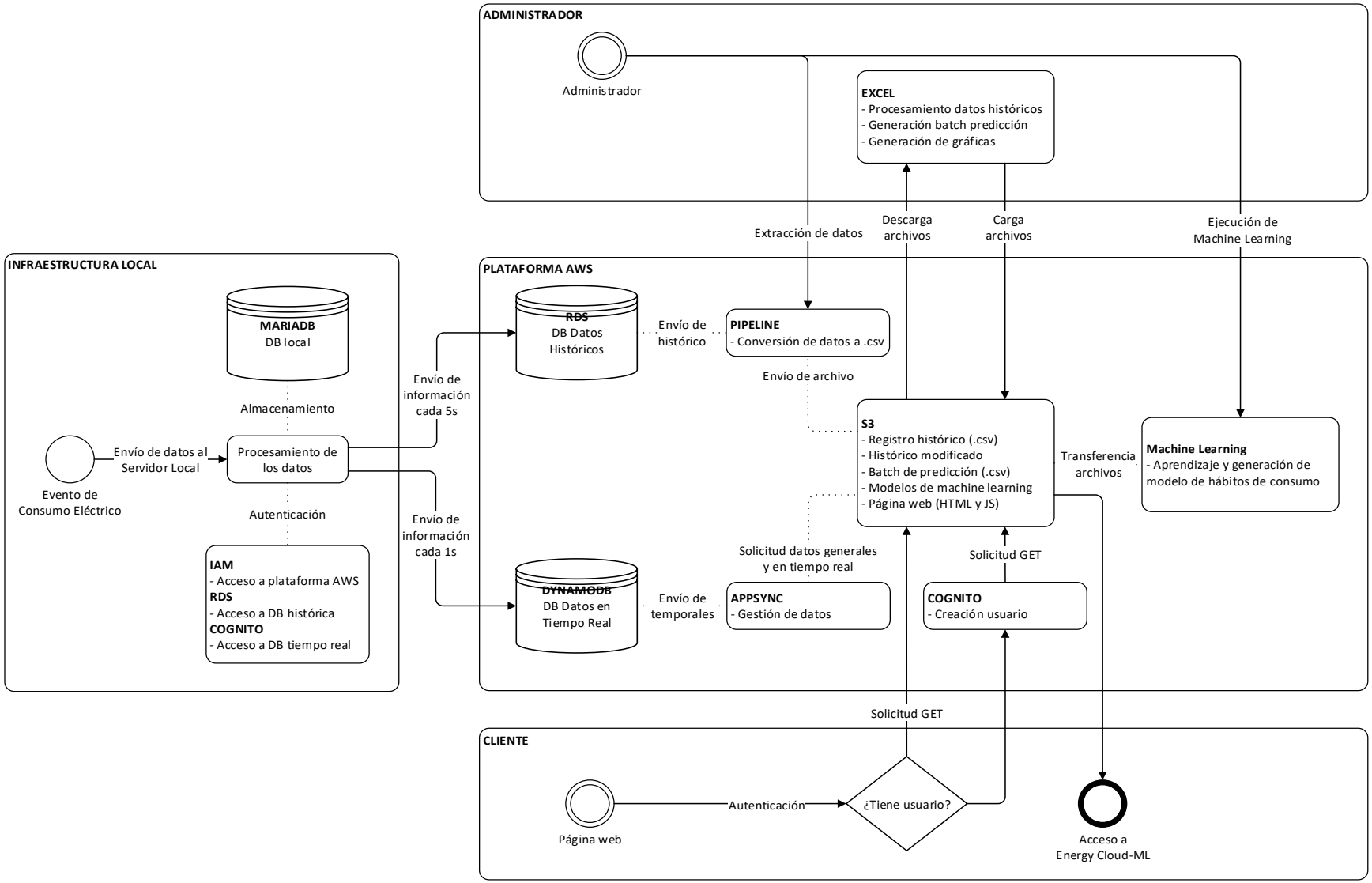


Figura 15. Diagrama BPMN de la arquitectura de sistema de gestión de consumo de energía en el hogar.

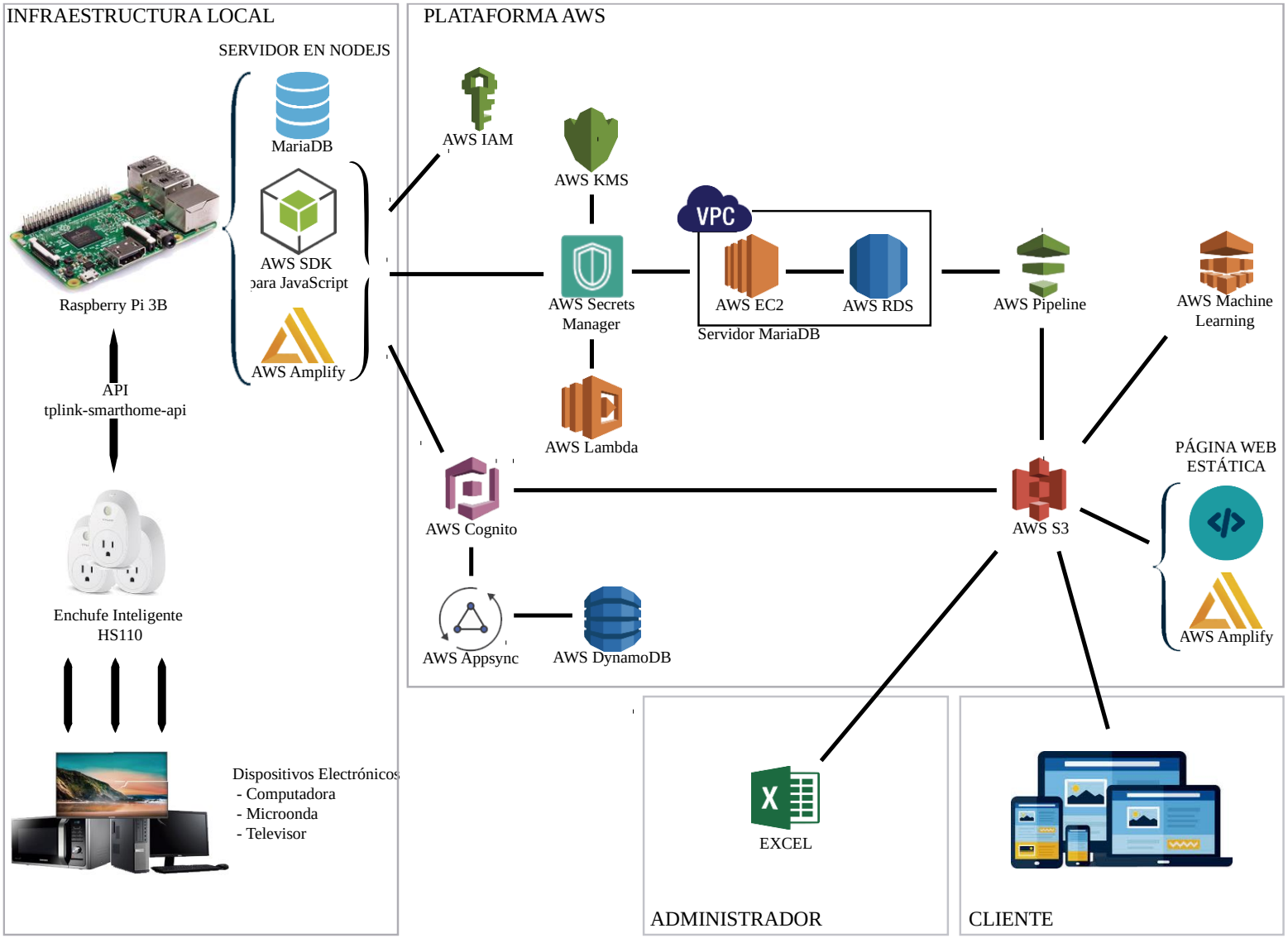


Figura 16. Esquema final de una propuesta de arquitectura de sistema de gestión de consumo de energía en el hogar.

En función de ello, se ha propuesto una arquitectura de sistema de gestión de consumo eléctrico en el hogar, como se puede ver en la ***Figura 16***. La cual consta de 4 partes fundamentales claramente divididas de acuerdo con el lugar donde se ejecuta los procesos, estos son:

- Infraestructura local
- Plataforma AWS
- Administrador
- Cliente

Además. se ha modelado un diagrama BPMN para ver de manera general los procesos inmersos en la propuesta de arquitectura, ver ***Figura 15***

3.2.1. Infraestructura Local

La infraestructura local consta de varios dispositivos físicos conectados y comunicándose entre sí, como se puede observar en la ***Figura 17***.

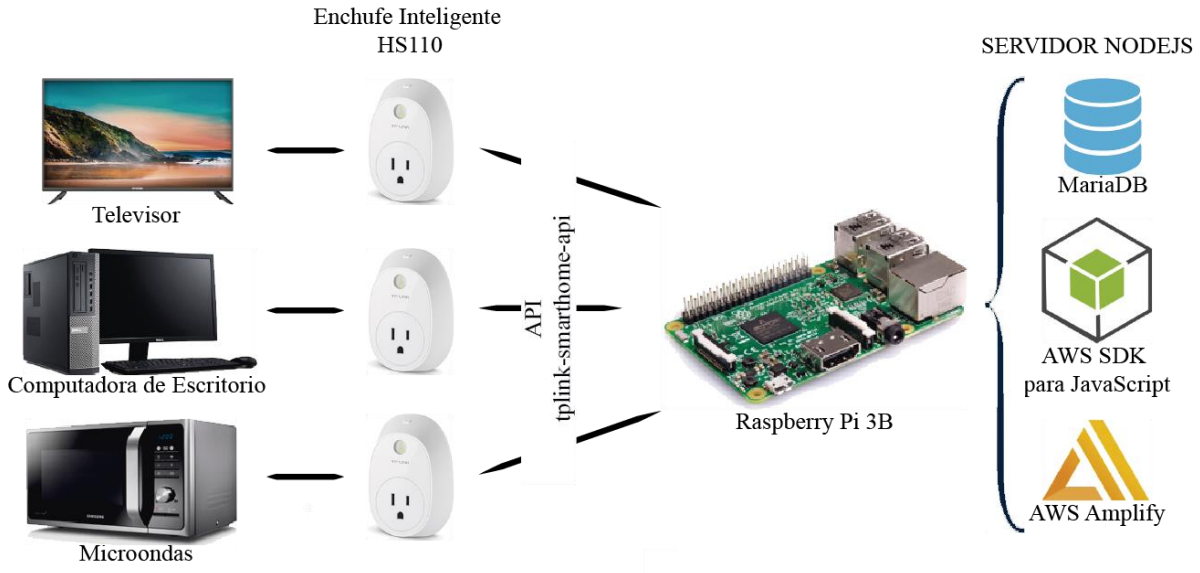


Figura 17. Esquema de conexión en la Infraestructura local.

Cada uno de los dispositivos cumple una función como se puede observar en la **Tabla 8**. Sin embargo, el diseño del servidor Node.js requiere definir con exactitud cuáles son las herramientas, cómo interactúan entre sí y cuál es su función dentro de la arquitectura.

Tabla 8
Función de dispositivos en la Infraestructura local.

Dispositivo	Función
Dispositivos electrónicos	• Ofrecer un servicio específico al usuario mediante el uso de la electricidad. Nota: En el presente trabajo de investigación se utilizará un Computador de escritorio, Microonda y Televisor.
Enchufe inteligente HS110	• Monitorizar el consumo eléctrico de dispositivos electrónicos conectados en tiempo real. • Conectar y desconectar los dispositivos electrónicos conectados del sistema eléctrico.
Raspberry Pi 3B	• Ejecución del Servidor Node.js o Registrar los enchufes inteligentes de la red local. o Adquirir datos relevantes mediante el uso la API. o Comunicar y enviar los datos a microservicios en la plataforma AWS. • Ejecutar base de datos local.

En primera instancia se definirá los diagramas para la comunicación con los enchufes inteligentes, posterior la comunicación con la plataforma AWS, y para finalizar su funcionamiento global.

3.2.1.1. Diseño para comunicación con los enchufes inteligentes

El diagrama de flujo permite conocer el proceso de interacción entre cada una de las funciones para tener comunicación con los enchufes inteligentes HS110. Para ello se ha definido el diagrama de flujo como se observa en la ***Figura 18***. Además, se estableció en la **Tabla 9** varios eventos que permitan ejecutar procesos asíncronos cuando son activados.

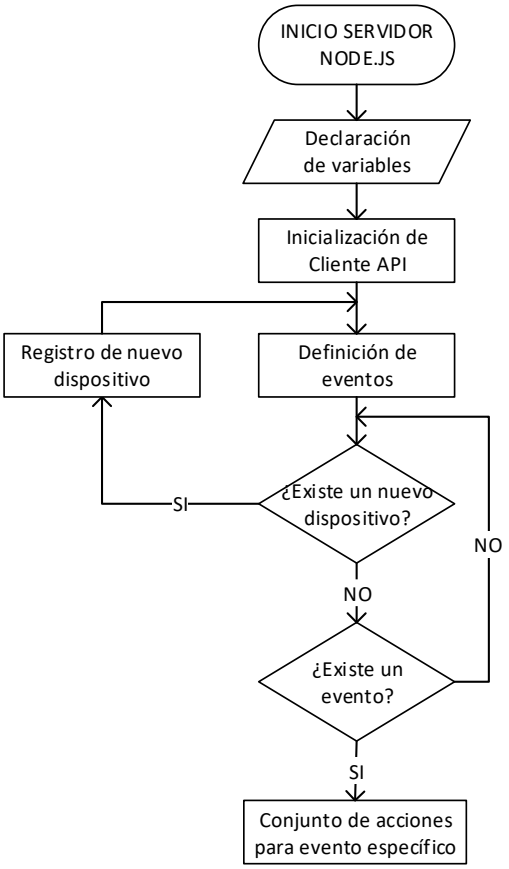


Figura 18. Diagrama de flujo de la comunicación con los enchufes inteligentes.

Tabla 9

Tabla de descripción para cada evento correspondiente al diagrama de flujo.

Evento	**Declaración**	**Función**
emeter-realtime-update	plug.on('emeter-realtime-update', (emeterRealtime) => {})	• Envía los parámetros de consumo eléctrico constantemente.
in-use	plug.on('in-use', () => {})	• Avisa cuando el dispositivo electrónico está encendido constantemente
not-in-use	plug.on('not-in-use', () => {})	• Aviso cuando el dispositivo electrónico se apaga por primera vez
power-on	plug.on('power-on', () => {})	• Avisa cuando el enchufe inteligente está encendido constantemente
power-off	plug.on('power-off', () => {})	• Aviso cuando el enchufe inteligente se apaga por primera vez

3.2.1.2. Diseño para comunicación con la plataforma AWS

Para acceder a la plataforma AWS ya sea mediante AWS SDK o AWS Amplify, se requiere un usuario y contraseña, para ello es necesario definirlo previamente. Entre los atributos requeridos para acceder a la plataforma AWS y microservicios requeridos se encuentran en la **Tabla 10**.

NOTA: Si bien la interrelación de cada microservicio aún no ha sido explicada, en el apartado 3.2.2 se explica a más detalle su diseño. Por lo pronto, estos son los atributos requeridos para establecer comunicación con la plataforma AWS.

Tabla 10

Variables requeridas para acceder a la plataforma AWS e inicializar cada microservicio.

Librería	Microservicio	Atributos	Función
AWS SDK	AWS IAM	• aws_project_region • aws_access_key_id • aws_secret_access_key	• Acceso a los servicios AWS
	AWS Secrets Manager	• aws_secret_manager	• Obtención de claves de acceso para la base de datos.
AWS Amplify	AWS Cognito	• aws_cognito_identity_pool_id • aws_cognito_region • aws_user_pools_id • aws_user_pools_web_client_id • aws_cognito_username • aws_cognito_password	• Acceso a la plataforma AWS mediante un pool de usuario registrado. • Utilizado para ingresar al microservicio AWS Appsync
	AWS Appsync	• aws_appsync_graphqlEndpoint • aws_appsync_region • aws_appsync_authenticationType	• Creación y actualización de datos en tiempo real.

3.2.1.3. Diseño de bases de datos

Existen 2 bases de datos en la presente arquitectura, la primera se encuentra en la infraestructura local, mientras que la otra en la plataforma AWS. Sin embargo, ambas comparten el mismo diagrama de clase descrito en la ***Figura 19***.

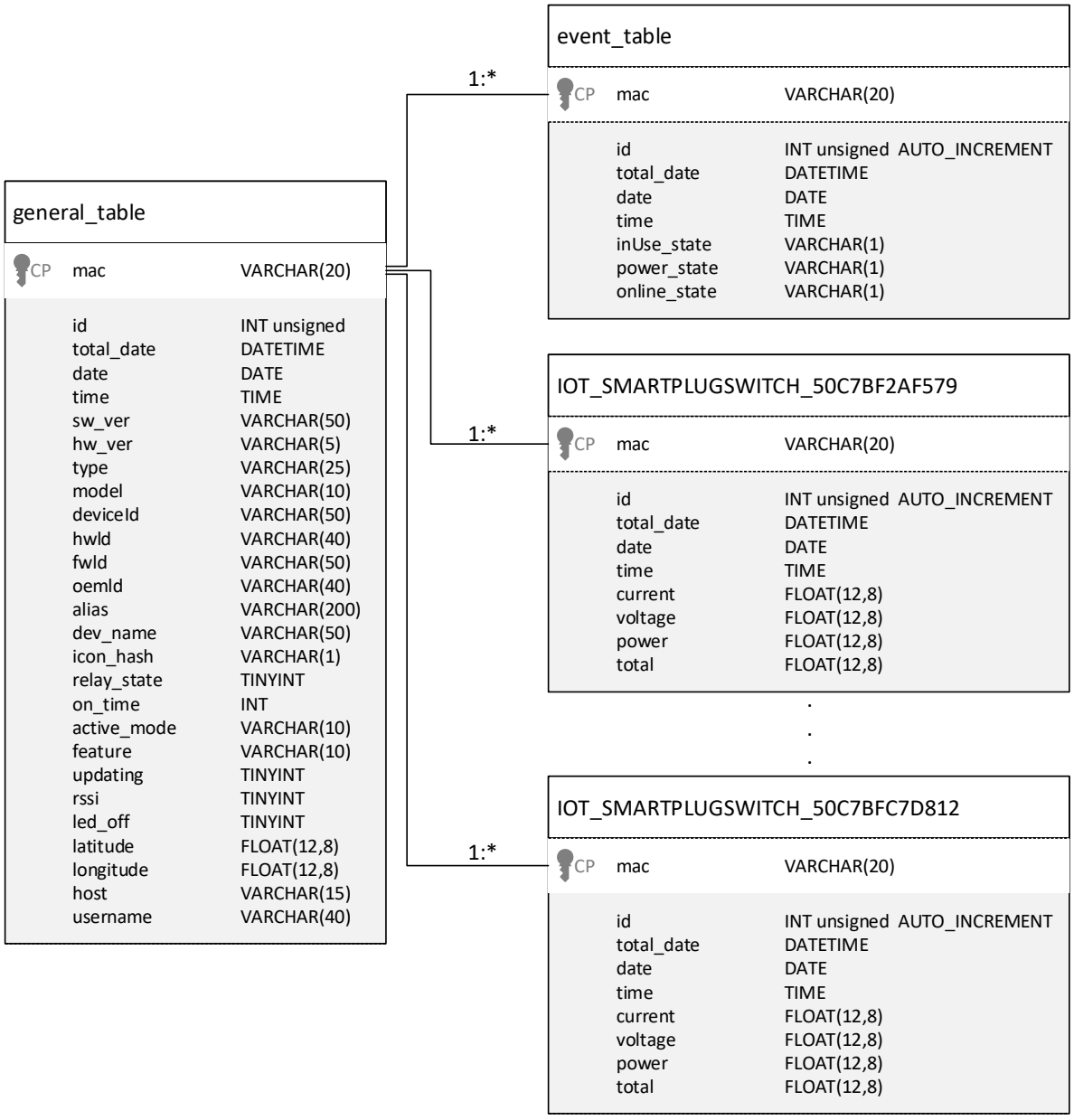


Figura 19. Diagrama de clase de la estructura de base de datos para la infraestructura local y la plataforma AWS.

Todos los parámetros son almacenados en función los datos proporcionados por los enchufes inteligentes, como se observa en la ***Figura 20***.

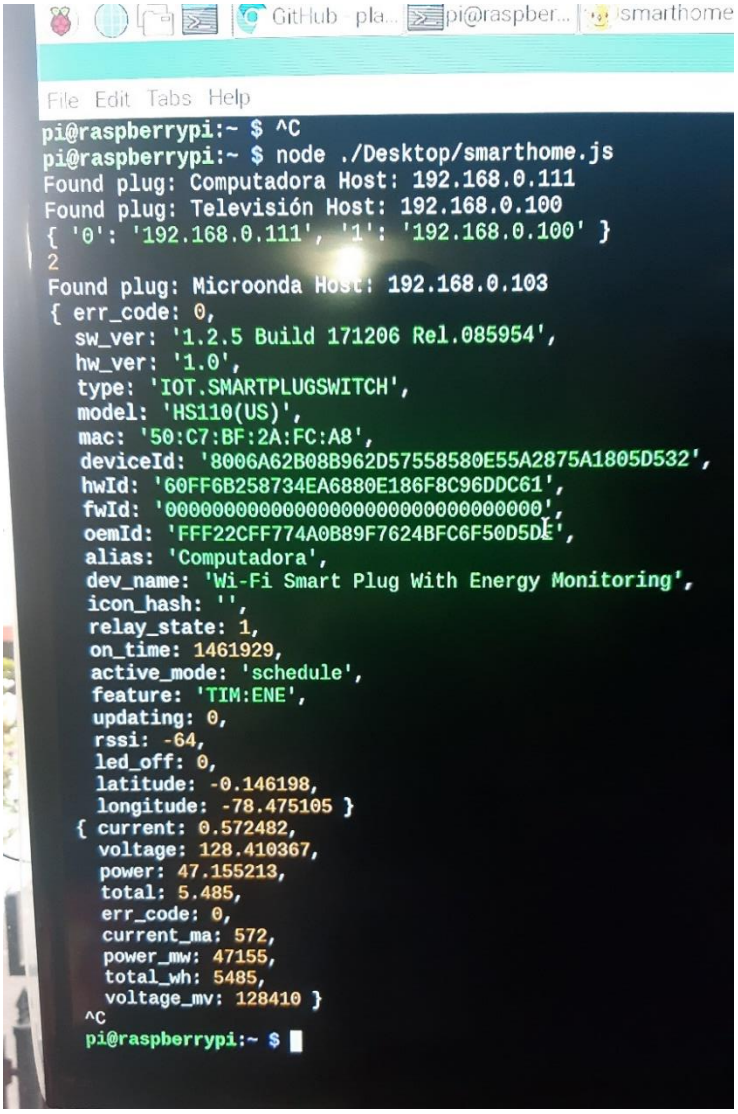


Figura 20. Datos proporcionados por la API, mediante sus funciones y eventos.

Cabe destacar que los datos son proporcionados mediante una estructura JSON, como se puede observar en la ***Figura 20***. Los parámetros son descritos en la **Tabla 11**. Además, el motor de las bases de datos es MariaDB, debido a:

- Compatibilidad y disponibilidad en los repositorios de Raspberry Pi 3B.
- Menores costes para implementación del servidor en la plataforma AWS.

Tabla 11

Descripción de los parámetros mostrados en la Figura 20.

Parámetro	**Descripción**
Err_code	• Define si existe un error o alarma en el dispositivo IoT
Sw_ver / Hw_ver	• Versión de software y hardware
Type / Model	• Tipo y modelo del dispositivo IoT
Mac	• Dirección MAC del dispositivo IoT
deviceId / hwId / fwId / oemId	• ID del dispositivo, hardware, entre otros del dispositivo IoT
Alias	• Nombre asignado al dispositivo IoT
Dev_name	• Nombre genérico del dispositivo IoT
Icon_hash	• Directorio del icono del dispositivo IoT
Relay_state	• Estado del relay
On_time	• Tiempo de actividad del dispositivo IoT
Active_mode / feature / updating	• Atributos del dispositivo IoT
rssi	• Potencia de RX
Led_off	• Estado del indicador led
Latitude / longitude	• Latitud y longitud en formato decimal
Current / current_ma	• Corriente que circula por el dispositivo IoT en amperios y miliamperios
Voltaje / voltaje_mv	• Voltaje del dispositivo IoT en voltios y milivoltios
Power / power_mw	• Potencia consumida por el dispositivo IoT

3.2.1.4. Diseño global de la infraestructura local

Cada uno de los diseños anteriores se analizaron independientemente, ahora se va a agrupar todos ellos para comprender como se relacionan estos, tal como se puede observar en la ***Figura 21***. Previo al desarrollo de la programación respectiva, se necesita definir todas las librerías que serán ocupadas, y su función dentro del servidor, como se observa en la **Tabla 12**.

Tabla 12
Librerías requeridas para el servidor Node.js.

Librería	Versión	Función
aws-amplify"	1.1.16	• Comunicación con AWS Appsync mediante AWS Cognito.
aws-sdk	2.326.0	• Comunicación con microservicios de la plataforma AWS
date-and-time	0.6.3	• Creación de formato específico para fecha y tiempo local
es6-promise	4.2.5	• Permite la definición de promesas para ES6
mariadb	2.0.2-rc	• Comunicación con DB que se ejecuten con el motor de MariaDB
tplink-smarthome-api	0.23.1	• Comunicación con enchufes inteligentes HS110
ws	6.0.0	• Generación de servidor que permita compatibilidad entre ES5 y ES6.
webpack	4.20.2	• Permite agrupar o empaquetar los archivos de JavaScript.
webpack-cli	3.1.2	• Habilita las funciones CLI de webpack

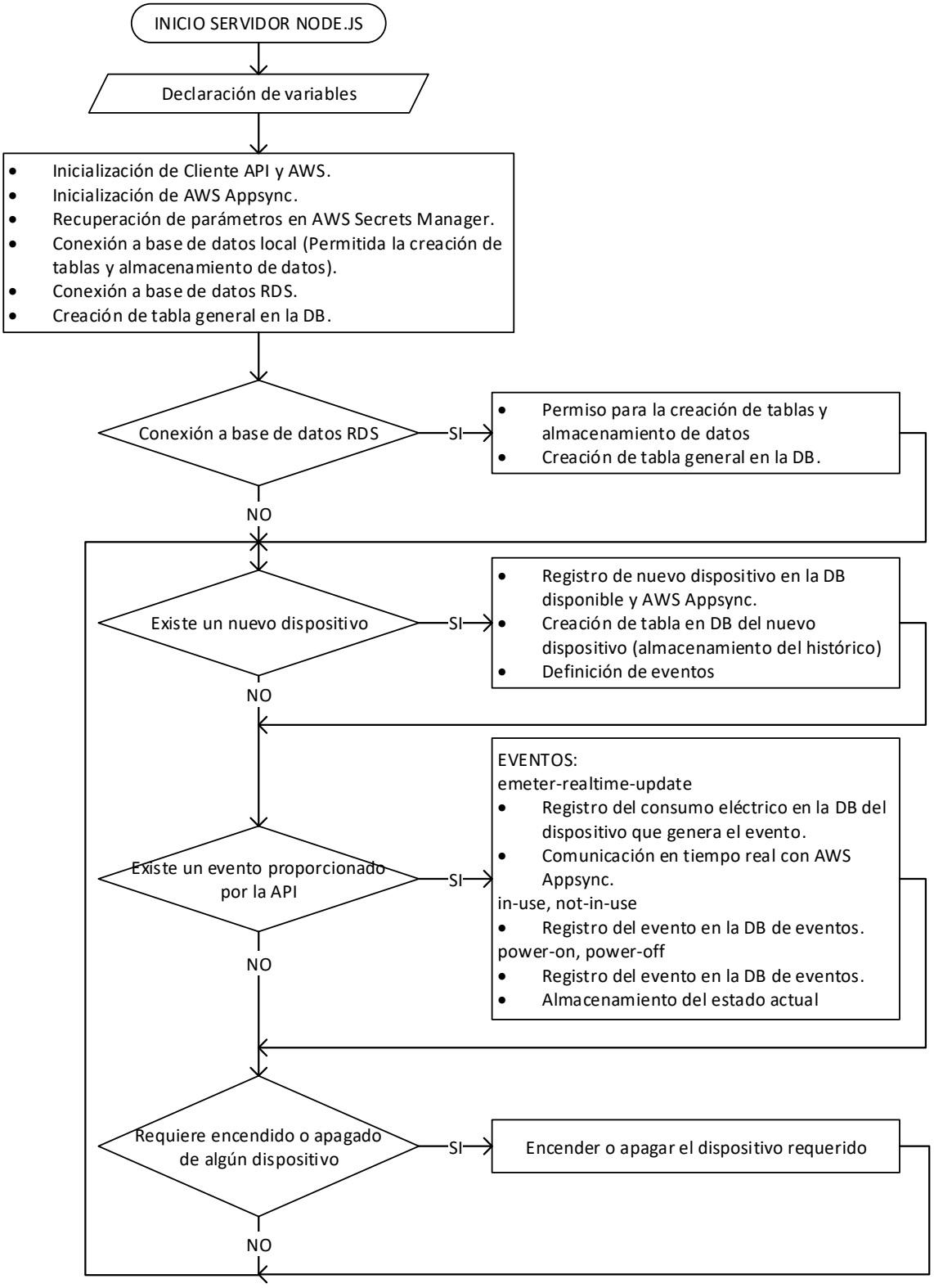


Figura 21. Diagrama de flujo de la infraestructura local.

3.2.2. Plataforma AWS

La relación entre microservicios en la plataforma AWS permite desarrollar servicios fundamentales, sin embargo, se requiere de un diseño previo que permita mostrar como cada microservicio interactúa para lograr su objetivo final.

El análisis de la comunicación de microservicios de la plataforma AWS hizo énfasis en la existencia de 3 servicios principales para el presente trabajo de investigación, pese a ello, 2 de estos comparten los mismos microservicios con la diferencia del flujo de los datos, es decir, la visualización de parámetros en tiempo real requiere de un flujo de datos desde el servidor hacia el cliente, mientras, el control de encendido y apagado de los enchufes inteligentes requiere de un flujo desde el cliente hacia el servidor.

Por lo tanto, en el diseño se ha considerado 2 servicios principales:

- Transmisión de datos en tiempo real
- Aprendizaje de hábitos de consumo

Además, el modo de ingreso es mediante un usuario Root, pero esto trae consigo acceso sin límites a todos los microservicios y servidores que proporciona la plataforma AWS. Para ello, el diseño requiere la creación de un nuevo usuario con beneficios limitados, únicamente para la administración de servicios en la consola de AWS, medio que permite acceder a cada servicio en la nube. Esto se realiza mediante el microservicio AWS IAM.

3.2.2.1. Transmisión de datos en tiempo real

En la ***Figura 22***, el esquema de la transmisión de datos en tiempo real comienza desde el servidor Node.js, hasta cuando los datos son visualizados en la página web. Sin embargo, se requiere varios microservicios en medio para el procesamiento respectivo, además es necesario ingresar a la plataforma de AWS de forma segura, para luego hacer uso de los microservicios.

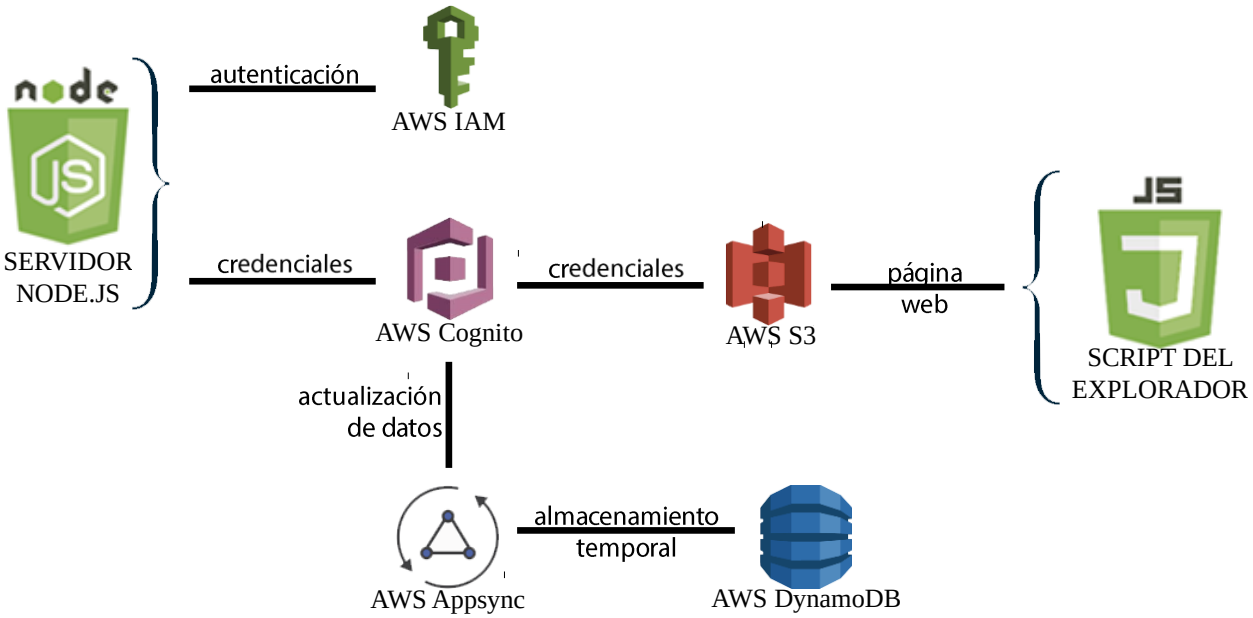


Figura 22. Esquema del servicio para la transmisión de datos en tiempo real.

El acceso a la plataforma AWS se debe realizar mediante un usuario y contraseña a través de AWS IAM (Identity and Access Manager), luego se puede acceder a cada microservicio mediante la AWS SDK en la Infraestructura local. Sin embargo, todo el desarrollo e implementación de los microservicios se realiza mediante la consola de AWS debido a su facilidad de administración y configuración.

La transmisión de datos en tiempo real hace uso de AWS Appsync como pieza clave. El microservicio permite manipular y consultar los datos mediante esquemas GraphQL, para transmitir los parámetros de consumo eléctrico, y enviar señales de control desde el cliente. Sin embargo, requiere un almacén de datos para almacenar la información. AWS DynamoDB es la base de datos que cumple los requerimientos mencionados en el análisis respectivo.

Los parámetros que se almacenarán se observan en la **Tabla 13**.

Tabla 13
Parámetros almacenados en AWS DynamoDB para manipular y consultar mediante AWS Appsync.

Parámetros	Tipo	Función
mac: **name:** **type:** **host:** **latitude:** **longitude:**	String! String String String Float Float	• Transmitir información general de cada dispositivo
mac: **date:** **voltage:** **current:** **power:** **state:**	String! AWSDateTime Float Float Float Boolean	• Transmitir parámetros de consumo eléctrico y estado actual del dispositivo
mac: **control:**	String! Boolean	• Enviar señales de control para encender y apagar cada enchufe inteligente.

AWS Appsync ofrece varios tipos de acceso al microservicio, para mayor seguridad y enfocado en el desarrollo de aplicaciones. Estas opciones son:

- AWS IAM
- Amazon Cognito User Pool
- OpenID Connect
- API key

Cada uno tiene características definidas, pese a ello, Amazon Cognito ofrece capacidades para crear usuarios y acceder a microservicios como AWS Appsync mediante aplicaciones móviles de forma rápida y simple. El proceso de autenticación se observa en la ***Figura 23***.

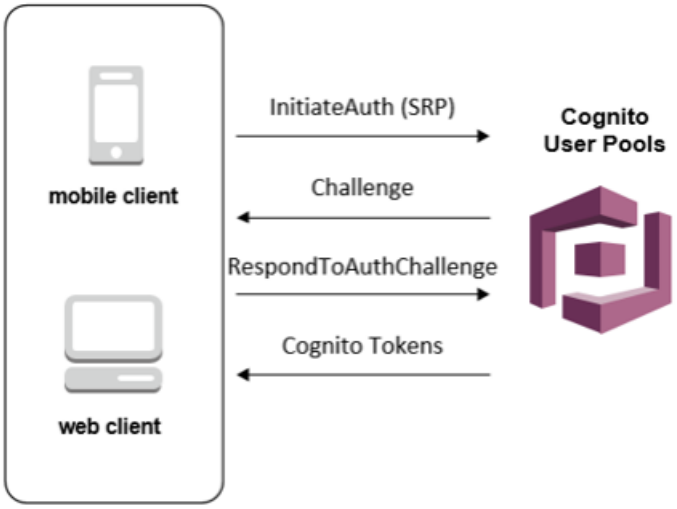


Figura 23. Proceso de autenticación de Amazon Cognito.
Fuente: (Amazon Web Services, 2018)

El usuario debe registrarse al pool de usuarios, mediante atributos como nombre de usuario y contraseña, además el servicio ofrece varios atributos adicionales como en la ***Figura 24***.

Obligatorio	Atributo	Obligatorio	Atributo
☐	address	☐	nickname
☐	birthdate	☐	phone number
☑	email	☐	picture
☐	family name	☐	preferred username
☐	gender	☐	profile
☐	given name	☐	zoneinfo
☐	locale	☐	updated at
☐	middle name	☐	website
☐	name		

Figura 24. Atributos para crear usuarios en Amazon Cognito.
Fuente: (Amazon Web Services , 2018)

La dirección de email o el número celular son atributos para verificar o supervisar la creación del usuario. Un código de verificación es enviado a estos, previo al registro en la plataforma, en caso de no aceptar la creación del usuario, el proceso se elimina en 7 días. En la propuesta de arquitectura, la dirección de email es el atributo para verificar la creación de la cuenta.

3.2.2.2. Aprendizaje de hábitos de consumo

En la ***Figura 25***, el esquema del aprendizaje de hábitos de consumo comienza desde el servidor Node.js, hasta la visualización de datos en la página web. En términos generales, el servicio almacena los datos en una instancia de bases de datos, extrae los datos y almacena en un archivo formato .csv (compatible con AWS Machine Learning), posteriormente lo transforma y procesa los datos para cargar en AWS Machine Learning, para luego generar las gráficas para la visualización de los hábitos de consumo eléctrico en el hogar.

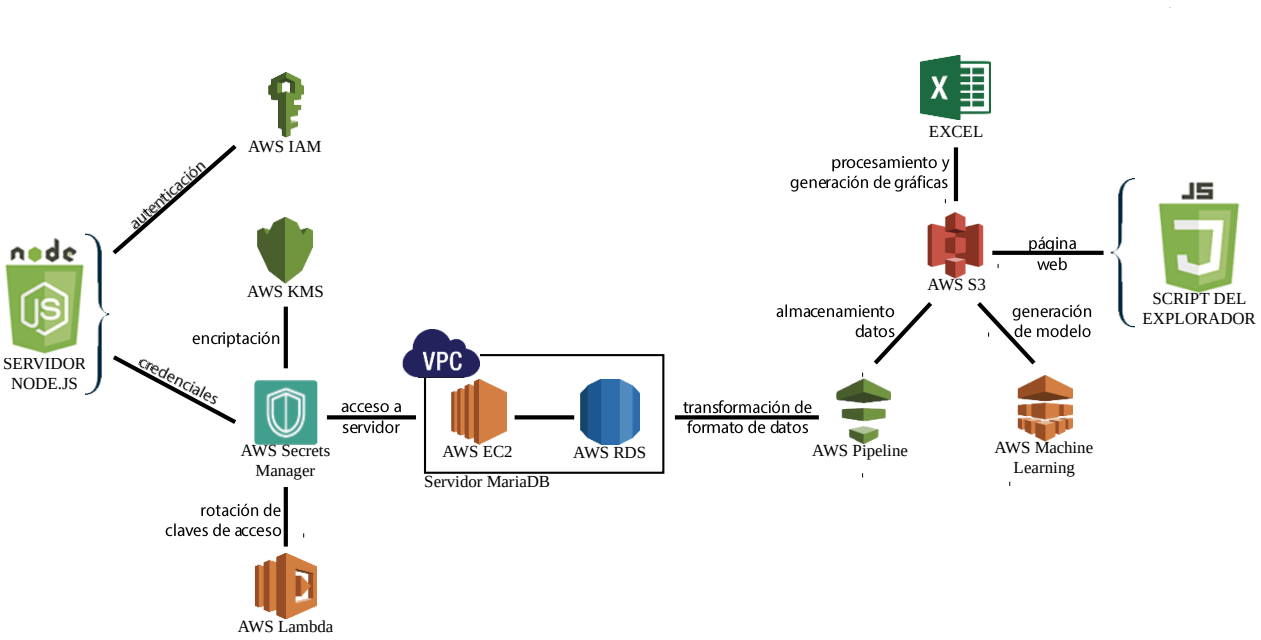


Figura 25. Esquema del servicio para aprendizaje de hábitos de consumo.

Sin embargo, cada uno de los microservicios debe ser diseñado de tal manera que sean capaces de interactuar armónicamente sin afectar al resultado final. Tomando como referencia a AWS Machine Learning, en primera instancia se define los atributos que permitan la creación del modelo con su respectivo tipo de datos y función, como se observa en la **Tabla 14**. Los datos suministrados son divididos en 70% para la creación del modelo y el 30% restante para la evaluación.

Tabla 14
Atributos definidos para el aprendizaje de hábitos de consumo mediante AWS Machine Learning.

Atributo	Tipo	Función
day_week:	Numérico	• Permite aprender de los hábitos de consumo de días diferentes de la semana.
hour:	Numérico	• Permite aprender de los hábitos de consumo en las diferentes horas el día.
min:	Numérico	• Permite aprender de los hábitos de consumo en los diferentes minutos de la hora.
use:	Binario	• Objetivo clave para la predicción. • Obtenido mediante el procesamiento del histórico de potencia almacenada en la base de datos.

A continuación, se define las fuentes de los datos compatibles con AWS Machine Learning:

- AWS S3 – Storage
- AWS Redshift – Base de datos
- AWS RDS – Base de datos

El análisis establecido anteriormente permite definir a AWS RDS como la fuente de datos requerida para el desarrollo del presente servicio. La capacidad de compatibilidad con la base de datos local y la capacidad dinámica del servidor, son esenciales para implementar el servicio.

AWS RDS es una base de datos implementada en una instancia de servidor sobre AWS EC2. Además, junto con AWS VPC permite ofrecer mayor seguridad a la base de datos mediante el filtrado de direcciones IP y protocolos especificados en un grupo de seguridad, solo las IP y protocolos permitidos son capaces de acceder al servidor.

A pesar de que AWS RDS es una fuente de datos compatible con AWS Machine Learning, requiere procesar y mover los datos hacia un bucket de AWS S3 en un archivo .csv. AWS S3 es el microservicio de almacenamiento por defecto para mantener comunicación con AWS Machine

Learning, debido a que permite almacenar archivos en formato csv Formato requerido para enviar los datos al microservicio y generar los modelos, evaluaciones, y predicciones. Para ello se hace uso de AWS Pipeline para capturar los datos en AWS RDS y almacenarlos en un archivo csv en AWS S3.

Sin embargo, los datos almacenados en AWS RDS son primitivos y no refleja los parámetros definidos en la **Tabla 14** para el aprendizaje de los hábitos de consumo. El microservicio AWS Glue permite realizar la transformación y procesamiento de la información, pero aún se encuentra en desarrollo y no ofrece las libertades requeridas para procesar los datos. Por ello, se hace uso de herramientas de cálculo como Microsoft Excel o Matlab, para procesar los datos y transformar en la estructura de datos requerida.

Al momento se ha definido únicamente microservicios en la plataforma AWS, pero ahora, el actor clave es el administrador, quien, mediante el uso de Excel, debido a que es la hoja de cálculo más utilizada a nivel mundial, permitirá manipular la información mediante modelos de datos. Aun así, Excel también permitirá generar batch de datos en archivos csv para realizar la predicción correspondiente en AWS Machine Learning. El batch de datos es un modelo de datos conformado por la permutación de los siguientes valores:

- Días de la semana
- Horas del día
- Minutos de la hora

El batch de datos es almacenado en AWS S3 y luego procesado en AWS Machine Learning para obtener la predicción correspondiente a todo el modelo de datos generado con anterioridad. La predicción es almacenada en AWS S3. Ahora el administrador podrá descargar la información

y con el uso de Excel podrá realizar el diagrama de barras correspondiente para luego subirla a la página web del cliente.

3.2.3. Cliente

Cliente es la persona o grupo de personas que podrá hacer uso de la arquitectura para realizar la gestión del consumo eléctrico en el hogar, esto será realizado mediante una aplicación web capaz de:

- Acceso mediante usuario y contraseña personal
- Creación de usuarios
- Visualización de datos generales de cada enchufe inteligente
- Encendido y apagado de enchufes inteligentes
- Visualización de datos y gráficos de parámetros de consumo eléctrico en tiempo real
- Visualización de hábitos de consumo
- Visualización de datos generales de la cuenta

3.2.3.1. Aplicación Web

El diseño del aplicativo web consta de 2 partes fundamentales, por una parte, el diseño gráfico y por otra el diseño funcional.

3.2.3.1.1. Diseño gráfico de la aplicación web

El mockup es una maqueta o modelo que hace referencia al diseño gráfico y web, para visualizar el aplicativo web antes de su desarrollo. Mediante herramientas online como “moqups”, se ha diseñado la ***Figura 26***, ***Figura 27***, ***Figura 28***, y ***Figura 29*** como modelo de la aplicación web.

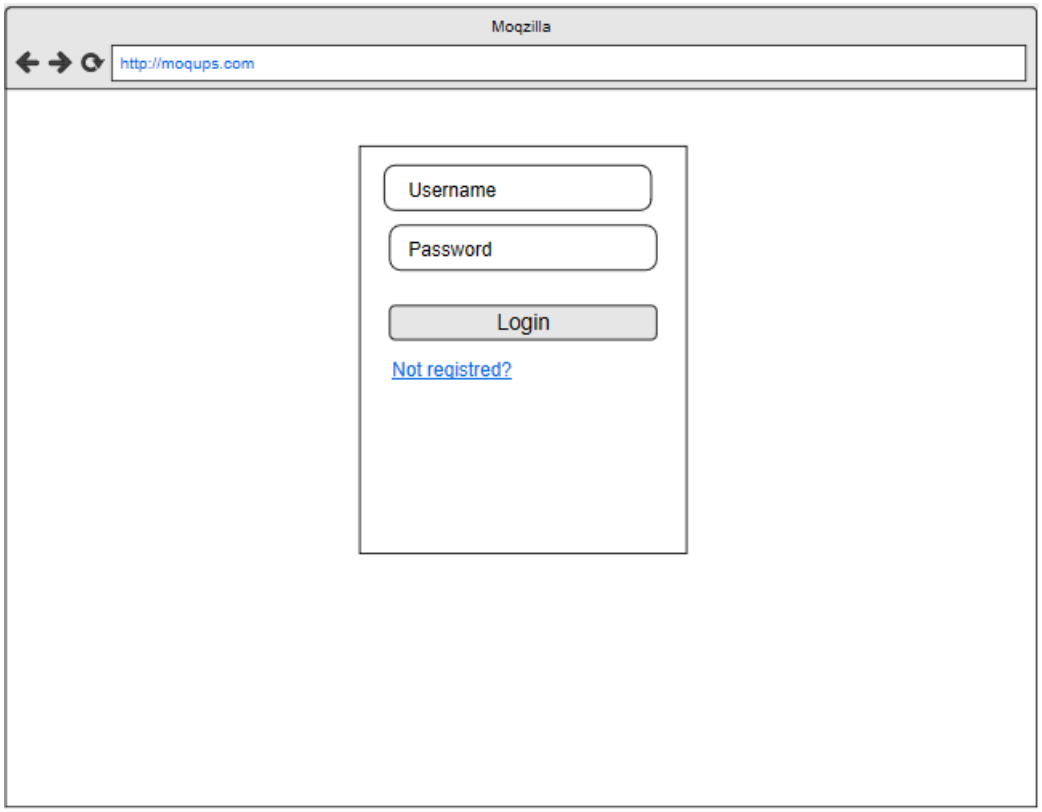


Figura 26. Diseño gráfico de la página principal para acceder a la página web de Energy Cloud - ML

El diseño gráfico de la ***Figura 26*** permite acceso a la página web de "Energy Cloud – ML", nombre definido para la página web debido a que se basa en la gestión de energía, es desarrollado en una plataforma cloud y además utiliza machine learning.

Como se observa en la ***Figura 26***, el acceso se realiza mediante el uso del nombre de usuario y contraseña, los cuales han sido creados y almacenados en el microservicio AWS Cognito de la plataforma AWS. Además, en caso de que no contar con una cuenta específica, es posible acceder a la página web de registro, ver ***Figura 27***. Mediante el ingreso del nombre de usuario, correo electrónico, contraseña, y código de verificación, es posible crear un nuevo usuario.

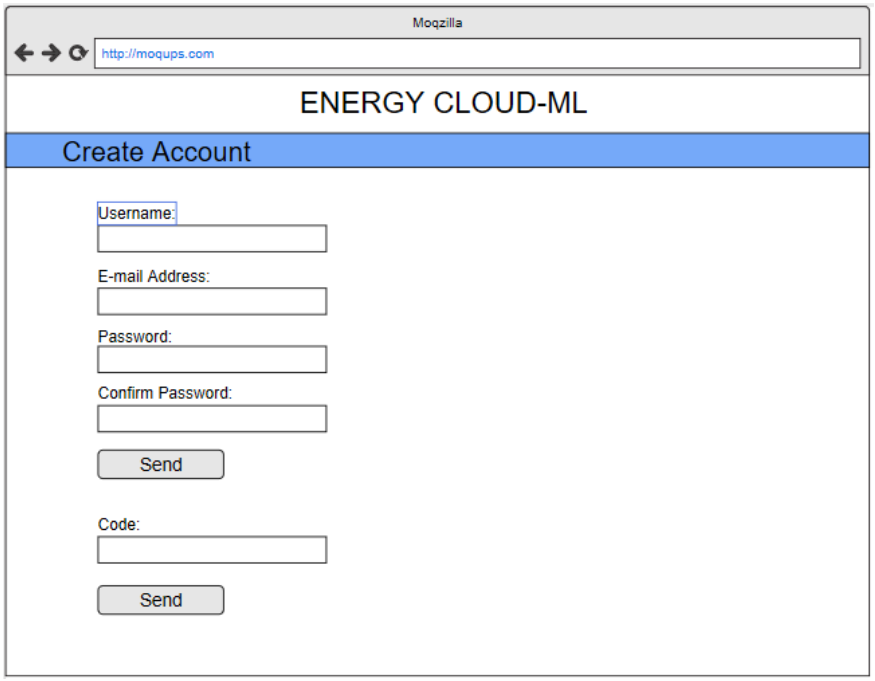


Figura 27. Diseño gráfico de la página de registro para crear usuarios y acceder a Energy Cloud - ML

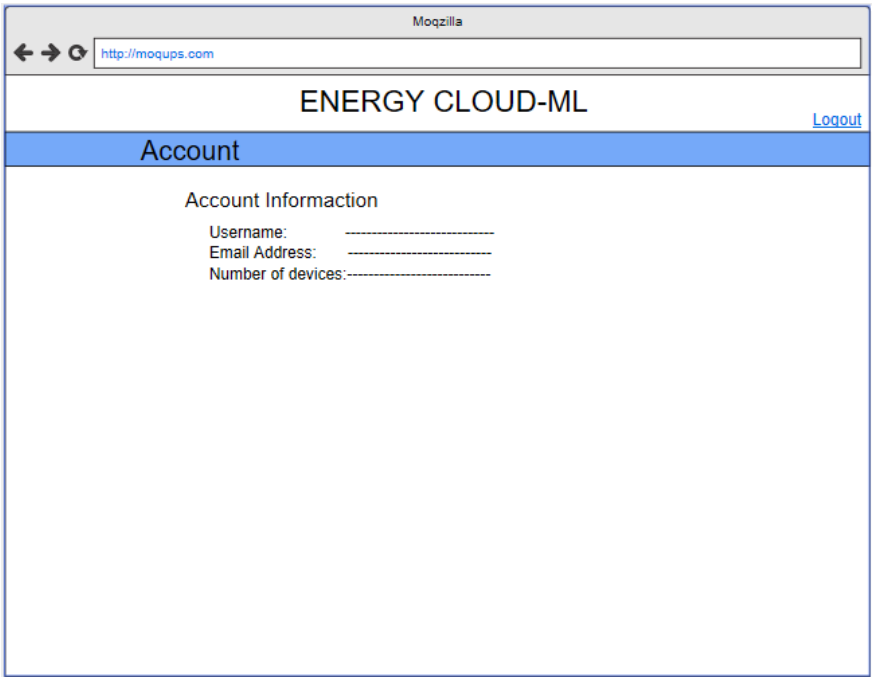


Figura 28. Diseño gráfico de la página principal para visualizar la información general de la cuenta de usuario.

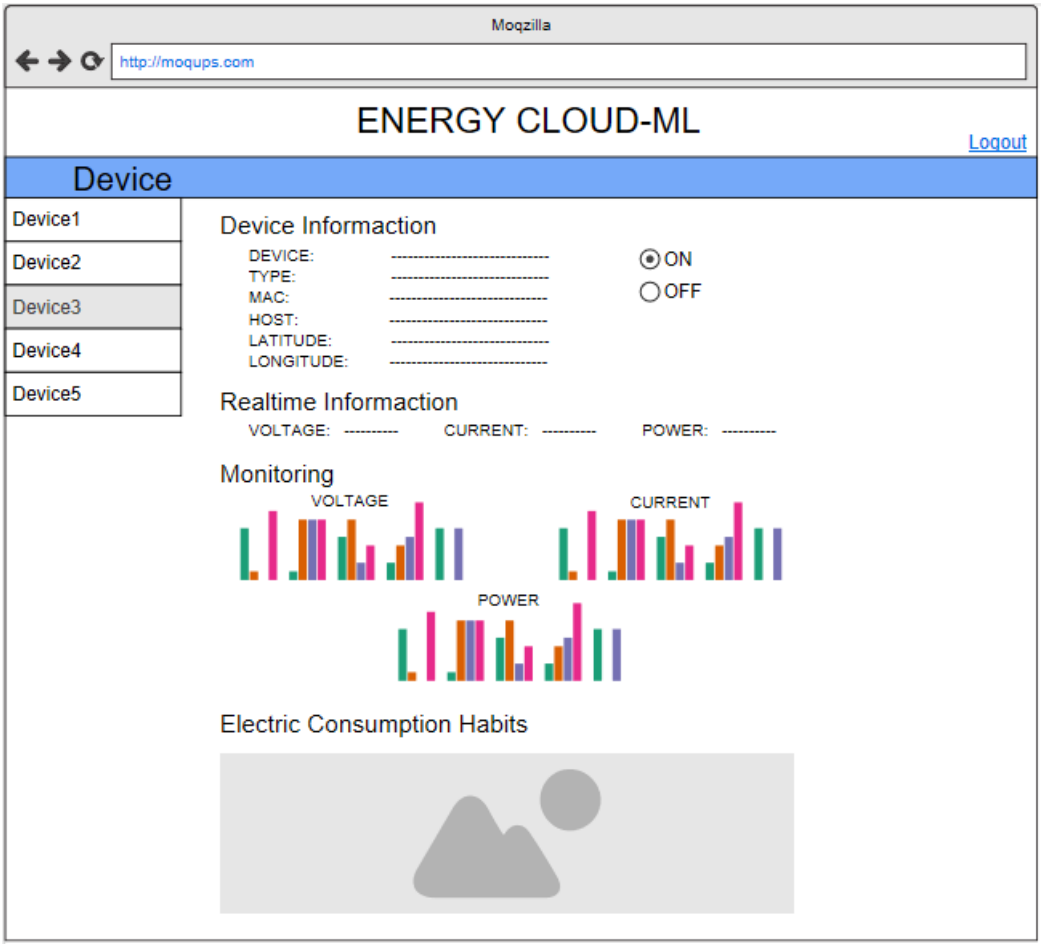


Figura 29. Diseño gráfico de la página principal para visualizar los valores y gráficos en tiempo real, visualizar la información de cada enchufe inteligente, y visualizar los hábitos de consumo eléctrico.

La ***Figura 28*** y ***Figura 29*** son menús de la página web principal de "Energy Cloud - ML". El primero llamado dispositivos (Devices), permite visualizar la información relevante correspondiente a cada uno de los dispositivos disponibles en el submenú. La información fue dividida en 4 bloques como se observa en la **Tabla 15**. El segundo corresponde al menú cuenta (Account), la información proporcionada es:

- Nombre de usuario
- Correo electrónico
- Número de dispositivos

Tabla 15
Descripción de los bloques establecidos en el menú Dispositivos (Devices)

Bloque	Función
Información del dispositivo (Device Information)	• Información de cada enchufe inteligente (Nombre de dispositivo, tipo, mac, host, latitud y longitud)
Información en tiempo real (Realtime information)	• Información en tiempo real de parámetros de consumo eléctrico (voltaje, corriente, potencia)
Monitoreo (Monitoring)	• Gráfica de los parámetros de consumo eléctrico en tiempo real
Hábitos de consumo de energía (Energy consumption habits)	• Imagen de Gráfica de los hábitos de consumo eléctrico generado por el administrador en Machine Learning

3.2.3.1.1. Diseño funcional de la aplicación web

La aplicación web es desarrollada mediante el protocolo HTML en conjunto con JavaScript, y un conjunto de librerías descritas en la **Tabla 16**.

Tabla 16
Librerías requeridas para el aplicativo web basado en HTML y JavaScript

Librería	Versión	Función
aws-amplify"	1.1.16	• Comunicación con AWS Appsync mediante AWS Cognito.
momentjs	2.0.0	• Creación de formato específico para fecha y tiempo local
chart.js	2.7.3	• Generación de gráficas
webpack	4.20.2	• Permite agrupar o empaquetar los archivos de JavaScript.
webpack-cli	3.1.2	• Habilita las funciones CLI de webpack
webpack-dev-server	3.1.5	• Desplegar el servidor html.
copy-webpack-plugin	4.5.2	• Copia archivos especificados para construir el archivo de ejecución.

Además, para acceder a la plataforma AWS se requieren los atributos de la **Tabla 17**.

Tabla 17
Variables requeridas para acceder a la plataforma AWS mediante el aplicativo web

Librería	Microservicio	Atributos	Función
AWS Amplify	AWS Cognito	• aws_cognito_identity_pool_id • aws_cognito_region • aws_user_pools_id • aws_user_pools_web_client_id	• Acceso a la plataforma AWS mediante un pool de usuario registrado. • Utilizado para ingresar al microservicio AWS Appsync
	AWS Appsync	• aws_appsync_graphqlEndpoint • aws_appsync_region • aws_appsync_authenticationType	• Creación y actualización de datos en tiempo real.

A partir del uso de librerías y atributos para acceder a la plataforma AWS, se realiza los diagramas de flujo para:

- Acceder a plataforma AWS, ver ***Figura 30***.

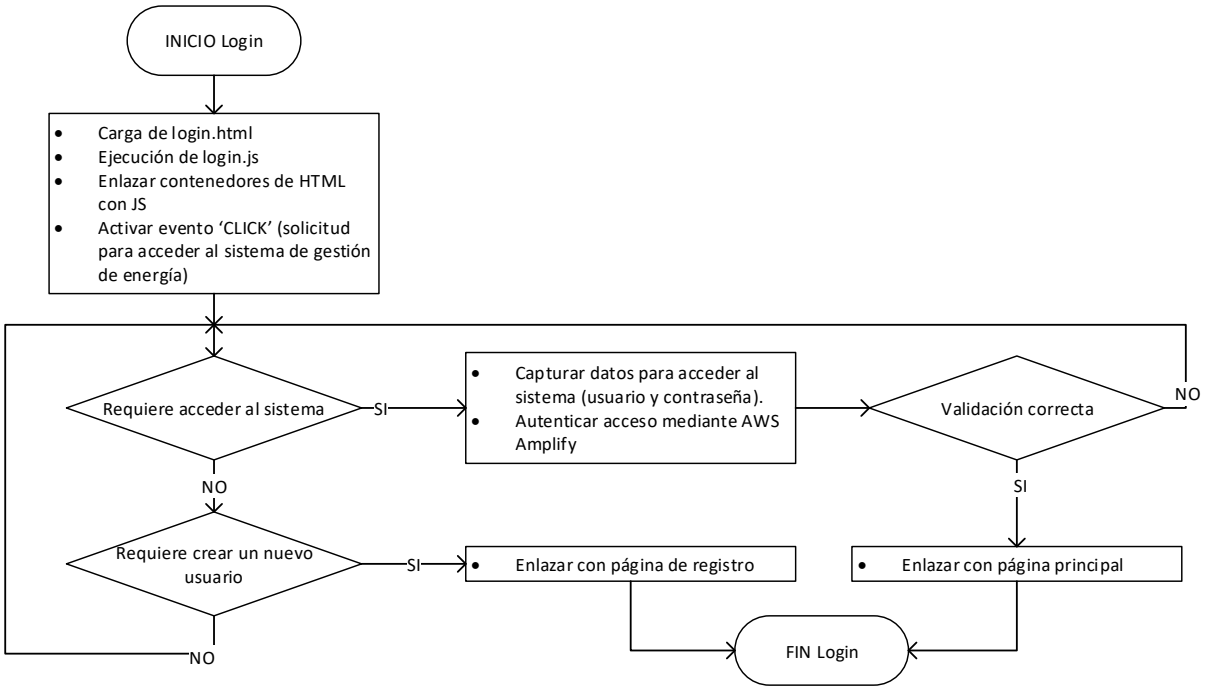


Figura 30. Diagrama de flujo para acceder al sistema de gestión de energía.

- Registrar a la plataforma AWS, ver ***Figura 31***.

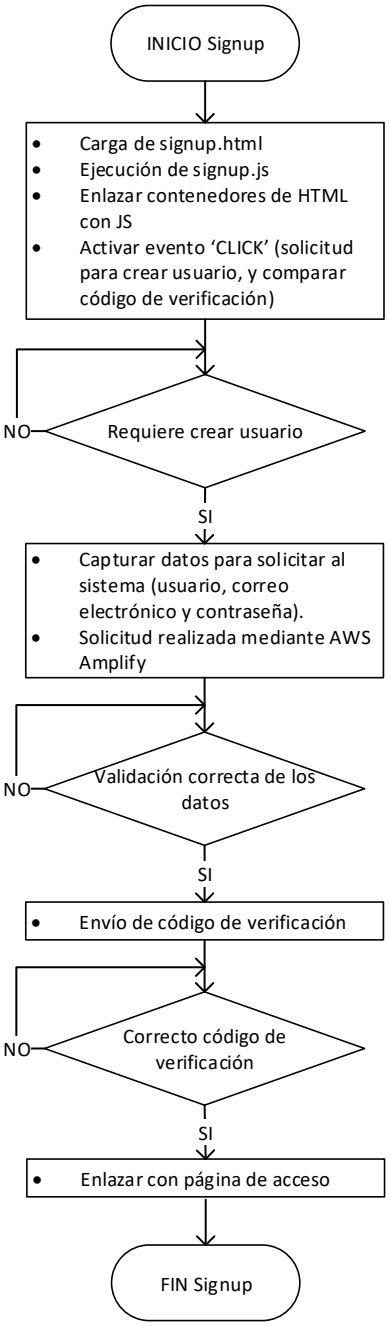


Figura 31. Diagrama de flujo para registrar un usuario al sistema de gestión de energía.

- Ejecutar microservicios para mantener comunicación en tiempo real, ver ***Figura 32***.

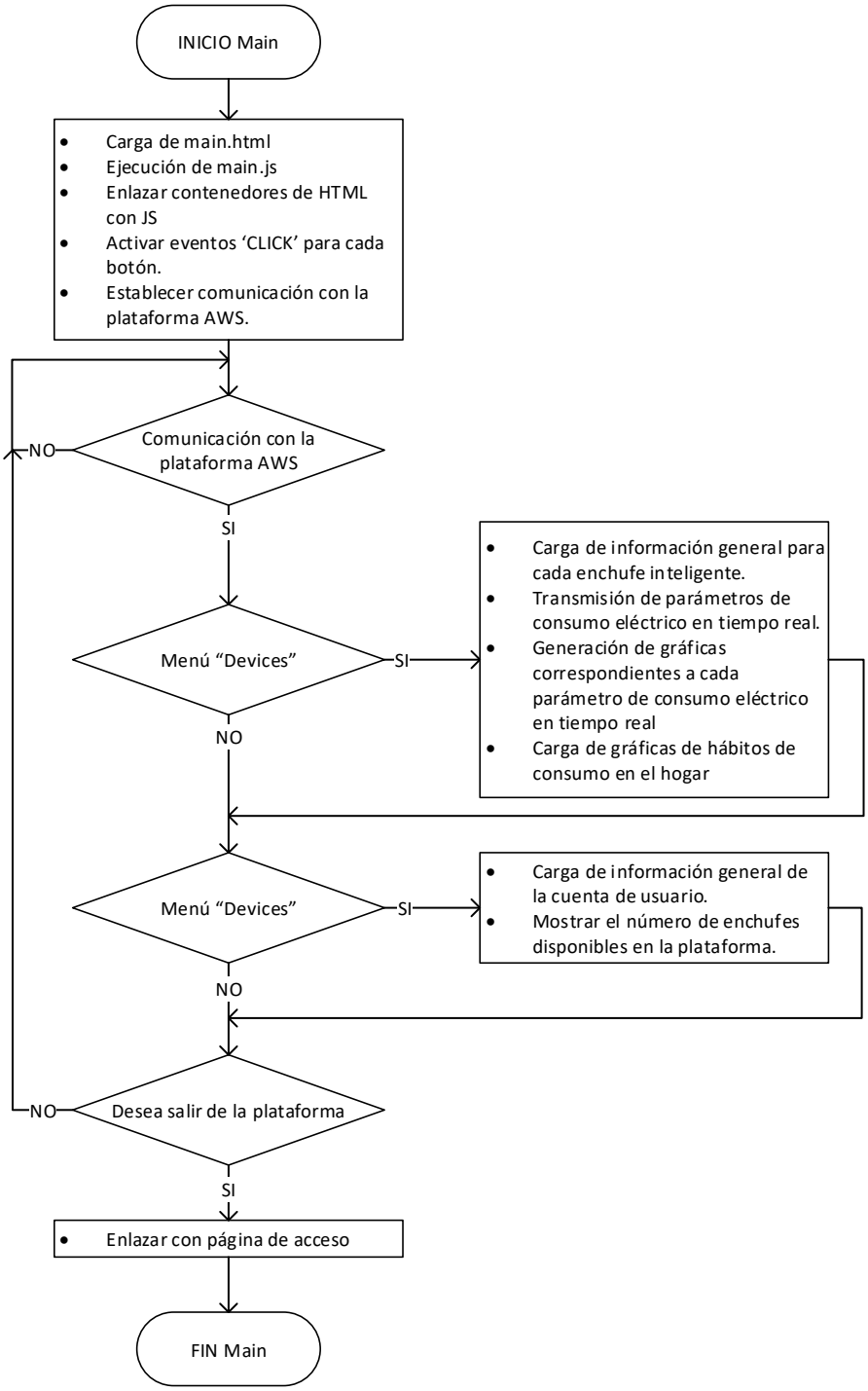


Figura 32. Diagrama de flujo para mantener comunicación con la plataforma AWS, y ejecutar los microservicios en la página web.

3.2.4. Administrador

El administrador es la persona o grupos de personas con capacidad de acceder mediante consola a la plataforma AWS. Allí es capaz de modificar los microservicios, crear nuevos, y modificar la arquitectura actual, son la finalidad de satisfacer nuevas necesidades. Una parte fundamental es la utilización de la herramienta de cálculo Excel, para procesar datos, generar un batch de predicción y graficar los hábitos de consumo eléctrico.

El procesamiento de datos se realiza mediante el uso de modelos de datos, donde se aplican varios procesos como se observan en la ***Figura 33***.

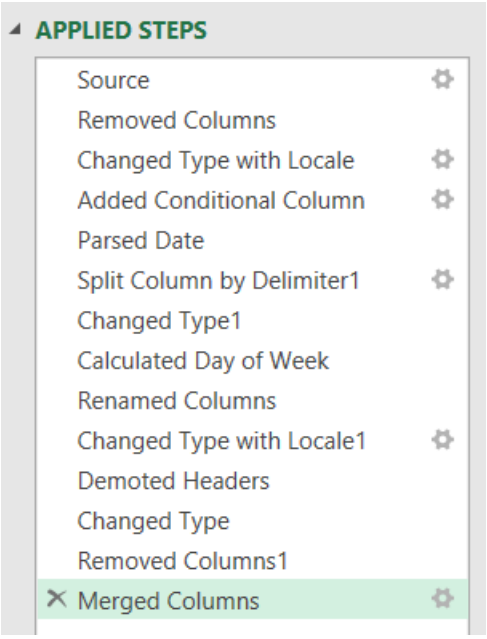


Figura 33. Procesamiento de datos obtenidos de AWS RDS para subir hacia AWS Machine Learning

CAPÍTULO 4

4. DESARROLLO E IMPLEMENTACIÓN

4.1. Desarrollo

El desarrollo se ha dividido en 2 partes, la primera se ha desarrollado mediante el uso de la Raspberry Pi 3 (servidor Node.js y aplicativo web), mientras que la segunda parte se ha desarrollado en la nube mediante la plataforma AWS.

4.1.1. Desarrollo en la nube

La plataforma AWS permite administrar los recursos mediante consola, además ofrece un periodo de prueba de 1 año con recursos limitados a varios microservicios y herramientas. A partir de ello, se hace uso de la plataforma gratuita con la finalidad de disminuir el costo de ejecución del presente trabajo de investigación.

En función de la ***Figura 16***, la propuesta de arquitectura de sistema de gestión de energía en el hogar se ha dividido en sus 2 servicios fundamentales.

4.1.1.1. Transmisión de datos en tiempo real

El microservicio destinado para el desarrollo de aplicativos en tiempo real se denomina AWS Appsync. A continuación de detalla los pasos necesarios para generar la instancia requerida:

- Configuración inicial, ver ***Figura 34***.
- Creación de modelo, ***Figura 35***.
- Creación de los recursos, ***Figura 36***.

La configuración inicial se realiza mediante la utilización del asistente para facilitar la creación automática de los recursos necesarios, y la comunicación entre microservicios. El asistente además

genera una tabla destinada al almacenamiento de los datos en tiempo real mediante el uso de la base de datos DynamoDB.

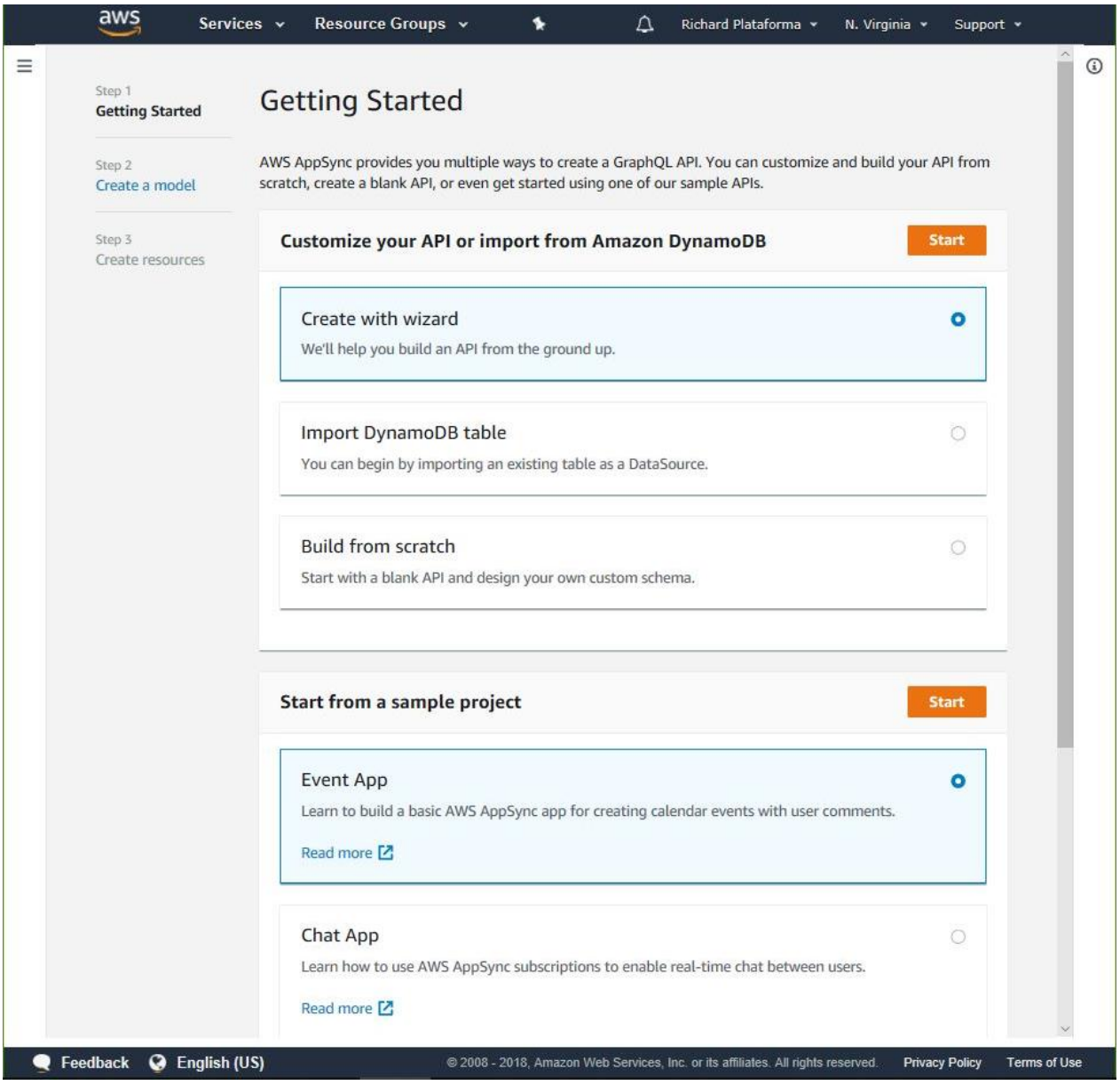


Figura 34. Configuración inicial de AWS Appsync mediante la consola de AWS.

El segundo paso corresponde a crear el modelo, ver ***Figura 35***, mediante la definición de todos los atributos requeridos por AWS Appsync, los cuales son creados en la tabla de DynamoDB. Para cada atributo se define el nombre y tipo de dato correspondiente.

Los atributos fueron definidos en la **Tabla 13**.

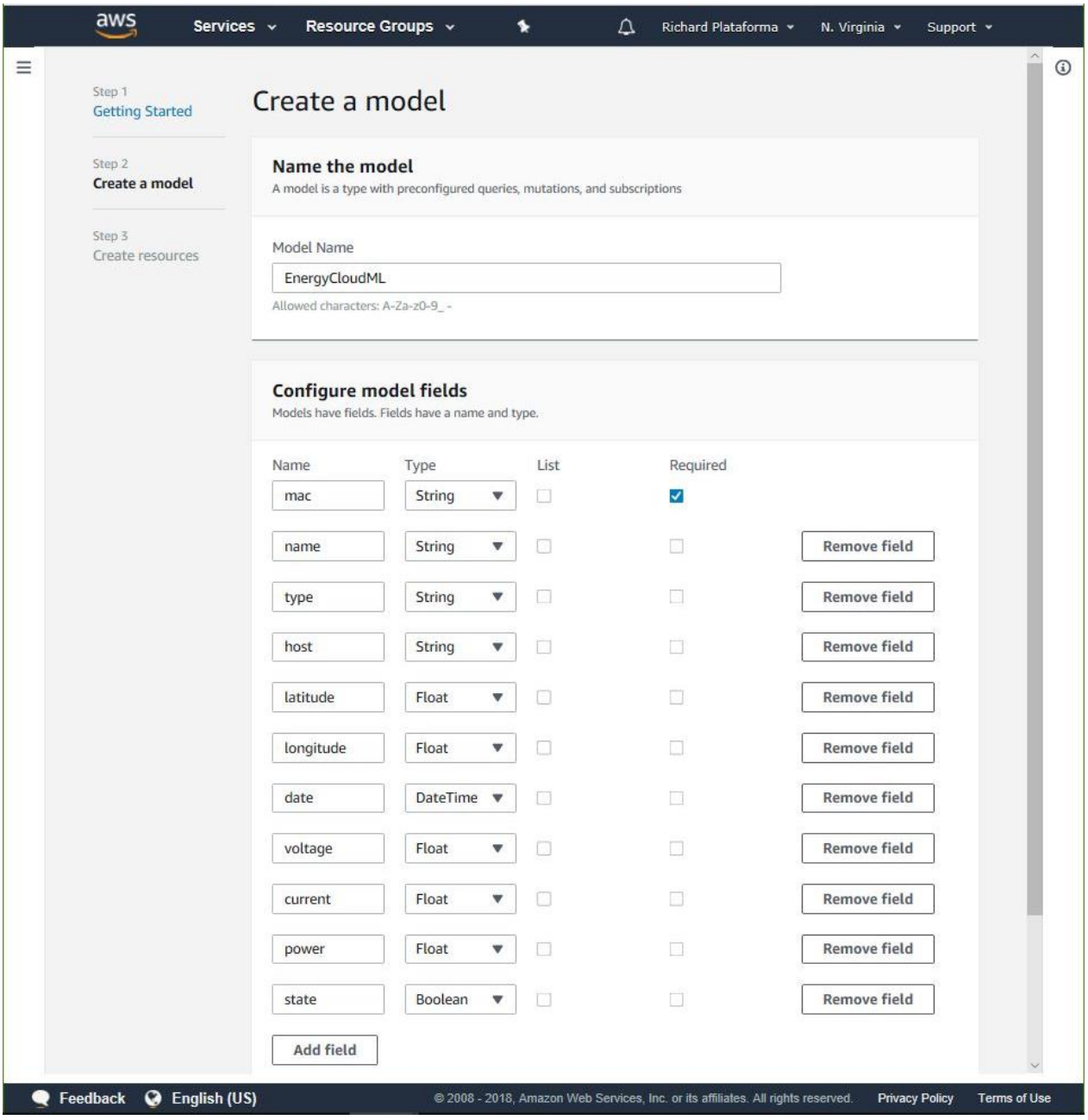


Figura 35. Creación el modelo de AWS Appsync mediante la consola de AWS.

En el tercer paso se define el nombre de la API, ver ***Figura 36***.

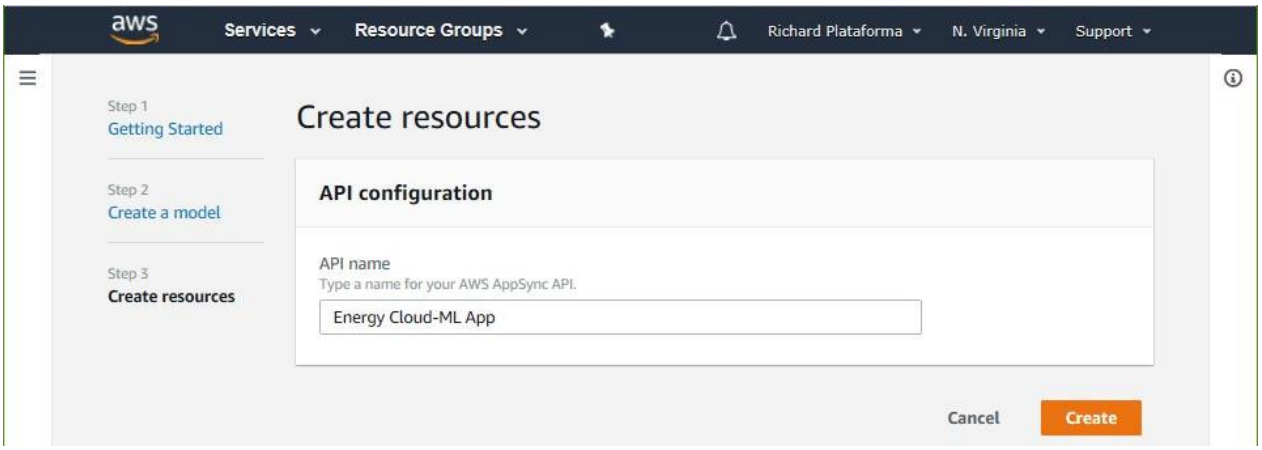


Figura 36. Creación del recurso de AWS Appsync mediante la consola de AWS.

Finalizada la creación del microservicio, ver ***Figura 37***, ahora se modifican los métodos y funcionalidades internas por defecto del microservicio, La configuración corresponden a:

- Edición del esquema para funcionalidad con el servidor node.js y el aplicativo web.
- Asignación de método de autenticación.

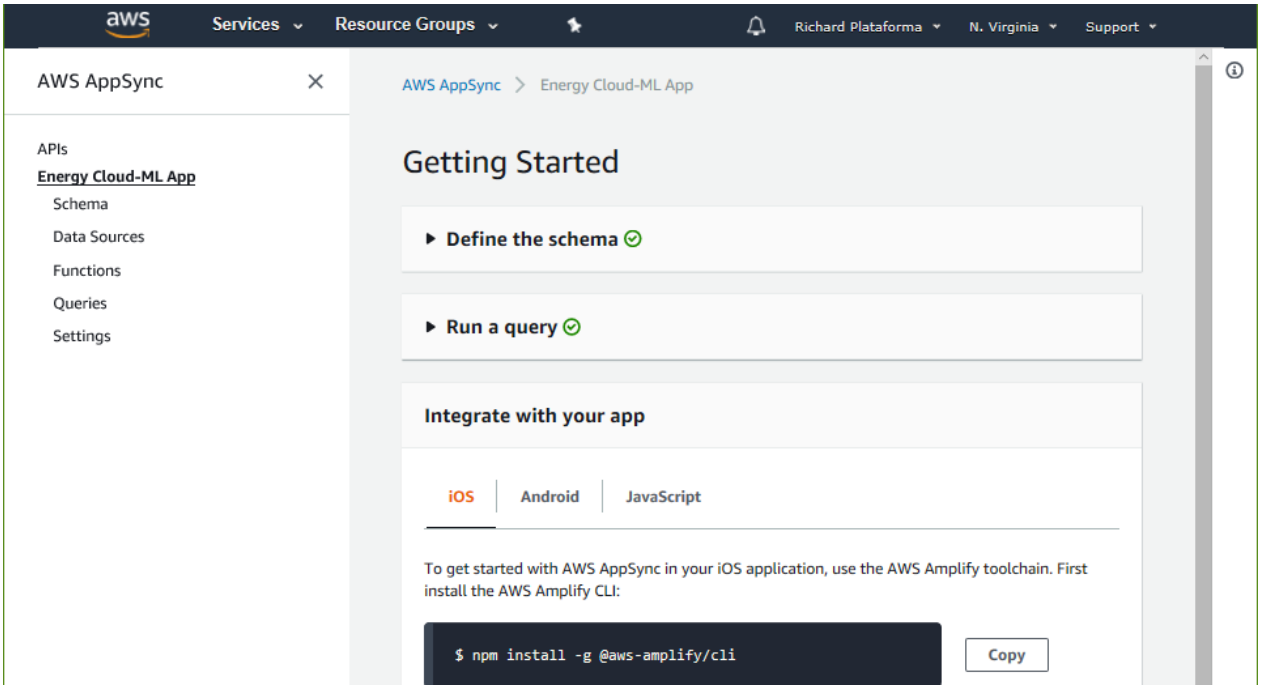


Figura 37. Página principal de la API generada en AWS Appsync.

4.1.1.1.1. Edición del esquema definido en AWS Appsync

La **Tabla 13** específica los atributos a utilizar, sin embargo, el esquema actual no diferencia las funcionalidades requeridas, sí no qué, engloba todas ellas en una sola petición aumentando la carga en el sistema, y solicitando atributos innecesarios el servidor y el aplicativo. Por tal motivo, el esquema se modifica de tal manera que cumplan las funcionalidades requeridas, como se observa en la **Tabla 18**.

Cabe destacar que todo el esquema generado se puede consultar en el Anexo 2.

Tabla 18
Métodos modificados en AWS Appsync para satisfacer las necesidades de la arquitectura.

Tipo	Función	Atributos	Función
Query	listControlStatus		• Consulta si existe alguna señal de control de todos los enchufes.
Mutation	createGeneralInfo	• Mac • Name • Type • Host • Latitude • Longitude	• Creación de nueva fila en la tabla con información general de cada dispositivo.
	updateGeneralInfo	• Mac • Name • Type • Host • Latitude • Longitude	• Actualización de la información general de cada dispositivo.
	updateRealtimeInfo	• Mac • Date • Voltage • Current • Power • State	• Actualización de los parámetros de consumo eléctrico, además del estado para cada dispositivo.
	control	• Mac • Control	• Envío de señal de control para encender o apagar el dispositivo
Suscription	onUpdateRealtimeInfo	• mac	• Suscripción para recibir datos proporcionados al actualizar la tabla mediante “updateRealtimeInfo”

La modificación del esquema implica establecer enlaces hacia los atributos definidos en DynamoDB. Los recursos hacen referencia a los atributos requeridos por cada función, los cuales son ingresados o extraídos desde AWS Appsync.

Por ejemplo, la ***Figura 38*** hace referencia al ingreso de información en la base de datos DynamoDB, mediante el comando "PutItem". El comando ingresa la información proporcionada por la función, y asocia a sus atributos mediante un identificador, en este caso la MAC.

Las funciones que satisfacen este recurso son: *createGeneralInfo* y *control*.

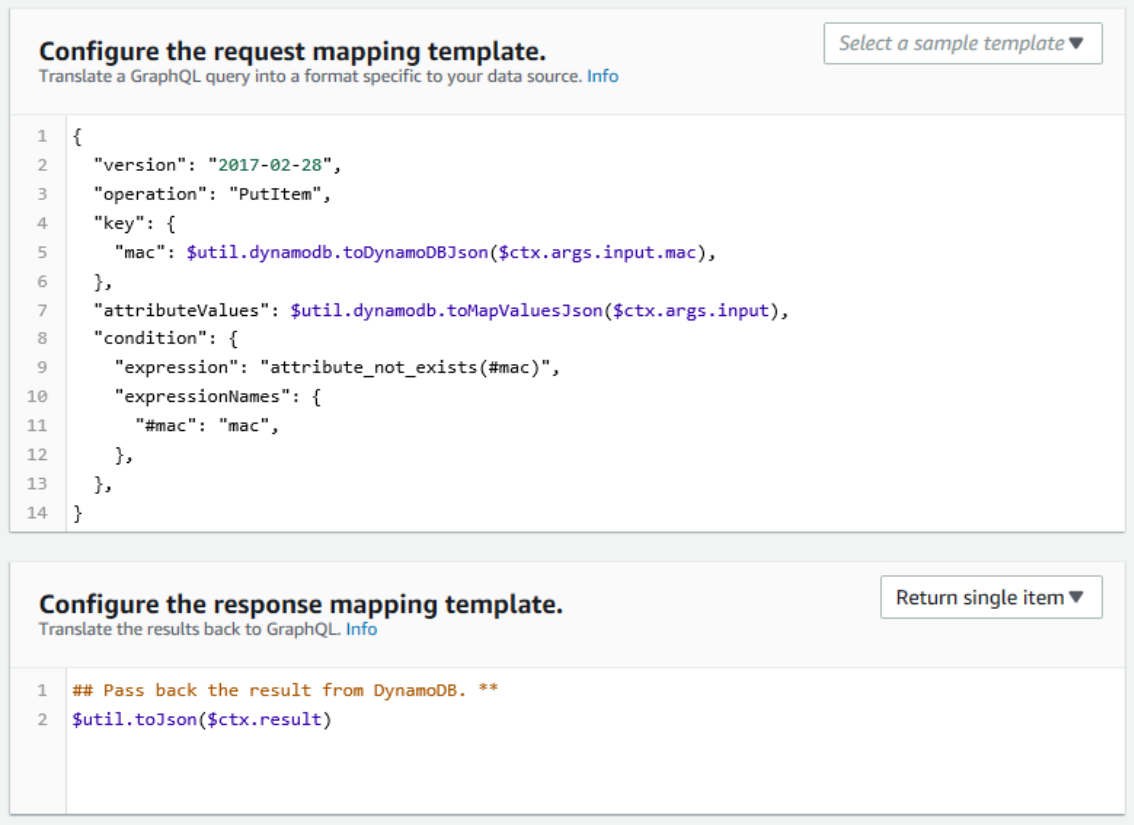


Figura 38. Solicitud y respuesta para establecer comunicación entre AWS Appsync y AWS DynamoSB, para las funciones createGeneralInfo y control.

Sin embargo, algunos recursos son definidos de manera más compleja, como actualizaciones, estas no solo evalúan los valores ingresados para actualizar los datos, sino que, además evalúan los

valores en blanco y nuevos atributos, para realizar actualizaciones más complejas, como se observa en la ***Figura 39***.

El recurso es aplicado para las funciones *updateGeneralInfo* y *updateRealtimeInfo*.

Configure the request mapping template.
Translate a GraphQL query into a format specific to your data source. Info

Select a sample template

```
{
  "version": "2017-02-28",
  "operation": "UpdateItem",
  "key": {
    "mac": $util.dynamodb.toDynamoDBJson($ctx.args.input.mac),
  },

  ## Set up some space to keep track of things we're updating **
  #set( $expNames  = {} )
  #set( $expValues = {} )
  #set( $expSet = {} )
  #set( $expAdd = {} )
  #set( $expRemove = [] )

  ## Iterate through each argument, skipping keys **
  #foreach( $entry in $util.map.copyAndRemoveAllKeys($ctx.args.input, ["mac"]).entrySet() )
    #if( $util.isNull($entry.value) )
      ## If the argument is set to "null", then remove that attribute from the item in DynamoDB **

      #set( $discard = ${expRemove.add("#${entry.key}")} )
      $!{expNames.put("#${entry.key}", "${entry.key}")}
    #else
      ## Otherwise set (or update) the attribute on the item in DynamoDB **

      $!{expSet.put("#${entry.key}", ":${entry.key}")}
      $!{expNames.put("#${entry.key}", "${entry.key}")}
      $!{expValues.put(":${entry.key}", $util.dynamodb.toDynamoDB($entry.value))}
    #end
  #end

  ## Start building the update expression, starting with attributes we're going to SET **
  #set( $expression = "" )
  #if( !${expSet.isEmpty()} )
    #set( $expression = "SET" )
    #foreach( $entry in $expSet.entrySet() )
      #set( $expression = "${expression} ${entry.key} = ${entry.value}" )
      #if ( $foreach.hasNext )
        #set( $expression = "${expression}," )
      #end
    #end
  #end
```

CONTINÚA

```
## Continue building the update expression, adding attributes we're going to ADD **
#if( !${expAdd.isEmpty()} )
  #set( $expression = "${expression} ADD" )
  #foreach( $entry in $expAdd.entrySet() )
    #set( $expression = "${expression} ${entry.key} ${entry.value}" )
    #if ( $foreach.hasNext )
      #set( $expression = "${expression}," )
    #end
  #end
#end

## Continue building the update expression, adding attributes we're going to REMOVE **
#if( !${expRemove.isEmpty()} )
  #set( $expression = "${expression} REMOVE" )

  #foreach( $entry in $expRemove )
    #set( $expression = "${expression} ${entry}" )
    #if ( $foreach.hasNext )
      #set( $expression = "${expression}," )
    #end
  #end
#end

## Finally, write the update expression into the document, along with any expressionNames and expressionVa
"update": {
  "expression": "${expression}",
  #if( !${expNames.isEmpty()} )
    "expressionNames": $utils.toJson($expNames),
  #end
  #if( !${expValues.isEmpty()} )
    "expressionValues": $utils.toJson($expValues),
  #end
},

"condition": {
  "expression": "attribute_exists(#mac)",
  "expressionNames": {
    "#mac": "mac",
  },
}
}
```

Configure the response mapping template.

Translate the results back to GraphQL. Info

Select a sample template

```
## Pass back the result from DynamoDB. **
$util.toJson($context.result)
```

Figura 39. Solicitud y respuesta para establecer comunicación entre AWS Appsync y AWS DynamoSB, para las funciones updateGeneralInfo y updateRealtimeInfo.

4.1.1.1.2. Asignación de método de autenticación

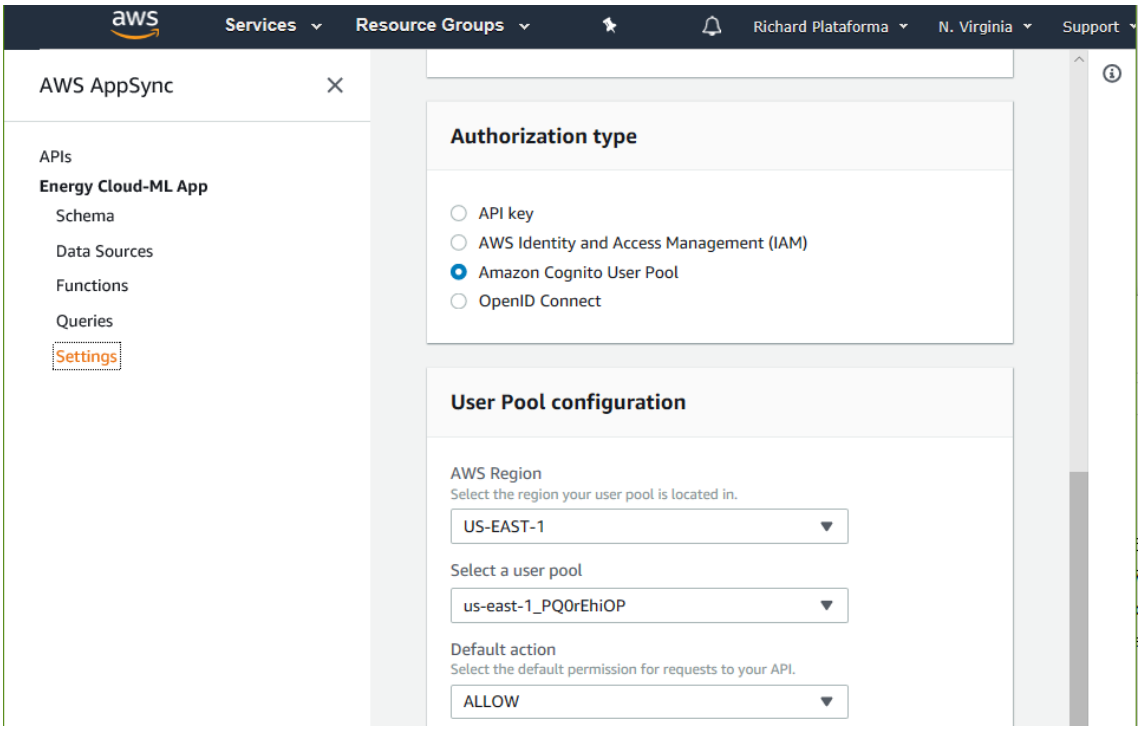


Figura 40. Configuración del modo de autenticación para AWS Appsync.

Otro factor fundamental es la modificación del método acceso al microservicio AWS Appsync, el modo de autenticación elegido es *Amazon Cognito User Pool* como se observa en la ***Figura 40***.

Para mantener comunicación con el pool de usuario, se tiene que definir el id del pool correspondiente. Cabe recalcar que la creación del recurso se puede observar en la sección 4.1.2.2. Aplicativo Web.

4.1.1.2. Aprendizaje de hábitos de consumo

El microservicio principal para el servicio es AWS Machine Learning. El desarrollo se realiza a partir de la creación de la instancia de base de datos en AWS RDS, procesamiento de los datos y ejecución del proceso en AWS Machine Learning.

4.1.1.2.1. Definición de la base de datos

Utilizando la consola AWS, se procede a crear la instancia respectiva con el motor de base de datos MariaDB bajo la capa de recursos de prueba, como se observa en la ***Figura 41***. Es necesario aclarar que debido a que la base de datos (DB) es un servidor SQL, se requiere montarlo en una máquina virtual mediante el uso de AWS EC2 para su acometida. Sin embargo, este proceso es transparente gracias al asistente de AWS RDS.

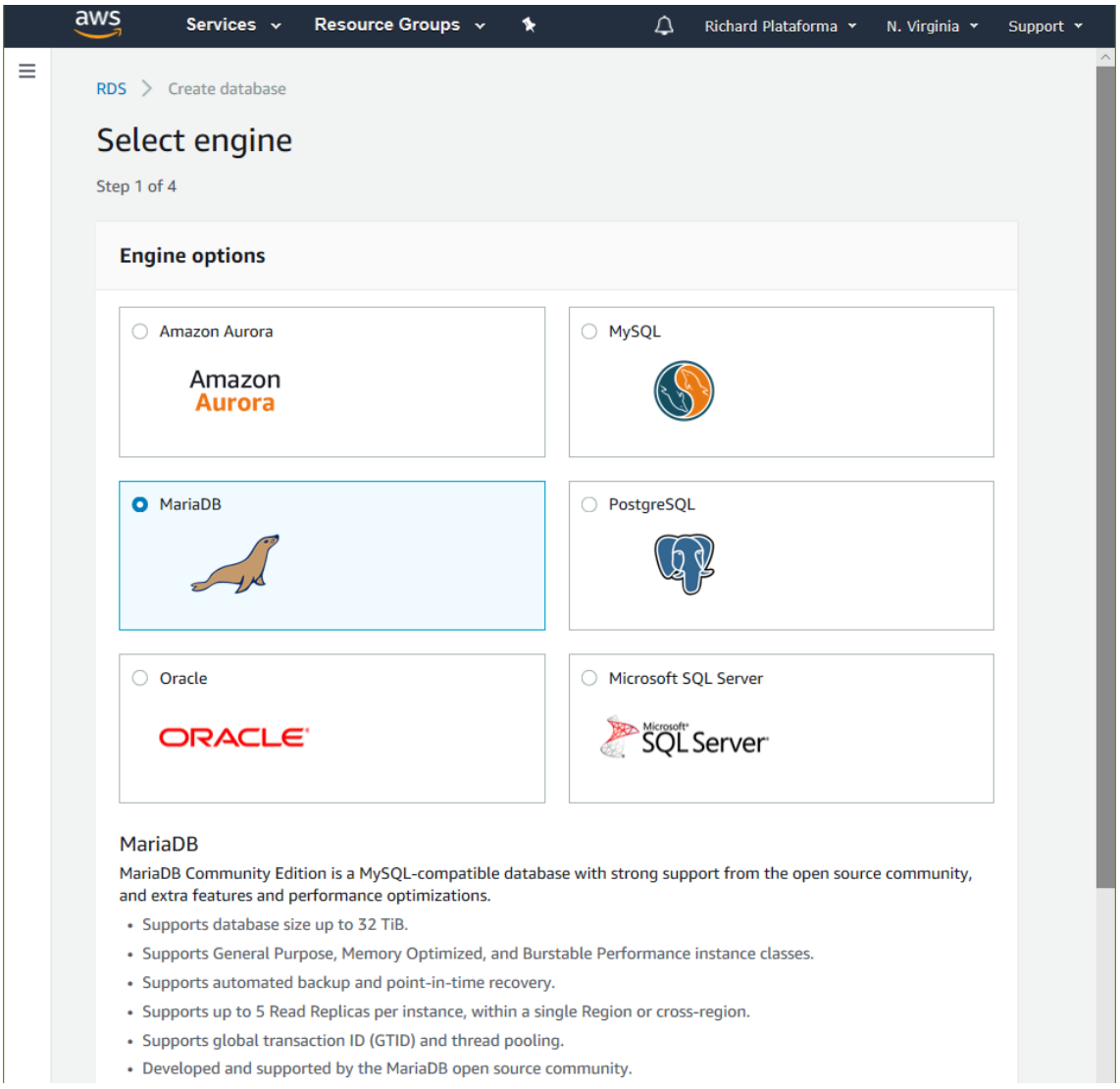


Figura 41. Selección de motor de AWS RDS mediante la consola de AWS.

En el paso 2 se configura detalles específicos de la base de datos, las principales opciones se muestran en la ***Figura 42***. Los atributos clave son: nombre de la instancia, nombre de usuario y contraseña, además de definir la memoria de disco, y CPU. En la capa gratuita, la memoria de disco y CPU son limitados a 20GB en memoria y 1G en RAM.

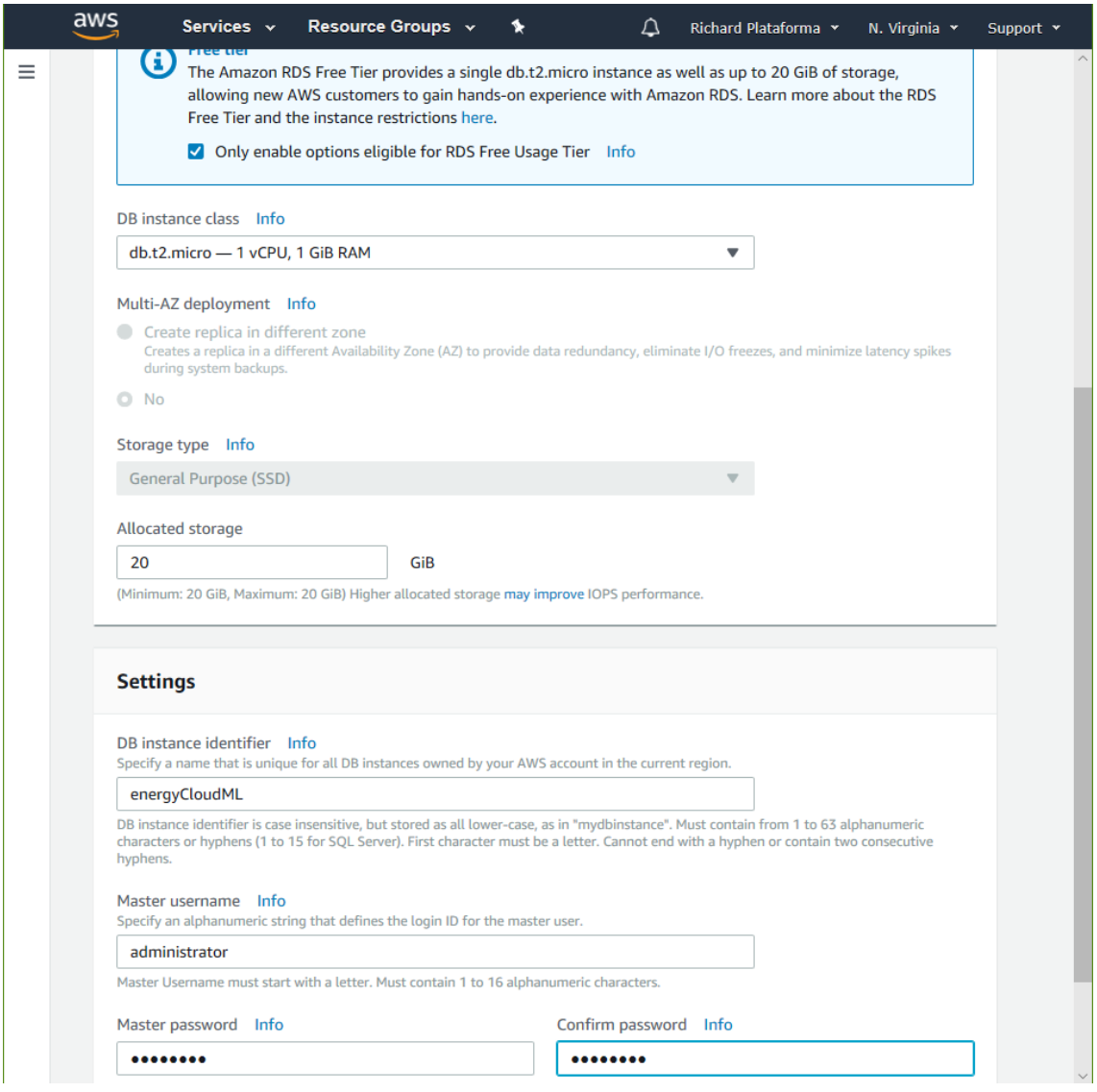


Figura 42. Configuración de atributos de AWS RDS mediante la consola de AWS.

Para finalizar, el paso 3 define configuraciones avanzadas, entre las más relevantes son:

- VPC - genera un grupo privado de acceso restringido.

- Subnet group – define conjunto de IP y protocolos de transporte permitidos (en caso de acceso mediante administradores SQL como *phpMyAdmin*).
- Public accessibility – permite acceso remoto a la DB.
- Nombre de base de datos

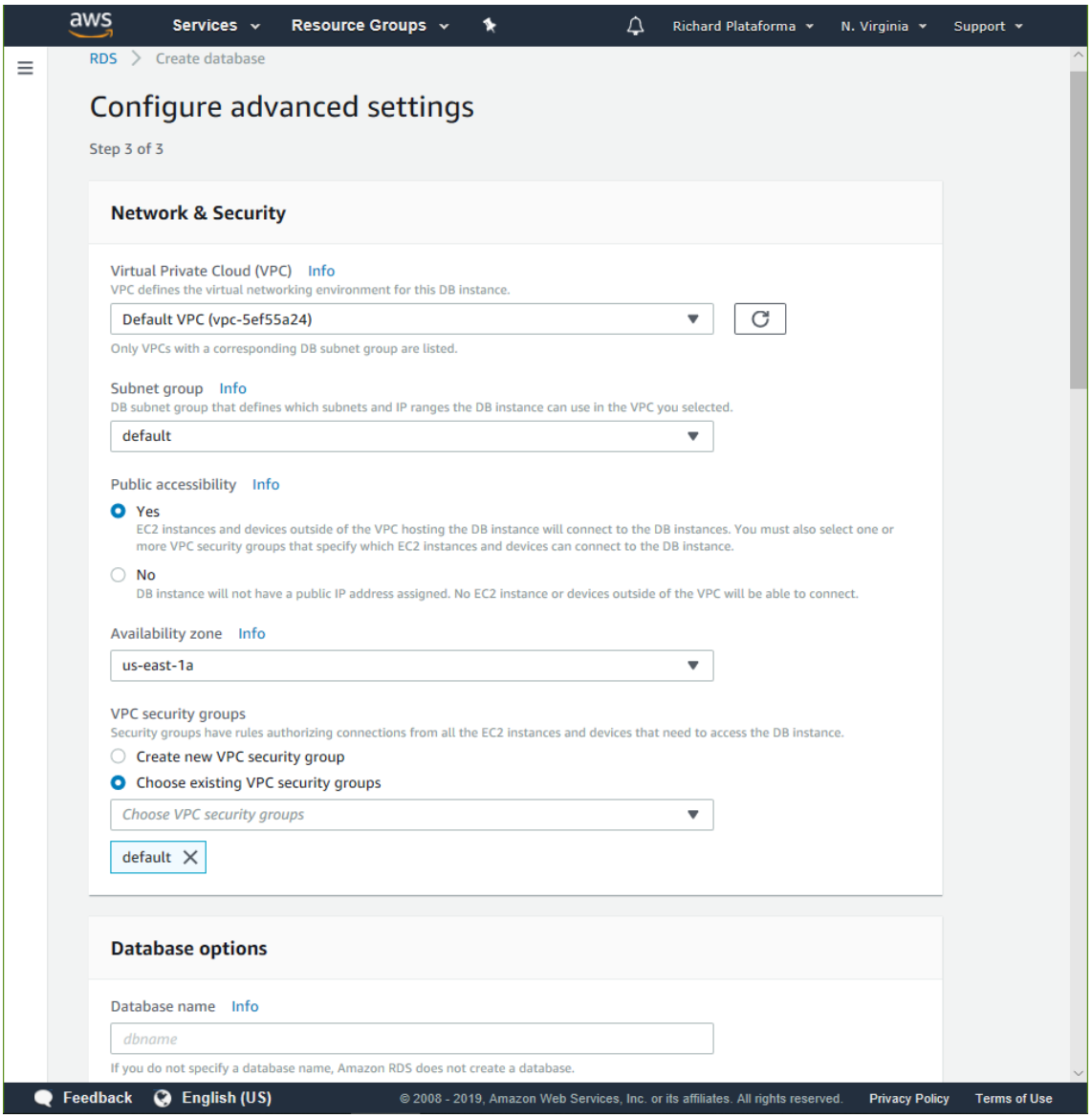


Figura 43. Configuración avanzada de atributos de AWS RDS mediante la consola de AWS. Permite definir seguridad en la red, atributos de la DB, encriptación, backup, monitoreo, mantenimiento y protección.

A pesar de las seguridades que ofrece AWS RDS, es posible dar mayor seguridad y facilidades de acceso a la DB con AWS Secrets Manager. Esta herramienta permite proporcionar las credenciales requeridas para acceder a la DB con el servidor y, genera contraseñas de acceso aleatorias definidas en un periodo de tiempo. Además, establece comunicación end-to-end cifrada mediante el uso de claves públicas distribuidas hacia AWS RDS.

La ***Figura 44*** muestra la configuración inicial para guardar el nuevo secreto, entre la información proporcionada se encuentra: nombre de usuario, contraseña y DB.

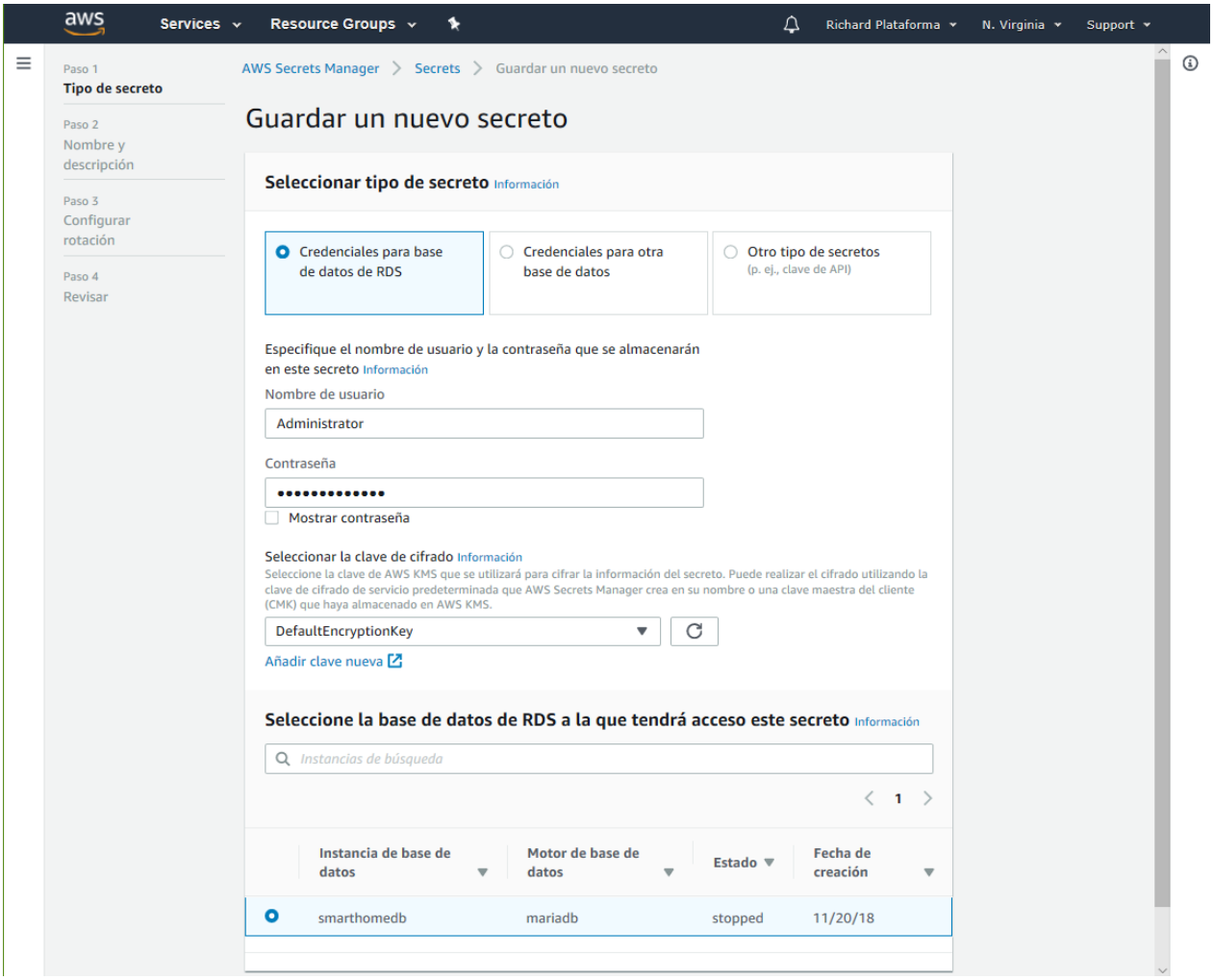


Figura 44. Selección de secreto de AWS Secrets Manager mediante la consola de AWS.

El paso 2 corresponde a proporcionar el nombre y descripción del secreto, esto facilita la administración de nuevos secretos e identificación de la DB que se utiliza.

El paso 3, ver ***Figura 45***, permite configurar contraseñas rotativas, es decir, el cambio automático de la contraseña de acceso a la DB, mediante el uso de AWS Lambda. El código generado se realiza automáticamente, por lo que no requiere realizar cambio alguno.

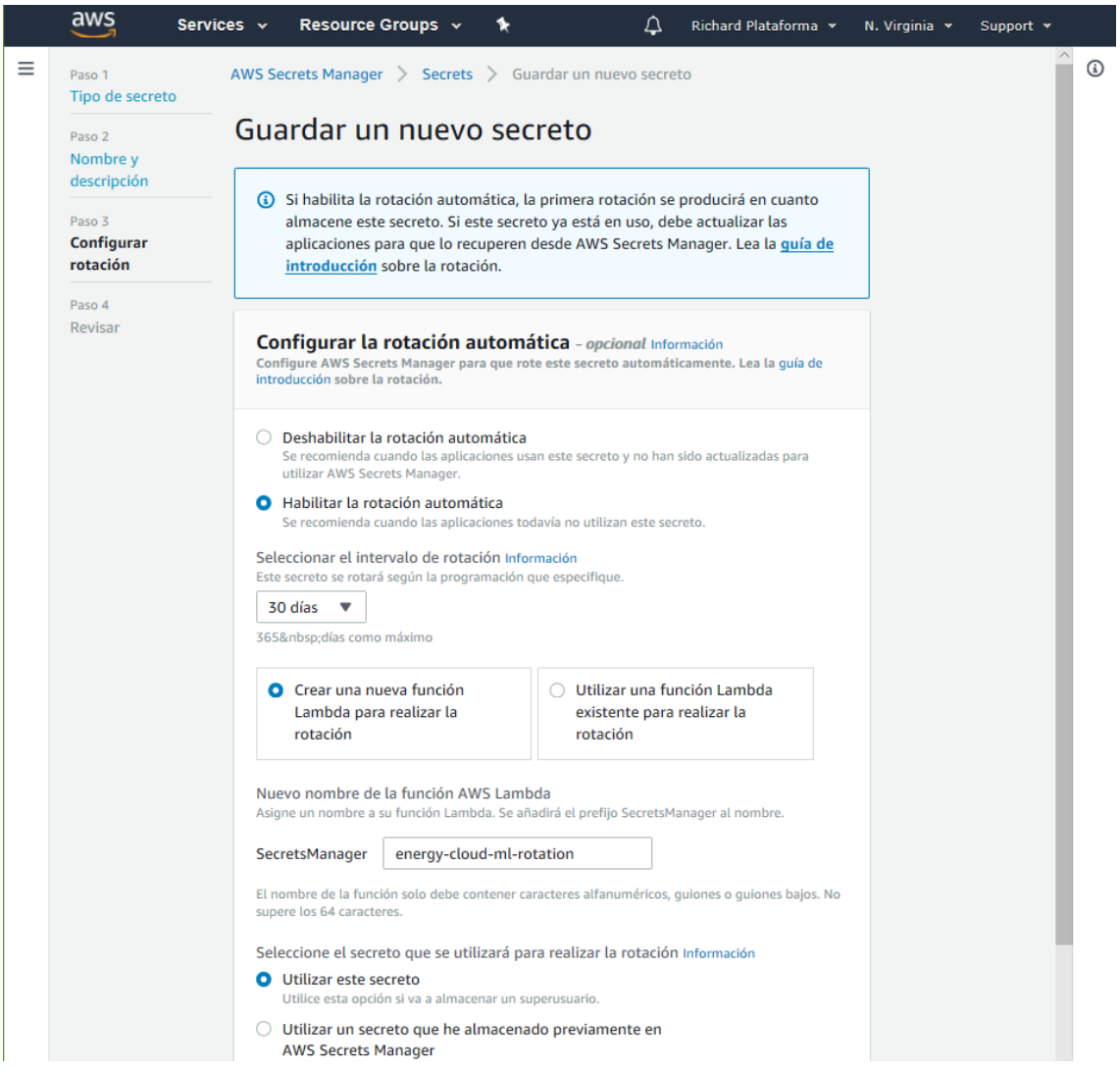


Figura 45. Configuración de contraseña rotativa en AWS Secrets Manager mediante la consola de AWS.

4.1.1.2.2. Procesamiento de los datos

Los datos almacenados en AWS RDS durante 5 meses son extraídos con AWS Pipeline. El proceso de extracción se realiza mediante el esquema definido en la ***Figura 46***.

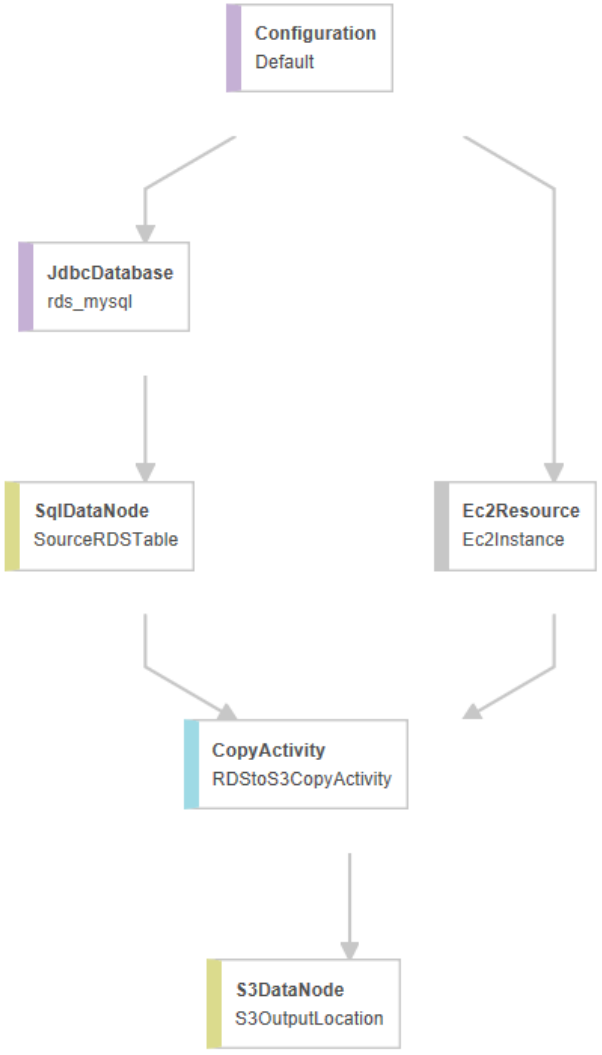


Figura 46. Esquema de extracción y procesamiento en AWS Pipeline mediante la consola de AWS.

El esquema se desarrolla en la consola AWS mediante la creación de un nuevo Pipeline, como se observa en la ***Figura 47***. Entre los atributos requeridos están: nombre y descripción del Pipeline,

tipo de procesamiento que va a realizar Pipeline, parámetros para acceder a la DB, directorio para almacenar los datos y logs, y cuando ejecutar el Pipeline.

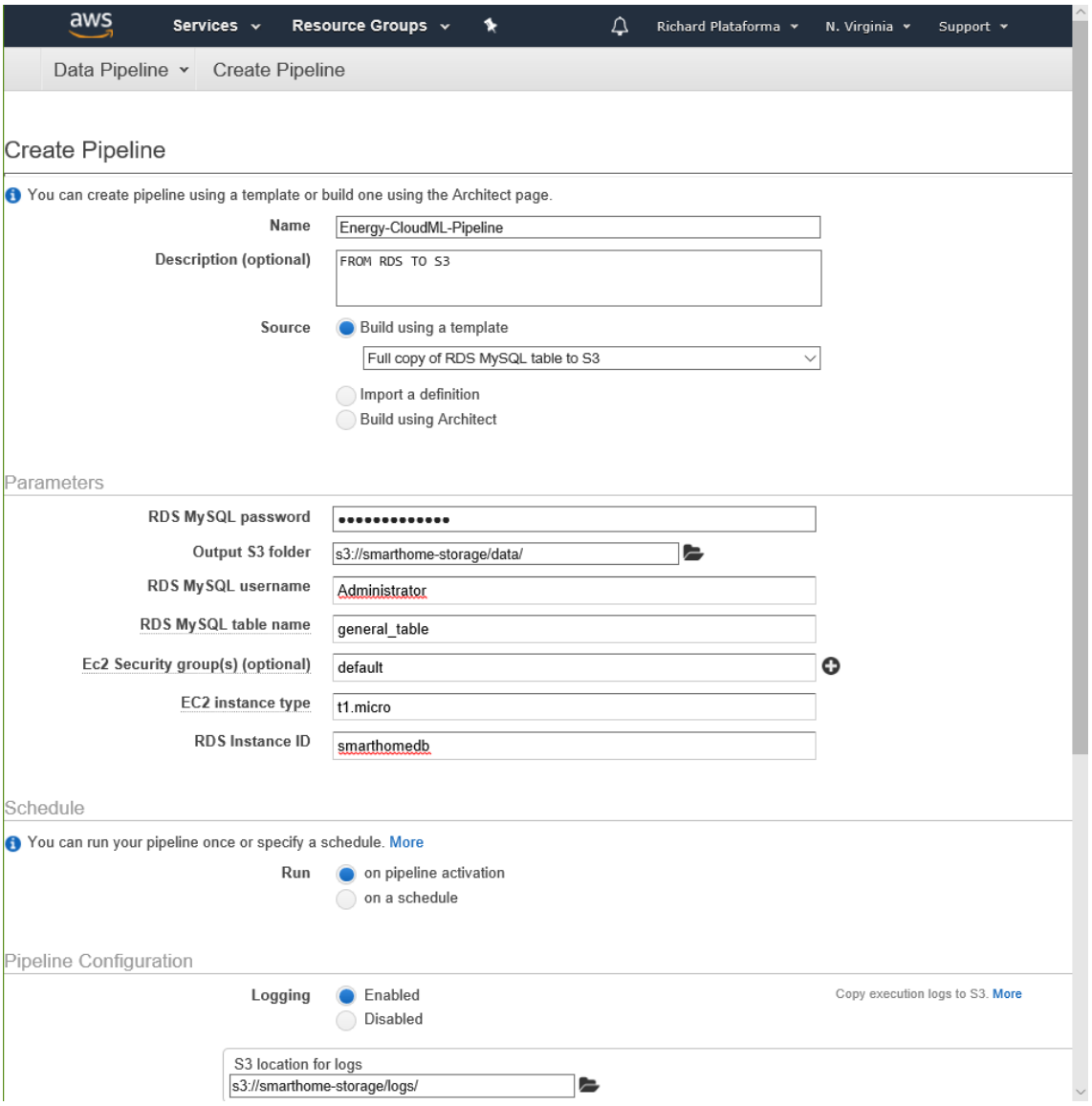


Figura 47. Creación del Pipeline de AWS Data Pipeline mediante la consola de AWS.

El esquema genera automáticamente las conexiones requeridas, sin embargo, hay que añadir la conexión hacia la DB, debido a que la DB aún no soporta o la configuración hacia una DB MariaDB

no ha sido establecida. El motor es *org.mariadb.jdbc.Driver*, y la conexión se genera a partir de la estructura tipo jdbc (Java DataBase Connection) que tiene la siguiente estructura, ver ***Figura 48***:

"jdbc:mariadb://localhost/db"

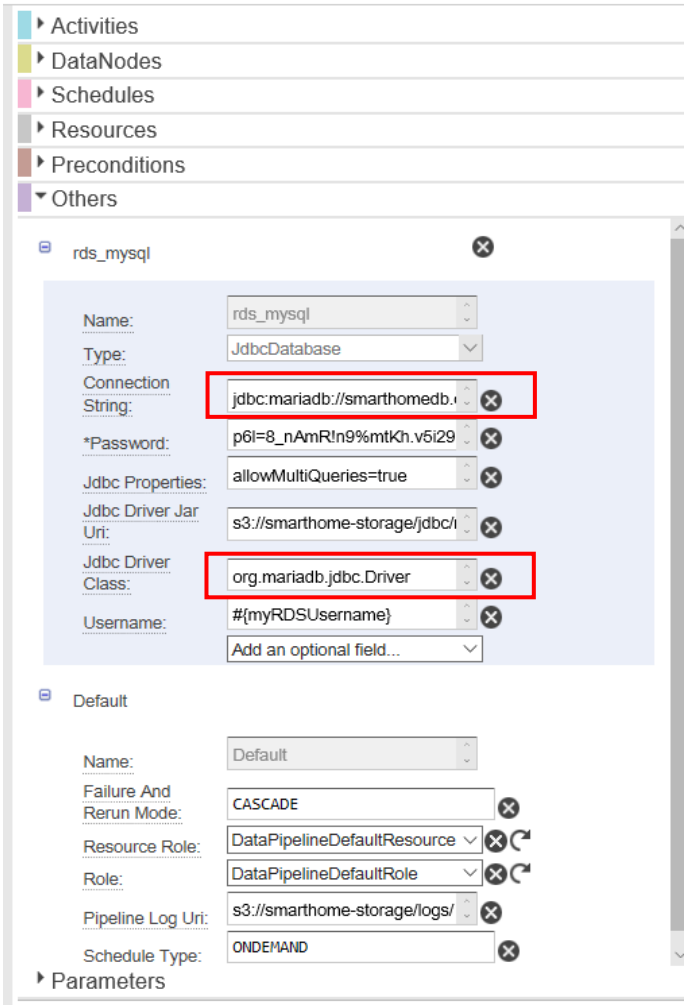


Figura 48. Modificación de parámetros en AWS Data Pipeline mediante la consola de AWS.

A partir de las modificaciones realizadas en la ***Figura 48***, se ejecuta el esquema de la ***Figura 46***, para cada tabla de datos histórico generadas, ver ***Figura 49***, en este caso son 3:

- IOT_SMARTPLUGSWITCH_50C7BF2AF579
- IOT_SMARTPLUGSWITCH_50C7BF2AFCA8

- IOT_SMARTPLUGSWITCH_50C7BFC7D812

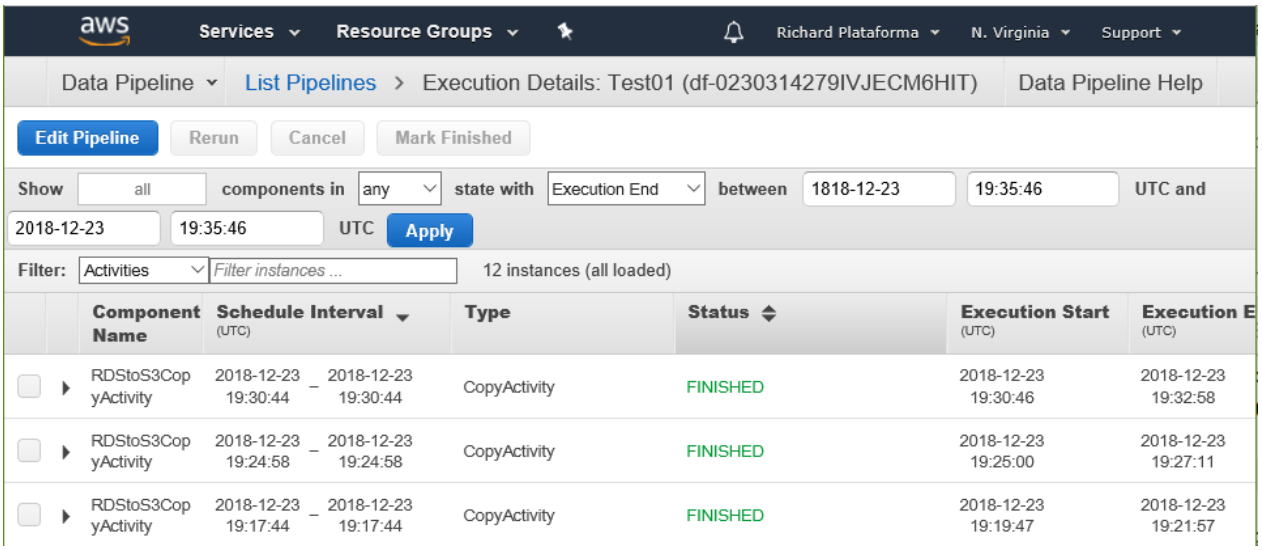


Figura 49. Ejecución del Pipeline para cada tabla de datos históricos en AWS Data Pipeline mediante la consola de AWS.

Los datos y logs generados en AWS DataPipeline son almacenados en AWS S3, ver ***Figura 50***.

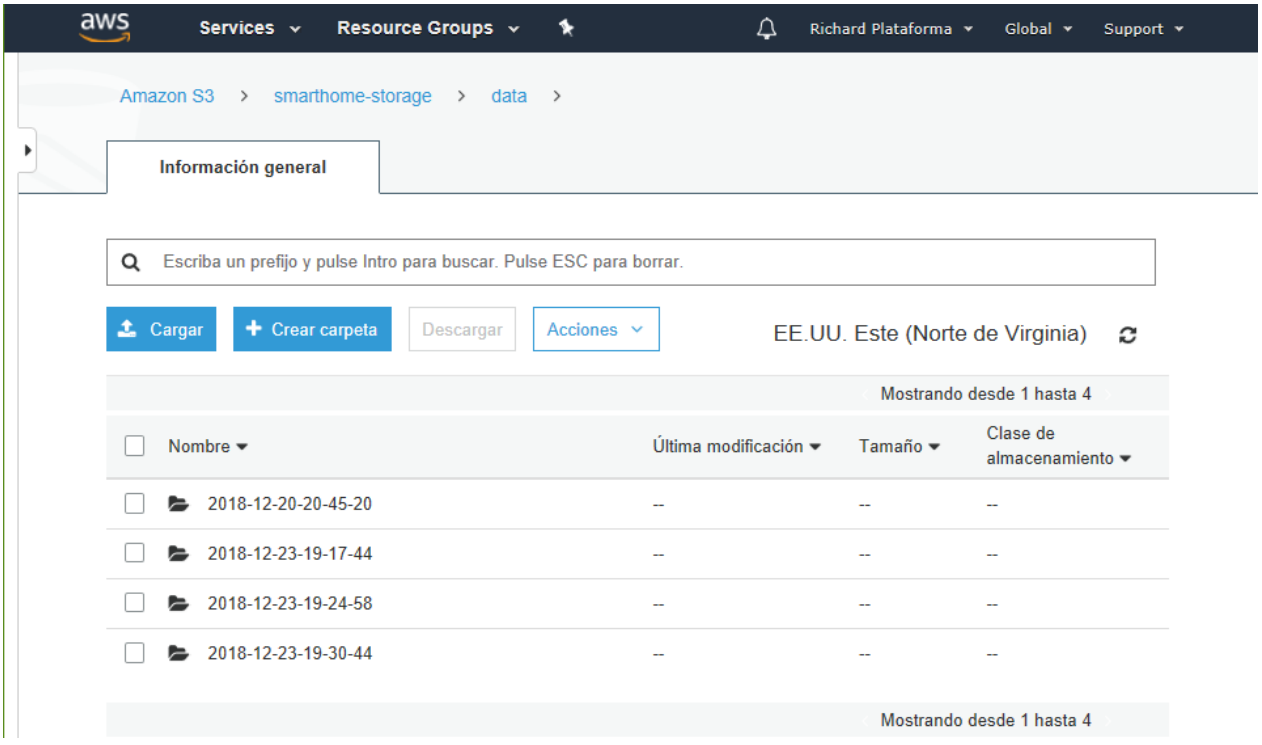


Figura 50. Archivo csv de datos generado por AWS DataPipeline mediante la consola de AWS.

Los archivos generados son:

- Carpeta 2018-12-23-19-17-44 → 3c40947e-2c95-4d67-ac3c-6e1ee71dc3e0.csv
- Carpeta 2018-12-23-19-24-58 → cb23ee5d-c323-4cc0-9555-b641f639720c.csv
- Carpeta 2018-12-23-19-30-44 → ec3dccdd-2a14-4701-afd2-8afb4a98631b.csv.

Por ejemplo, el archivo *3c40947e-2c95-4d67-ac3c-6e1ee71dc3e0.csv* hace referencia al enchufe inteligente con MAC=50:C7:BF:C7:D8:12 que corresponde al Televisor. La información proporcionada corresponde a los atributos definidos en la ***Figura 19*** y ***Figura 20***, ver ***Figura 51***.

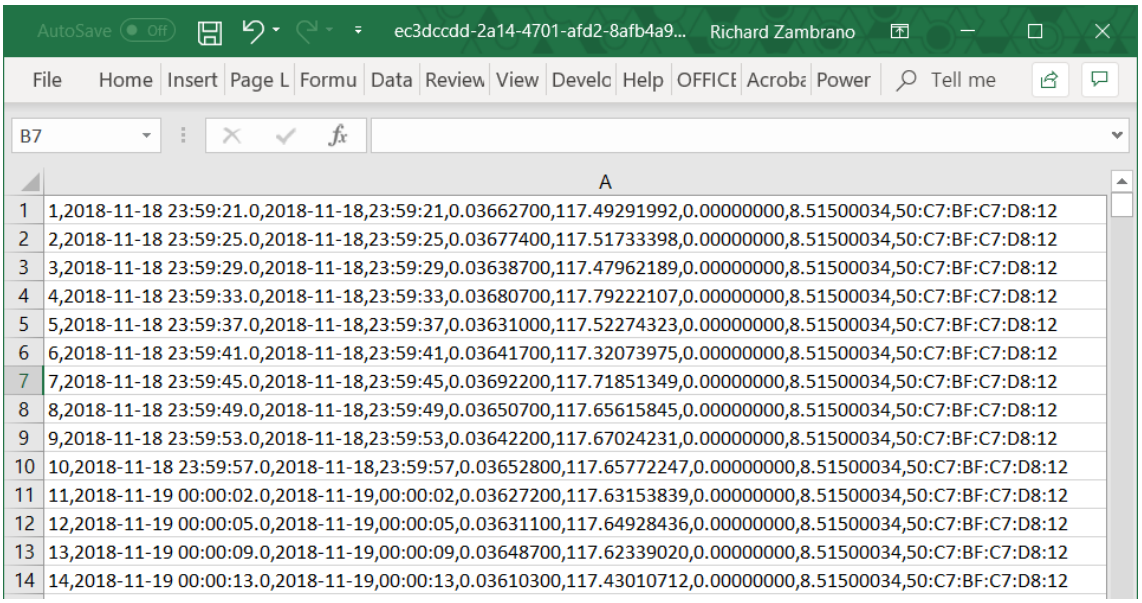


Figura 51. Archivo 3c40947e-2c95-4d67-ac3c-6e1ee71dc3e0.csv de datos históricos del enchufe con mac "50:C7:BF:C7:D8:12" abierto mediante Microsoft Excel 2016

Los archivos csv de cada enchufe son procesados mediante el uso de herramientas para modelar datos como Microsoft Excel. A partir de ello, los datos de la ***Figura 51*** son transformados hacia la estructura de datos de la ***Figura 52***, los pasos requeridos se observan en la ***Figura 33***, los cuales son mantienen información relevante para el procesamiento en AWS Machine Learning.

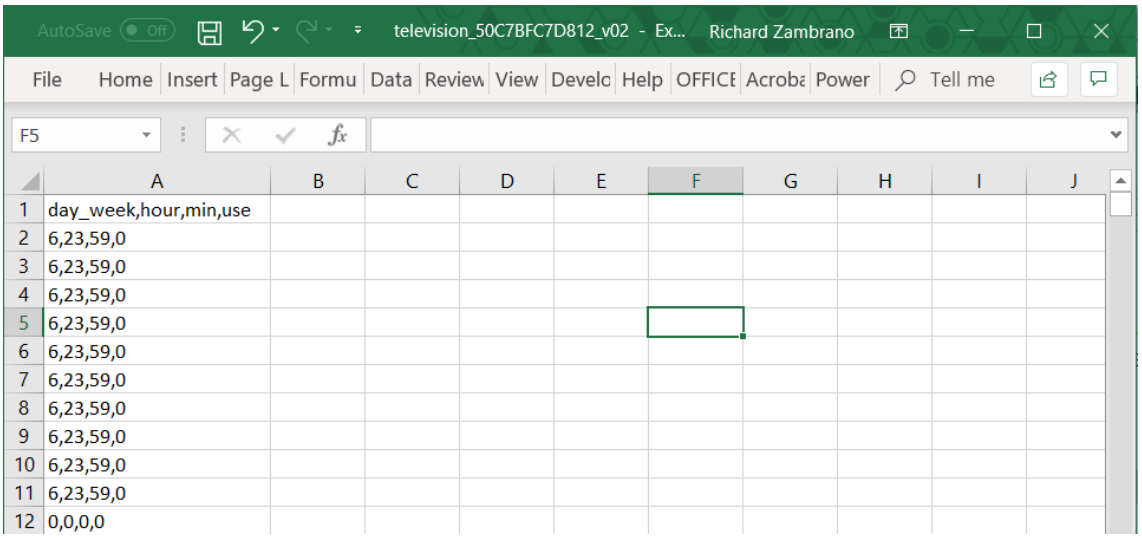


Figura 52. Archivo csv de datos históricos procesados del enchufe con mac "50:C7:BF:C7:D8:12" abierto mediante Microsoft Excel 016

El procesamiento se realiza para cada uno de los datos históricos, cuyos archivos procesados se muestran a continuación:

- 3c40947e-2c95-4d67-ac3c-6e1ee71dc3e0.csv → television_50C7BFC7D812.csv
- cb23ee5d-c323-4cc0-9555-b641f639720c.csv → microonda_50C7BF2AF579.csv
- ec3dccdd-2a14-4701-afd2-8afb4a98631b.csv → computadora_50C7B72AFCA8.csv

Sin embargo, además de los datos históricos, AWS Machine Learning requiere la definición de los atributos predicción, para ello se genera un batch donde se enlista las franjas horarias para cada día de la semana. El desarrollo del batch de predicción se lo realiza mediante VBA en Excel, a través de la macro visualizada en la ***Figura 54***. La estructura de datos se observa en la ***Figura 53***.

BATCH DE PREDICCIÓN
Día (0 - 6) Hora (0 - 23) Minuto (0 - 59; cada 5min)

Figura 53. Diagrama UML del Batch de predicción

```
(General)                                                    batchPredition

Sub batchPredition()
    Dim batch As String
    Dim dia As Byte
    Dim hora As Byte
    Dim minuto As Byte
    Dim contador As Integer
    contador = 2

    Cells(contador - 1, 1) = "day_week,hour,min"

    For dia = 0 To 6
        For hora = 0 To 23
            For minuto = 0 To 59
                Cells(contador, 1) = CStr(dia) + "," + CStr(hora) + "," + CStr(minuto)
                contador = contador + 1
            Next minuto
        Next hora
    Next dia
End Sub
```

Figura 54. Código de VBA desarrollada en Excel para generar el batch de predicción.

El resultado es el archivo *batch_smarthome.csv* con atributos de días de la semana, horas, y minutos, ver ***Figura 53*** y ***Figura 55***.

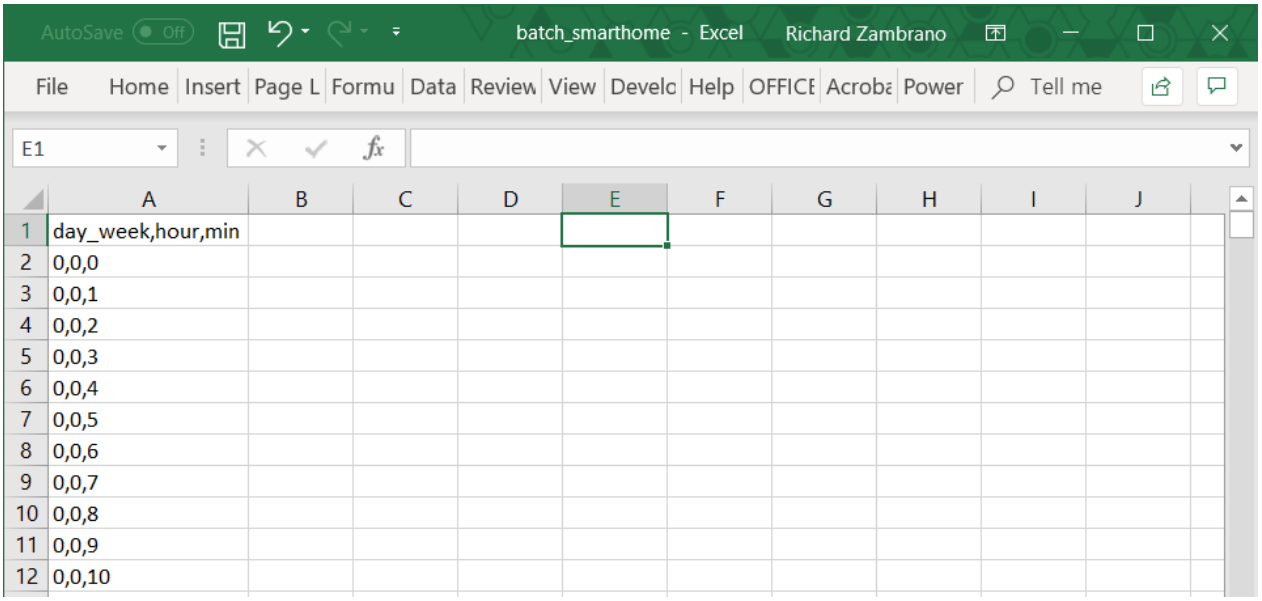


Figura 55. Batch de predicción generado por el código de VBA desarrollado en Excel

Los archivos csv (datos históricos de AWS RDS y el batch de predicción) son almacenados en WS S3 para realizar el procesamiento en AWS Machine Learning, ver ***Figura 56***.

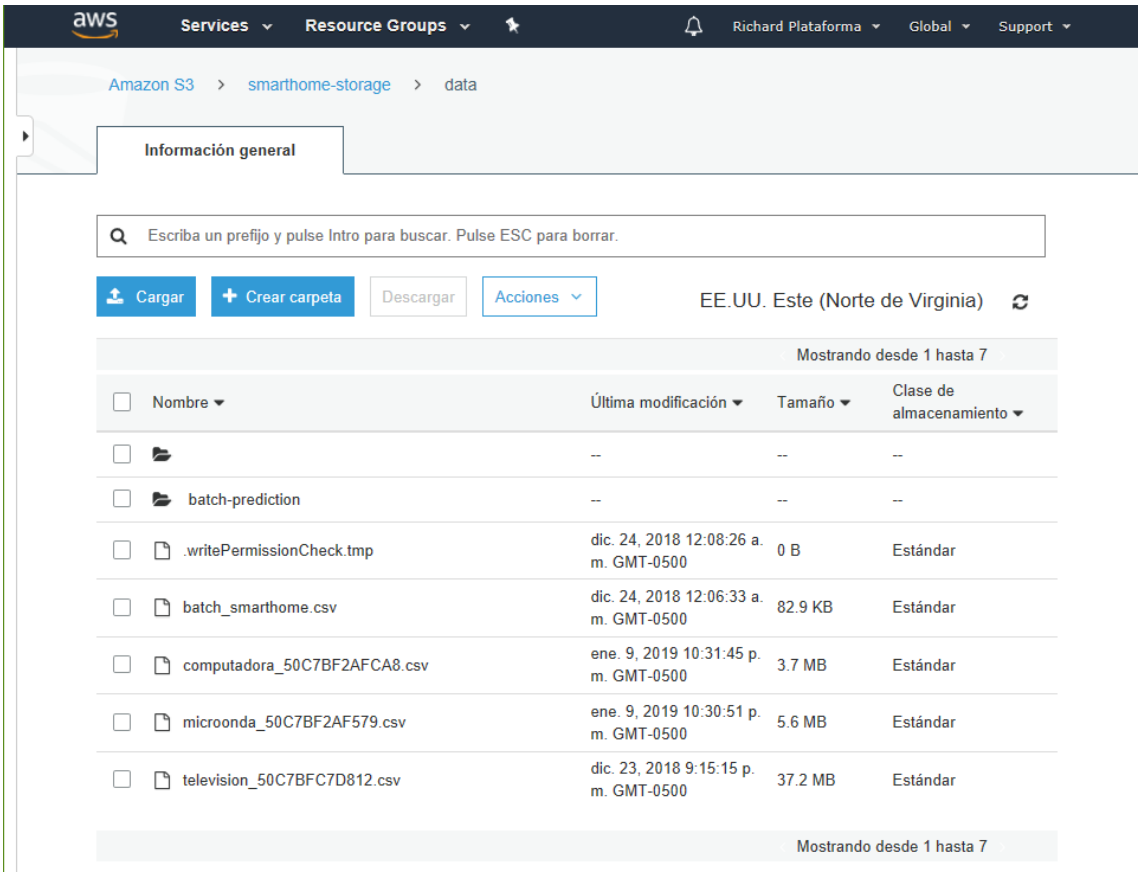


Figura 56. Archivo csv generados luego del procesamiento de los datos históricos de AWS RDS y el batch de predicción mediante Microsoft Excel.

4.1.1.2.3. Procesamiento en AWS Machine Learning

El procesamiento en AWS Machine Learning se realiza a partir de los archivos generados en la sección anterior. Para realizar el procesamiento se realiza los siguientes pasos:

En primer lugar, se define el almacén de datos, para ello se ingresa la dirección URL del archivo csv alojado en AWS S3, como se observa en la ***Figura 57***.

Los URL son los siguientes:

- https://s3.amazonaws.com/smarthome-storage/data/computadora_50C7BF2AFCA8.csv
- https://s3.amazonaws.com/smarthome-storage/data/microonda_50C7BF2AF579.csv
- https://s3.amazonaws.com/smarthome-storage/data/television_50C7BFC7D812.csv

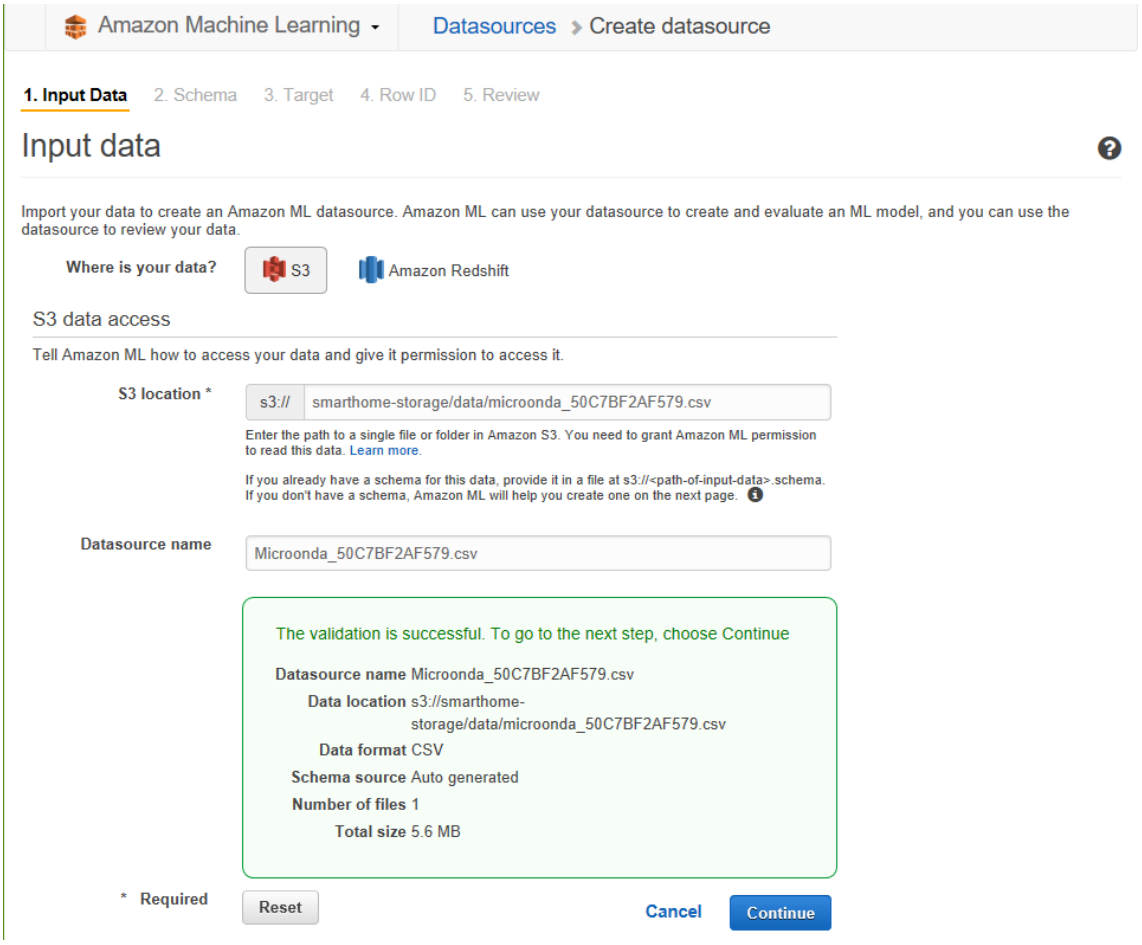


Figura 57. Ingreso de datos para generar el modelo en AWS Machine Learning mediante la consola de AWS.

A continuación, el esquema de datos se establece de tal manera de que permita identificar con exactitud el tipo de datos y atributos requeridos para generar el modelo ML, ver ***Figura 58***. Mientras que la ***Figura 59*** muestra el resumen del modelo de datos.

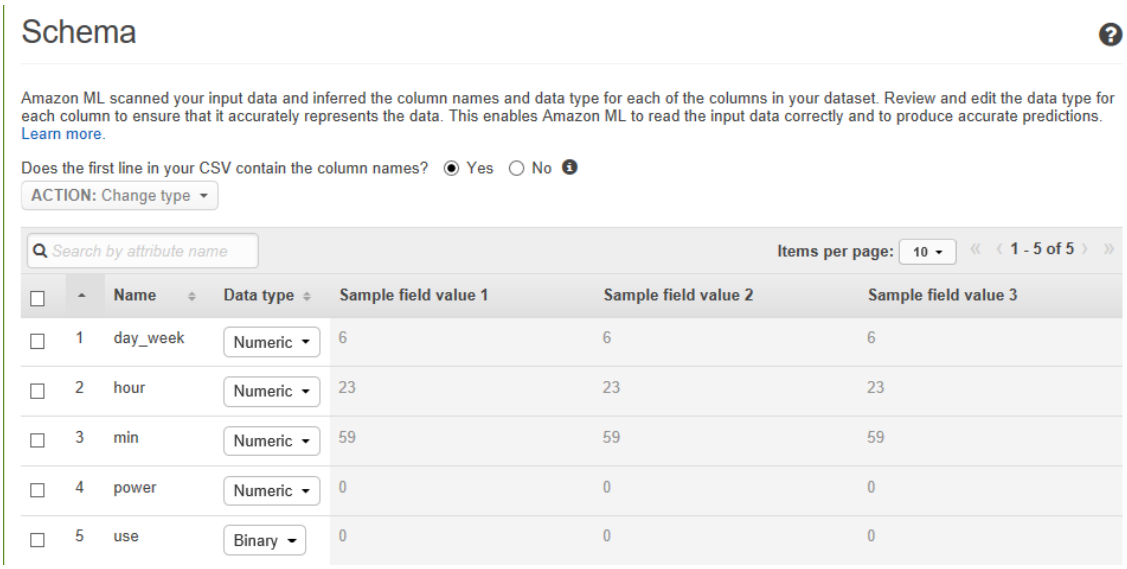


Figura 58. Esquema de datos definidos en AWS Machine Learning mediante la consola de AWS.

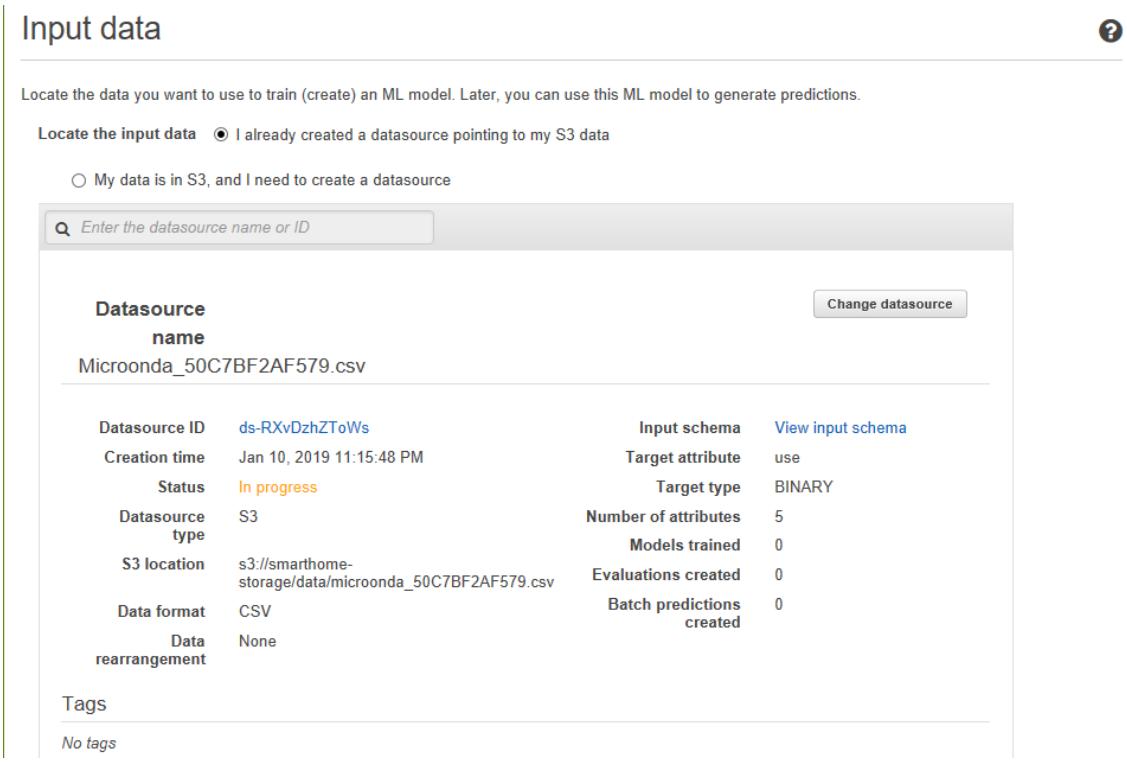


Figura 59. Resumen del origen de datos creado en AWS Machine Learning mediante la consola de AWS.

El siguiente paso es crear el modelo ML para definir como procesar el modelo de datos generado. El modelo ML seleccionado corresponde a un modelo binario con 70% de entrenamiento y 30% de evaluación, es decir, al sistema de gestión de energía le permitirá saber si un dispositivo electrónico se encuentra ENCENDIDO o APAGADO, mediante el uso del 70% de los datos suministrados y seleccionados aleatoriamente para aprender los hábitos de consumo, y el 30% restante para evaluar la eficiencia del modelo, la ***Figura 60*** se muestra el resumen del modelo ML.

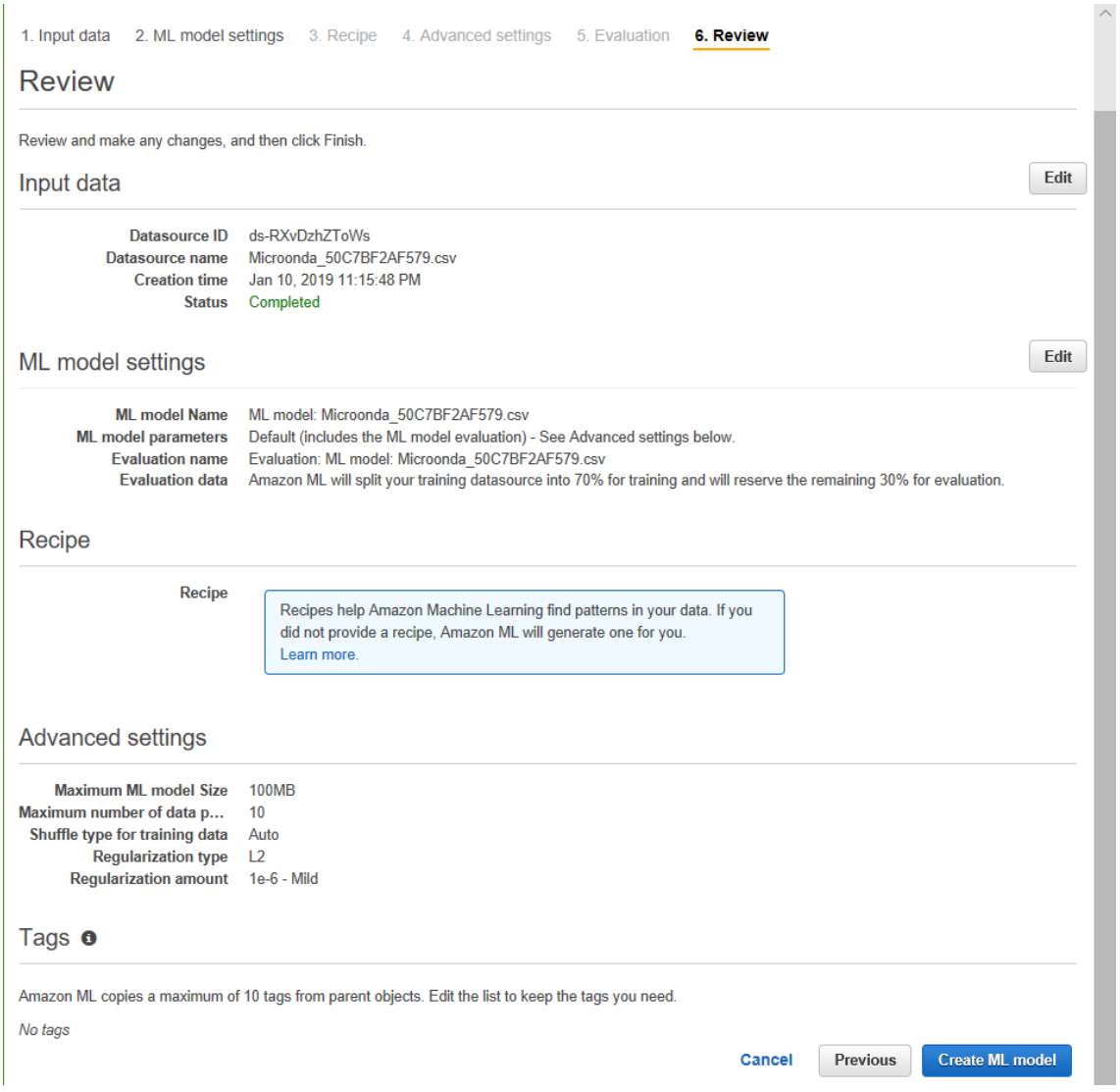


Figura 60. Resumen del modelo ML creado en AWS Machine Learning mediante la consola de AWS.

Al finalizar la creación del modelo ML, AWS Machine Learning almacena los registros de los archivos generados como se observa en la ***Figura 61***. Entre los archivos se encuentran:

- Fuente de datos
- Modelos ML
- Evaluaciones de modelos ML
- Predicciones

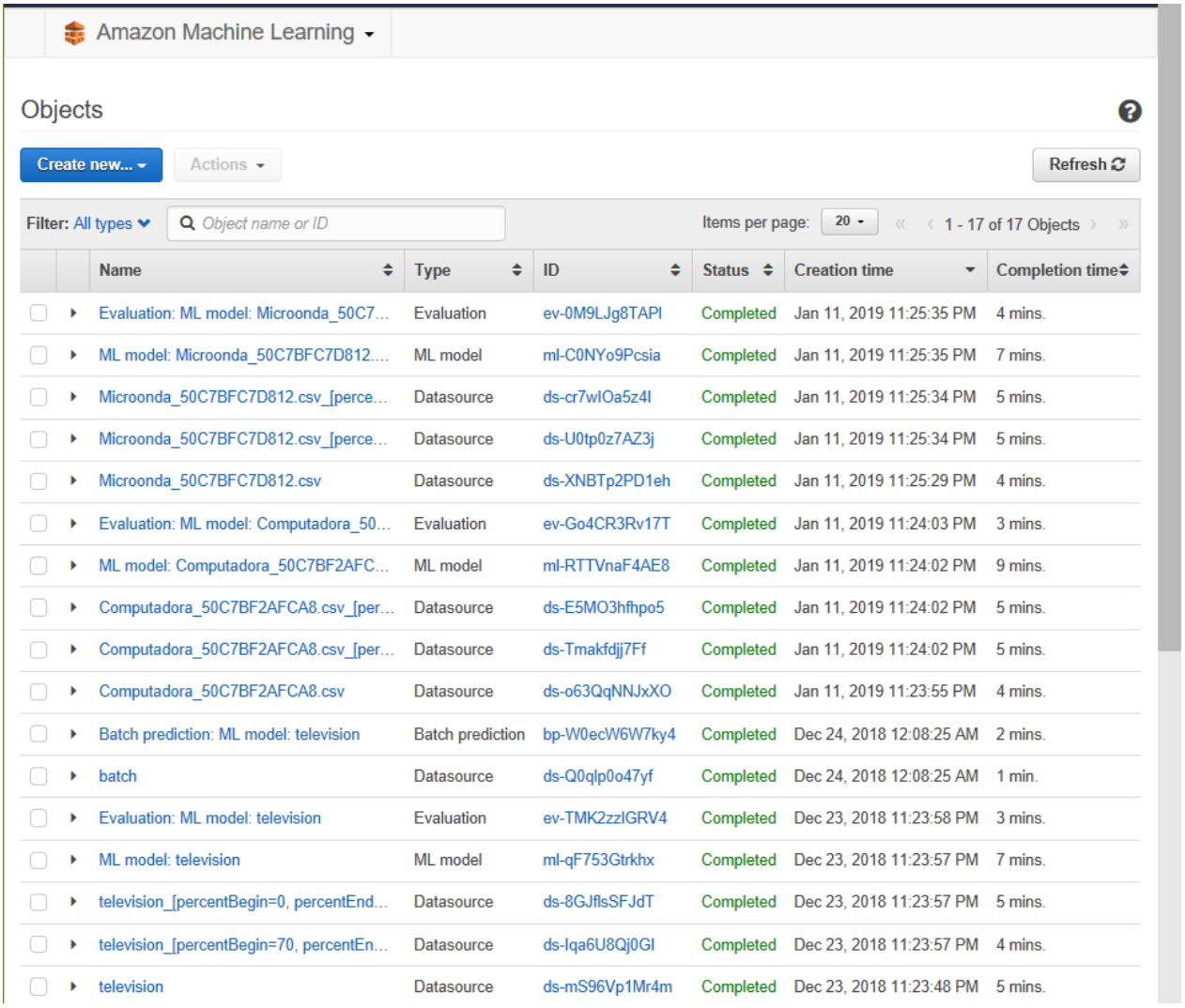


Figura 61. Almacenamiento de fuentes de datos, modelos ML, evaluación ML y batch de predicciones en AWS Machine Learning mediante la consola de AWS.

A continuación, se procede a ingresar el batch de predicción al modelo ML para cada uno de los modelos. El resumen de la creación se observa en la ***Figura 62***.

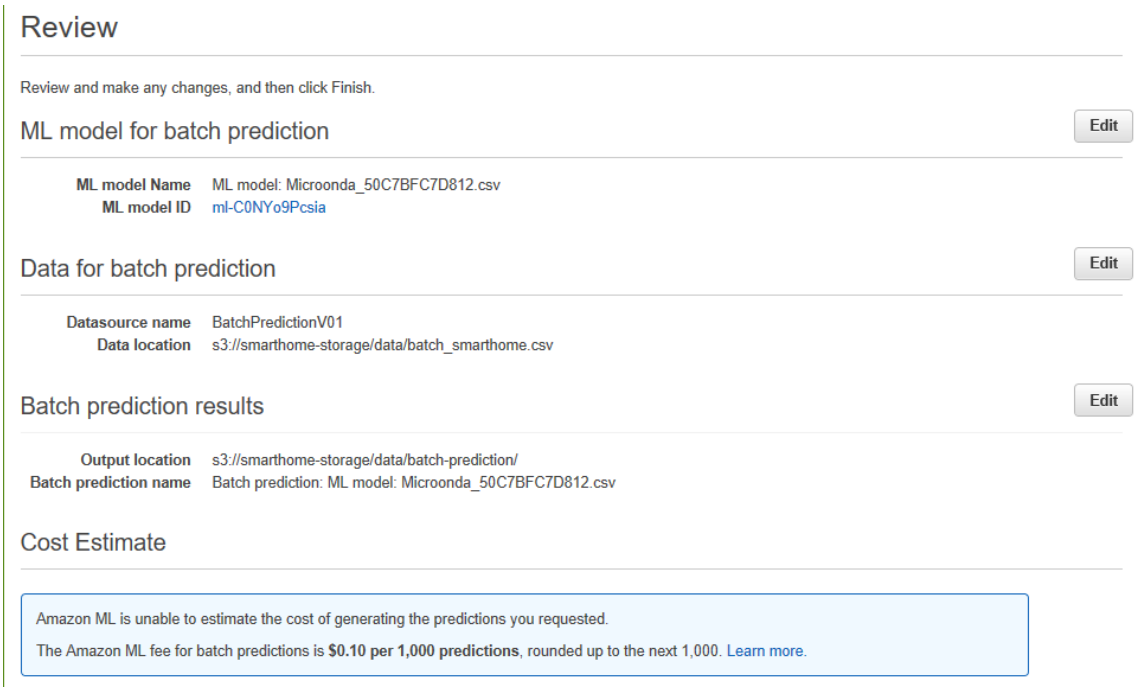


Figura 62. Creación de predicción del enchufe conectado a la microonda en AWS Machine Learning mediante la consola de AWS.

El resultado es almacenado en AWS S3 como se observa en la ***Figura 63***.

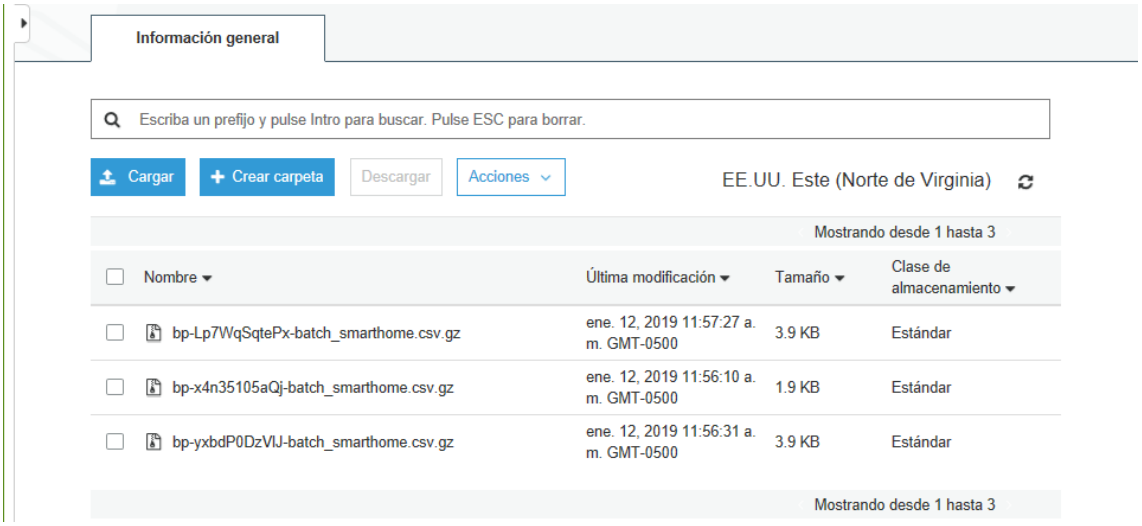


Figura 63. Archivos csv que contiene la predicción de cada modelo ML en AWS S3.

El archivo de batch de predicción y la predicción para cada uno de los enchufes es combinado en uno solo mediante el uso de Microsoft Excel, de tal manera que se forme una tabla con información sobre: número de la semana, horas, minutos, predicción, y precisión de la predicción, ver ***Figura 64***.

	A	B	C	D	E	F	G	H	I	J	K
1	day_week	hour	min	bestAnswer	score	day					
2	0	0	0	0	1289.099	DOMINGO					
3	0	0	5	0	1289.099	DOMINGO					
4	0	0	10	0	1090.926	DOMINGO					
5	0	0	15	0	1268.358	DOMINGO					
6	0	0	20	0	1231.909	DOMINGO					
7	0	0	25	0	1342.149	DOMINGO					
8	0	0	30	0	1316.986	DOMINGO					
9	0	0	35	0	1316.986	DOMINGO					
10	0	0	45	0	1238.253	DOMINGO					
11	0	0	50	0	1121.457	DOMINGO					
12	0	0	55	0	1269.789	DOMINGO					
13	0	1	0	0	1289.099	DOMINGO					

Figura 64. Datos combinados del batch y el resultado de la predicción de la computadora de escritorio generado en Microsoft Excel.

La nueva tabla creada permite realizar gráficos de barras para visualizar los hábitos de consumo de cada dispositivo electrónico (computadora de escritorio, microonda, y televisor). Por ejemplo, la gráfica del hábito de consumo de la computadora de escritorio se observa en la ***Figura 65***.

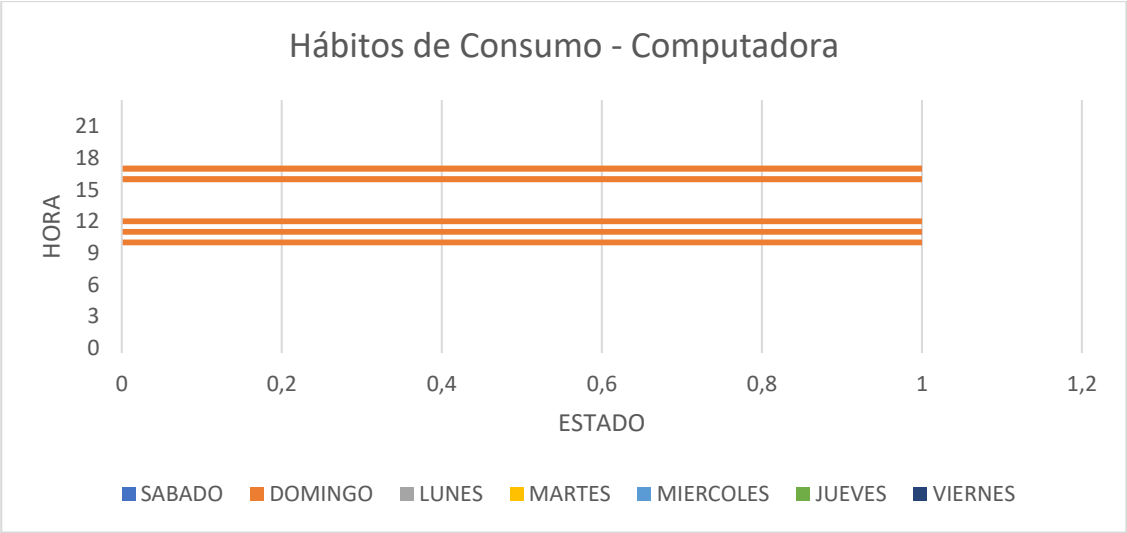


Figura 65. Gráfico de hábitos de consumo eléctrico de la computadora de escritorio generado en Microsoft Excel.

4.1.2. Desarrollo local

4.1.2.1. Servidor Node.js

El desarrollo del servidor consta del siguiente árbol de archivos, ver ***Figura 66***.

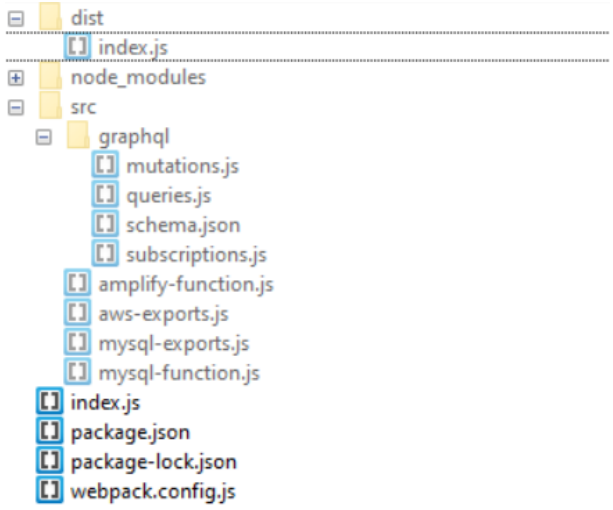


Figura 66. Árbol de archivos del Servidor Node.js

El árbol de archivos de la ***Figura 66*** muestra la mayoría de los archivos, a excepción del contenido de la carpeta "node_modules" debido al gran número de archivos que alberga. La descripción de cada una de las carpetas principales se encuentra en la **Tabla 19**.

Todo el código del servidor node.js se encuentra en el Anexo 3.

Tabla 19
Descripción de carpetas principales del árbol de archivos del servidor Node.js.

Carpeta	Descripción
./	Contiene todos archivos del servidor, entre ellos: carpetas y archivos de ejecución
dist	Contiene el ejecutable del servidor, generado mediante la librería webpack
node_modules	Contiene todas las librerías requeridas por el servidor.
src	Contiene archivos de ejecución secundarios y alberga información requerida para inicialización.
graphql	Contiene el esquema y los archivos de consulta para AWS Appsync, obtenidos a partir de la herramienta Amplify CLI. La librería AWS Amplify permite trabajar mediante CLI para facilitar la integración con la plataforma AWS.

En primera instancia hay que configurar el archivo package.json, el cual contiene toda la información correspondiente a dependencias y archivos de ejecución del servidor. La ***Figura 67*** muestra el archivo package.json generado mediante un editor de código llamado “Brackets”.

```json
{
  "name": "aws-appsync-nodejs-smarthome",
  "version": "1.0.0",
  "description": "App to get power data and sent to the AWS",
  "main": "index.js",
  "scripts": {
    "build": "webpack",
    "test": "node dist/index.js"
  },
  "author": "Richard Zambrano",
  "license": "",
  "dependencies": {
    "aws-amplify": "^1.1.16",
    "aws-appsync": "^1.3.4",
    "aws-sdk": "^2.326.0",
    "date-and-time": "^0.6.3",
    "es6-promise": "^4.2.5",
    "graphql": "^14.0.2",
    "graphql-tag": "^2.9.2",
    "isomorphic-fetch": "^2.2.1",
    "mariadb": "^2.0.2-rc",
    "tplink-smarthome-api": "^0.23.1",
    "ws": "^6.0.0"
  },
  "devDependencies": {
    "webpack": "^4.20.2",
    "webpack-cli": "^3.1.2"
  }
}
```

Figura 67. Contenido del archivo package.json perteneciente al servidor Node.js.

Antes de la programación, se requiere crear el entorno de desarrollo node.js en servidor web mediante el archivo webpack.config.js, ver ***Figura 68***.

```
module.exports = {
      context: __dirname,
      mode: 'development',
      entry: "./index.js",
      target: "node",
      devtool: "eval-source-map",
      output: {
        filename: "index.js"
      }
    }
```

Figura 68. Contenido del archivo webpack.config.js perteneciente al servidor Node.js.

La programación del servidor node.js requiere la inicialización del proyecto en la plataforma AWS, para ello se hace uso de la herramienta de desarrollo AWS Amplify mediante comandos CLI. Los recursos de Backend se crean a través del comando: *"amplify init"*, ver ***Figura 69***.

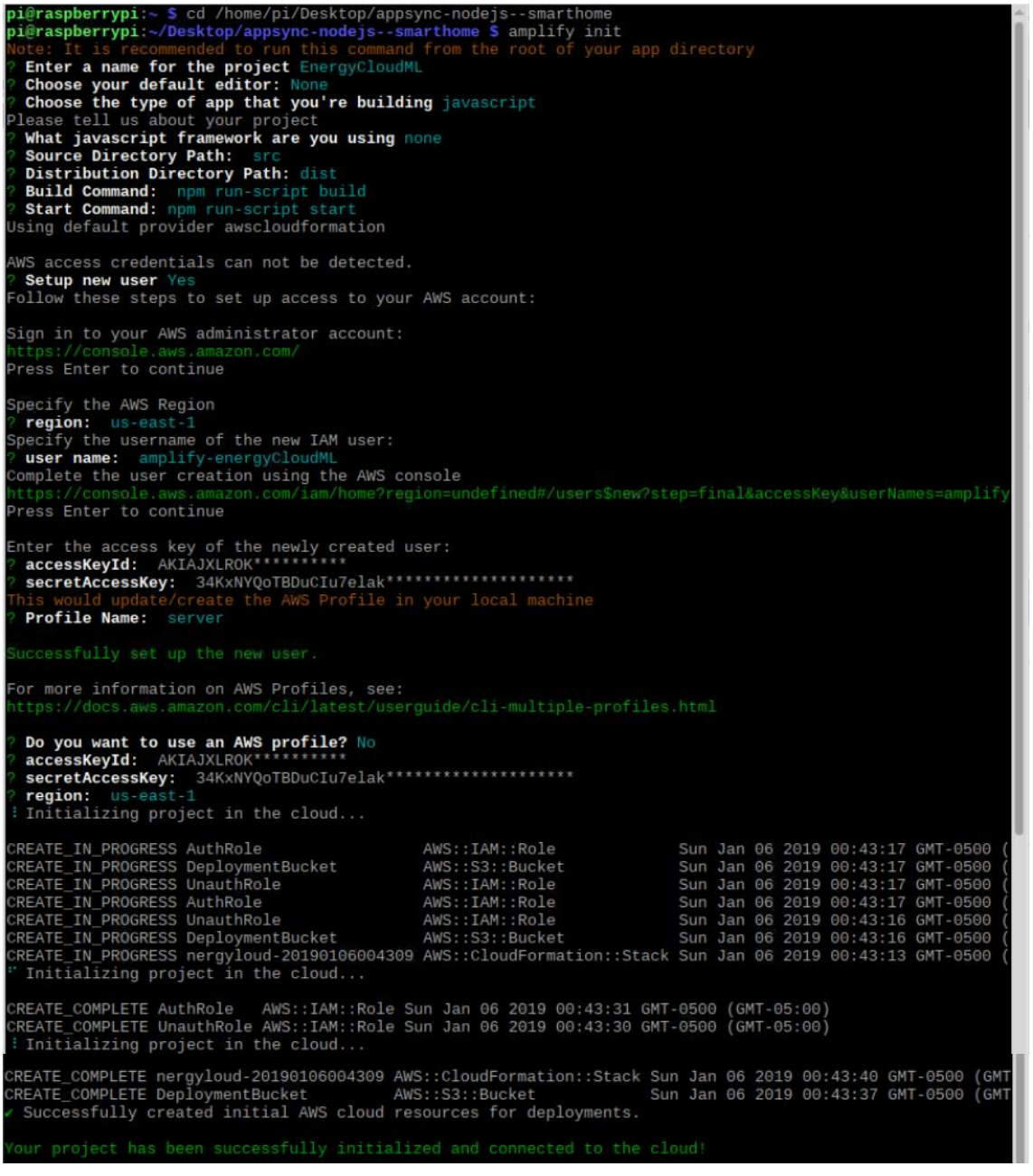


Figura 69. Inicialización del servidor Node.js en la plataforma AWS

Durante la inicialización se define o crea un nuevo usuario para acceder a la plataforma AWS, la ***Figura 70*** muestra el resumen de la creación del nuevo usuario. El usuario creado se llama "*amplify-energyCloudML*" el cual tiene únicamente acceso de administrador para programar, mediante el uso de credenciales.

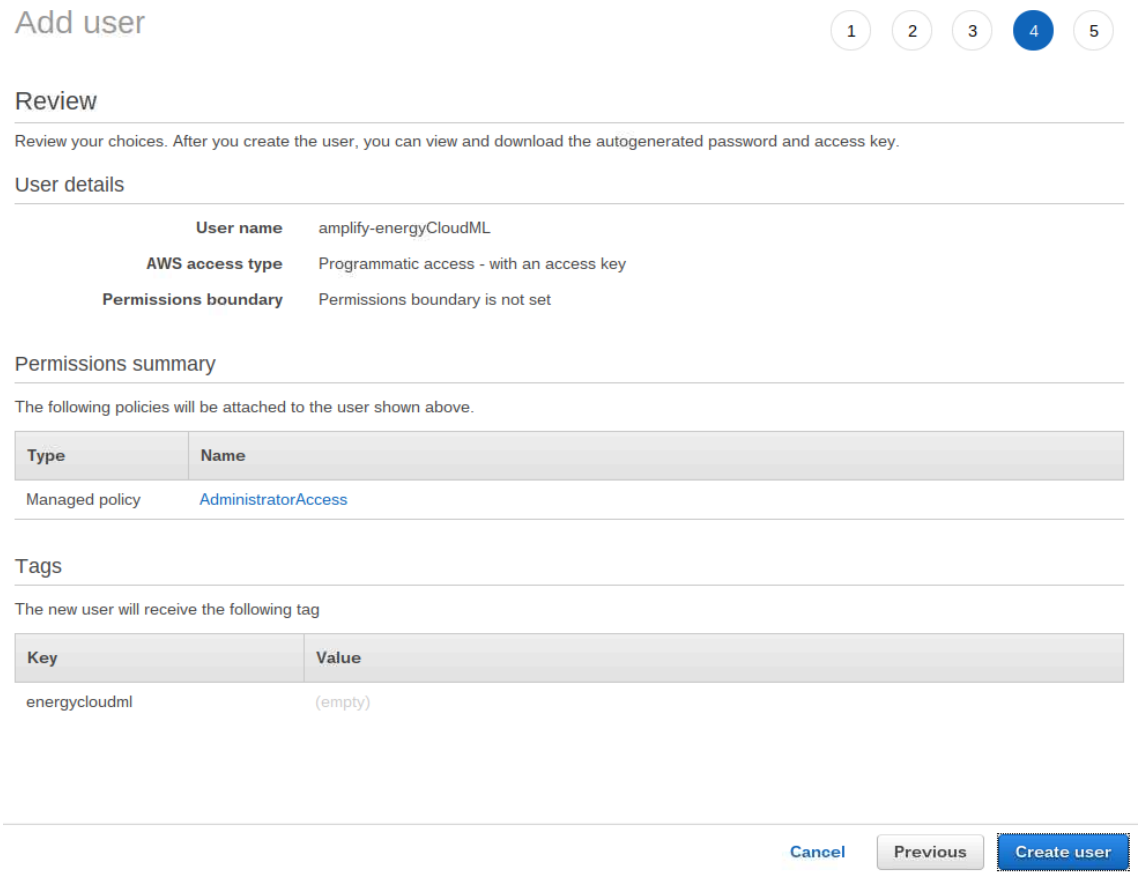


Figura 70. Resumen de la creación de usuario programador para acceder a la plataforma AWS mediante AWS IAM en la consola de AWS.

Ahora se integra al servidor local al microservicio AWS Appsync, a través del comando especificado en la plataforma de AWS Appsync, ver ***Figura 71***, para mantener comunicación con los datos en tiempo real, ver ***Figura 72***. Además, se define el ambiente como JavaScript para proceder a programar el servidor mediante Node.js

Add the codegen category to your project.

```
$ amplify add codegen --apiId wanv63wvg5e5ve7n4onf7hn6mu
```

Copy

This will automatically generate GraphQL documents (queries, mutations, and subscriptions) and generate types for your iOS application. If you modify the generated documents or your API's schema, you can regenerate the client code anytime with:

```
$ amplify codegen
```

Copy

Figura 71. Comandos requeridos para integrar AWS Appsync al proyecto en el servidor Node.js.

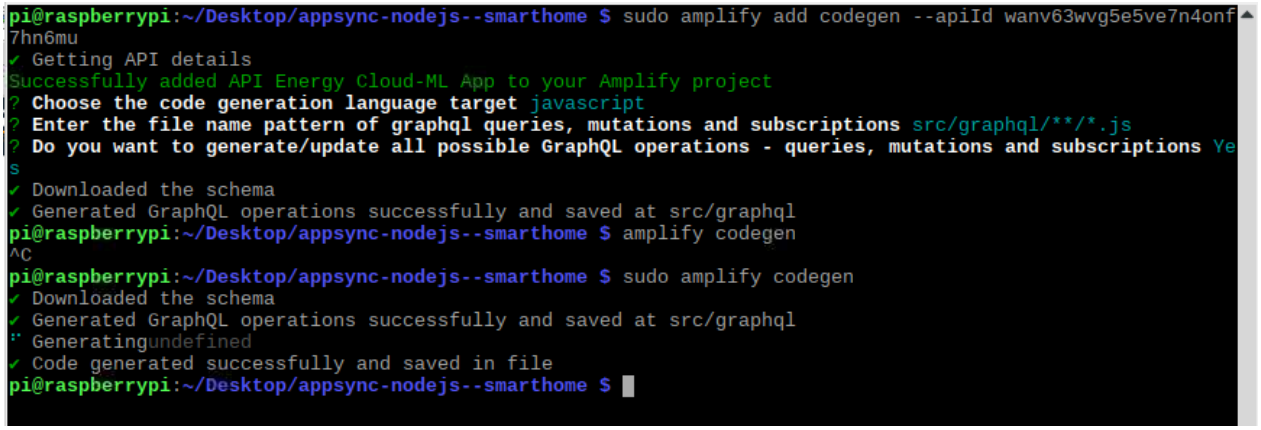


Figura 72. Integración de AWS Appsync al proyecto en el servidor Node.js.

En el archivo principal (index.js) se define todas las variables requeridas e inicializa la comunicación de la API y la plataforma AWS. Para ello se siguen los pasos descritos en el diseño de la infraestructura local en la ***Figura 21***.

En primer lugar, se mantiene comunicación con AWS Appsync como se puede observar en la ***Figura 73***. Mediante acceso al microservicio por AWS Amplify mediante usuario y contraseña como método de autenticación. Cabe recalcar que el acceso a la plataforma AWS es diferente al acceso hacia AWS Amplify, el primero es destinado al administrador, mientras que el segundo es destinado al usuario.

```
AppSync.buildAppSync(aws_exports)
.then(() => {realtimeSuscription ();
             AppSync.updateRealtimeMutation(inputMutation, mutationState)
                 .then((data) => {mutationState = data;});
            })
.catch(err => console.log(err));

export async function buildAppSync (config){
    var aws_exports = config;
    Amplify.configure(aws_exports);

    var promise = await Amplify.Auth.signIn(aws_exports.aws_cognito_username,
    aws_exports.aws_cognito_password)
    return promise;
}
```

Figura 73. Código requerido para acceder al microservicio AWS Appsync mediante AWS Amplify.

A continuación, se mantiene comunicación con la plataforma AWS, y accede a AWS Secrets Manager para inicializar la comunicación con la DB. En la ***Figura 74*** se observa el código para mantener comunicación con la plataforma AWS, AWS Secrets Manager y AWS RDS.

```
// AWS CONFIGURARION
AWS.config.update({
    region: aws_exports.aws_project_region,
    credentials: new AWS.Credentials({
        accessKeyId: aws_exports.AWS_ACCESS_KEY_ID,
        secretAccessKey: aws_exports.AWS_SECRET_ACCESS_KEY
    })
});
getRDSdb();

// RDS Initialation
function getRDSdb() {
    var clientAWS = new AWS.SecretsManager({
        region: aws_exports.aws_project_region
    });

    clientAWS.getSecretValue({SecretId: aws_exports.aws_secret_manager},
    function(err, data) {
        if (err) console.log(err)
        else {
            poolRDS =
            mysql_functions.RDSInitialization(JSON.parse(data['SecretString']));

             mysql_functions.createTable(poolRDS,titleGeneralTable,headerGeneralT
            able, 'RDS');
            mysql_functions.createTable(poolRDS,titleEventTable,headerEventTable,
            'RDS');
        }
    });
}
```

Figura 74. Código requerido para mantener comunicación con la plataforma AWS y la DB en AWS RDS.

En las instancias de DB local y AWS RDS se crea las tablas de datos que corresponden a *general_data* y *event_data* para alamacenar información relacionada con información básica del enchufe inteligente y los eventos generados al encender o apagar cada uno de ellos.

En la ***Figura 75*** se observa las principales funciones MYSQL para estructurar las DBs.

```
exports.dBInitialization = function dBInitializaion (){
    pool = mariadb.createPool({
         host: mysql_exports.host,
         user: mysql_exports.user,
         password: mysql_exports.password,
         database: mysql_exports.database
         //connectionLimit: 5
    });
    return pool;
}

//mysql
exports.createTable = function createTable (pooldB, name, header, dB){
    var sql = "CREATE TABLE " + name + " (" + header + ")";
    if (dB === 'Local') pool.query(sql)
        .then((data)=>{console.log(name," table was created in the ",dB,' database.');})
        .catch(err=>{console.log(name," table wasn't created in the ",dB," database. Error:
        ",err['code'])});
    if (dB === 'RDS') poolRDS.query(sql)
        .then((data)=>{console.log(name," table was created in the ",dB,' database.');})
        .catch(err=>{console.log(name," table wasn't created in the ",dB," database. Error:
        ",err['code'])});
};

exports.insertGeneralData = function insertGeneralData (name,header,datos){
    var sql = "INSERT INTO " + name + " (" + header + ") VALUES
    (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
    if (pool) pool.query(sql,datos)
        .then((data)=>{console.log("Record saved in ", name,'. Database: Local');})
        .catch(err=>{console.log("Record didn't save in ", name,". Database: Local. Error:
        ",err['code'])});
    if (poolRDS) poolRDS.query(sql,datos)
        .then((data)=>{console.log("Record saved in ", name,'. Database: RDS');})
        .catch(err=>{console.log("Record didn't save in ", name,". Database: RDS. Error:
        ",err['code'])});
};

exports.updateGeneralData = function updateGeneralData (name,header, key, datos){
    var sql = "UPDATE " + name + " SET " + header + " WHERE mac = '" + key + "' ;";
    if (pool) pool.query(sql,datos)
        .then((data)=>{console.log("Record updated in in ", name,'. Database: Local');})
        .catch(err=>{console.log("Record didn't save in ", name,". Database: Local. Error:
        ",err['code'])});
    if (poolRDS) poolRDS.query(sql,datos)
        .then((data)=>{console.log("New record updated in ", name,'. Database: RDS');})
        .catch(err=>{console.log("Record didn't save in ", name,". Database: RDS. Error:
        ",err['code'])});
};
```

Figura 75. Código requerido para interactuar con las DBs mediante consultas mysql.

Luego de establecer comunicación con las DBs y con AWS Appsync se inicializa la comunicación con la API, la ***Figura 76*** muestra el código principal que permite descubrir todos los dispositivos IoT TP-Link conectados a la red local. Los dispositivos son configurados de tal manera, que envíen la información instantánea en un periodo de 2 segundos, los cuales son almacenados en las tablas de las DBs y AWS Appsync respectivamente.

```
// Search for all plugs
clientApi.on('plug-new', function (plug) {
    console.log('Found plug:', plug.alias, 'Host:', plug.host);
    count[plug.mac] = 0;

    if (plug['_sysInfo'].relay_state === 1) state[plug.mac] = true;
    else {state[plug.mac] = false;}
    numDev++;

    if (mutationState) {
        console.log('New device Appsync');
        AppSync.createGeneralMutation(AppSync.generalInfo(plug), mutationDevice)
            .then((data) => mutationDevice = data);
        mutationDevice[plug.mac] = 'Ready';
    }
    generalData = mysql_functions.commonData(numDev);
    generalData = generalData.concat(Object.values(plug['_sysInfo']).splice(1),plug.host,
    null);
    mysql_functions.insertGeneralData(titleGeneralTable, headerGeneralData, generalData);

    titleEmeterTable[numDev-1] = plug.type.replace(".","_") + '_' +
    plug.mac.replace(/:/g,"");
    mysql_functions.createTable(titleEmeterTable[numDev-1],headerEmeterTable);

    plug.startPolling(2000);
});
```

Figura 76. Código requerido para establecer comunicación con los dispositivos IoT de TP-Link mediante la API.

Además, en la ***Figura 77*** se declaran eventos destinados a conocer los parámetros de consumo eléctrico en tiempo real de los dispositivos conectados como el encendido y apagado de los enchufes, y si se encuentran en uso o no (existen fallas en reconocimiento).

El evento *'emeter-realtime-update'* envía los parámetros de consumo eléctrico cada 2s definido en la ***Figura 76***. Los valores son almacenados en las DBs y enviados a AWS Appsync para transmitir los datos en tiempo real en la aplicación web. El resto de los eventos son almacenados

únicamente en la tabla *event_data* de la DB para tener un registro y futura utilización de los datos proporcionados.

```
plug.on('emeter-realtime-update', (emeterRealtime) => {
    if (count[plug.mac] === 2){
        var title = plug.type.replace(".","_") + '_' + plug.mac.replace(/:/g,"");
        emeterData = mysql_functions.commonData(null);
        emeterData = emeterData.concat(Object.values(emeterRealtime).splice(0,4),plug.mac);
        mysql_functions.insertEmeterData(title, headerEmeterData, emeterData);
        count[plug.mac] = 0;
    }
    else {count[plug.mac]++;}
    if (mutationState) {
        if (mutationDevice[plug.mac]) {
            console.log('Update...',plug.mac);
        }
        else {
            console.log('Creation...',plug.mac);
            AppSync.createGeneralMutation(AppSync.generalInfo(plug), mutationDevice)
                .then((data) => mutationDevice = data);
            mutationDevice[plug.mac] = 'Ready';
        }
    }
});

plug.on('in-use', () => {
    eventData = mysql_functions.commonData(null);
    eventData = eventData.concat('1',null,null,plug.mac);
    mysql_functions.insertEventData(titleEventTable, headerEventData, eventData);
});

plug.on('not-in-use', () => {
    eventData = mysql_functions.commonData(null);
    eventData = eventData.concat('0',null,null,plug.mac);
    mysql_functions.insertEventData(titleEventTable, headerEventData, eventData);
});

plug.on('power-on', () => {
    eventData = mysql_functions.commonData(null);
    eventData = eventData.concat(null,'1',null,plug.mac);
    mysql_functions.insertEventData(titleEventTable, headerEventData, eventData);
    state[plug.mac] = true;
    console.log('current state: ',state);
});

plug.on('power-off', () => {
    eventData = mysql_functions.commonData(null);
    eventData = eventData.concat(null,'0',null,plug.mac);
    mysql_functions.insertEventData(titleEventTable, headerEventData, eventData);
    state[plug.mac] = false;
    console.log('current state: ',state);
});
```

Figura 77. Código requerido para declarar los eventos mediante la API.

Finalmente, se emplea las consultas en AWS Appsync para mantener comunicación con el usuario mediante la aplicación web. Cada vez que el usuario envía una señal de control, es decir, el usuario enciende o apaga el enchufe inteligente desde el aplicativo, el servidor node.js ejecuta el código definido en la ***Figura 78***. Cabe recalcar que el servidor realiza la consulta al microservicio cada 30s para evitar saturar la red local.

```
setInterval(
        ()=>{getControlSignal();},
        30000
    );

function getControlSignal(){
    AppSync.stateQuery()
        .then(data => {
            var aux = data.data.listControlStatus.items;
            var control = {};
            for (var i = 0; i < aux.length; i++){
                control[aux[i].mac] = aux[i].control;
            }
            var keys = Object.keys(state);

            for (var i = 0; i< keys.length; i++){
                if (isNaN(control[keys[i]])) console.log(keys[i], 'no existe valor');
                else {
                    if (state[keys[i]] === control[keys[i]])
                        console.log(keys[i], 'no requiere cambio');
                    else
                        console.log(keys[i], 'requiere cambio');
                }
            }
        })
        .catch(err => {console.log(err);});
}

export async function stateQuery (){
    const control = await Amplify.API.graphql(graphqlOperation(query.listControlStatus));
    return control;
}
```

Figura 78. Código requerido para encender y apagar los enchufes inteligentes desde el aplicativo web.

4.1.2.2. Aplicativo web

El árbol de archivos de la ***Figura 79*** muestra la estructura de archivos del aplicativo web, a excepción del contenido de la carpeta *"node_modules"* debido al gran número de archivos que contiene. La descripción de cada una de las carpetas principales se encuentra en la **Tabla 20**.

Todo el código del servidor node.js se encuentra en el Anexo 4.

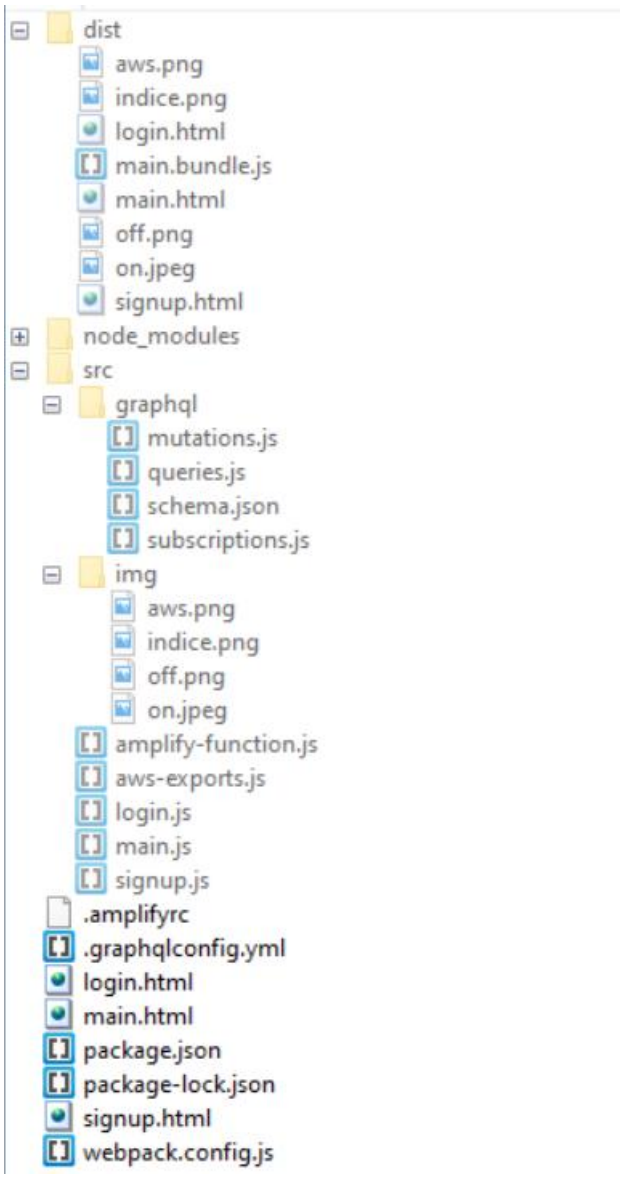


Figura 79. Árbol de archivos del aplicativo web

Tabla 20
Descripción de carpetas principales del árbol de archivos de la aplicación web.

Carpeta	Descripción
./	Contiene todos archivos del servidor, entre ellos: carpetas y archivos de ejecución
dist	Contiene las páginas HTML, archivo condensado de JavaScript, e imágenes del aplicativo generados mediante la librería webpack
node_modules	Contiene todas las librerías requeridas por la aplicación web
src	Contiene archivos de ejecución secundarios y alberga información requerida para inicialización.
graphql	Contiene el esquema y los archivos de consulta para AWS Appsync, obtenidos a partir de la herramienta Amplify CLI. La librería AWS Amplify permite trabajar mediante CLI para facilitar la integración con la plataforma AWS.
img	Contiene las imágenes requeridas por la aplicación web.

Primero hay que configurar el archivo package.json, el cual contiene toda la información correspondiente a las dependencias y archivos de ejecución del servido. La ***Figura 80*** muestra el archivo package.json generado mediante un editor de código llamado “Brackets”.

```
{
  "name": "amplify-js-app-smarthome",
  "version": "1.0.0",
  "description": "AWS Smarthome JavaScript",
  "dependencies": {
    "aws-amplify": "^1.1.10",
    "chart.js": "^2.7.3",
    "mariadb": "^2.0.2-rc",
    "momentjs": "^2.0.0"
  },
  "devDependencies": {
    "webpack": "^4.17.1",
    "webpack-cli": "^3.1.0",
    "copy-webpack-plugin": "^4.5.2",
    "webpack-dev-server": "^3.1.5"
  },
  "scripts": {
    "start": "webpack && webpack-dev-server --mode development",
    "build": "webpack"
  }
}

```

Figura 80. Contenido del archivo package.json perteneciente a la aplicación web.

Además, se requiere crear el entorno de ejecución de la aplicación web mediante el archivo webpack.config.js, ver ***Figura 81***. Entre el código desarrollado se detalla la ubicación de los archivos de javascript y html, además de la versión de compilación de JS.

```
const CopyWebpackPlugin = require('copy-webpack-plugin');
const webpack = require('webpack');
const path = require('path');

module.exports = {
    mode: 'development',
    entry: ['./src/app.js','./src/login.js','./src/signup.js','./src/main.js'],
    output: {
        filename: '[name].bundle.js',
        path: path.resolve(__dirname, 'dist')
    },
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/
            }
        ]
    },
    devServer: {
        contentBase: './dist',
        overlay: true,
        hot: true
    },
    plugins: [
        new
        CopyWebpackPlugin(['login.html','signup.html','main.html','src/img']),
        new webpack.HotModuleReplacementPlugin()
    ]
};
```

Figura 81. Contenido del archivo webpack.config.js perteneciente a la aplicación web.

En general, la ***Figura 80*** y ***Figura 81*** define los requerimientos necesarios para la aplicación. Por ejemplo, *"webpack-dev-server"* crea el lanzador de la página web, mientras que *"webpack.config"* permite consolidar el código de JavaScript y definir los archivos de ejecución.

Al igual que el servidor node.js, la aplicación web requiere la inicialización del proyecto en la plataforma AWS y AWS Amplify. El usuario definido para la aplicación corresponde al usuario

de administración ya generado en el servidor, ver ***Figura 70***, para lo cual se hace uso de "accessKeyId" y "secretAccessKey" como credenciales de acceso, ver ***Figura 82***.

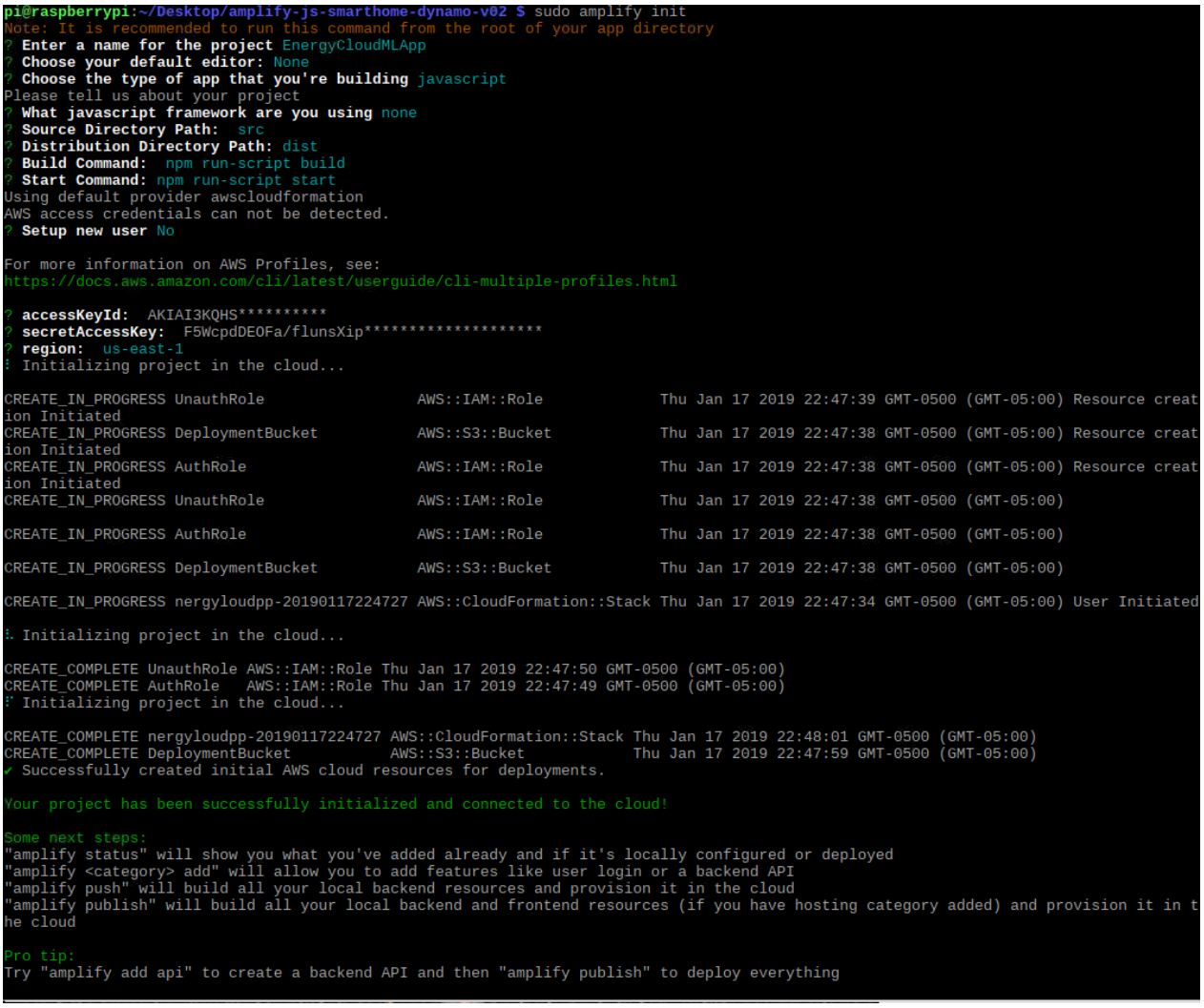


Figura 82. Inicialización de la aplicación web en la plataforma AWS

Ahora definimos 2 microservicios esenciales para la aplicación web, ver ***Figura 83***, como son:

- AWS Cognito
- AWS Appsync

AWS Appsync se define a partir del API ID definido en la consola de AWS como se observó en la ***Figura 71***. Mientras tanto, AWS Cognito se genera automáticamente desde cero como método de autenticación, ver ***Figura 83***.

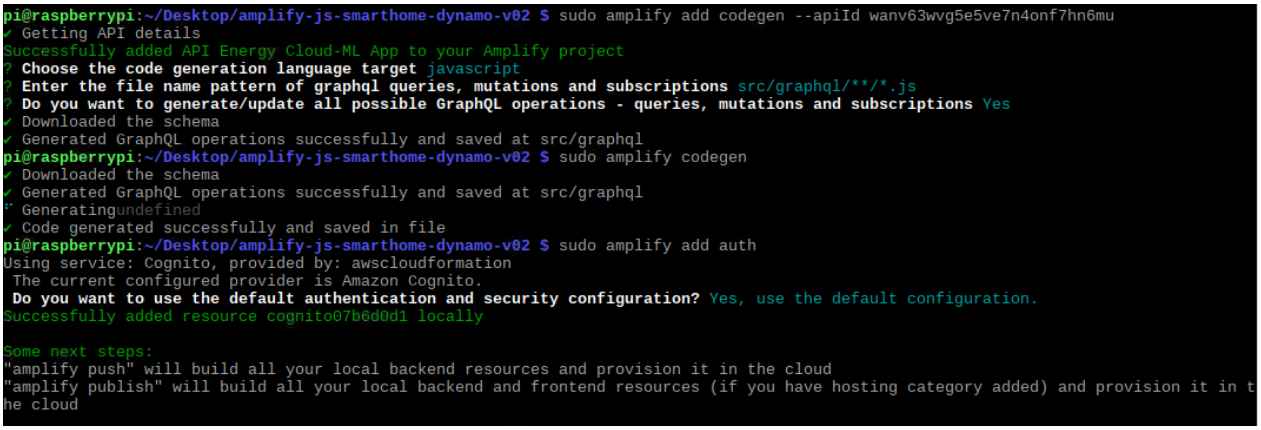


Figura 83. Integración de AWS Appsync y AWS Cognito a la aplicación web.

La configuración de AWS Cognito se realizó de tal manera que cumpla con los requisitos establecidos en el apartado 3.2.3.11. Además, como se observa en la ***Figura 84***, se estableció como método de verificación el correo electrónico, y políticas de seguridad en la contraseña, que son:

- Uso de caracteres especiales
- Uso de letras mayúsculas y minúsculas
- Uso de caracteres numérico.

Además, la aplicación web requiere tener un servidor web, por lo tanto, mediante CLI se define el hosting correspondiente, es decir, se almacena los archivos de ejecución en un bucket de AWS S3 como servidor estático para acceso de cada uno de los usuarios. Y finalmente se crea todo el backend mediante el comando *"sudo amplify publish"*, ver ***Figura 85***.

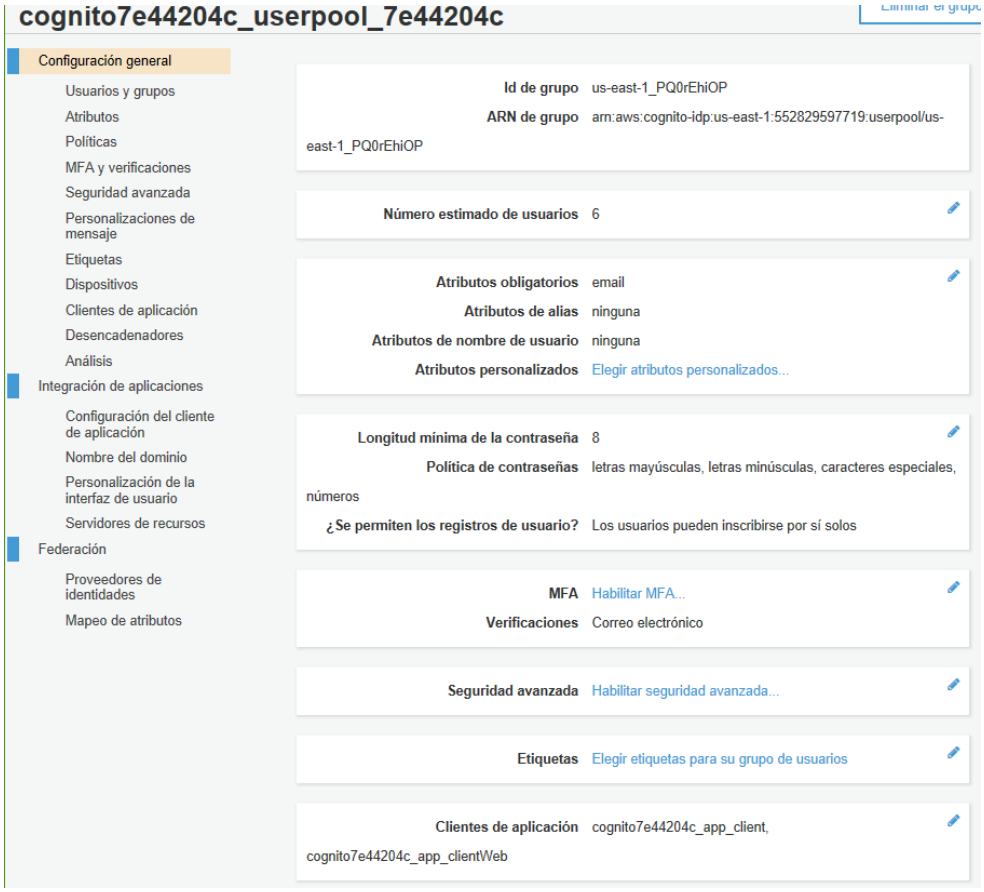


Figura 84. Configuración general de AWS Cognito mediante CLI.

```
pi@raspberrypi:~/Desktop/amplify-js-smarthome-dynamo-v02 $ sudo amplify publish

Please add hosting to your project before publishing your project
Command: amplify hosting add

pi@raspberrypi:~/Desktop/amplify-js-smarthome-dynamo-v02 $ sudo amplify add hosting
? Select the environment setup: DEV (S3 only with HTTP)
? hosting bucket name EnergyCloudMLHosting
? index doc for the website index.html
? error doc for the website index.html

You can now publish your app using the following command:
Command: amplify publish

pi@raspberrypi:~/Desktop/amplify-js-smarthome-dynamo-v02 $ sudo amplify publish
| Category | Resource name          | Operation | Provider plugin   |
| -------- | ---------------------- | --------- | ----------------- |
| Auth     | cognito07b6d0d1        | Create    | awscloudformation |
| Hosting  | S3AndCloudFront        | Create    | awscloudformation |
| Api      | Energy Cloud-ML App | No Change |                   |
? Are you sure you want to continue? true
⠦ Updating resources in the cloud. This may take a few minutes...

CREATE_IN_PROGRESS hostingS3AndCloudFront     AWS::CloudFormation::Stack Thu Jan 17 2019 22:59:23 GMT-0500 (GMT-05:00) Resource creat
ion Initiated
CREATE_IN_PROGRESS authcognito07b6d0d1        AWS::CloudFormation::Stack Thu Jan 17 2019 22:59:22 GMT-0500 (GMT-05:00) Resource creat
ion Initiated
CREATE_IN_PROGRESS hostingS3AndCloudFront     AWS::CloudFormation::Stack Thu Jan 17 2019 22:59:22 GMT-0500 (GMT-05:00)

CREATE_IN_PROGRESS authcognito07b6d0d1        AWS::CloudFormation::Stack Thu Jan 17 2019 22:59:21 GMT-0500 (GMT-05:00)

UPDATE_IN_PROGRESS nergyloudpp-20190117224727 AWS::CloudFormation::Stack Thu Jan 17 2019 22:59:17 GMT-0500 (GMT-05:00) User Initiated

⠏ Updating resources in the cloud. This may take a few minutes...
```

Figura 85. Creación del hosting e integración de la aplicación web con la plataforma de AWS.

Definido las herramientas requeridas para la aplicación, ahora se procede a programar la aplicación web. La página web principal es *"main.html"* y el archivo de ejecución principal es *"main.js"*, sin embargo, primero se genera el código HTML y JS de la página de autenticación, llamados *"login.html"* y *"login.js"* respectivamente. La ***Figura 86*** muestra la página web visualizada en un explorador.

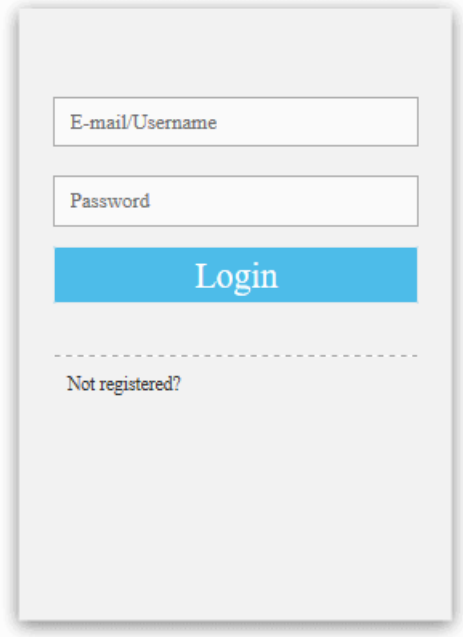


Figura 86. Visualización de la página web login.html

La ***Figura 86*** muestra la visualización de la página web, pero la ***Figura 87*** es el código realizado con JavaScript como complemento de la página web para realizar la autenticación con AWS Cognito, caso contrario, no es posible acceder al *"main.html"*, y menos aún a los servicios de AWS.

```
LoginEventButton.addEventListener('click', (evt) => {
    let username = LoginUsername.value;
    let password = LoginPassword.value;
    Auth.signIn(username, password)
        .then((user) => {document.location.href = LoginUrl;})
        .catch(err => console.log(err)) ;
```

Figura 87. Código requerido para autenticar al usuario y contraseña con AWS Cognito.

Como se observa en la ***Figura 86***, hay un link ubicado en la parte inferior del botón *"Login"* denominado "*not registered?"*, este enlace nos da la posibilidad de crear nuevos usuarios al sistema de gestión. Al dar clic, nos redirecciona a la ***Figura 88*** cuya página web se encuentra conformada por los archivos *"signup.html"* y *"signup.js"*. Entre los campos requeridos constan el nombre de usuario, email, y contraseña.

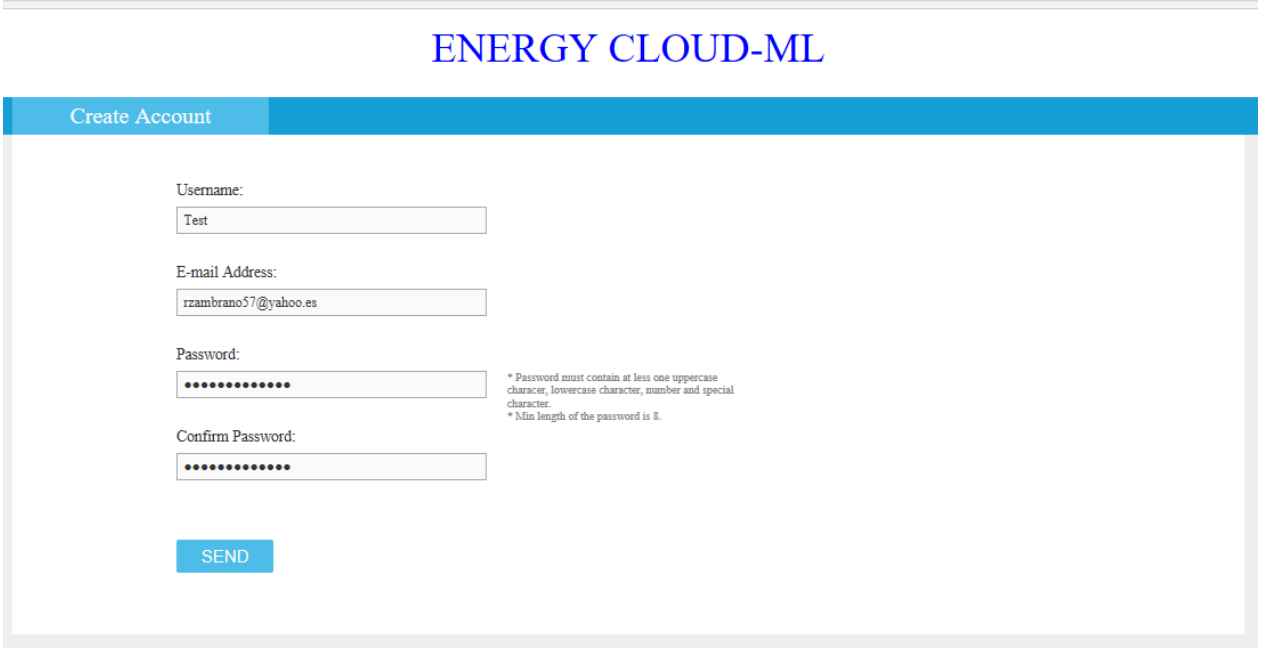


Figura 88. Visualización de la página web signup.html

El archivo *"signup.js"* permite interactuar con AWS Cognito para crear un nuevo usuario y verificar mediante correo electrónico. La ***Figura 91*** muestra el código generado para autenticar y verificar los datos proporcionados en la ***Figura 89***.

```
function signup(){
    let username = SignupUsername.value;
    let email = SignupEmail.value;
    let password = SignupPassword.value;
    console.log(username,email,password);

    Auth.signUp({
        username,
        password,
        attributes: {
            email
        },
        validationData: []  //optional
    })
    .then(data => {console.log(data);
                   confirmation();})
    .catch(err => {console.log(err);
                   let msg = 'ERROR: ' + err.message;
                   passwordReset(msg);});

}

function verificationCode (){
    let username = SignupUsername.value;
    let code = SignupCode.value;
    // After retrieveing the confirmation code from the user
    Auth.confirmSignUp(username, code, {
        // Optional. Force user confirmation irrespective of existing alias.
        By default set to True.
        forceAliasCreation: true})
    .then(data => {console.log(data);
                   document.location.href = SignupUrl;})
    .catch(err => {console.log(err);
                   let msg = 'ERROR: ' + err.message;
                   alert(msg);
                   SignupCode.value = "";});
}
```

Figura 89. Código requerido para crear y verificar la nueva cuenta creada en AWS Cognito.

Cabe destacar que, durante el proceso de autenticación, AWS Cognito envía un código de verificación al correo, el cual debe ser ingresado en la plataforma, caso contrario el usuario será eliminado en 7 días a partir de la creación.

El nombre de usuario y contraseña son ingresados en la ***Figura 86*** para acceder al sistema de gestión de energía, que a partir de ahora se denominará *"Energy Cloud-ML"*.

Energy Cloud-ML está conformada de 2 pestañas o contenidos principales. La ***Figura 91*** corresponde a la información del dispositivo, mientras que la ***Figura 90*** corresponde a información de la cuenta de usuario.

Figura 90. Visualización de la página web main.html (información sobre los dispositivos)

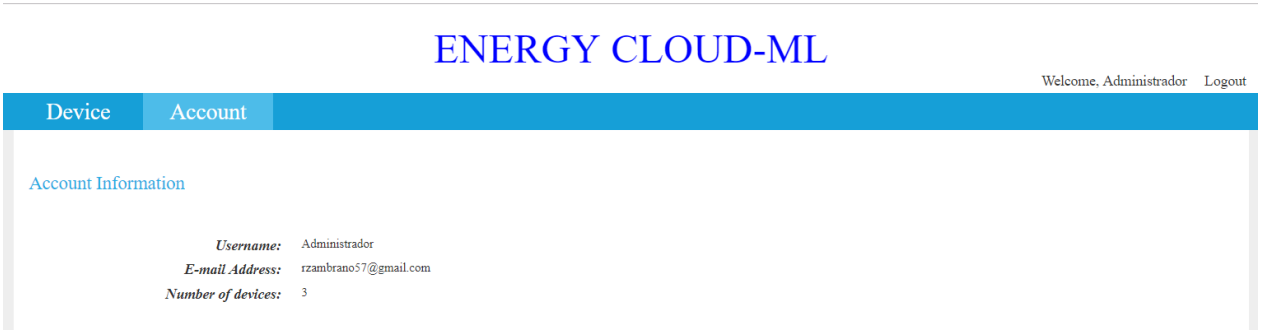


Figura 91. Visualización de la página web main.html (información sobre la cuenta)

La información del dispositivo se subdivide en cuatro partes:

1) **Información del dispositivo (Device Information)**

Muestra la información útil para el usuario como: nombre, tipo, mac, host, latitud y longitud del dispositivo. Para ello se acceder a AWS Appsync para obtener los dispositivos registrados en la plataforma, esto se realiza mediante el código de la ***Figura 92***.

```
Appsync.generalInfoQuery()
    .then(data => {
        deviceInfo = data;
        devicesAccount.innerHTML = Object.keys(deviceInfo).length;
        macID = Object.keys(deviceInfo)[0];
        realtimeSuscription (macID);
    });

export async function generalInfoQuery (){
    const appsyncquery = await
    API.graphql(graphqlOperation(queries.listEnergyCloudMls))
    .then(data => {
        var aux = data.data.listEnergyCloudMLS.items;
        for (var i = 0; i < aux.length; i++){
            if (aux[i].mac !== '2') {
                deviceInfo[aux[i].mac] = aux[i];
                controlData[aux[i].mac] = aux[i].state;
            }
        }
        navFunction(deviceInfo);
        generalInfoData(deviceInfo[aux[0].mac]);
        return deviceInfo;
    })
    .catch(err => console.log);
    return appsyncquery;
}

export async function generalInfoData (data){
    getGraphic(ctxVoltage,chartErase,'VOLTAGE');
    getGraphic(ctxCurrent,chartErase,'CURRENT');
    getGraphic(ctxPower,chartErase,'POWER');

    nameDevice.innerHTML = data.name;
    typeDevice.innerHTML = data.type;
    macDevice.innerHTML = data.mac;
    hostDevice.innerHTML = data.host;
    latitudeDevice.innerHTML = data.latitude;
    longitudeDevice.innerHTML = data.longitude;
    if (data.state) {imagenSrc.src ='on.jpeg';}
    else {imagenSrc.src ='off.png';}
}
```

Figura 92. Código requerido para adquirir los datos generales de cada dispositivo registrados en AWS Appsync.

Los datos adquiridos mediante la query *"ListEnergyCloudMls"*, son almacenados en el array deviceInfo, y los valores son visualizados en los campos de *"Device Information"*.

2) Información en Tiempo real (Realtime Information)

Muestra la información en tiempo real de los parámetros de consumo eléctrico (voltaje, corriente y potencia) enviada por el servidor local. Para obtener los datos, se suscribe al método *"UpdateRealtimeInfo"* definido en AWS Appsync. La ***Figura 93*** muestra el código definido para adquirir los datos en tiempo real.

```
function realtimeSuscription (macID) {
    subscription =
    API.graphql(graphqlOperation(subscriptions.onUpdateRealtimeInfo,{mac:
    macID}))
        .subscribe({
        next: (todoData) => {//console.log('Datos actualizados:
        ',todoData.value.data.onUpdateRealtimeInfo);
                            deviceInfo =
                            Appsync.update(todoData.value.data.onUpdateRealtimeI
                            nfo, macID);},
        complete: (data) => {console.log('DATOS: ',data)},
        error: (error) =>   {console.warn('ERROR: ',error)}
        });
}

export function update (data, mac){
    console.log('Control: ',controlData[mac],'Estado: ',data.state)
    if (controlData[mac] === data.state){
        if (data.state) {imagenSrc.src ='on.jpeg';}
        else {imagenSrc.src ='off.png';}
    }

    dataVoltage = dataVoltage.concat({x:new Date(), y:data.voltage});
    dataCurrent = dataCurrent.concat({x:new Date(), y:data.current});
    dataPower = dataPower.concat({x:new Date(), y:data.power});

    getGraphic(ctxVoltage,dataVoltage,'VOLTAGE');
    getGraphic(ctxCurrent,dataCurrent,'CURRENT');
    getGraphic(ctxPower,dataPower,'POWER');

    realtimeVoltage.innerHTML = data.voltage;
    realtimeCurrent.innerHTML = data.current;
    realtimePower.innerHTML = data.power;

    deviceInfo[mac].state = data.state;

    return deviceInfo;
}
```

Figura 93. Código requerido para adquirir los parámetros de consumo eléctrico en tiempo real de cada dispositivo registrados en AWS Appsync.

Además, se establece los estados del enchufe inteligente mediante la variable "*status"* como se observa en la ***Figura 94***.

Figura 94. Imágenes que representan cada uno de los estados de los dispositivos.

Los estados en la página web están definidos también como botones, de tal manera que facilite el control de encendido y apagado de los dispositivos, el código se observa en la ***Figura 95***. Además, debido a que existe un tiempo entre cambio de estado, se ha definido un estado de transición para que el dispositivo ejecute la orden establecida.

```
if (evt.srcElement.localName === "img"){
    switch(evt.srcElement.attributes.src.value){
        case "on.jpeg":
            Appsync.controlSignal(macID, false);
            break;
        case "off.png":
            Appsync.controlSignal(macID, true);
        break;
        case "indice.png":
            alert('Wait for the current execution');
            break;
        default:
            console.log('Error');
    }
}

export async function controlSignal(mac,control){
    imagenSrc.src ='indice.png';
    controlData[mac] = control;
    if (control) {console.log('on');}
    else {console.log('off');}

    var input = {mac: mac, control:control};
    const controlMutation = API.graphql(graphqlOperation(mutations.control,
    {input:input}))
                                .then(data => console.log)
                                .catch(err => console.log);
}
```

Figura 95. Código requerido para enviar la señal de control hacia el servidor node.js mediante AWS Appsync.

3) Monitoreo (Monitoring)

Permite visualizar los parámetros de consumo eléctrico de forma gráfica. Los datos son capturados a partir del momento que selecciona un dispositivo, con la finalidad de representar y almacenar los datos en tiempo real en la memoria cache del explorador. La ***Figura 96*** muestra el código definido para graficar los parámetros.

```
export async function getGraphic(ctx,datos,title){
    let chart = new Chart(ctx, {
        type: 'line',
        data: {
            datasets: [{
                data: datos,
                //fill: false,
            }]
        },
        options: {
            events: ['click'],
            legend: {
                display: false,
            },
            //responsive: true,
            title: {
                        display: true,
                        text: title
                    },
            scales: {
                xAxes: [{
                    type: 'time',
                    time: {
                        //format: timeFormat,
                        tooltipFormat: 'll HH:mm'
                    }
                }]
            }
        }
    });
}
```

Figura 96. Código requerido para graficar los parámetros de consumo eléctrico en tiempo real obtenidos mediante AWS Appsync.

4) Hábitos de consumo de energía (Energy consumption habits)

Imágenes pre-procesadas correspondientes a cada dispositivo son cargadas en AWS S3, y visualizadas gráficamente en el explorador. Como se observa en la ***Figura 97***, los datos históricos

han permitido aprender los hábitos de consumo de la computadora de escritorio perteneciente al escenario de prueba definido en la sección 4.2.

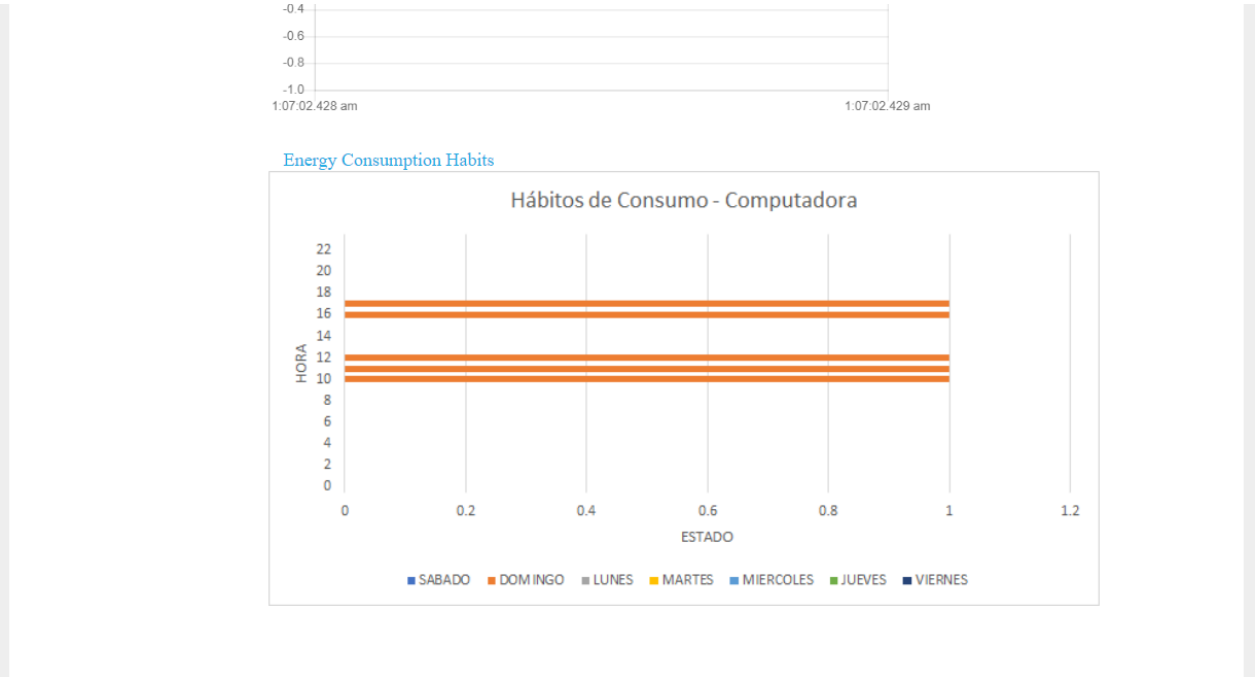


Figura 97. Visualización de la página web main.html (hábitos de consumo de energía del computador)

La segunda parte de la aplicación web corresponde a información relacionada con la cuenta de usuario. El código JavaScript únicamente adquiere información del usuario autenticado, y cuenta el número de dispositivos registrados, como se observa en la ***Figura 98***.

```
Auth.currentAuthenticatedUser()
.then((user) => {
                username_info.innerHTML = user.username;
                usernameAccount.innerHTML = user.username;
                emailAccount.innerHTML = user.attributes.email;
                devicesAccount.innerHTML = Object.keys(deviceInfo).length;
              })
.catch(err => console.log(err));
```

Figura 98. Código requerido para obtener la información de la cuenta actual.

4.2. Implementación

4.2.1. Descripción de Escenario de Prueba

El presente proyecto de investigación es desarrollado en un hogar tipo C+ conformado por 4 miembros durante 5 meses. El detalle de los miembros del hogar se detalla en la **Tabla 21**.

Tabla 21
Miembros que conforman el escenario de prueba del hogar tipo C+.

Miembro	Nombre	Edad	Educación
Padre	Cristóbal	52	Superior
Madre	Nelly	45	Tercer nivel
Hija mayor	Erika	12	Secundaria
Hija menor	Doménica	9	Primaria

La vivienda cuenta con varios electrodomésticos y equipos electrónicos, sin embargo, el presente proyecto se analiza 3 dispositivos electrónicos detallados en la **Tabla 22**.

Tabla 22
Dispositivos electrónicos que conforman el escenario de prueba en hogar tipo C+.

Nombre	Marca	Modelo	Año de fabricación	Tiempo de uso	Rango de funcionamiento
Computadora	Samsung		2003	10 años	
Microonda	Panasonic	NN-SA768W	2012	7 años	12,7ª – 120V (AC)
Televisor	Samsung	UN48J6500AK	2015	4 años	100 – 240V (AC)

Cada uno de los dispositivos electrónicos son conectados al enchufe inteligente TP-Link HS110, ver **Tabla 23**, y en conjunto al tomacorriente más cercano. El aplicativo móvil KASA permite registrar los enchufes a la plataforma de TP-Link y además establece una dirección IP en la red local. Los enchufes conectados la red WAN son descubiertos y registrados mediante la API, y con ello se establece comunicación con Energy Cloud-ML.

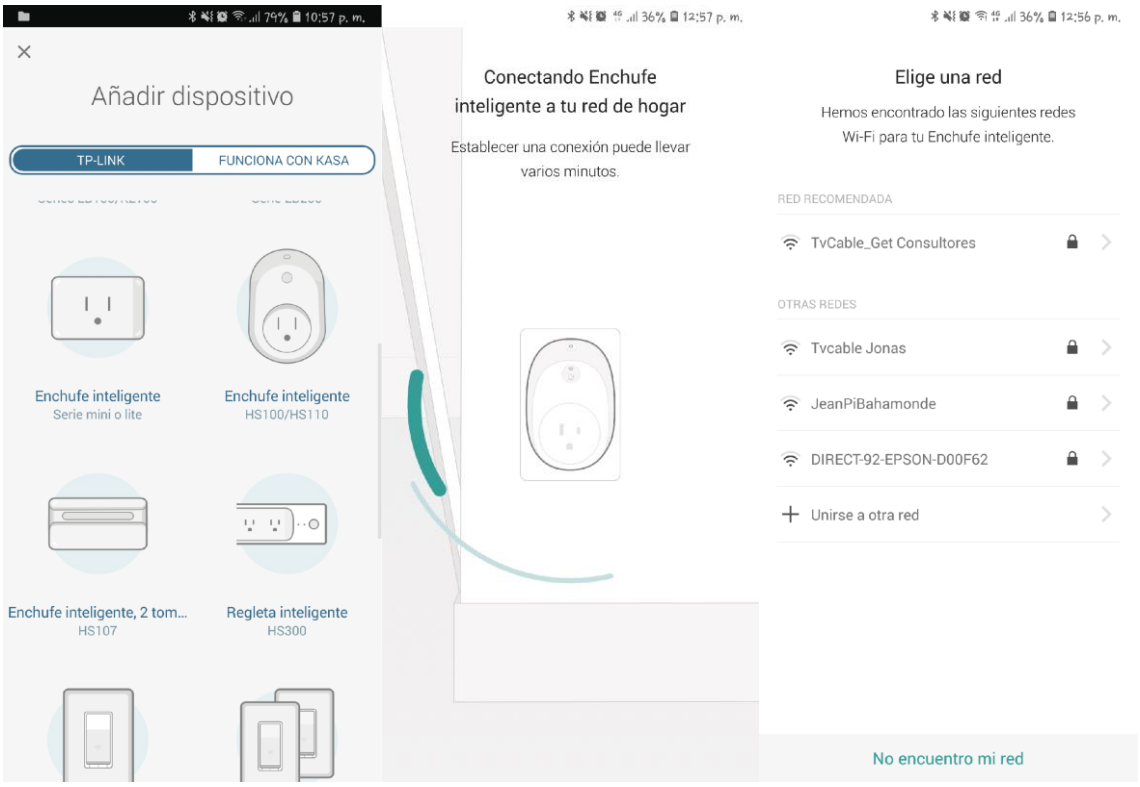
Tabla 23
Correspondencia entre dispositivos electrónicos y el enchufe inteligente.

Dispositivo electrónico	Enchufe Inteligente	MAC de Enchufe	Alias de Enchufe
Computadora	TP-Link HS-110	50:C7:BF:2ª:FC:A8	Computadora
Microonda	TP-Link HS-110	50:C7:BF:2ª:F5:79	Microonda
Televisor	TP-Link HS-110	50:C7:BF:C7:D8:12	Televisión

4.2.2. Registro de Enchufes Inteligentes

Los pasos para registrar los enchufes al aplicativo KASA se detallan a continuación:

1. Descargar el aplicativo KASA en smartphone y crear cuenta de administrador.
2. Conectar el enchufe inteligente HS-110 a la fuente de energía (110V DC – 60Hz).
3. Añadir cada uno de los enchufes al aplicativo KASA de acuerdo a la **Tabla 23**, seguir pasos de la app como se observa en la ***Figura 99***.



Continua

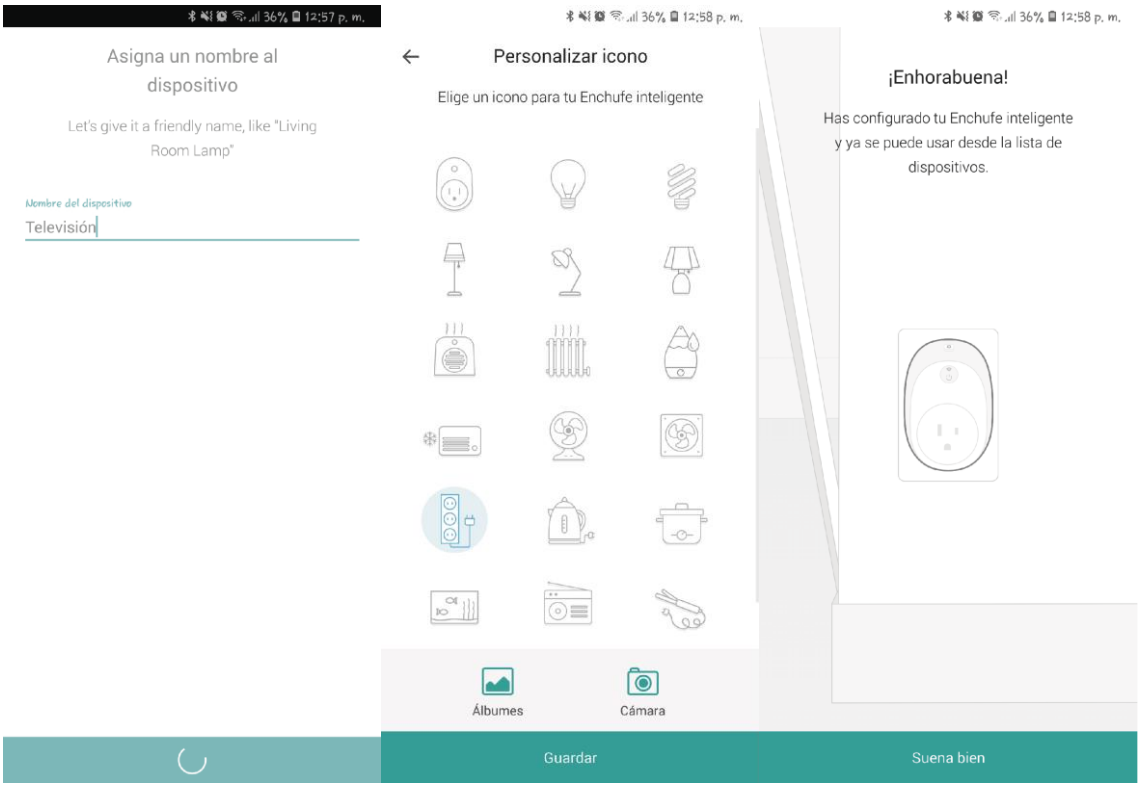


Figura 99. Pasos para registro de enchufe inteligente HS-110.

4. Al finalizar se registra cada enchufe inteligente en el aplicativo KASA. Ver ***Figura 100***.

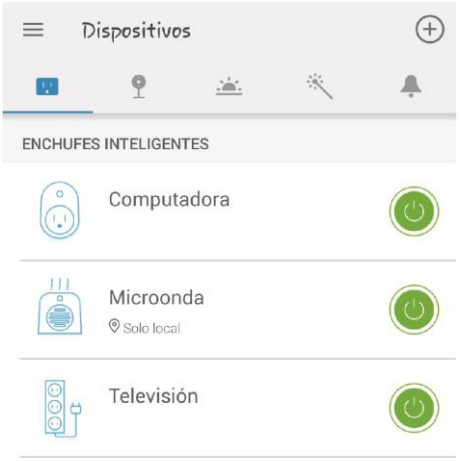


Figura 100. Lista de dispositivos registrados en la APP KASA.

4.2.3. Inicialización del Servidor Node.js

El servidor es almacenado en la carpeta raíz del usuario pi de la Raspberry PI 3B. Al ingresar al entorno de consola, se ejecuta la siguiente línea de comandos:

cd appsync-nodejs-smarthome

npm run build && npm run test

Al ejecutar los comandos, automáticamente empieza a descubrir y registrar los enchufes inteligentes presentes en la red y futuros enchufes también. La ***Figura 101*** y ***Figura 102*** muestra la descripción de los logs generados desde la inicialización del servidor node.js, y durante su funcionamiento.

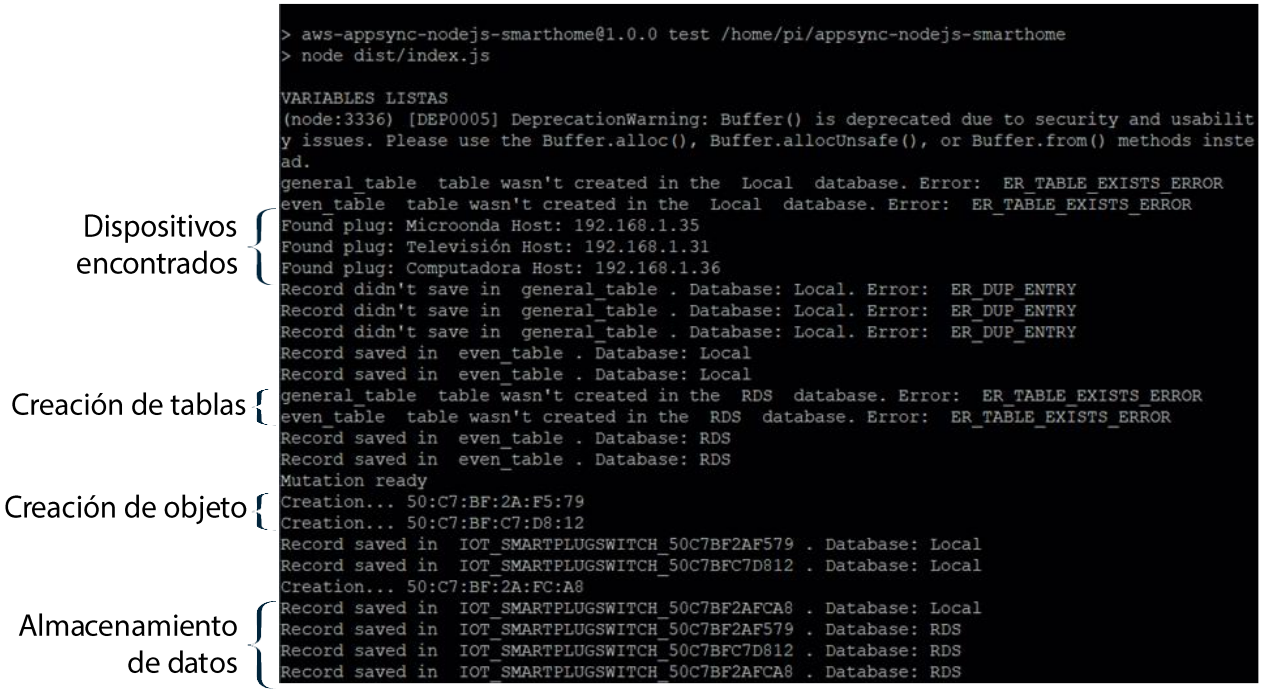


Figura 101. Descripción de logs generados por consola al inicializar el servidor node.js.

Figura 102. Descripción de logs generados por consola del servidor node.js.

CAPÍTULO 5

5. RESULTADOS Y ANÁLISIS

5.1. Resultados

5.1.1. Pruebas de Implementación

La información del dispositivo o Device Information es representada para cada uno de los enchufes inteligentes HS-110, como se puede observar en la ***Figura 103***, ***Figura 104***, y ***Figura 105***. Cada uno de los enchufes muestra su respectiva información, aunque cabe destacar que el valor de latitud y longitud no varía debido a que se encuentran a una distancia muy cercana, y la precisión de los enchufes es baja. Los datos tabulados se observan en la **Tabla 24**.

Figura 103. Información de dispositivo correspondiente a la Televisión.

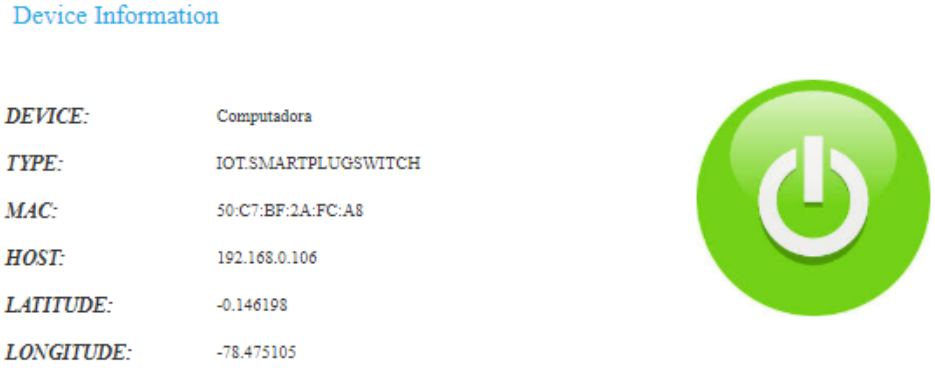


Figura 104. Información de dispositivo correspondiente a la Computadora.

Figura 105. Información de dispositivo correspondiente al Microonda.

Tabla 24
Información de dispositivo tabulada para cada enchufe registrado

Dispositivo	Tipo	MAC	Host	Latitud	Longitud
Televisor	IOT.SMARTPLUGSWITCH	50:C7:BF:C7:D8:12	192.168.0.101	-0.146198	-78.47510
Computadora	IOT.SMARTPLUGSWITCH	50:C7:BF:2A:FC:A8	192.168.0.106	-0.146198	-78.47510
Microonda	IOT.SMARTPLUGSWITCH	50:C7:BF:2A:F5:79	192.168.0.102	-0.146198	-78.47510

5.1.2. Información en tiempo real

Los parámetros de consumo eléctrico (voltaje, corriente, y potencia) son representados numéricamente en la sección de información en tiempo real o realtime information proporcionada por los enchufes inteligentes cada segundo, como se observa en la ***Figura 106***.

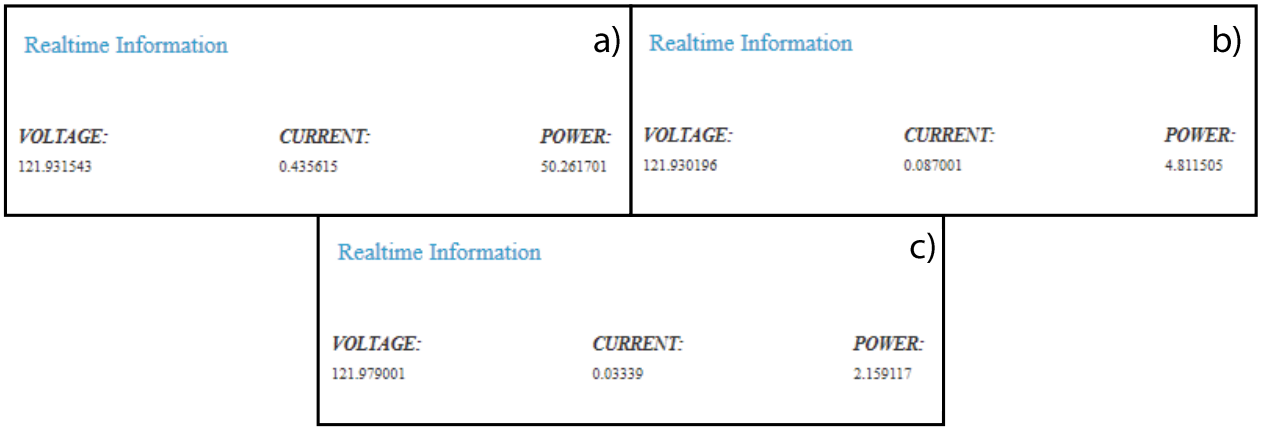


Figura 106. Información en tiempo real. a) Televisión; b) Computadora; y c) Microonda.

5.1.3. Monitoreo

El monitoreo o monitoring permite visualizar el flujo de los parámetros de consumo eléctrico en tiempo real. El aplicativo web representa cada muestra obtenida de tiempo real mediante una representación lineal en función del tiempo. Como se observa en la ***Figura 107***, ***Figura 108***, y ***Figura 109***, el voltaje suministrado a cada uno de los enchufes inteligentes tiene una media estimada de 116,5V, 127V, y 127V respectivamente, y presenta variaciones centesimales a lo largo del tiempo.

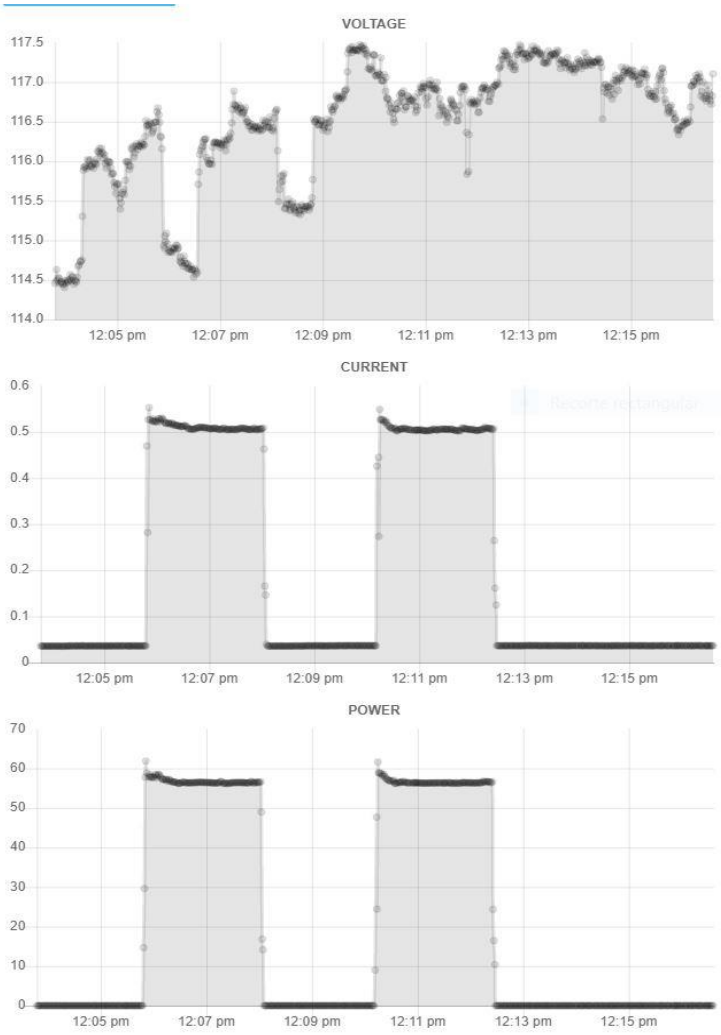


Figura 107. Monitoreo de parámetros de consumo eléctrico de la Televisión.

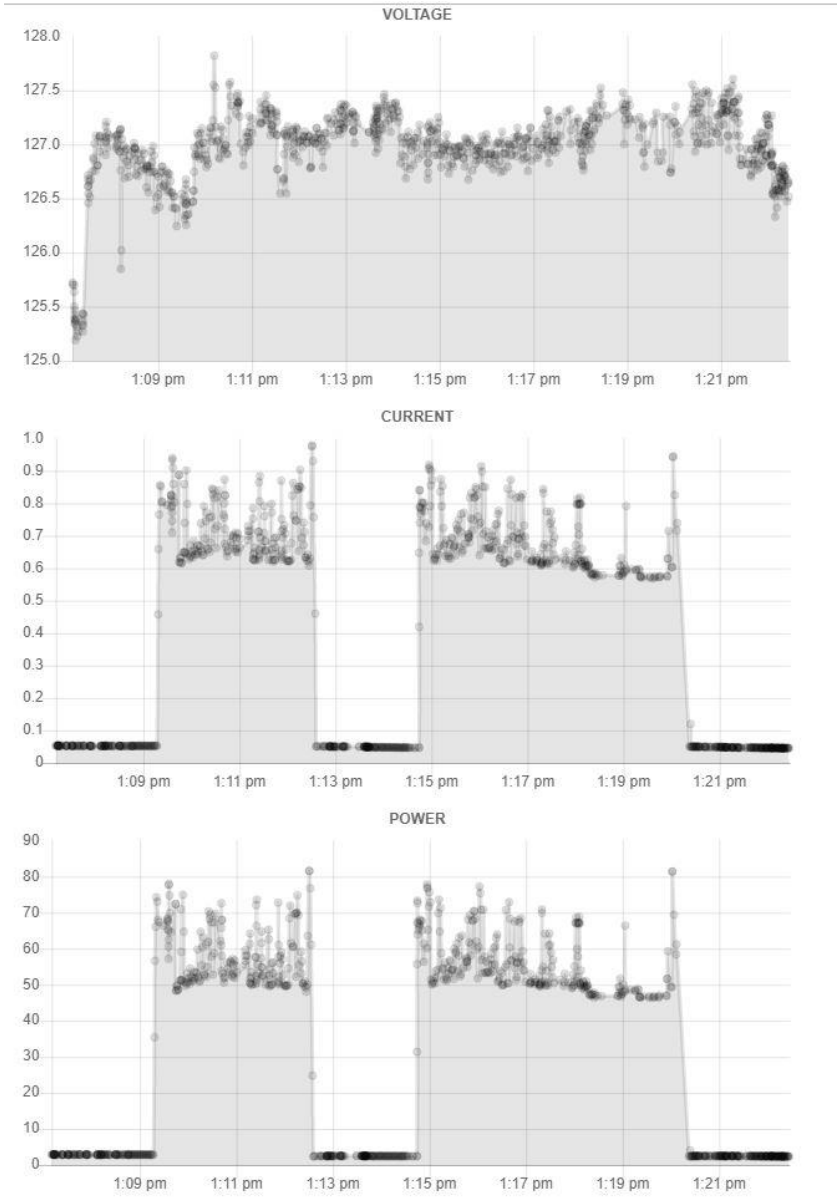


Figura 108. Monitoreo de parámetros de consumo eléctrico de la Computadora.

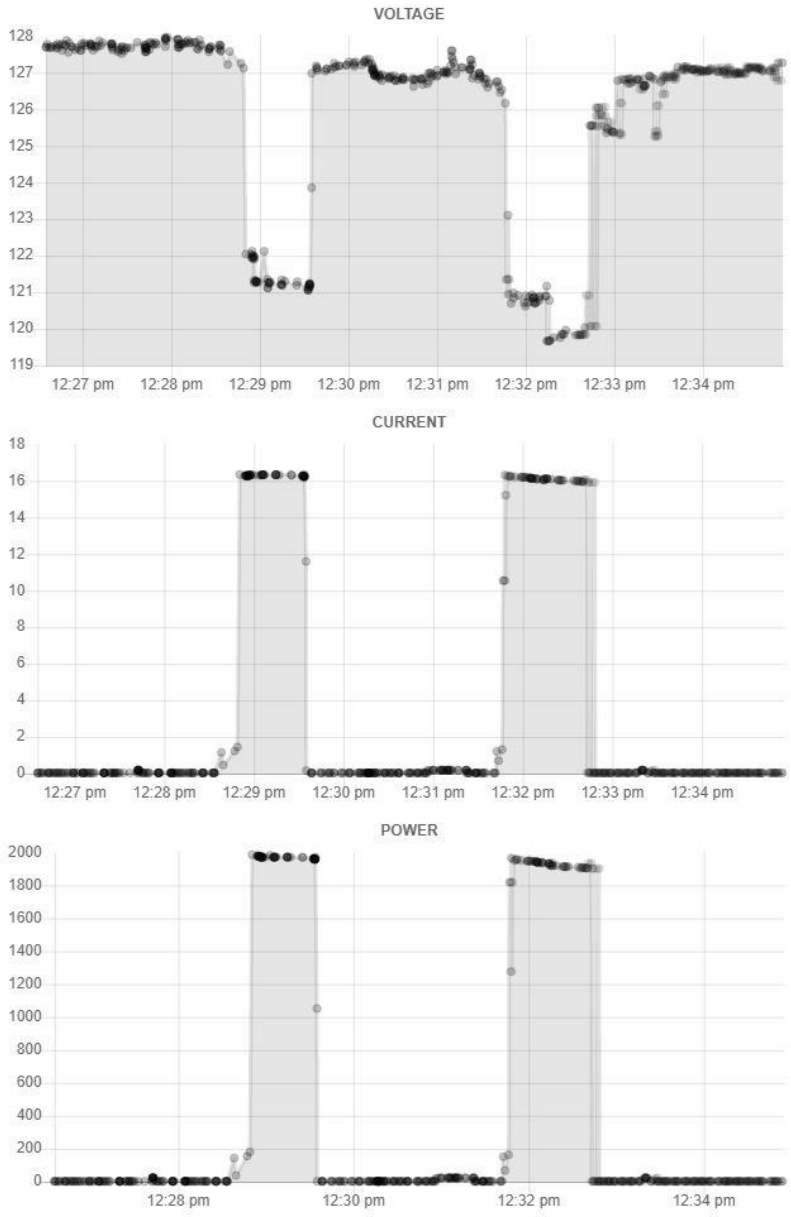


Figura 109. Monitoreo de parámetros de consumo eléctrico del Microonda.

Los resultados fueron obtenidos de tal manera que permita evidenciar el contraste entre los diferentes estados de cada dispositivo electrónico conectado. Los datos fueron capturados durante un periodo aproximado de 12 min (13min, 15min, y 8min respectivamente), donde se observa 2 veces cuando se encuentra en uso, y 3 veces sin uso el dispositivo electrónico.

El comportamiento de cada dispositivo varia a lo largo del tiempo y el estado en que se encuentre, en particular cuando los dispositivos electrónicos están apagados no deberían consumir energía, sin embargo, en la ***Figura 108*** y ***Figura 109*** se observa como algunos dispositivos consumen cierta cantidad de energía a pesar de estar sin uso (apagados), el promedio de los datos recolectados se ha representado en la **Tabla 25**.

Tabla 25
Promedio de potencia consumida del dispositivo electrónico durante el estado de encendido y apagado.

Dispositivo	Fecha de fabricación	Potencia (en uso)	Potencia (sin uso)
Televisor	2015	58 W	0 W
Computadora	~2010	66 W	3.1W
Microonda	2012	1980 W	2.5 W

Este es un claro ejemplo de que a pesar de que los dispositivos electrónicos no se encuentran en uso, existe un consumo mínimo de energía, lo que permite justificar el presente sistema de gestión de energía. Además, se puede observar que a medida que la tecnología evoluciona, la eficiencia energética es un punto clave para fabricantes de dispositivos electrónico.

5.1.4. Hábitos de consumo eléctrico

Los datos almacenados en AWS RDS son el histórico de consumo eléctrico de los enchufes inteligentes cada 5 segundos proporcionado por el servidor Node.js durante 3 meses, la cantidad de datos es capaz de ser procesada mediante AWS Machine Learning.

Los hábitos de consumo eléctrico o Energy Consumption Habits son visualizados a partir de imágenes precargadas por el administrador al aplicativo web. Las imágenes permiten observar el

rango horario y diario de consumo de cada dispositivo conectado, ver ***Figura 111***, ***Figura 113***, y ***Figura 114***.

Como se puede observar en la ***Figura 111***, la televisión presenta un comportamiento predecible debido a que los usuarios mantienen un uso constante y rutinario, en general, el uso del televisor se encuentra en las siguientes franjas horarias:

- 6am – 8pm
- 2pm-4pm
- 7pm-0pm

Lo contrario ocurre con el microondas y se puede decir también el computador, debido a que el uso es poco frecuente y no es posible definir horarios definidos donde los usuarios hacen uso de los dispositivos electrónicos.

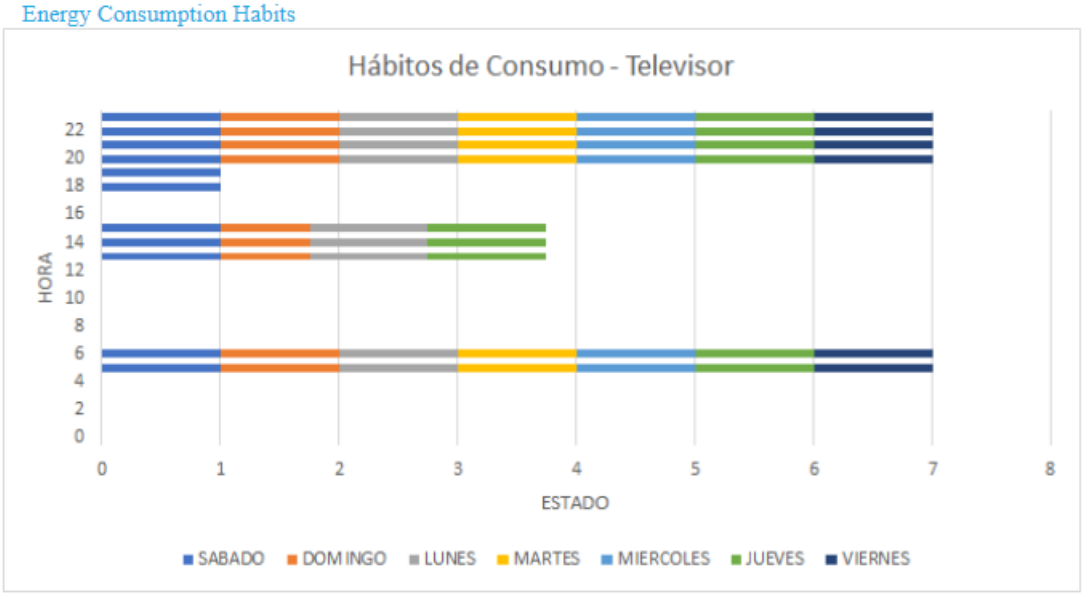


Figura 110. Hábitos de consumo correspondiente al Televisor.

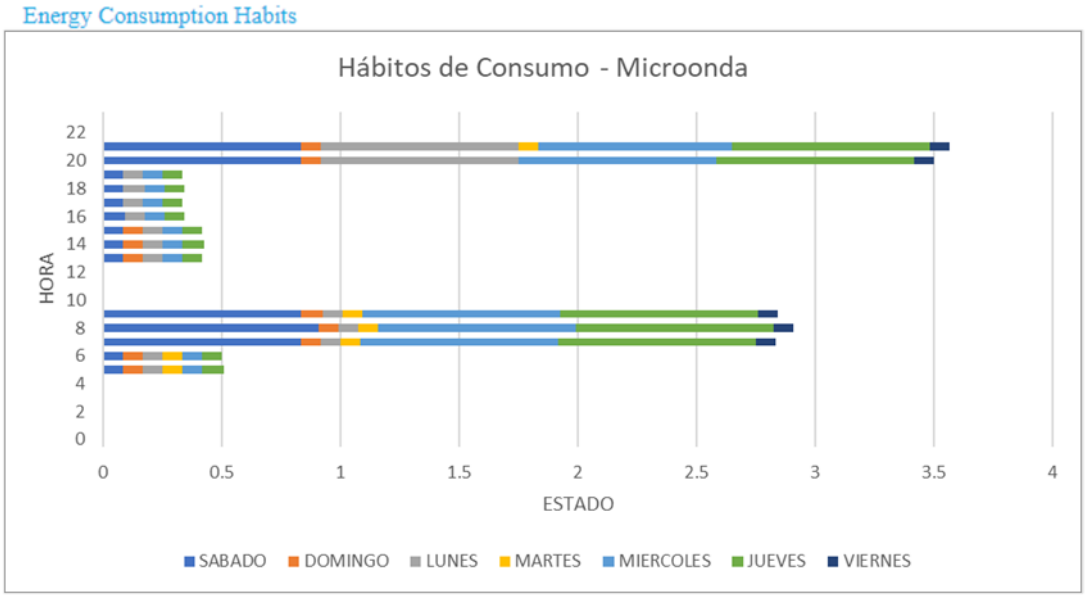


Figura 111. Hábitos de consumo correspondiente al Microonda.

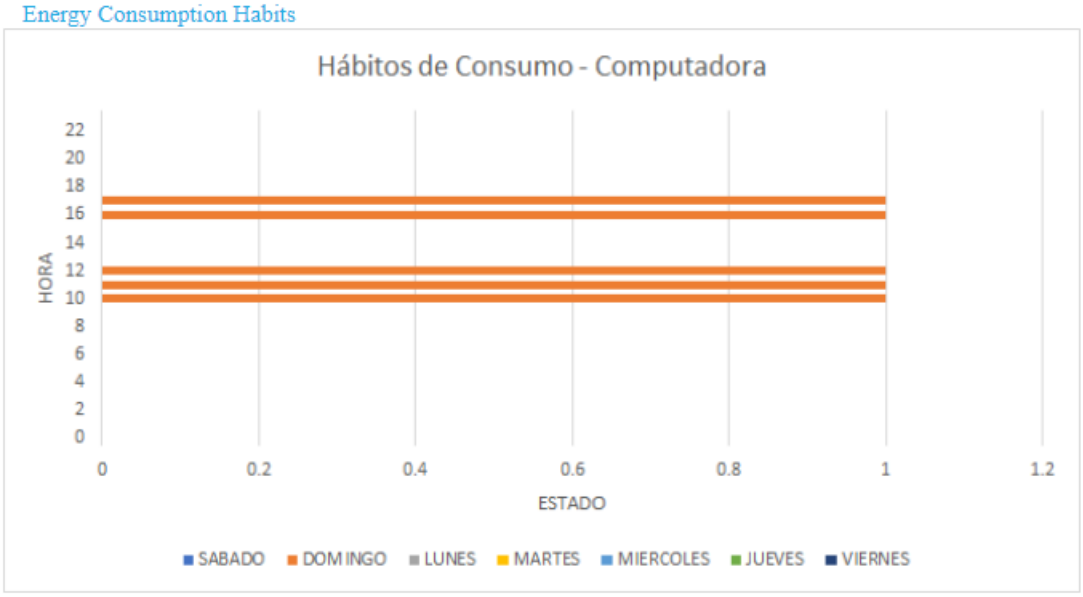


Figura 112. Hábitos de consumo correspondiente al Computador.

5.2. Análisis

En el presente trabajo de investigación hay varios factores o variables que se pueden analizar. Entre ellas está la energía consumida por cada enchufe inteligente cuando se encuentra o no en funcionamiento, también el porcentaje de memoria y cpu consumido en la instancia de base de datos, o analizar si la predicción realizada corrobora los datos proporcionados en el escenario de prueba. Por ello se ha dividido la sección en varios apartados.

5.2.1. Parametrización del enchufe inteligente

Como se observa en la **Tabla 23**, el presente trabajo de investigación monitorea los parámetros de consumo eléctrico de tres dispositivos electrónicos a través de tres enchufes inteligentes. Cada enchufe inteligente está conectado a una toma de energía para alimentar los circuitos internos, y los dispositivos electrónicos conectados. Sin embargo, la energía consumida por el enchufe es desconocida, para ello se ha medido la corriente mediante el uso de un multímetro de acuerdo al circuito de la ***Figura 113***. La medición se ha realizado cuando el enchufe sin carga se encuentra apagado (relé abierto) y encendido (relé cerrado).

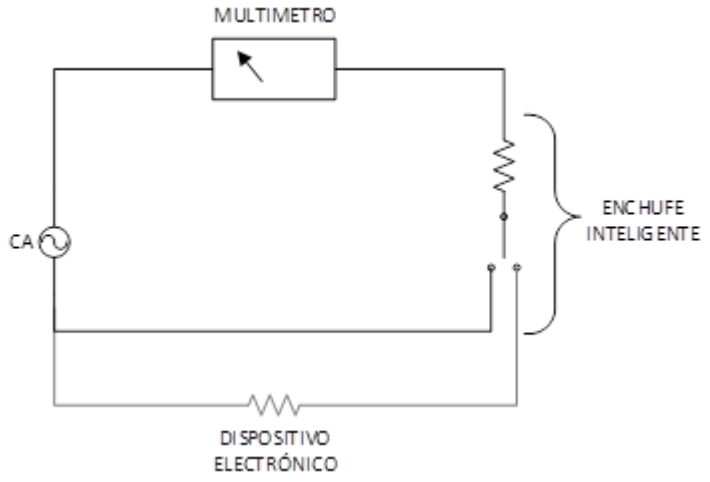


Figura 113. Diagrama del circuito para medir la corriente consumida por cada enchufe inteligente sin carga.

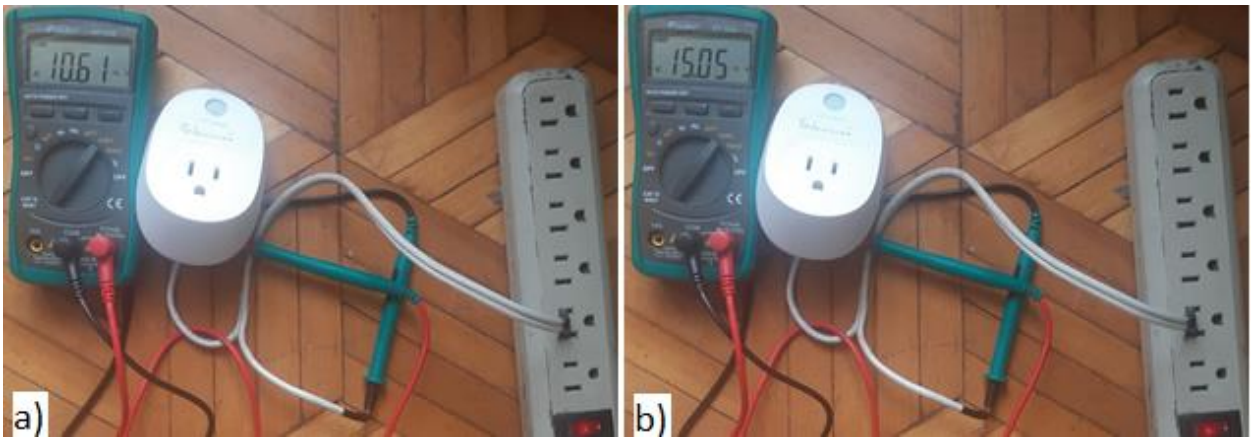


Figura 114. Medición de corriente de enchufe inteligente (alias: Televisor) en estado: a) apagado; b) encendido.

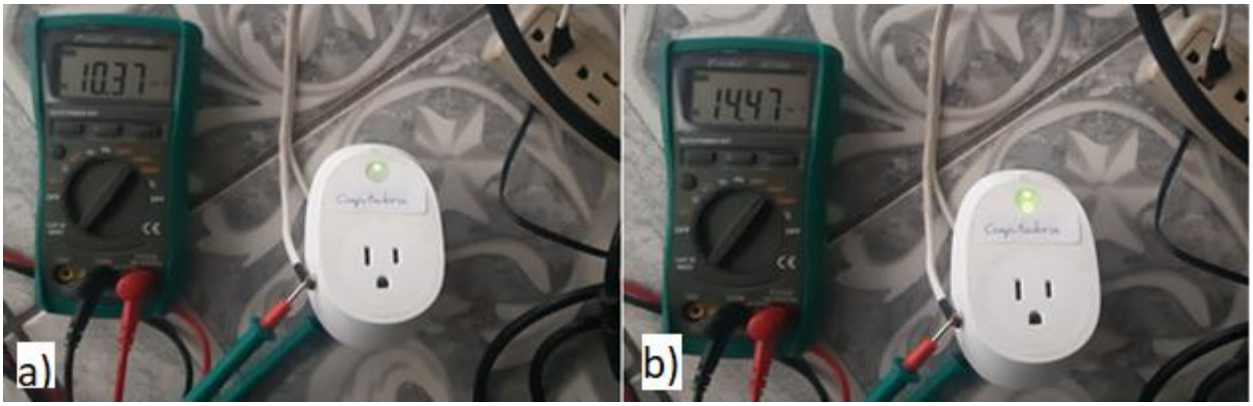


Figura 115. Medición de corriente de enchufe inteligente (alias: Computadora) en estado: a) apagado; b) encendido.

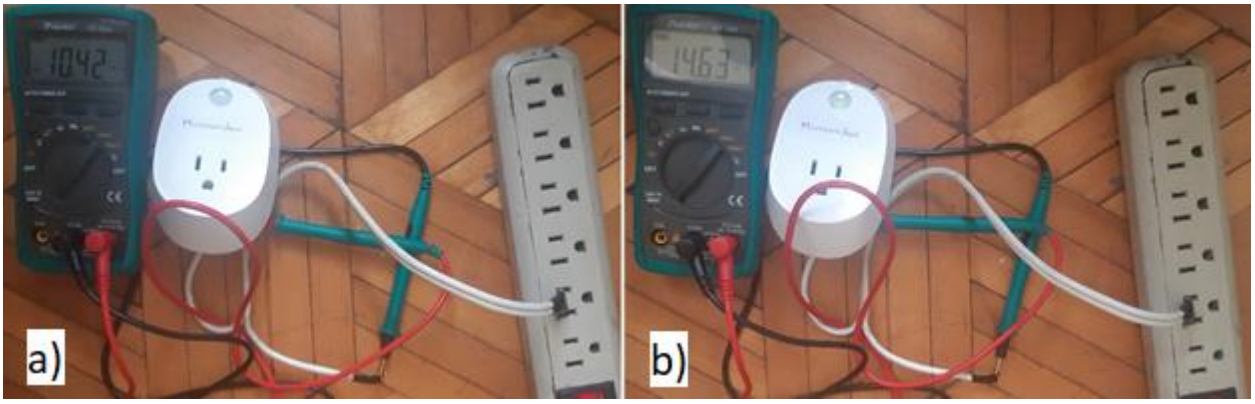


Figura 116. Medición de corriente de enchufe inteligente (alias: Microonda) en estado: a) apagado; b) encendido.

La **Tabla 26** muestra los resultados obtenidos a partir de la ***Figura 113***, además se ha considerado un voltaje referencia previo a la medida de la corriente como medida para la potencia. Los datos son tabulados en la **Tabla 26**, obtenidas a partir de la medición mediante multímetro como se observa en la ***Figura 114***, ***Figura 115***, y ***Figura 116***.

Tabla 26
Parámetros de consumo eléctrico de los enchufes inteligentes.

Alias	MAC	Voltaje	Corriente (OFF)	Corriente (ON)	Potencia (OFF)	Potencia (ON)
Televisor	50:C7:BF:C7:D8:12	127.9V	10.61mA	15.05mA	1.36W	1.92W
Computadora	50:C7:BF:2A:FC:A8	127.9V	10,37mA	14,47mA	1.33W	1.85W
Microonda	50:C7:BF:2A:F5:79	127.9V	10.42mA	14.63mA	1.33W	1.87W

Los datos proporcionados por la **Tabla 26** muestra que no existe una diferencia considerable entre los estados de encendido y apagado de los enchufes inteligentes, además, sus valores son similares en cada enchufe. La potencia promedio en estado apagado es 1.34W, mientras que la potencia promedio en estado encendido es de 1.88W, por lo tanto, se evidencia que el consumo es mínimo y similar al consumo de los dispositivos electrónicos conectados medidos y tabulados en la **Tabla 25** en estado apagado.

5.2.2. Confiabilidad de datos en tiempo real

Otro factor fundamental es la confiabilidad de la información que transmite los enchufes inteligentes, para ello se ha medido la corriente como se observa en la ***Figura 117***.

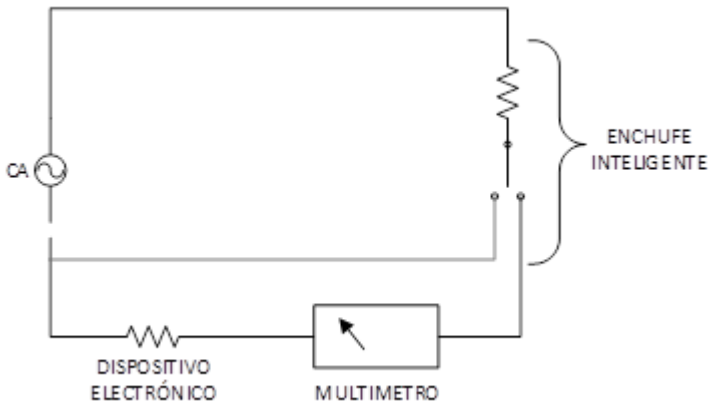


Figura 117. Diagrama del circuito para medir la corriente consumida por cada dispositivo electrónico

La **Tabla 27** muestra los valores tabulados a partir de la ***Figura 119***, ***Figura 118***, y ***Figura 120***; cuyos datos fueron medidos cuando los dispositivos electrónicos se encuentran en uso y sin uso. Esto permite verificar que los valores mostrados en el sistema de gestión de energía son similares o aproximados a los valores obtenidos por medio del multímetro, es decir, permite conocer si los datos proporcionados por el sistema son confiables.

Tabla 27

Parámetros de consumo eléctrico de los dispositivos electrónicos.

Alias	Voltaje	MULTIMETRO		APP	
		Corriente (OFF)	Corriente (ON)	Corriente (OFF)	Corriente (ON)
Televisor	127.9V	0.042A	0.454A	0.036194A	0.494047A
Computadora	127.9V	0.033A	0.439A	0.055149A	0.790138A
Microonda	127.9V	0.045A	15.80A	0.0374A	16.440394A

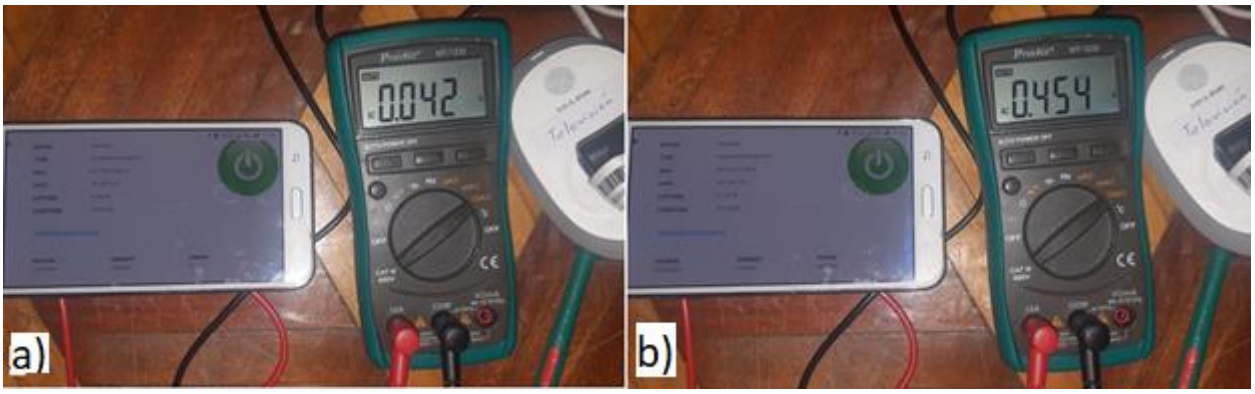


Figura 118. Medición de corriente de dispositivo electrónico (alias: Televisor) en estado: a) apagado; b) encendido.

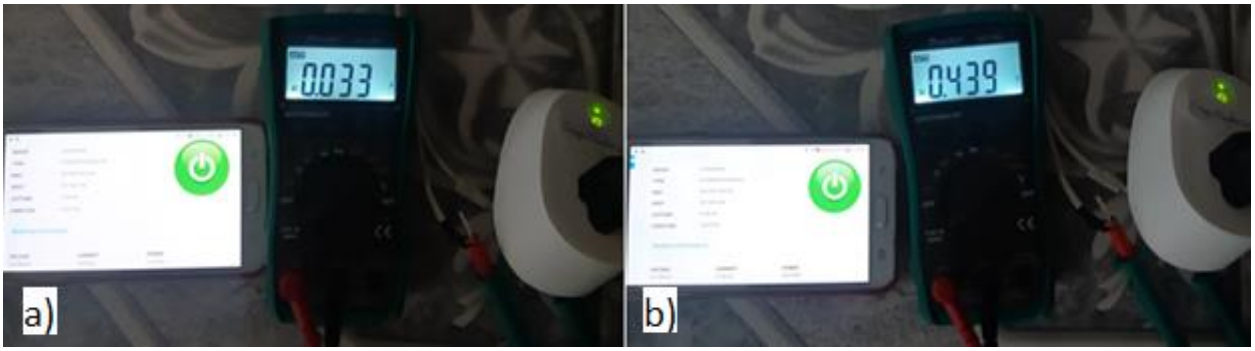


Figura 119. Medición de corriente de dispositivo electrónico (alias: Computadora) en estado: a) apagado; b) encendido.

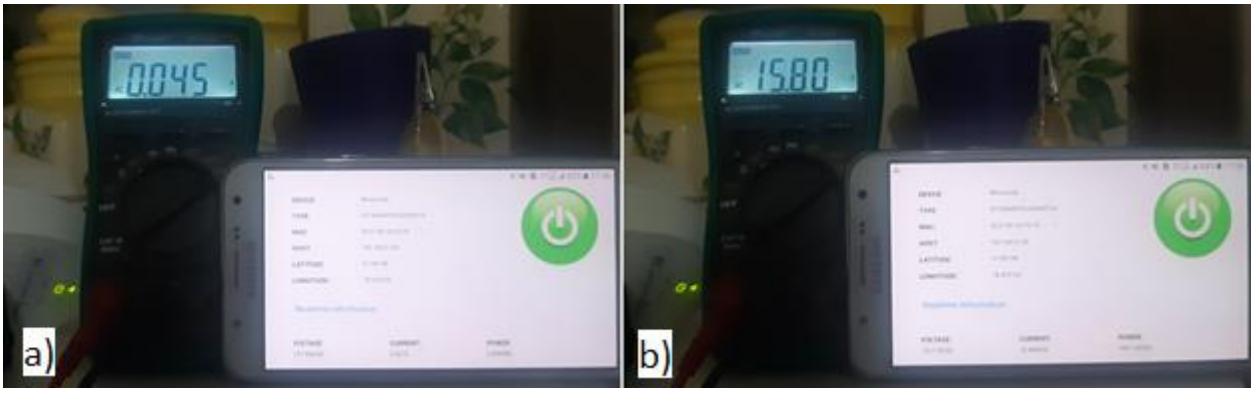


Figura 120. Medición de corriente de dispositivo electrónico (alias: Microonda) en estado: a) apagado; b) encendido.

Los datos de la **Tabla 27** son calculados con las siguientes fórmulas:

- Error Absoluto

$$e = \left|Valor_{real} - Valor_{aprox}\right|$$

- Error Relativo Porcentual

$$\varepsilon(\%) = \frac{\left|Valor_{real} - Valor_{aprox}\right|}{Valor_{real}} * 100\%$$

La información obtenida permite observar el error relativo de cada uno de los enchufes inteligentes, tomando como referencia a la medición obtenida por el multímetro. La información se puede observar en la **Tabla 28**.

Tabla 28
Parámetros de consumo eléctrico de los dispositivos electrónicos.

Alias	Voltaje	Corriente (OFF)		Corriente (ON)	
		e	$\varepsilon(\%)$	e	$\varepsilon(\%)$
Televisor	127.9V	5,81mA	13,82%	40,05mA	8,82%
Computadora	127.9V	22,15mA	67,12%	351,14mA	79,99%
Microonda	127.9V	7,60mA	16.89%	640,39mA	4,05%

5.2.3. Tiempo de retardo

5.2.3.1. Información en tiempo real

Los parámetros de consumo eléctrico son actualizados cada segundo, sin embargo, debido a retardos, perdidas de paquetes, o tráfico en la red local y la nube, la información proporcionada por el sistema de gestión de energía puede ser afectada. Por ello se ha tomado 25 muestras de la diferencia de tiempo entre cada actualización para cada enchufe inteligente.

Los resultados se observan en la **Tabla 29**.

Tabla 29

Muestreo de tiempo de actualización para cada enchufe inteligente.

Muestra	Computadora	Microonda	Televisor
1	0.95’	0.93’	1.06’
2	1.01’	1.09’	0.61’
3	0.94’	0.94’	0.66’
4	0.82’	0.95’	1.08’
5	1.01’	1.14’	0.89’
6	1.97’	1.05’	1.00’
7	0.28’	0.94’	1.26’
8	0.99’	0.97’	0.67’
9	1.27’	1.45’	1.26’
10	1.13’	1.15’	0.92’
11	1.46’	0.68’	0.88’
12	1.01’	1.58’	1.39’
13	1.14’	0.63’	0.51’
14	0.99’	0.70’	1.09’
15	0.81’	0.95’	0.94’
16	0.74’	0.87’	0.94’
17	0.048’	0.89’	1.34’
18	0.082’	0.92’	1.27’
19	0.85’	1.18’	0.71’
20	0.96’	0.88’	0.86’
21	0.94’	0.89’	1.02’
22	1.15’	1.13’	0.96’
23	1.09’	1.22’	1.22’
24	1.26’	1.01’	0.71’
25	0.70’	1.27’	1.40’

La **Tabla 29** permite observar como los retardos no son mayores a 2 segundos. Además, mediante la tabulación de los datos, la **Tabla 30** muestra como el promedio bordea en 1 segundo,

sin pérdida de paquetes. Sin embargo, la desviación estándar varía considerablemente, la computadora presenta un mayor valor con alrededor 0.4 segundos, mientras que el televisor con 0.22 segundos tiene un menor valor. A nivel de usuario los tiempos son aceptables para obtener valores en tiempo real.

Tabla 30
Promedio y desviación estándar en función de la Tabla 27

	Computadora	**Microonda**	**Televisor**
Promedio	0.944’	1.017’	0.986’
Desviación estándar	0.398’	0.220’	0.254’

Cabe destacar que la distancia entre el router y el computador es mayor a la distancia con la microonda y el televisor. Además, el computador requiere el uso de un repetidor para acceder a la red Internet.

5.2.3.2. Control de enchufe inteligente

Otra medida importante es el tiempo que tarda el sistema de gestión de energía en ejecutar la orden de encendido o apagado de los enchufes inteligentes. Por ello se ha tomado 11 muestras para verificar el tiempo promedio de ejecución de las órdenes para cada enchufe, como se puede observar en la **Tabla 31**. Los datos también fueron aplicados y obtenidos a partir del aplicativo KASA vía remota.

Tabla 31
Muestreo del tiempo que tarda en ejecutar el sistema de gestión de energía vs aplicativo Kasa

Muestra	**Sistema de Gestión de Energía**			**Aplicativo KASA**		
	Computadora	Microonda	Televisor	Computadora	Microonda	Televisor
1	17.11’	28.34’	18.49’	0.88’	0.85’	1.09’
2	24.24’	25.04’	27.98’	0.64’	1.59’	0.61’
3	28.40’	28.01’	11.02’	0.78’	0.85’	0.69’

4	27.38’	26.99’	18.54’	0.61’	0.87’	0.62’
5	20.08’	29.85’	9.70’	0.82’	0.82’	0.76’
6	23.91’	25.96’	31.84’	1.24’	1.71’	0.78’
7	29.65’	28.08’	13.16’	1.05’	0.92’	0.66’
8	20.88’	27.90’	23.00’	0.64’	0.48’	0.62’
9	26.00’	27.66’	28.73’	0.67’	0.74’	0.63’
10	23.44’	26.38’	23.75’	1.19’	0.75’	0.71’
11	26.86’	21.41’	19.83’	1.00’	0.70’	0.71’

La **Tabla 32** representa el promedio y desviación estándar de los datos proporcionados en la **Tabla 31**. El promedio de respuesta es directamente proporcional con el tiempo de consulta definido en AWS Appsync, 30 segundos. Esto es posible ser reducido, sin embargo, al disminuir el tiempo de consulta, aumenta el tráfico en la red.

Tabla 32

Promedio y desviación estándar en función de la Tabla 29

	Sistema de Gestión de Energía			**Aplicativo KASA**		
	Computadora	Microonda	Televisor	Computadora	Microonda	Televisor
Promedio	24.359’	26.875’	20.549’	0.865’	0.935’	0.716’
Desviación estándar	3.826’	2.230’	7.333’	0.226’	0.374’	0.137’

Cabe mencionar que AWS Appsync aún está en desarrollo, sin embargo, en un futuro será posible aplicar eventos para enviar ordenes desde el aplicativo hacia el servidor y así reducir el tiempo promedio de respuesta actual de alrededor de 24 segundos, a tiempos similares a los obtenidos con el Aplicativo KASA de alrededor de 1 segundo que es sumamente menor.

Además, la desviación estándar varia constantemente en el aplicativo web desarrollado, mientras que el aplicativo KASA presenta mejores prestaciones

5.2.4. Base de datos RDS

5.2.4.1. Carga de enchufes al sistema

La base de datos nos permite almacenar los datos históricos, sin embargo, el porcentaje de uso podría afectar el desenvolvimiento de este debido al número de dispositivos conectados al sistema de gestión de energía. Por ello se ha cargado al sistema un enchufe a la vez para mostrar la variación de uso del CPU en función del número de enchufes enlazados, el cronograma fue el siguiente:

- Desconexión hacia la DB y enchufes inteligentes (20 min)
- Conexión hacia la DB (5min)
- Conexión de primer enchufe (televisor) (5 min)
- Conexión de segundo enchufe (computadora) (5min)
- Conexión de tercer enchufe (microonda) (25min)

Mediante AWS CloudWatch, podemos monitorizar ciertas funcionalidades de la instancia de base de datos. La ***Figura 121*** muestra las estadísticas más importantes obtenidas a partir del cronograma establecido, allí se obtuvo lo siguiente:

Conexiones DB

- Por defecto el servidor utiliza 10 conexiones para acceder a AWS RDS.
- A pesar de que el número de enchufes inteligentes conectados aumenta, el número de conexiones no, debido a que servidor centraliza las conexiones de los enchufes.

CPU

- El porcentaje de utilización permanece constante durante todo el proceso, únicamente se puede ver una variación de 0.3% entre el valor inicial y el valor final.

- La capa gratuita nos ofrece 1G de memoria RAM, la cual es suficiente para los usos actuales del servidor y permite escalar el número de dispositivos hasta alrededor de 900.
- El tiempo de uso de CPU es un factor sumamente importante, durante un periodo de 960h es decir 40 días la capa gratuita como su nombre lo dice su uso es gratuito, sin embargo, luego de este periodo se debe pagar $0.017 por cada hora de uso de la DB.

Escritura

- El throughput de escritura es de alrededor de 0.035 Mbps por cada enchufe conectado. Cuando los 3 enchufes se encuentran en funcionamiento el valor es de 0.01 Mbps, mientras que el valor de throughput en la red es de 0.0015Mbps.
- El valor de throughput es importante, debido a que limita el tiempo de almacenamiento en disco necesario antes de alcanzar el límite de la capa gratuita que es de 20 GB.
- En función del throughput de actual, el almacenamiento gratuito durará alrededor de 23 días, es decir menos de 1 mes, luego de este periodo se tiene pagar un valor de $0.115 por GB al mes utilizado.
- Los valores de latencia son sumamente menores, alrededor de 0.6ms.

Lectura

- El thoughput de lectura presenta pico cada 5min de alrededor de 0.001 Mbps a pesar de que no se solicitan peticiones de consulta, esto se puede deber a periodos durante el cual AWS RDS realiza una inspección de la DB.
- El valor de throughput de la red no es relevante para el sistema de gestión de energía, sin embargo, el valor obtenido es constante con alrededor de 0.0025 Mbps.
- Los valores de latencia son menores, alrededor de 1ms.

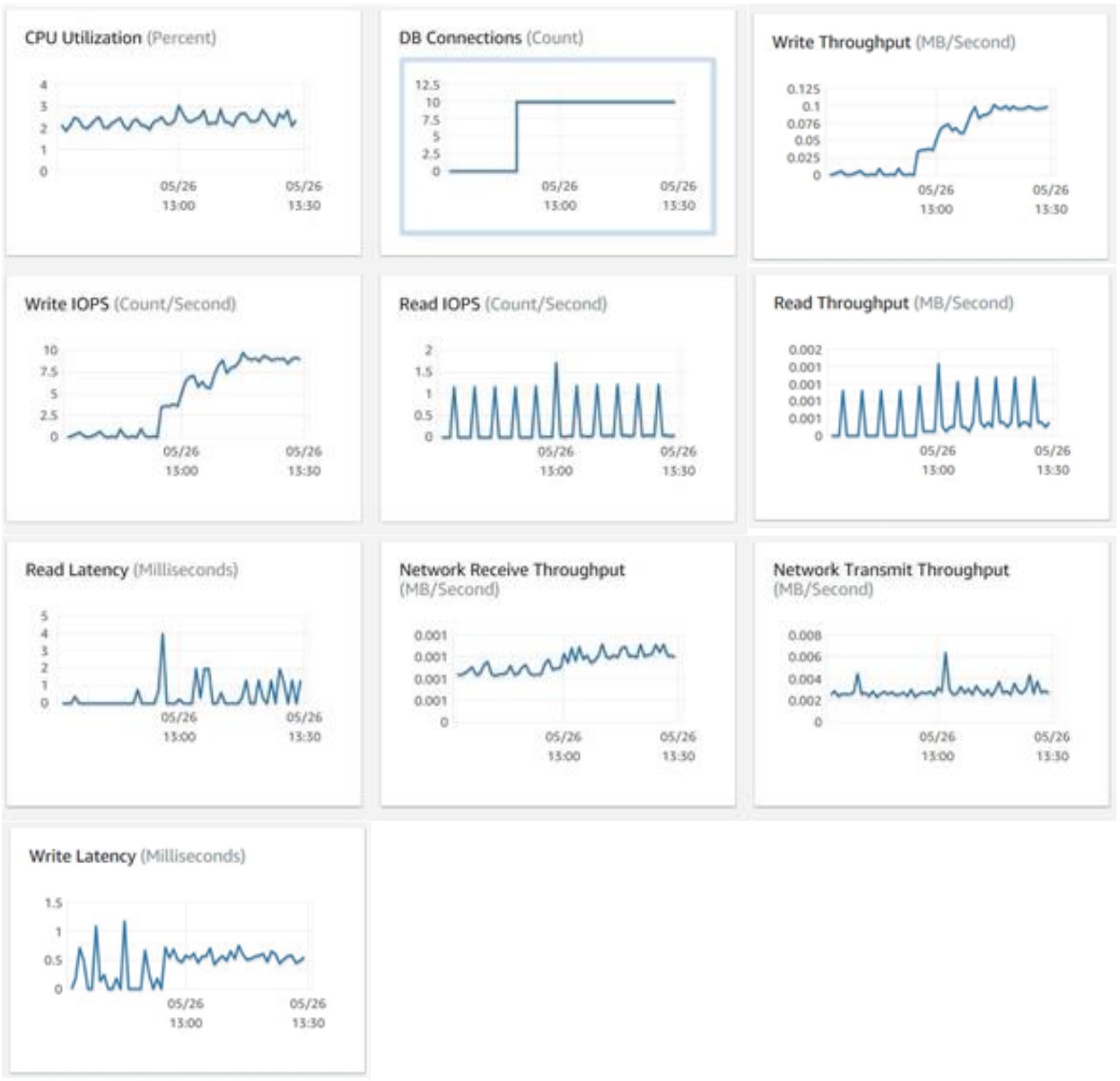


Figura 121. Gráficos estadísticos obtenidos mediante CloudWatch para la conexión secuencial del servidor y enchufes inteligentes hacia AWS RDS.

Un comportamiento similar se puede observar en las gráficas obtenidas en AWS Dynamo como se observa en la ***Figura 122*** y ***Figura 123***, aunque los gráficos mostrados pertenecen a la misma prueba, el tiempo varía considerablemente debido que los relojes de las DB no se encuentran sincronizados.

La ***Figura 122*** muestra la cantidad de dispositivos conectados a AWS Dynamo, como se puede observar la capa gratuita tiene un límite de 5 dispositivos conectados al mismo tiempo, por lo tanto, en caso de requerir aumentar el número de dispositivos no podrá ser conectado debido a esta limitante.

Figura 122. Gráfico estadístico obtenido mediante CloudWatch de capacidad de escritura en AWS Dynamo.

La ***Figura 123*** muestra la latencia de los datos en DynamoDB, cabe recalcar que la DB es utilizada para la transmisión de los datos en tiempo real, el cual tiene un valor estimado de 7ms, que es un valor menor a los 100ms requeridos para transmisión de datos en tiempo real.

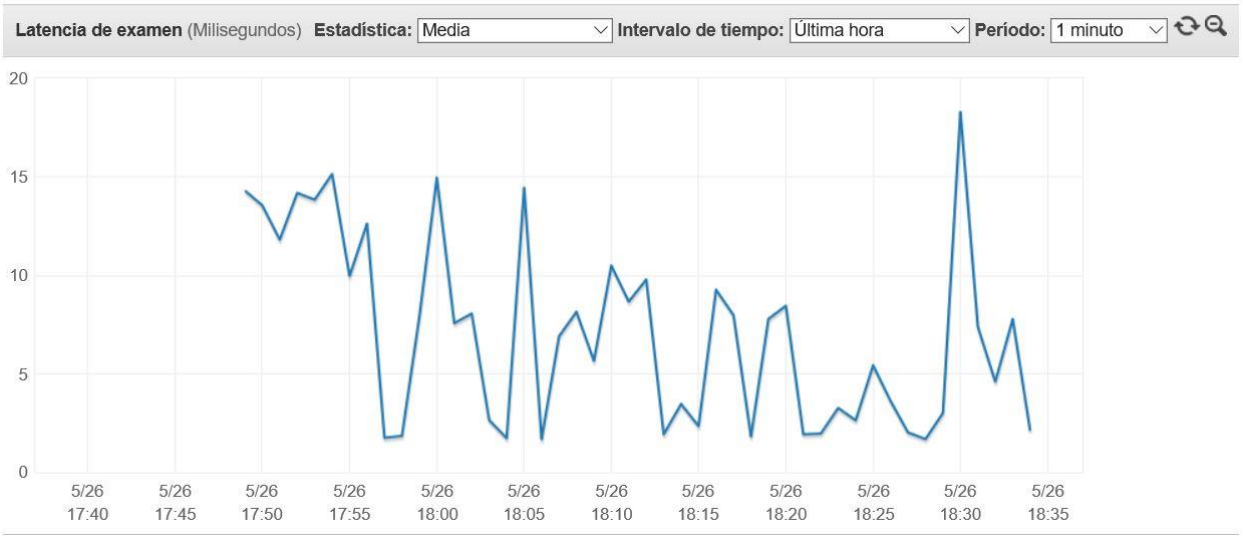


Figura 123. Gráfico estadístico obtenido mediante CloudWatch de latencia en AWS Dynamo.

5.2.4.2. Desconexión de la base de datos

Otra forma de analizar la base de datos es observar los estados de conexión y desconexión del de la base de datos RDS. Esto se debe a varios factores, como fallas del hardware del servidor local, o fallas en acceso al servicio de internet, ver ***Figura 124***.

Las estadísticas obtenidas en la ***Figura 122*** fueron obtenidas a partir de una simulación de falla de hardware, por lo que se detuvo el servidor local durante un tiempo prudente. El resultado muestra como alrededor de las 8:00pm y 10:00pm fue interrumpido la comunicación con AWS RDS, por lo tanto, el almacenamiento de los datos no es posible realizarlo. Esto se puede evidenciar en gráficas como el número de conexiones DB, o el número de IOPS de escritura. Sin embargo, la gráfica correspondiente al porcentaje de uso de la instancia permanece constante en el tiempo con un valor de alrededor 2%.

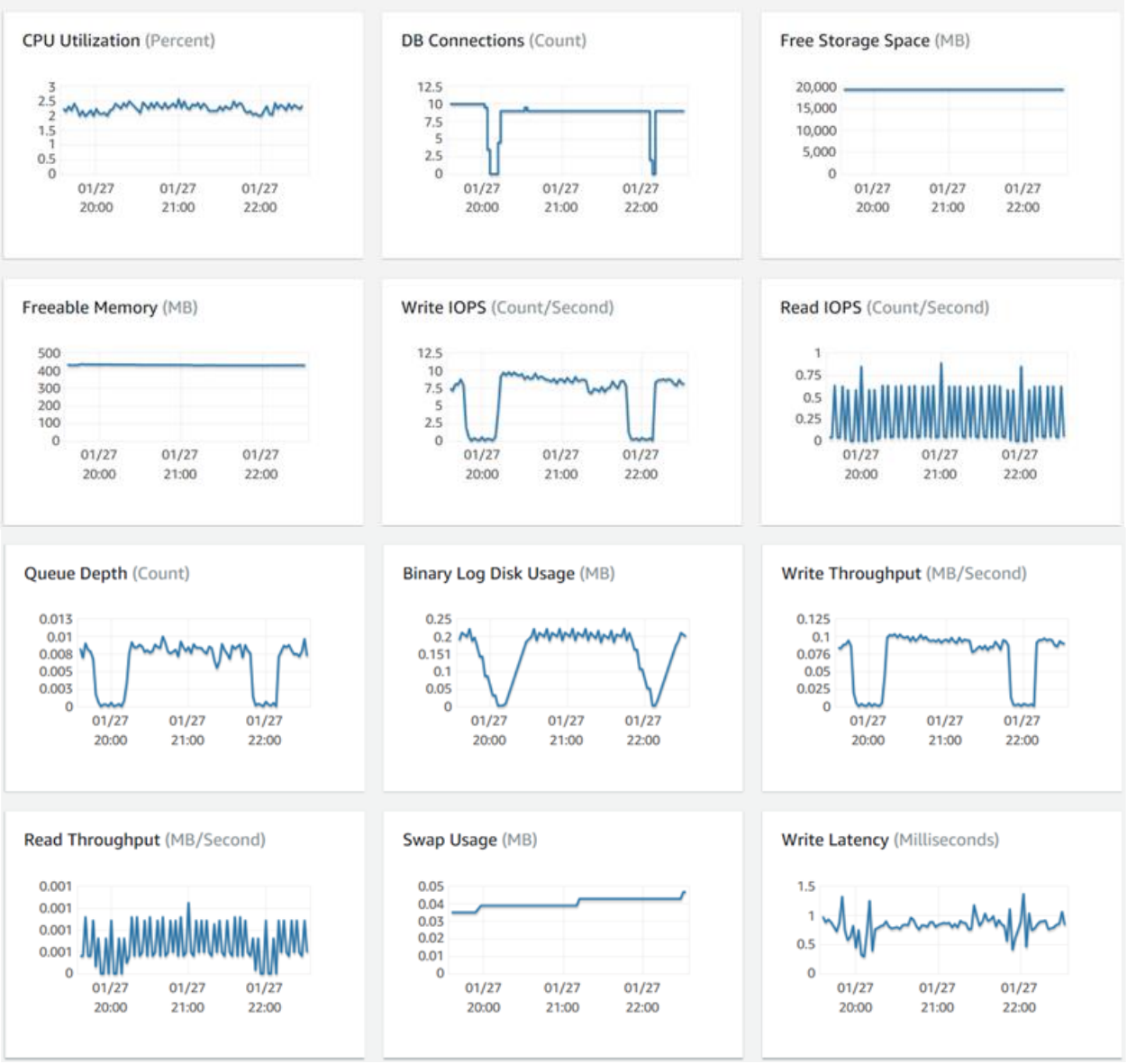


Figura 124. Gráficos estadísticos obtenidos mediante CloudWatch para los estados de conexión y desconexión a AWS RDS.

El mismo comportamiento se puede observar en las gráficas obtenidas en AWS Dynamo como se observa en la ***Figura 125***. Cabe recalcar que fueron tomados en instantes diferentes, pero aun así se observa cómo hay actualización de 3 elementos diferentes en la DB

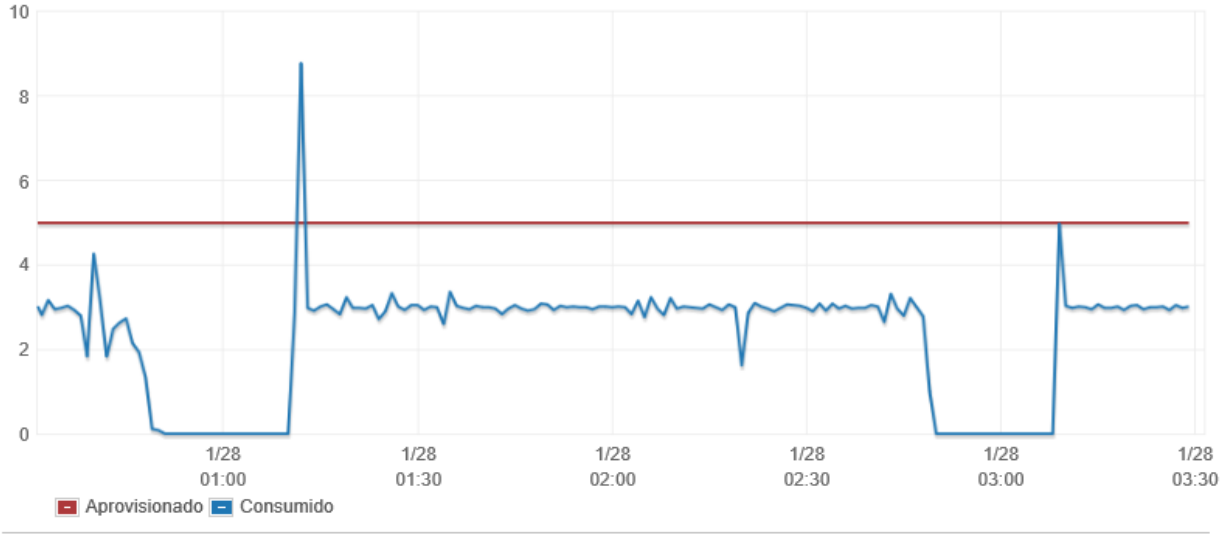


Figura 125. Gráfico estadístico obtenido mediante CloudWatch de la capacidad de escritura para los estados de conexión y desconexión a AWS Dynamo.

5.2.5. Hábitos de consumo

Las gráficas proporcionadas en el aplicativo web en la ***Figura 110***, ***Figura 112***, y ***Figura 111***; permitió observar la capacidad de AWS Machine Learning para identificar los patrones de consumo eléctrico de los dispositivos electrónicos conectados, sin embargo, es necesario comprobar su validez. Para ello se ha generado una encuesta (ver Anexo 5) a los miembros del hogar para determinar las horas del día durante el cual se hace uso del televisor, computador, y microonda. El resultado se ha tabulado y representado gráficamente para su comparación en la ***Figura 126***, ***Figura 127***, y ***Figura 128***, sin embargo, cabe recalcar que el modelo de machine learning es una predicción, mientras que la encuesta hace referencia al promedio del consumo escrito por cada miembro de la familia.

En la ***Figura 126*** se observa como ambas gráficas coinciden que el televisor se enciende en la mañana entre 5am y 7am; la tarde entre 1pm y 3pm; y la noche entre 7pm y 12am. Además, la encuesta proporciona otras franjas horarias donde posiblemente se haga uso del televisor.

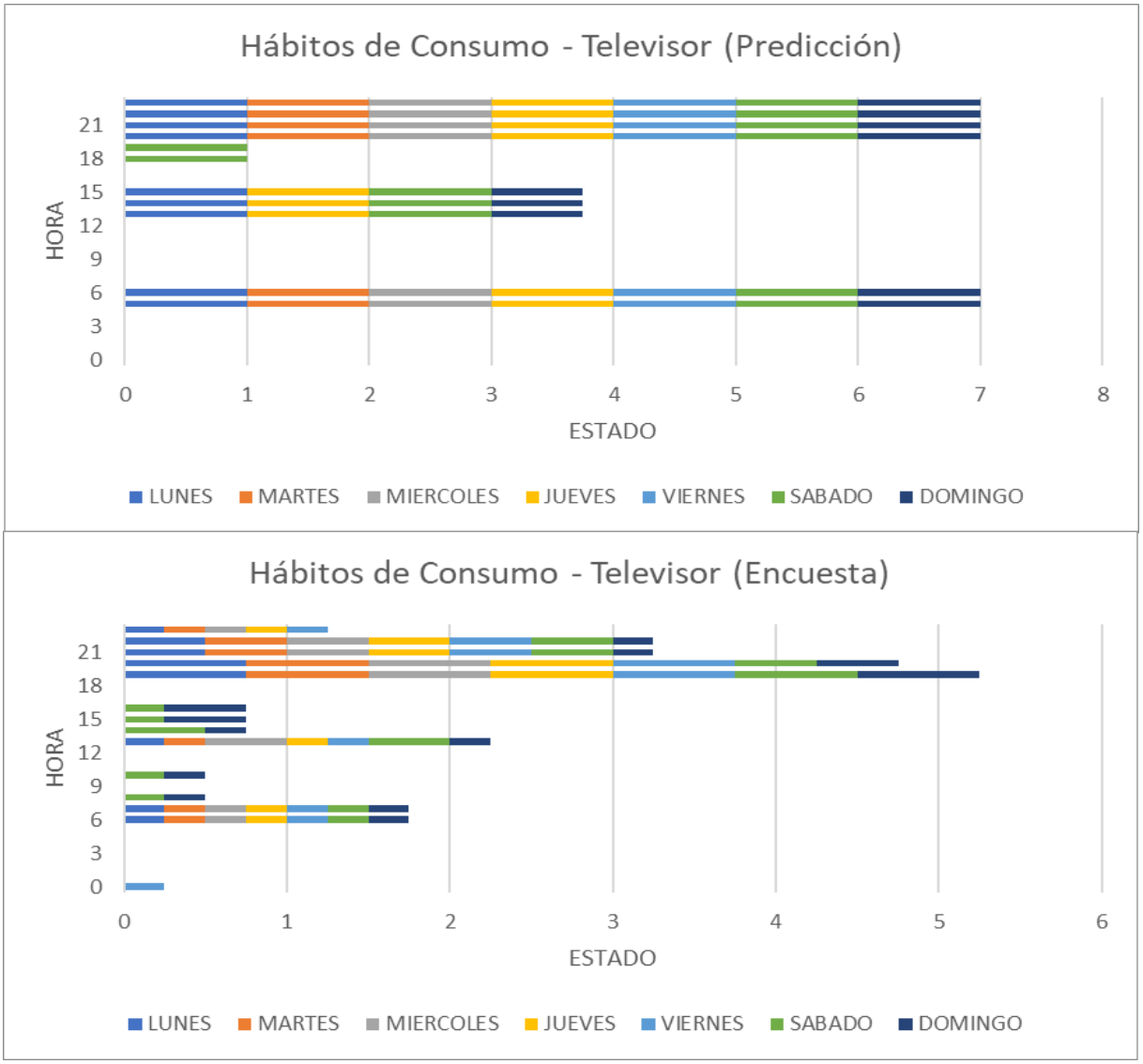


Figura 126. Comparación de gráficas de hábitos de consumo del televisor entre la predicción y la encuesta.

En la ***Figura 127*** se muestra como las gráficas no tienen relación, a pesar de que únicamente coinciden el domingo a 11m y 5pm. Esto se debe a que la predicción considera el uso únicamente de la computadora de escritorio, mientras tanto la encuesta hace referencia a la computadora en

general, es decir, computadora de escritorio y portátil. De acuerdo a información suministrada por el hogar, la computadora de escritorio únicamente se solía utilizar para realizar impresiones, por tal motivo el uso es nulo.

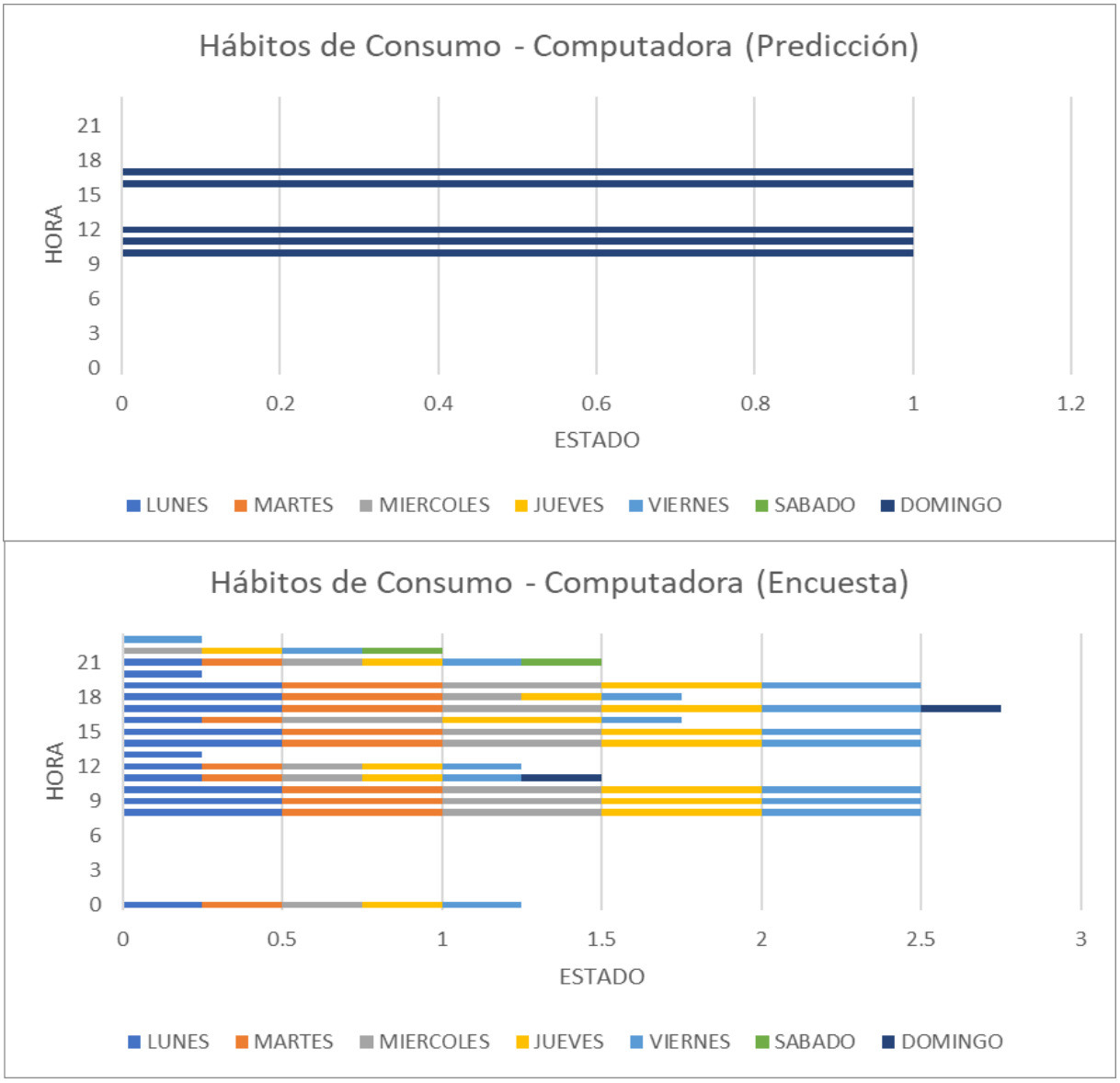


Figura 127. Comparación de gráficas de hábitos de consumo del computador entre la predicción y la encuesta.

En la ***Figura 128*** se muestra como las gráficas tienen una baja correlación, es decir, ciertas franjas horarias son similares y otras no. En general, la microonda se utiliza en la mañana entre

4am y 6am; la tarde entre 12pm y 3pm; y la noche entre 7pm y 9pm. Sin embargo, en la encuesta se hace referencia a otras franjas horarias como entre las 6am y 10am; y entre 3pm y 7pm. En este caso se evidencia claramente que el modelo de machine learning no se podría aplicar a escenarios, donde su uso sea esporádico, y no se tenga una rutina establecida como se observa en la gráfica.

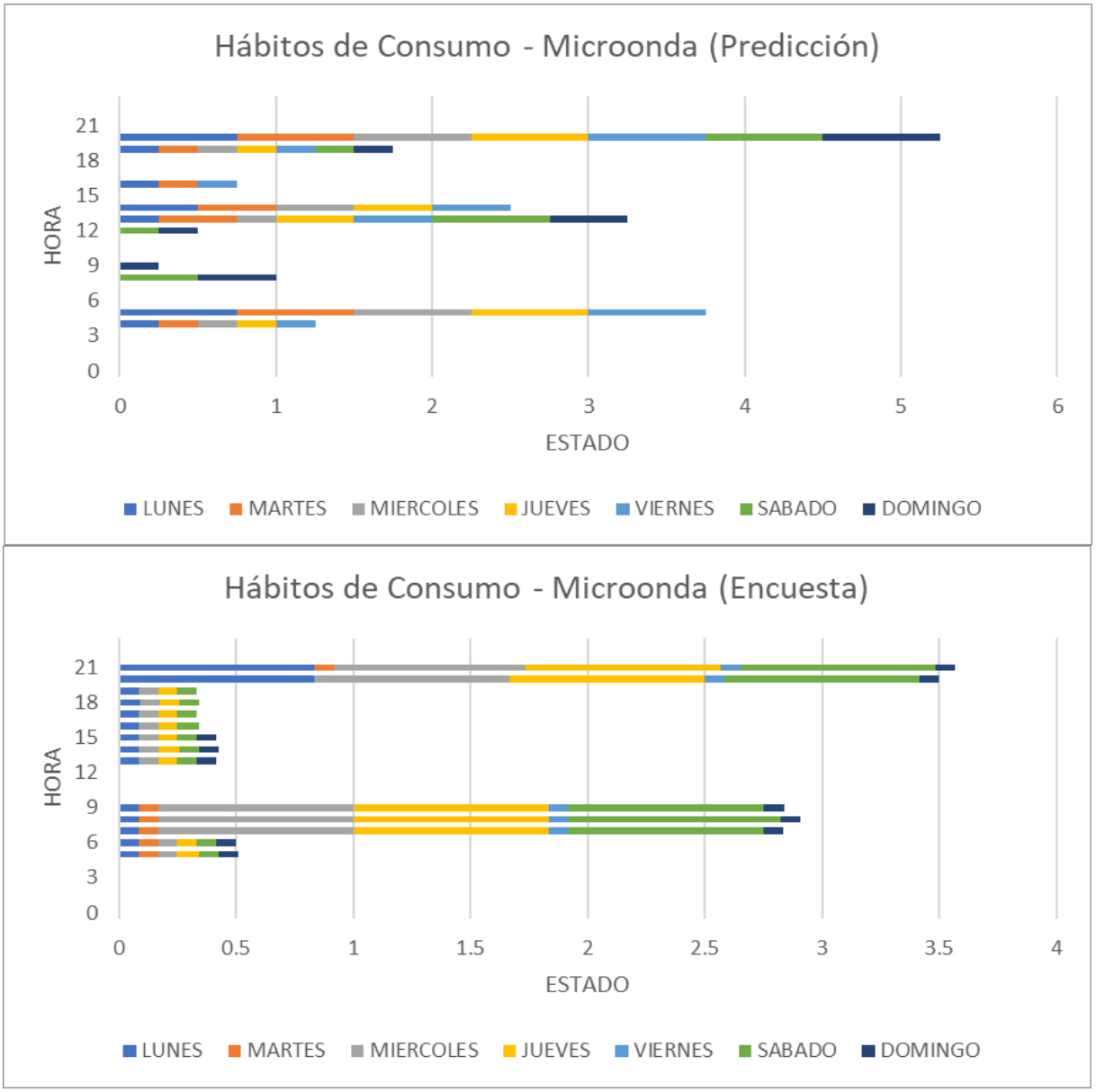


Figura 128. Comparación de gráficas de hábitos de consumo de la microonda entre la predicción y la encuesta.

5.2.6. Pruebas con SonarQube

SonarQube es un software libre y usa diversas herramientas de análisis estático para evaluar código fuente (SONARQUBE, 2019). Esta herramienta va a permitir auditar el código desarrollado, donde se podrá observar la calidad tanto para el servidor local como para el servidor web. En primera instancia el software fue descargado desde la plataforma de SonarQube, el servidor SonarQube y Sonnar-Scanner se encuentra en (SONARQUBE, 2019), y luego fue ejecutado en el SO Windows 10. A partir de la **Tabla 33**, se hizo uso de "sonar-scanner" para analizar el código de cada servidor, ver ***Figura 129***.

Tabla 33
Parámetros para ejecución de Sonar-Scanner

Parámetros	Servidor Node.js	Aplicativo web
Key	my:server	my:app
Nombre	My Local Server	My Application
Origen	./appsync-nodejs-smarthome-test	./amplify-js-app-smarthome-test

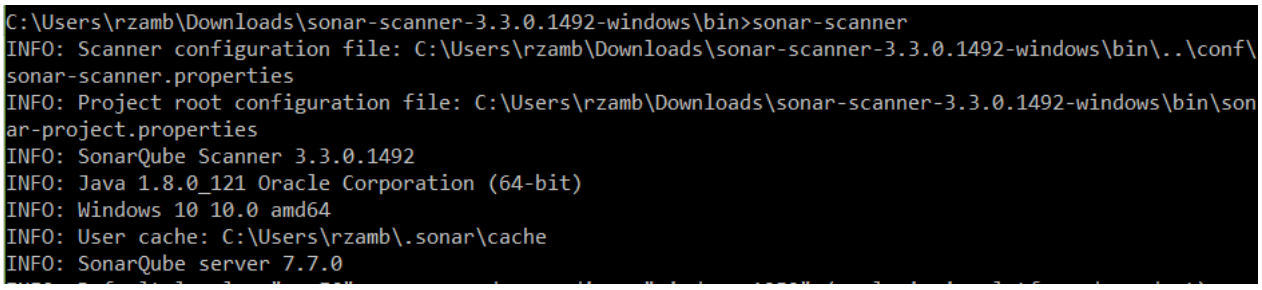


Figura 129. Ejecución de "sonar-scanner" en Windows10 para analizar el código del servidor.

Luego de ejecutar se ingresa al servidor SonarQube, mediante el explorador Google Chrome, se ingresa a la URL http://localhost:9000/ donde se observa el análisis realizado al código. Como se observa en la ***Figura 130***, cada uno de los servidores han pasado las pruebas realizadas por SonarQube, sin embargo, hay algunos bug y vulnerabilidades, como se observa en la ***Figura 131*** y ***Figura 132***.

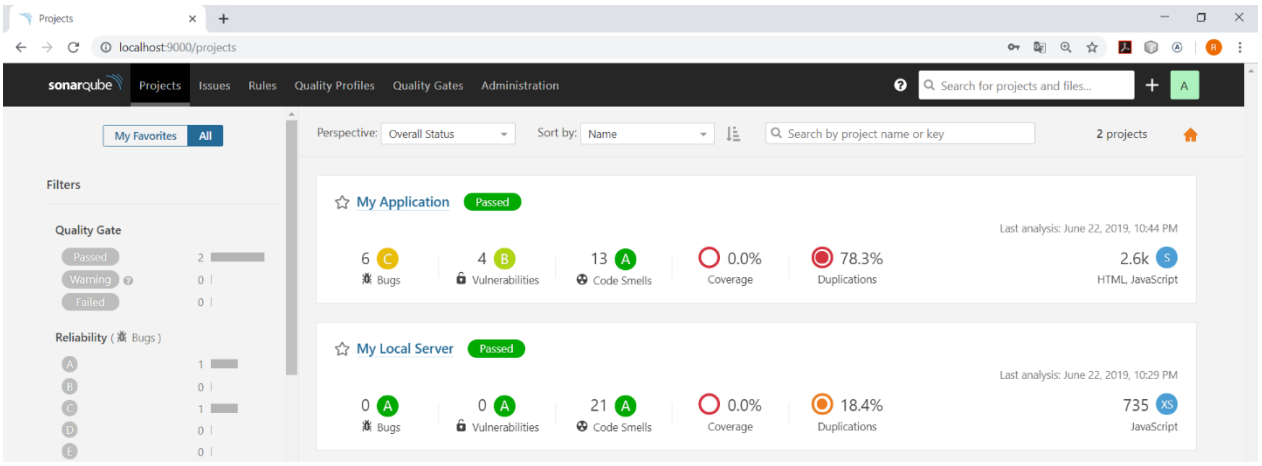


Figura 130. Lista de proyectos analizados en SonarQube por primera vez.

Entre los bugs detectados en la ***Figura 131***, la mayoría hacen referencia a sugerencias para agregar el atributo ALT en las imágenes añadidas, mientras que la ***Figura 132*** muestra vulnerabilidades que hacen referencia a la etiqueta *alert*, pese a ello, puede ser omitido debido a que son diálogos informativos para el usuario. A partir de las observaciones realizas, se procede a realizar las correcciones necesarias para cada proyecto. ***Figura 131***

Figura 131. Ejemplo de algunos Bugs detectados en código del aplicativo web.

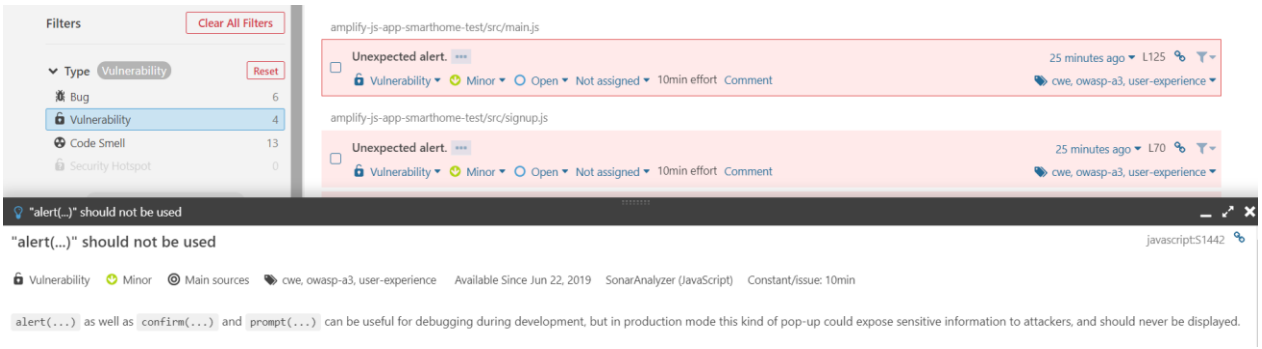


Figura 132. Ejemplo de algunas Vulnerabilidades detectadas en código del aplicativo web.

Luego de realizar las correcciones, el resultado se observa en la ***Figura 133***. Los bugs y códigos basura han sido corregidos, pese a ello, hay un indicador denominado "*Duplications"* que evidencia un alto valor porcentual de líneas duplicadas. Esto se debe a que al ejecutar cada uno de los servidores, el código del archivo webpack, ver ***Figura 68*** y ***Figura 81***, genera copias de los archivos principales en el directorio "*./dist/*", entonces esto genera una duplicidad de código en SonarQube.

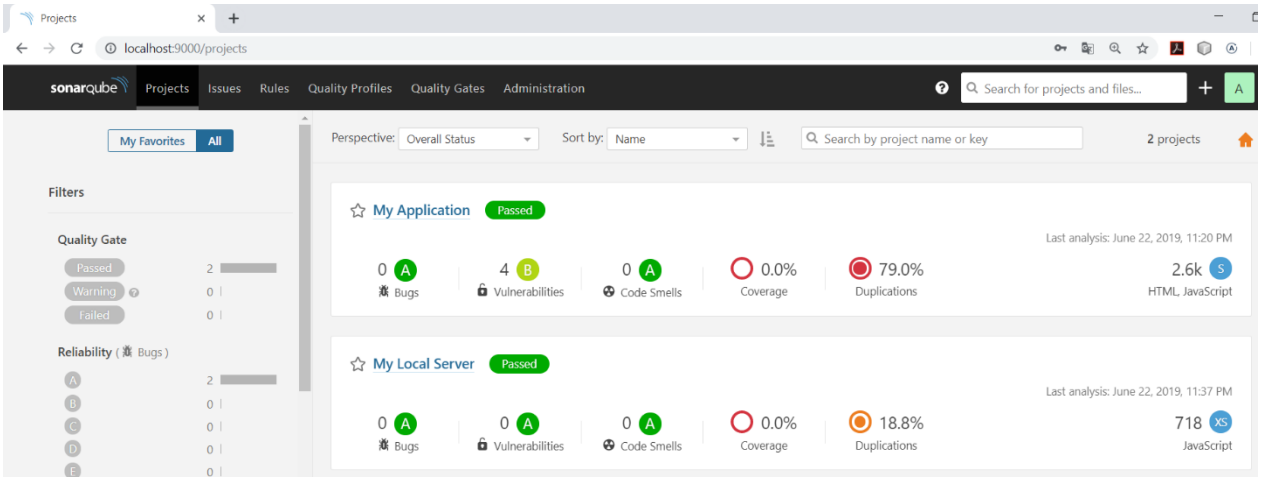


Figura 133. Lista de proyectos analizados en SonarQube luego de las correcciones.

La ***Figura 134*** y ***Figura 135*** muestra las líneas duplicadas del aplicativo web más detalladamente, allí se observa claramente como existen duplicidad de los archivos en la carpeta raíz y la carpeta dist, además, los archivos proporcionados por AWS Appsync almacenan líneas duplicadas debido a tipo de estructura de array que manejan.

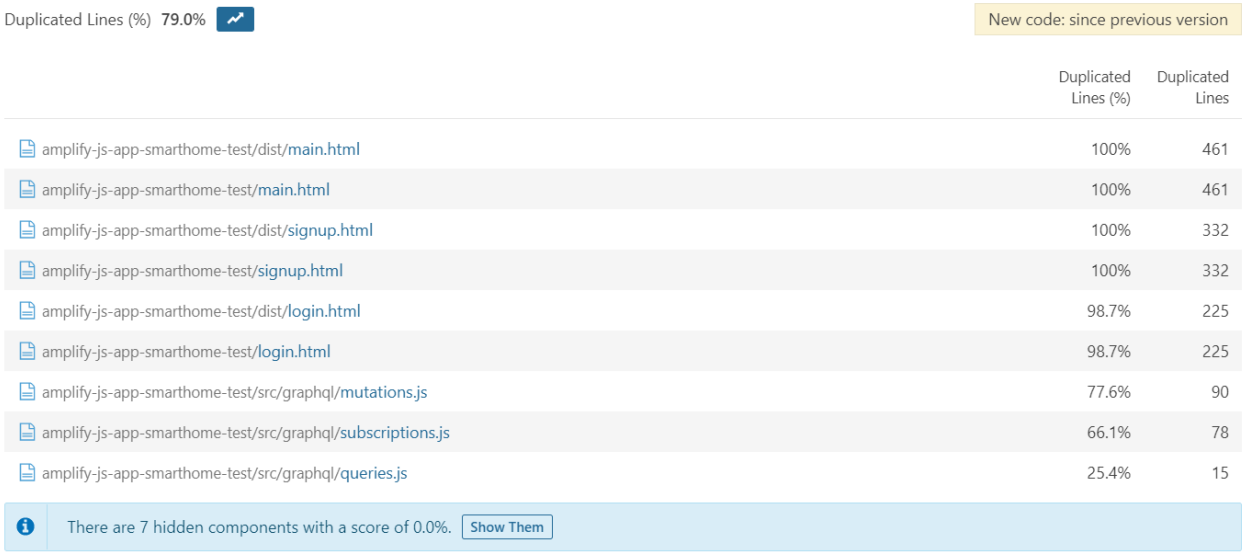


Figura 134. Listado de archivos con líneas duplicadas en código del aplicativo web.

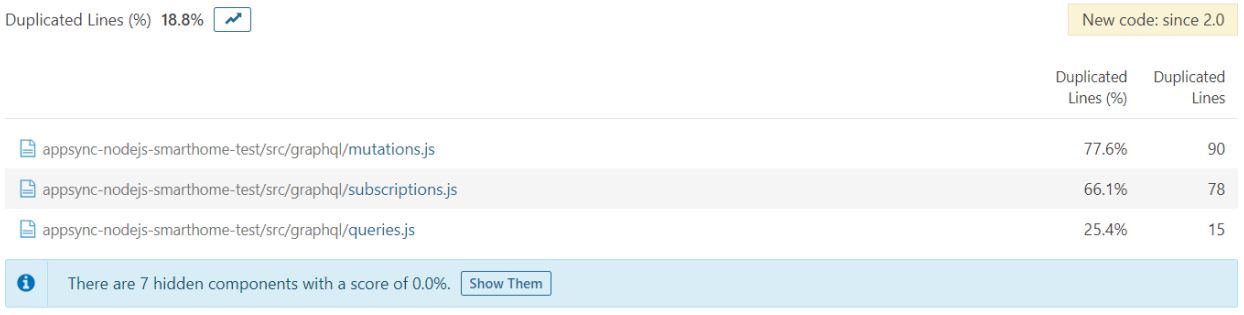


Figura 135. Listado de archivos con líneas duplicadas en código del servidor local.

Tabla 34
Resumen de bugs y vulnerabilidades encontrados en el código fuente.

Origen	Descripción	Acción
Aplicativo Web	Agregar atributo ALT en la imagen	Atributo agregado a cada imagen
Aplicativo Web	<!DOCTYPE> no declarado	Declaración en los archivos faltantes
Aplicativo Web / Servidor local	Librería sin usar	Borrar librería en el código
Aplicativo Web / Servidor local	Duplicidad de variables	Renombrar variables
Servidor local	Variables sin uso	Mantiene. Hace referencia a funciones
Aplicativo Web	Atributo alert declarado	Mantiene. Pop-ups informativos

En resumen, los bugs y vulnerabilidades son descritos en la **Tabla 34** junto con la acción correctiva tomada para solucionar los riesgos de seguridad.

5.2.7. Pruebas con JMeter

La aplicación Apache JMeter es un software de código abierto diseñada para realizar pruebas de cargar y medir el rendimiento mediante escenarios de prueba (The Apache Software Fundation, 2019). El apache originalmente fue diseñado para probar aplicaciones web, sin embargo, ahora es posible realizar pruebas en diferentes tipos de plataformas, como por ejemplo a servidores mysql.

El aplicativo web y la base de datos van a ser sometidas a pruebas de cargas, con la finalidad de medir el rendimiento de cada una de ellas, las pruebas se describen a continuación:

5.2.7.1. Pruebas al Aplicativo Web

En primer lugar, la URL del aplicativo web es la siguiente:

http://smarthomeapp-20181129221710--hostingbucket.s3-website-us-east-1.amazonaws.com/

El Apache JMeter es ejecutado en SO Windows 10. En ventana principal se configura un Servidor Proxy HTTP tal como se observa en la ***Figura 136***. Esta función nos permite capturar los

paquetes REST enviados y recibidos por el explorador web cuando se navega a través del aplicativo web del sistema de gestión de energía. Además, se establece el puerto proxy a 8888, tanto en JMeter, ver ***Figura 136***, como en la configuración de Windows, ver ***Figura 137***.

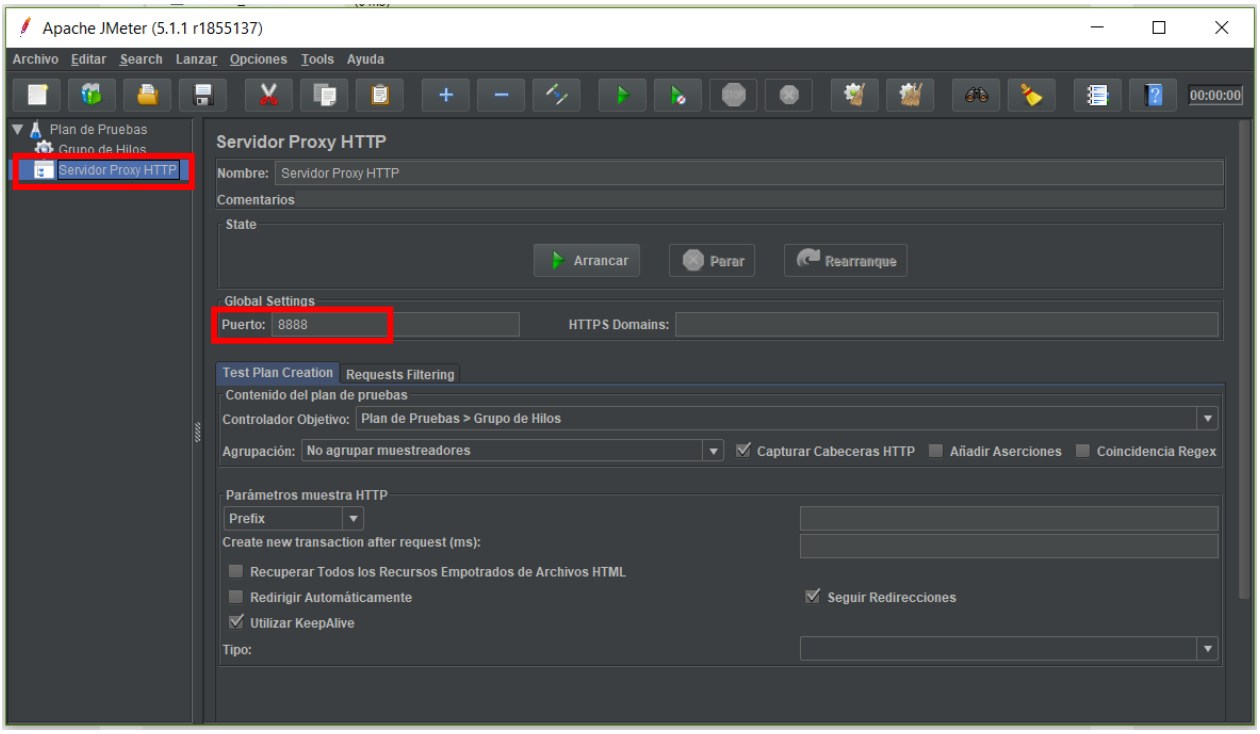


Figura 136. Configuración de servidor proxy HTTP en JMeter.

Figura 137. Configuración manual de proxy en Windows 10.

Luego se arranca la captura, y mediante el explorador web se simula la navegación de un usuario común, el cual implica las siguientes acciones:

- Carga de página principal de Energy Cloud-ML
- Acceso a Energy Cloud-ML
- Navegación en Energy Cloud-ML
- Cierre de sesión de Energy Cloud-ML

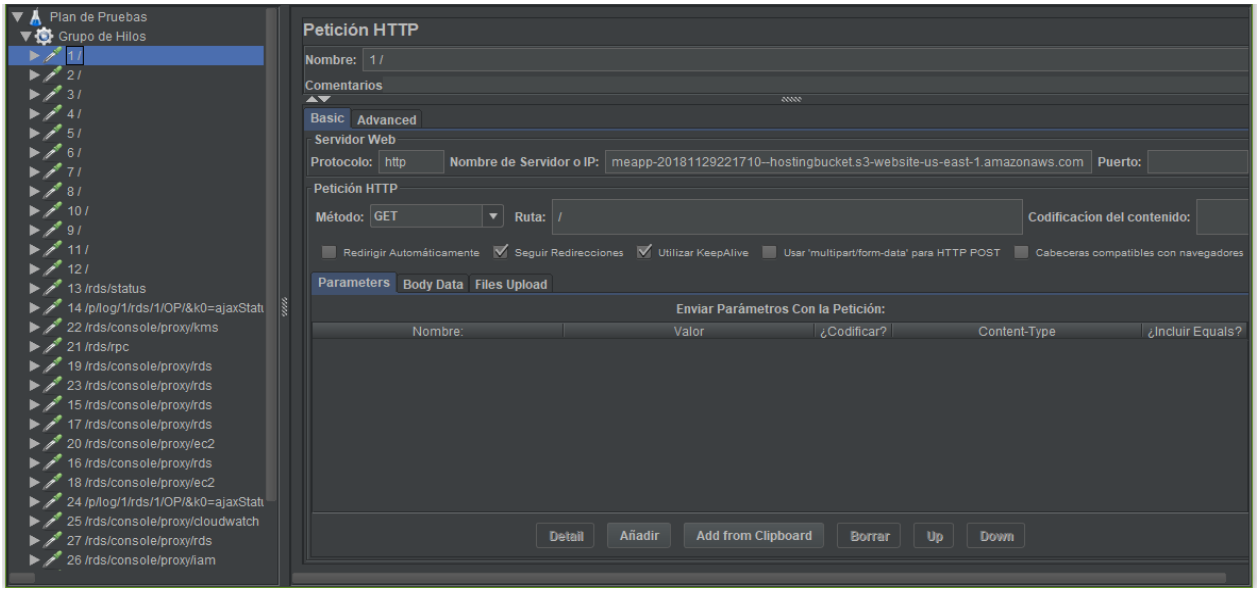


Figura 138. Captura de paquetes mediante uso del servidor proxy HTTP.

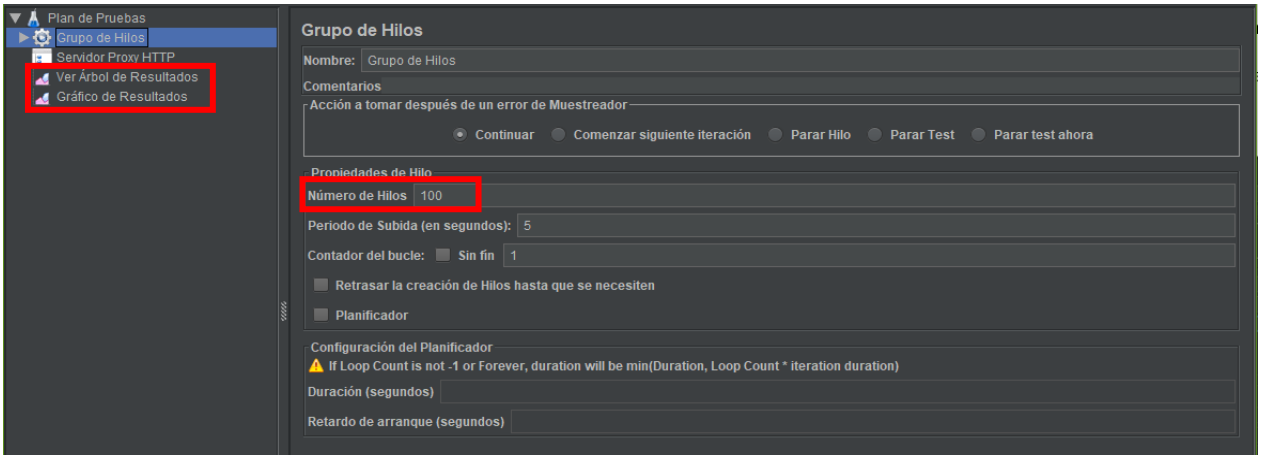


Figura 139. Configuración de Grupos de Hilos para simulación de 100 usuarios en JMeter.

Los paquetes capturados se observan en la ***Figura 138***. Ahora, la simulación de un usuario común se replica para *n* cantidad de usuarios. Para ello se configura el Grupo de Hilos a 100, para simular el uso de 100 usuarios concurrentes del aplicativo web, como se observa en la ***Figura 139***.

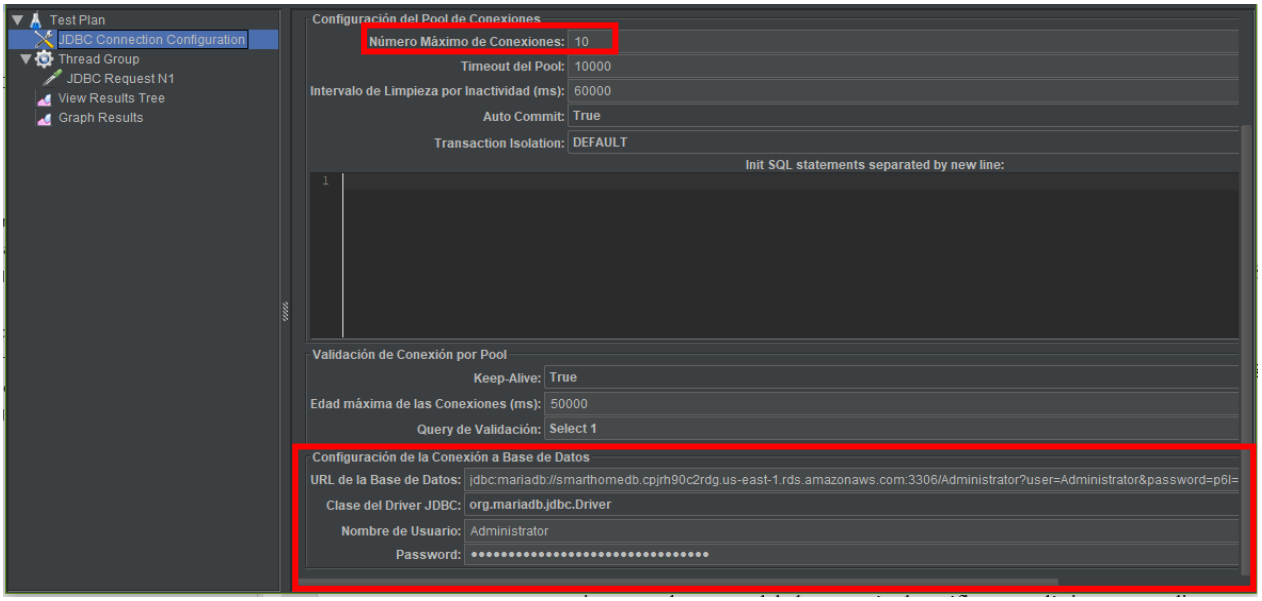


Figura 140. Configuración de Conexión JDBC en JMeter.

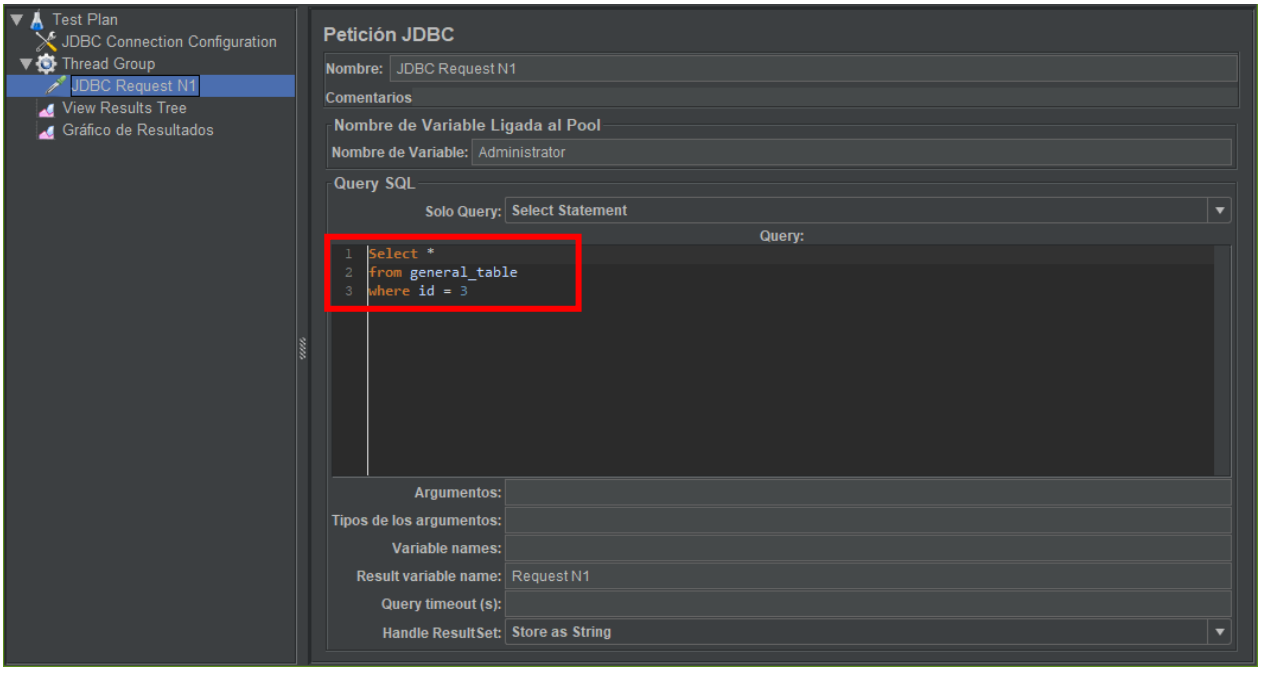


Figura 141. Configuración de Solicitud JDBC en JMeter.

Los resultados obtenidos muestran que en promedio la carga del aplicativo web tarda 189ms, mientras que el rendimiento estimado se establece en 16.890,307 solicitudes por minuto, valor sumamente alto considerando que se cargan alrededor de 280 paquetes por segundo, en función de las solicitudes o peticiones capturadas.

5.2.7.1. Pruebas a Servidor MYSQL

JMeter además permite realizar pruebas de carga a diversas bases de datos estructuradas, a continuación, se procede a realizar una prueba de carga a la DB en AWS RDS. En primer lugar, se arma el Plan de prueba, y se establece la conexión hacia la DB mediante JDBC ("Java Database Connection"), como se observa en la ***Figura 140***.

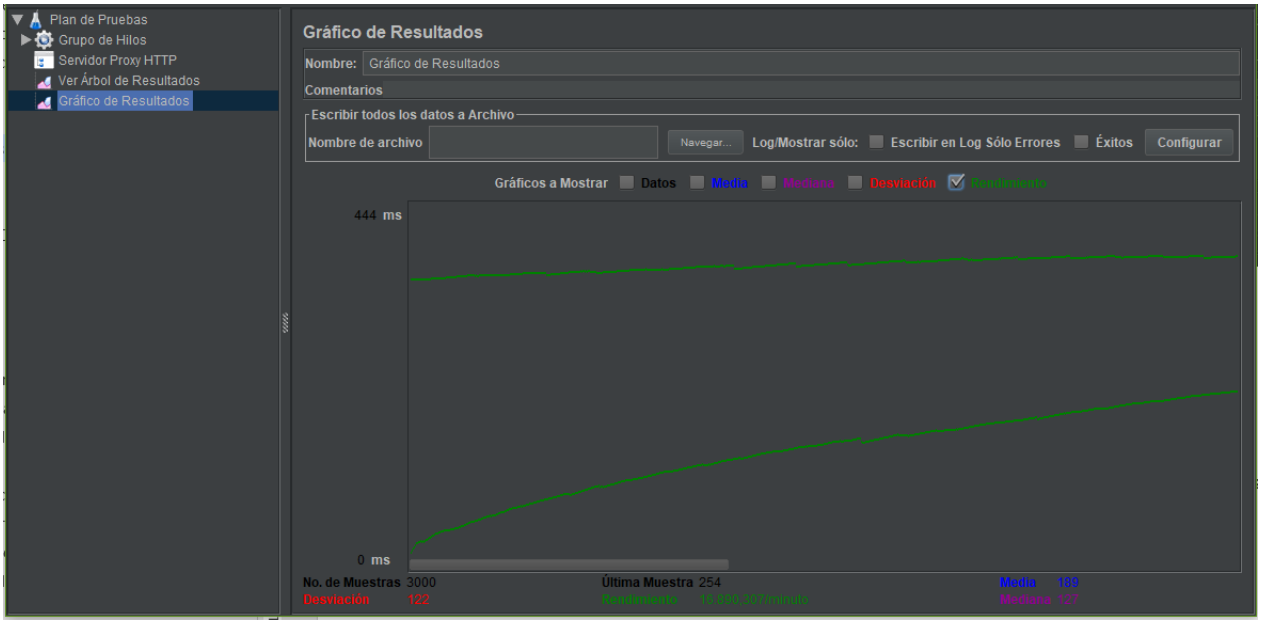


Figura 142. Resultados de la prueba al aplicativo web con JMeter.

Entre los atributos requeridos son: número de conexiones hacia la DB, URL (formato JDBC), driver JDBC (motor MariaDB), usuario, y password.

En grupo de hilos se establece el número de solicitudes a ser realizadas a la DB, para este caso el número se define a 6000, ver ***Figura 143***, mientras que en la solicitud JDBC se especifica la consulta SQL a ser efectuada para cada una de las solicitudes como se observa en la ***Figura 141***.

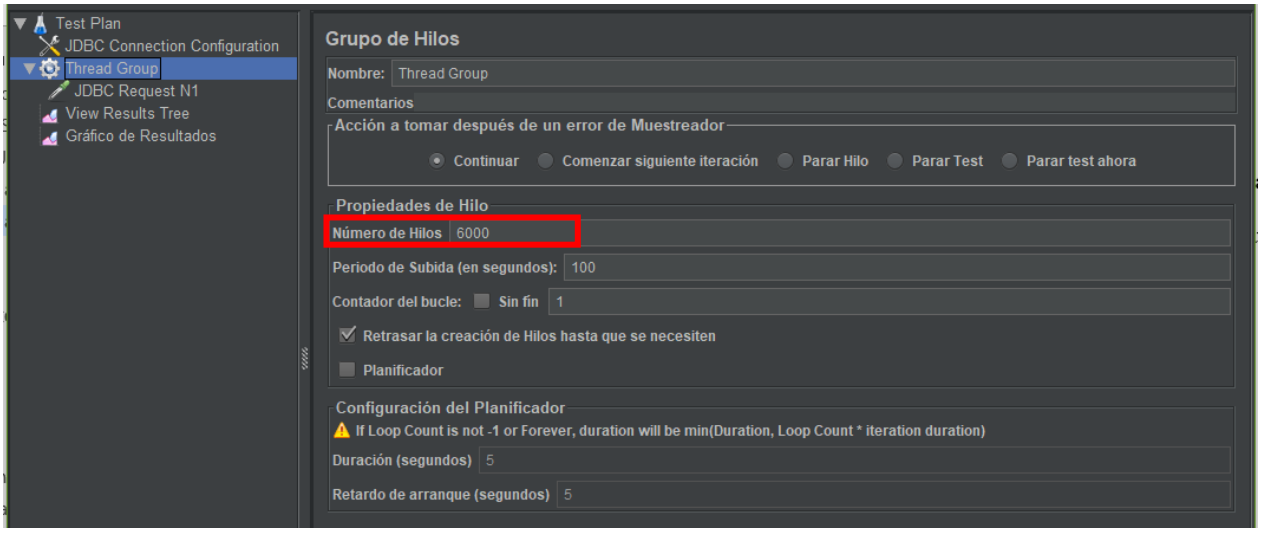


Figura 143. Configuración de Grupo de Hilos en JMeter.

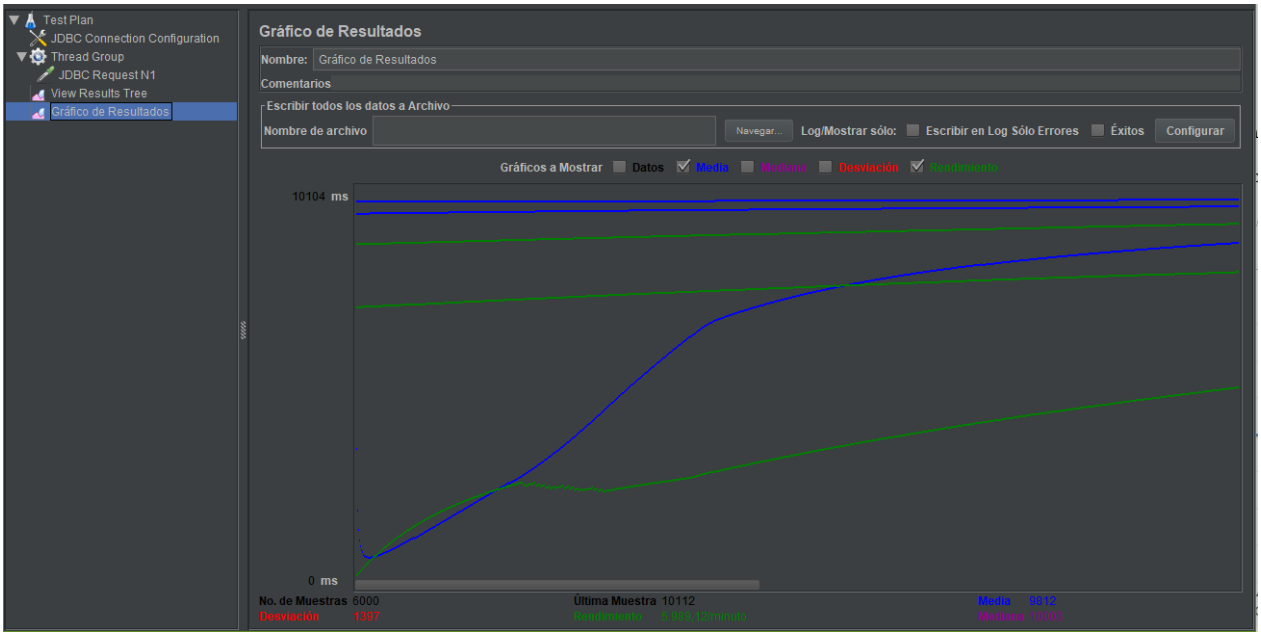


Figura 144. Resultados de la prueba a servidor MYSQL con JMeter.

Los resultados de la ***Figura 144*** establecen que en promedio la sesión MYSQL tarda 9812ms en cerrar, tiempo durante el cual se efectúa la solicitud establecida en la ***Figura 141***, además, el rendimiento de la DB muestra que en promedio se realizaron 5989,12 solicitudes por minuto, es decir, cada consulta es ejecutada en aproximadamente en 1s.

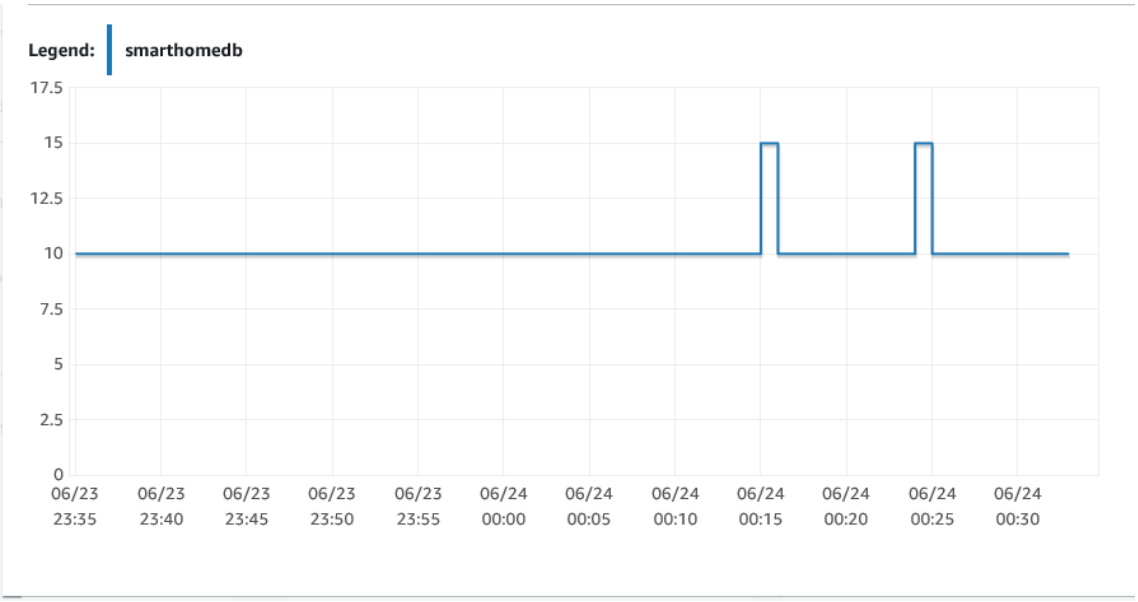


Figura 145. Número de conexiones generadas en AWS RDS.

Como se observa en la ***Figura 145***, el número de conexiones aumento a 5, menor a la configuración realizada en la ***Figura 140***, y luego de realizar la carga el servidor siguió ejecutándose con normalidad, es decir, la carga efectuada no saturó al servidor RDS.

5.2.8. Otros

Además, en la encuesta se utilizó como herramienta para verificar el conocimiento sobre el consumo eléctrico de sus equipos electrónicos, entre otros (Ver Anexo 5). Las preguntas realizadas fueron la siguientes:

1) Considera que los dispositivos electrónicos conectados al tomacorriente consumen energía mientras se encuentran apagados.
2) Si pudiera controlar la conexión y desconexión de dispositivos electrónicos conectados al tomacorriente, ¿Compraría un dispositivo capaz de ejecutar esta tarea?
3) Considera que el uso o consumo diario de dispositivos electrónicos conectados al tomacorriente puede ser modelado a través de gráficos estadísticos como diagrama de barras, histogramas, entre otros.
4) Si pudiera modelar los hábitos de consumo de los dispositivos electrónicos conectados al tomacorriente, ¿Qué parámetros de tiempo consideraría necesarios? (minuto, hora, día, etc)
5) Considera que los dispositivos electrónicos conectados al tomacorriente requieren ser integrados en un sistema de gestión, capaz de tener 100% control tanto en la casa como vía remota.

Tabla 35
Promedio y desviación estándar en función de la Tabla 29

Pregunta	Padre	Madre	Hija 1	Hija 2
1	SI	SI	NO	SI
2	SI	Depende (beneficio)	NO	SI
3	SI	NO SABE	SI	SI
4	HORA	HORA	20 MIN	HORA
5	SI	SI	NO	SI

Los resultados en la encuesta fueron tabulados en la **Tabla 35**. Allí se obtuvo que:

- Los miembros del hogar tienen un claro conocimiento que sus equipos consumen energía, a pesar de que se encuentren apagados, aunque no tienen conocimiento de la cantidad de energía, y menos aún el costo que implica.
- Ellos están parcialmente de acuerdo en comprar dispositivos que permitan controlar la conexión y desconexión, siempre y cuando haya un beneficio económico y funcional.
- Además, están de acuerdo de que es posible modelar los hábitos de consumo de todos los equipos electrónicos, aunque tienen dudas respecto a equipo se usan de vez en cuando, por ejemplo, plancha, minicomponente, microonda, entre otros.
- Los hábitos podrían ser representados gráficamente en franjas horarias. (tomada como referencia para las gráficas)
- Un sistema de gestión sería fundamental para el control de los dispositivos electrónicos, en especial en hogares para el control parental a los hijos, además, se podría aplicar para desconectar a todos los equipos electrónicos al salir de viaje.

CAPÍTULO 6

6. CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

- La implementación del sistema de gestión de energía permitió validar el uso de dispositivos IoT (enchufe inteligente HS110), un dispositivo central (Raspberry PI3), en una arquitectura orientada a servicios, para capturar y almacenar los datos proporcionados, mostrar los parámetros en tiempo real, y proporcionar a los usuarios el hábito de consumo de los dispositivos electrónicos conectados a este.
- El proyecto de investigación fue diseñado de forma modular, de tal manera de que cada segmento sea totalmente independiente como se observa en el Diagrama BPMN y Esquema final en la ***Figura 15*** y ***Figura 16***. El diseño modular optimiza el tiempo de construcción y permite definir funciones específicas a cada uno de los bloques que son descritos a continuación. La infraestructura local conformada por la Raspberry PI3 interactúa con los dispositivos IoT y la plataforma AWS para establecer un canal de comunicación entre ellos. La plataforma AWS hace uso de servicios para transmitir la información proporcionada por los enchufes al frontend, y aplica modelos de machine learning. El Cliente accede al frontend para gestionar los dispositivos IoT. El Administrador encargado de la gestión y optimización del backend y frontend.
- El sistema de gestión de energía hace uso de dispositivos IoT para monitorizar y controlar los equipos electrónicos conectados. Los dispositivos funcionan a partir de una arquitectura REST, por tal motivo se puede interactuar mediante flujo de peticiones HTTP. La interacción se puede realizar de 2 formas, la primera mediante peticiones HTTP, y la

segunda mediante el uso de una API abierta. Sin embargo, la API facilita la gestión de datos y control sobre los dispositivos, y evita la autenticación con la plataforma TPLINK cuando es ejecutada dentro de la red local, a diferencia de las peticiones HTTP que requieren credenciales para acceder al endpoint, capturar los datos y enviar las señales de control a los dispositivos, siendo totalmente dependientes de la plataforma del fabricante. Los datos suministrados por la API se pueden observar en la ***Figura 20*** y la descripción en la **Tabla 11**, a pesar de ello, necesita tener una máquina local que ejecute el código, y se enlace a la red local. Por esta razón se hace uso de la Raspberry PI3 como el servidor local del proyecto.

- La arquitectura propuesta fue desarrollada en la plataforma AWS e integrada al servidor local mediante la gestión de los servicios en la consola de administración, y librerías de desarrollo. La consola se utilizó durante la etapa de configuración de los servicios debido a que presenta una interfaz visual amigable e intuitiva para la configuración de los servicios, a diferencia de la gestión por CLI que requiere un conocimiento más avanzado referente a los atributos y parámetros a considerar por los servicios. Las librerías de desarrollo (AWS SDK y Amplify) fueron clave para la integración del servidor local a la plataforma AWS, y desarrollo del frontend porque interactúa con servicios como AWS IAM y Cognito para autenticación, AWS Secret Manager para obtener credenciales de acceso hacia la DB, y AWS Appsync para flujo de datos en tiempo real. Sin embargo, AWS Amplify generó varios inconvenientes de interoperabilidad durante el desarrollo del servidor local, ya que el código desarrollado corresponde al lenguaje Javascript ES5, y AWS Amplify funciona únicamente bajo JS ES6. Por tal motivo, se hizo uso de librerías como ES6-promise para mantener un funcionamiento de compatibilidad con funcionalidades reducidas.

- El flujo de datos en tiempo real en el sistema de gestión de energía fue desarrollado considerando una segura administración de los datos, y uso de lenguaje de consulta ligeros con sintaxis intuitiva y flexible para describir los datos. Entre los servicios se encontró AWS Appsync, el cual a partir del lenguaje GraphQ permite solicitar, modificar y suscribirse a los datos exactos que se necesita en una única solicitud de red para el envío de los parámetros de consumo eléctrico y señales de control entre el servidor local y el frontend en tiempo real, y permite al usuario interactuar con los dispositivos IoT de forma rápida, segura, y sin restricción de tiempo. Sin embargo, las librerías de nodeJS aún están en desarrollo, por tal motivo no fue posible generar el mismo comportamiento desde el frontend hacia el servidor, a pesar de ello se aplicó un timer de 30s para censar las señales de control y mantener el esquema de tiempo real como se observa en ***Figura 78***.
- La arquitectura AWS del sistema de gestión de energía permite por un lado la transmisión de parámetros de consumo eléctrico en tiempo real, y por el otro la aplicación de un modelo de machine learning, la arquitectura propuesta se puede observar en la ***Figura 16***. Sin embargo, requerimientos como latencia, tamaño del almacenamiento, número de conexiones y dinámica de los datos han sido factores que definieron el uso de 2 diferentes flujos de datos. El primero conformado por datos temporales en tiempo real, y el segundo por datos históricos almacenados secuencialmente. Los datos en tiempo real hacen uso de DynamoDB para el almacenamiento datos dinámicos que varían durante el tiempo, y la latencia durante el proceso de escritura y lectura es baja. Los datos históricos hacen uso de AWS RDS para el almacenamiento de gran cantidad de datos estáticos y estructurados que incrementan gradualmente, además, la latencia no es factor primordial durante la lectura y escritura.

- La seguridad de la información en desarrollo del Sistema de Gestión de Energía busca resguardar y proteger la información del usuario a través de un conjunto de mecanismos de seguridad proporcionados por la plataforma AWS como IAM, Cognito y Secret Manager. AWS IAM mediante el uso de usuario y contraseña de administración para la plataforma AWS, define reglas o políticas de acceso a los servicios, es decir, permite o niega accesos de lectura, escritura, y/o ejecución de servicios definidos. AWS Cognito permite la creación de usuarios en el frontend mediante claves de verificación, y brinda la posibilidad de vincular los usuarios con plataformas como Facebook, Yahoo o Google, entre otras. AWS Secret Manager por su parte, es un mecanismo de gestión de credenciales para AWS RDS, que almacena información necesaria para acceder a la DB como instancia, nombre, endpoint, y contraseña (ver ***Figura 44***), y además, mediante solicitudes cada 30 días a AWS Lambda, permite el cambio automático de la contraseña de acceso como se puede observar en la ***Figura 45***.
- El desarrollo del sistema de gestión de energía hace uso de machine learning para aprender los hábitos de consumo en un escenario de prueba mediante la provisión de datos históricos almacenados en una base de datos. El almacenamiento en AWS RDS puede ser realizado sobre diversos motores como MariaDB, PostgresSQL, Microsoft SQL Server, entre otros (ver motores disponibles en la ***Figura 41***). Sin embargo, MariaDB proporciona varias ventajas como interoperabilidad con la DB local, librerías compatibles con lenguaje Javascript, y menor coste, a diferencia de los demás motores. Además, el motor requiere definir una instancia de ejecución sobre un servidor en la plataforma AWS EC2, y la integración a la red virtual AWS VPC para la gestión de acceso remoto mediante puertos y direccionamiento IP público.

- El aprendizaje de hábitos de consumo en el sistema de gestión de energía hace uso de datos históricos para aprender, modelar, y predecir eventos en función de datos proporcionados por diferentes servicios de almacenamiento como AWS RDS, RedShift o S3, a pesar de ello, AWS Machine Learning únicamente importa y exporta archivos .csv desde S3, entonces los datos almacenados en RDS deben ser primero exportados en archivos .csv y almacenados en AWS S3. La extracción de los datos es realizada mediante AWS Pipeline (ver diagrama de flujo de la ***Figura 46***) de tal manera que las tablas de consumo eléctrico de cada dispositivo IoT sean almacenadas en archivos .csv en AWS S3 para ser utilizadas posteriormente en AWS Machine Learning.
- El aprendizaje de los hábitos de consumo en el sistema de gestión de energía consiste en reconocer el patrón de uso de un equipo electrónico conectado al dispositivo IoT cuyo estado es encendido o apagado. A partir del estado, en AWS Machine Learning se aplicó un modelo binario para predecir entre 1 y 0, traduciendo al sistema hace referencia cuando el equipo se encuentra encendido o apagado. Los datos históricos suministrados al servicio, permitió generar un patrón de datos de tal manera que fue posible ilustrar claramente los hábitos de consumo del escenario de prueba mediante diagramas de barra segmentados en días de la semana y franjas horarias como se puede ver en la ***Figura 110***, ***Figura 112***, y ***Figura 111***.
- Los dispositivos IoT son enchufes electrónicos que controlan y monitorizan la energía de equipos electrónicos conectados (televisor, horno microonda, y computadora de escritorio). La potencia eléctrica consuma por los enchufes sin carga como se observa en la **Tabla 26**, oscila entre 1,34W y 1,88W. Sin embargo, cuando se conecta un equipo electrónico apagado en los enchufes, su valor promedio aumenta a 4,3W (ver **Tabla 27**), porque a pesar de que

el equipo se encuentra apagado, internamente fluyen corrientes parásitas, y corrientes sobre elementos pasivos como indicadores led que consumen energía constantemente.

- La confiabilidad de la información permite definir que tanto podemos creer en los datos que proporciona los dispositivos IoT, para ello se realizó una comparativa de mediciones de corriente con multímetro y los datos visualizados de corriente en el frontend. Los resultados de la **Tabla 27** concluyen que el enchufe inteligente correspondiente a la computadora se encuentra averiado o mal calibrado, debido a que en la **Tabla 28** se observa un error relativo elevado de 88%, mientras que los demás enchufes presentan un error inferior al 10%. Sin embargo, esto no quiere decir que las gráficas de hábitos de consumo son erróneas, debido a que es posible diferenciar entre estado de encendido y apagado, únicamente los parámetros eléctricos en tiempo real presentan un valor diferente. Además, se observa en la **Tabla 28** como el flujo de corriente en el rango de *mA* presenta un error relativo de alrededor de 15,4%, mientras que el flujo de corriente en el rango de *A* presenta un error de 6,4%. Por tal motivo, se concluye que los enchufes tienen mejor exactitud cuando la corriente que fluye a través de sus circuitos es en el rango de los *A*.
- La disponibilidad de la información en el sistema de gestión de energía es clave durante la transmisión de datos en tiempo real, la cual debe estar disponible cuando el usuario la necesite. Para ello se realizó el muestreo de los parámetros de consumo eléctrico en el frontend (ver **Tabla 29**). Los resultados finales de la **Tabla 30** evidencian que en promedio los parámetros de consumo eléctrico en tiempo real se cargan en la página web cada 1s con una desviación estándar de 0.2s. Estos valores son aceptables para aplicaciones en tiempo real, y está dentro del límite de muestreo de la API utilizada el cual se definió a 1s. Además, durante el muestreo no existió perdida de datos que afecten la disponibilidad del sistema.

- El código fuente del servidor local y frontend desarrollado en el sistema de gestión de energía fue sometido a pruebas de calidad de software para localizar defectos y comprobar el funcionamiento de módulos, programas, objetos, clases, etc. Las pruebas realizadas con SonarQube al inicio detectaron varios bug y vulnerabilidades debido a librerías o variables sin uso, falta de etiquetas en el aplicativo web, entre otros; el resumen correspondiente se encuentra en la **Tabla 34**. Los errores fueron corregidos uno por uno con la finalidad de reducir al mínimo posible los defectos y mejorar la calidad del código. Sin embargo, la única vulnerabilidad que presenta actualmente el frontend es el pop-up generado cuando se accede erróneamente al aplicativo web, o cuando no coinciden las contraseñas durante el registro de un nuevo usuario al sistema.
- Las pruebas de carga del sistema de gestión de energía miden cuánto tarda el frontend en realizar diversas tareas y funciones bajo condiciones normales o predefinidas, para evaluar el rendimiento del sistema. Las pruebas de JMeter realizadas al frontend miden el tiempo de respuesta durante la navegación por la página web y al realizar consultas a la base de datos en RDS, cuando son sometidas a valores de carga de 100 y 6000 respectivamente. Las gráficas de rendimiento de la ***Figura 142*** y ***Figura 144*** muestran que el tiempo promedio de respuesta durante la navegación es 189ms, valor aceptable considerando que la visualización de datos en tiempo real se realiza cada 1s, mientras que el tiempo de consulta en la base de datos es de 9812ms, cuyo valor demuestra que AWS RDS tardaría al menos 10s en responder una solicitud y no sería una opción a considerar para flujo de datos en tiempo real.

6.2. Recomendación

- Mecanismos de seguridad como AWS IAM, Cognito, y Secret Manager se recomienda y fueron considerados en el diseño del sistema de gestión de energía para resguardar y proteger la información del usuario. AWS IAM mediante usuario y contraseña, define reglas o políticas de acceso a los servicios AWS. AWS Cognito permite la creación de usuarios en el frontend mediante claves de verificación, y brinda la posibilidad de vincular los usuarios con plataformas como Facebook, Yahoo o Google, entre otras. AWS Secret Manager es un gestor de credenciales para AWS RDS, que almacena información necesaria para acceder a la DB como instancia, nombre, endpoint, y contraseña (ver ***Figura 44***), y además, permite el cambio automático de la contraseña de acceso como se puede observar en la ***Figura 45***.
- Políticas de seguridad de acceso remoto a bases de datos partiendo por definición de rango de IPs y puertos se sugiere y fueron aplicadas en el diseño del sistema de gestión de energía para restringir el acceso a host no autorizados con la finalidad de agregar una capa extra de protección. Las políticas son definidas en AWS VPC como se observa en la ***Figura 43***, y limitan el acceso a instancias en EC2 como por ejemplo las bases de datos de AWS RDS.
- El flujo de datos en sistema en tiempo real considera una segura administración de los datos, y uso de lenguaje de consulta ligeros con sintaxis intuitiva y flexible, se sugiere y fue considerado en el proyecto el uso de AWS Appsync para solicitar, modificar y suscribirse a los datos mediante una única solicitud de red. Además, la aplicación de métodos de autenticación como AWS Cognito para la integración hacia el frontend, y AWS IAM para acceder a través de las librerías de desarrollo.

- Un sistema de archivos estructurado y jerárquico se sugiere mantener desde el inicio del proyecto no solo a nivel investigativo sino también durante el desarrollo. En particular, se recomienda mantener en orden los archivos almacenados en AWS S3 debido a que es el almacén de archivos base para cualquier desarrollo de frontend y backend en la AWS.

6.3. Trabajos futuros

- Diseñar un hardware para monitorizar el consumo eléctrico en hogares o fabricas capaz medir niveles altos de corriente. El hardware puede ser desarrollado mediante microcontroladoras con módulo de conectividad inalámbrica Wireless o Bluetooth integrado, cuya arquitectura conste de nodos distribuidos a través de un escenario de prueba conectados hacia un nodo central para integración hacia la plataforma AWS.
- Implementar y comparar la arquitectura propuesta bajo la plataforma AWS y la plataforma Google Cloud. La comparación busca medir el rendimiento, e integración de los servicios del sistema de gestión de energía bajo diferentes plataformas, además de considerar que ambas plataformas son privadas, por lo tanto, se podría realizar gráficas de costo beneficio para cada uno de los sistemas.
- Implementar la arquitectura propuesta en los Departamentos de Eléctrica y Electrónica de la Universidad de las Fuerzas Armadas – ESPE mediante el uso de infraestructura local y herramientas de código abierto para la transmisión de datos en tiempo real mediante una arquitectura REST, y aprendizaje de hábitos de consumo en oficinas de prueba mediante programas como Scikit-Learn, Shogun, Hadoop, Spark, entre otros.

- Diseñar e implementar una nueva propuesta de arquitectura capaz de interactuar directamente con los dispositivos IoT sin necesidad de mantener un servidor local activo. La comunicación debe permitir el flujo en tiempo real de los parámetros de consumo, y el envío de peticiones HTTP a la plataforma cloud del fabricante. Además, considerar la posibilidad de establecer un modo de control automático donde a partir del modelo de machine learning, encender o apagar los dispositivos automáticamente.

REFERENCIAS

Agencia de Regulación y Contro de Electricidad. (1 de February de 2018). *Estadística del Sector Eléctrico*. Obtenido de http://www.regulacionelectrica.gob.ec/estadistica-del-sector-electrico/

Agencia de Regulación y Control de Electricidad. (December de 2015). *Plan Maestro de Electrificación 2012-2021*. Recuperado el 30 de January de 2018, de http://www.regulacionelectrica.gob.ec/wp-content/uploads/downloads/2015/12/PME-2012-2021.pdf

Aimacaña, S., & Sango, W. (2015). *Diseño e implementación de un sistema de monitoreo de recursos energéticos primarios en el campus "Gral. Guillermo Rodríguez Lara" de la Universidad de las Fuerzas Armadas - ESPE.* Latacunga: Universidad de las Fuerzas Armadas ESPE.

Akhil, G. (2015). *A Survey of 5G Network: Architecture and Emerging Technologies* . IEEE .

Amazon Web Services . (December de 2018). *Overview of Amazon Web Services.* Obtenido de Amazon Web Services : https://d1.awsstatic.com/whitepapers/aws-overview.pdf

Amazon Web Services. (Febrero de 2016). *Diseño de arquitecturas para la nube: Prácticas recomendadas de AWS* . Obtenido de https://d1.awsstatic.com/whitepapers/es_ES/AWS_Cloud_Best_Practices.pdf

Amazon Web Services. (2 de Agosto de 2016). *Amazon Machine Learning*. Obtenido de https://docs.aws.amazon.com/machine-learning/latest/dg/history.html

Amazon Web Services. (13 de Julio de 2018). *AWS Key Management Service* . Obtenido de https://docs.aws.amazon.com/es_es/kms/latest/developerguide/dochistory.html

Amazon Web Services. (20 de Julio de 2018). *AWS SDK for JavaScript* . Obtenido de https://docs.aws.amazon.com/es_es/sdk-for-javascript/v2/developer-guide/welcome.html

Amazon Web Services. (2018). *Informática en la nube con Amazon Web Services*. Obtenido de https://aws.amazon.com/es/what-is-aws/

Ashton, K. (2009). That 'Internet of Things' Thing. *RFID Journal*, 1.

Ávila, O. (19 de May de 2011). Computación en la nube. *ContactoS*, págs. 45-82.

Bocchio, F. (2013). *ESTUDIO COMPARATIVO DE PLATAFORMAS CLOUD COMPUTING PARA ARQUITECTURAS SOA* . Buenos Aires: Universidad Tecnológica Nacional.

Chamba, C. (2015). *Diseño e implementación de un sistema de visualización y monitoreo móvil del consumo eléctrico a través de una red Zigbee con módulo XBEE.* Sangolquí: Universidad de las Fuerzas Armadas ESPE.

Chandi, L., & Roldán, G. (2015). *IMPLEMENTACIÓN DE UN APLICATIVO WEB COMO SERVICIO SAAS, BAJO UNA INFRAESTRUCTURA EN LA NUBE IAAS, PARA LA COOPERATIVA SAN VICENTE DEL SUR-MATRIZ.* Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

CISCO. (s.f.). *Securing the Internet of Things: A Proposed Framework*. Obtenido de https://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html

CloudTech. (6 de September de 2016). *AWS vs. Azure: IT pros weigh the pros and cons*. Recuperado el 6 de February de 2018, de https://www.cloudcomputing-news.net/news/2016/sep/06/aws-vs-azure-it-pros-weigh-pros-and-cons/

Diego, M. (2016). *Diseño e implementación de modelo a escala de una SmartHome con una red local doméstica (HAN) para monitorizar tres variables ambientales y transmitir las*

mediciones a un usuario remoto empleando protocolo IP. Bogotá: UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS .

Dumont, A. (17 de Diciembre de 2018). *Github*. Obtenido de https://github.com/adumont/tplink-cloud-api

Egas, J. (2014). *DESPLIEGUE DE UNA NUBE PRIVADA CON MODELO DE SERVICIO IAAS; Y ANÁLISIS E IMPLEMENTACIÓN DE UN GESTOR DE VERSIONES PARA FARMAENLACE CIA.LTDA.* Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

Estévez, D. (2005). *Monitoreo, Control y Adquisición de Datos de Energía Eléctrica mediante Internet.* Sangolquí: ESPE.

Fernandez, M., & Guerrero, C. (2016). *DECISION CLOUD: SISTEMA DE GESTIÓN DE SERVICIO AL CLIENTE E INTEGRACIÓN DE APLICACIONES, BASADO EN LA ARQUITECTURA ORIENTADA A MICROSERVICIOS, PARA DECISIÓN C.A.* Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

Flores, K. (2017). *Implementación de un sistema de monitoreo de energía y análisis de problemas eléctricos que afectan a la calidad de energía en la empresa sociedad de Destilación de Alcoholes S.A.* Sangolquí: Universidad de las Fuerzas Armadas ESPE.

Freire, W. (2008). *Diseño y construcción de un módulo de monitoreo y control del suministro de energía eléctrica a un data center a través de la internet.* Quito: Escuela Politécnica Nacional.

Godoy, P. (2017). *Un primer enfoque para el reconocimiento de lenguaje de señas basado en un guante inteligente que utiliza técnicas de Machine Learning.* Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

Gonzalez, D., & Verdugo, A. (2018). *Diseño e implementación de una arquitectura IoT para robótica colaborativa.* Sangolquí.

Hernández, R. (02 de Diciembre de 2016). *12 herramientas imprescindibles para asegurar la calidad del software (y sus alternativas)*. Obtenido de https://www.genbeta.com/desarrollo/12-herramientas-imprescindibles-para-asegurar-la-calidad-del-software-y-sus-alternativas

Herrera, L., & Salguero, M. (2010). *Implementación de un sistema de monitoreo con internet para una central de medida de parámetros eléctricos.* Sangolquí: ESPE.

Herrera, V. (2013). *Descripción de Redes Inteligentes (Smart Grids) y su aplicación en los sistemas de distribución eléctrica.* Quito: Escuela Politécnica Nacional.

Instituto Nacional de Estadísticas y Censos. (25 de March de 2018). *Estadísticas*. Obtenido de http://www.ecuadorencifras.gob.ec/estadisticas/

Linares, P. (2009). EFICIENCIA ENERGÉTICA Y MEDIO AMBIENTE. *Economía y Medio Ambiente ICE*, 75-92.

Matija, N., & Andrej, S. (19 de September de 2013). Concept of SmartHome and SmartGrids integration. *Energy (IYCE), 2013 4th International Youth Conference on* , 1-5.

Mera, J. (2016). Análisis del proceso de pruebas de calidad de software. *Ingeniería Solidaria*, 14.

Microsoft Azure. (s.f.). *Guía de la arquitectura de aplicaciones en Azure*. Obtenido de https://docs.microsoft.com/es-es/azure/architecture/guide/

Ministerio de Electricidad y Energía Renovable. (2018). *Redes Inteligentes y Generación Distribuida.* Recuperado el 1 de February de 2018, de http://www.iner.gob.ec/wp-content/uploads/downloads/2013/07/07_Generación-solar-distribuida-y-redes-inteligentes_PE.pdf

Morales, D. (2011). *Diseño e implementación de un sistema de monitoreo mediante telemedición del consumo de energía eléctrica de clientes especiales, de la Empresa Eléctrica Ambato Regional Centro Norte S.A.* Quito: Escuela Politécnica Nacional.

Morejón, J. (2015). *Análisis del desempeño de algoritmos de detección de eventos vulcanológicos basados en Machine Learning.* Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

PiAustralia. (s.f.). *Raspberry Pi 3 Model B*. Obtenido de https://raspberry.piaustralia.com.au/raspberry-pi-3-model-b

Redcentric. (s.f.). *What is Infrastructure as a Service (IaaS)?* Obtenido de https://www.redcentricplc.com/resources/articles/what-is-iaas/

REDES ZONE. (s.f.). *TP-Link*. Obtenido de https://www.redeszone.net/tp-link/

Relica, Y. P. (2014). ANÁLISIS Y EVALUACIÓN DE PARÁMETROS DE CALIDAD EN SERVICIOS CLOUD COMPUTING EN EL ECUADOR . *Departamento de Eléctrica y Electrónica, Universidad de las Fuerzas Armadas - ESPE*, 9.

Rodriguez, T. (5 de April de 2017). *Machine Learning y Deep Learning: cómo entender las claves del presente y futuro de la inteligencia artificial*. (Xataka) Recuperado el 1 de February de 2018, de https://www.xataka.com/robotica-e-ia/machine-learning-y-deep-learning-como-entender-las-claves-del-presente-y-futuro-de-la-inteligencia-artificial

Routing the World. (4 de Marzo de 2012). *Nube pública, nube privada, nube híbrida y nube súper-híbrida*. Obtenido de https://routingtheworld.wordpress.com/spanish-content/nube-publica-nube-privada-nube-hibrida-y-nube-super-hibrida/

Ruiz, C., & Tello, I. (2017). *Mejoramiento del proceso de asignación de turnos en emergencia con triage del Hospital de Especialidades de las Fuerzas Armadas Nº1 mediante la*

automatización y el uso de Machine Learning. Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

Sánchez, T. (2014). *DISEÑO DE UN MODELO DE PLANIFICACIÓN PARA IMPLEMENTAR ARQUITECTURA SOA EN EMPRESAS DE TELECOMUNICACIONES BASADOS EN LA GUIA DE PMBOOK CUARTA EDICIÓN*. Sangolquí: Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE.

Seal, P. (1 de Diciembre de 2018). *GitHub*. (TP-Link Smarthome WiFi API (formerly hs100-api)) Recuperado el 20 de February de 2018, de https://github.com/plasticrake/tplink-smarthome-api

SONARQUBE. (2019). *The leading product for*. Obtenido de https://www.sonarqube.org/

The Apache Software Fundation. (2019). *Apache JMeter™*. Obtenido de https://jmeter.apache.org/index.html

tp-link. (s.f.). *Enchufe Inteligente Wi-Fi con Monitorización de Energía* . (tp-link) Recuperado el 5 de February de 2018, de http://www.tp-link.com/es/products/details/cat-5258_HS110.html#specifications

TP-Link. (s.f.). *Enchufe Inteligente Wi-Fi con Monitorización de Energía* . Obtenido de https://www.tp-link.com/es/products/details/cat-5258_HS110.html

Universidad Tecnológica de Chile INACAP. (s.f.). *Oportunidades de negocio para internet de las cosas*. (MIRIADAX) Recuperado el 5 de February de 2018, de https://miriadax.net/web/oportunidades-de-negocio-para-internet-de-las-cosas/reto?p_p_id=activityViewer_WAR_liferaylmsportlet&p_p_lifecycle=0&p_p_state=normal&p_p_mode=view&p_p_col_id=column-1&p_p_col_count=1&p_r_p_564233524_actId=128002&p_r_p_564233524_m

Valdiosera, A. (2013). *Diseño de un medidor inteligente e implementación de sistema de comunicación bidireccional.* México D.F. : Instituto Politécnico Nacional.

Zapata, C., & Cardona, C. (2011). Feature comparison among some software tools for load testing. *Revista Avances en Sistemas e Informática*, 12.