



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Sistema IoT para Agricultura de Precisión basado en Infraestructura como Código (IaC)

AUTOR: Jorge de Diego Rubio

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR: Vicente Hernández Díaz

DEPARTAMENTO: Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Pedro García del Pino

TUTOR: Vicente Hernández Díaz

SECRETARIO: Lourdes López Santidrián

Fecha de lectura: 21/07/2020

Calificación:

El Secretario,

Resumen

Actualmente la tecnología está presente en prácticamente cualquier escenario, actividad, negocio, situación, etc. Se ha modernizado la mayor parte de los trabajos y actividades productivas de la sociedad. En esta transformación hacia el mundo digital se ha comenzado a incluir a una de las actividades más antiguas del ser humano: la agricultura.

La inclusión de la tecnología en la agricultura facilita una gran cantidad de tareas llevadas a cabo en las tierras de cultivo, sin embargo, aún es un campo con un gran camino a recorrer. El auge de la computación en la nube y los dispositivos IoT ha permitido nuevas formas de aplicar la tecnología a la agricultura. Es así como nace la agricultura de precisión.

Gracias a la agricultura de precisión, es posible controlar una plantación y actuar sobre ella de la manera más eficiente posible, lo que lleva a un mejor desarrollo de dicha actividad. En este proyecto se propone una forma de trabajo y unas tecnologías y herramientas para ello que incluyen el uso de la nube pública, uso de dispositivos IoT y uso de infraestructura como código para así poder desarrollar un sistema completo que dé solución a la necesidad de monitorización de una tierra de cultivo. Se ha desarrollado pensando tanto en su escalabilidad como en su integración con otras tecnologías y en el precio total del sistema desplegado.

Este sistema servirá de base para establecer una monitorización completa sobre variables como la presión atmosférica, la temperatura y humedad del aire, la humedad del terreno, etc. que mediante su posterior análisis sea posible realizar actuaciones automatizadas sobre los campos de cultivo, lo que supondrá una gran mejora en la eficiencia de la agricultura actual.

Abstract

Nowadays technology is present in practically any scenario, activity, business, situation, etc. Most of the work and productive activities of society have been modernized. Agriculture, which is one of the oldest activities of the human being has begun to be included in this transformation towards the digital world.

The inclusion of technology in agriculture facilitates a great number of tasks carried out on farmland, yet it is still a field with a long way to go. The rise of cloud computing and IoT devices has enabled new ways of applying technology to agriculture. This is how smart farming was born.

Thanks to smart farming, it is possible to control a cropland and perform actions on it in the most efficient way, which leads to a better development of this activity. This project proposes a way of acting and working, making use of different technologies and tools for this purpose that include the use of the public cloud, IoT devices and infrastructure as code in order to be able to develop a complete system that provides a solution for the need of monitoring a farmland. It has been developed thinking about both scalability and integration with other technologies and the total price of the deployed system.

This system will act as a base to establish a complete monitoring on variables such as atmospheric pressure, air temperature, air humidity, soil moisture, etc. that through its analysis will be possible to carry out automated actions on the crop fields, which will represent a great improvement in the efficiency of the today's agriculture.

Índice de Contenidos

1. Introducción	6
2. Antecedentes y Marco Tecnológico	8
2.1. Agricultura de Precisión	8
2.2. Internet de las Cosas (IoT)	9
2.3. Computación en la Nube	11
2.4. Hardware de dispositivos conectados a IoT	13
2.5. Protocolos de comunicación de dispositivos conectados a IoT	14
2.6. Ejemplos de Agricultura de Precisión	15
2.7. Infraestructura como Código	16
3. Especificaciones y restricciones de diseño	17
3.1. Descripción de las tecnologías y servicios necesarios	18
3.1.1. Amazon Web Services (AWS)	18
3.1.2. Terraform	21
3.1.3. Python y AWS SDK para Python (boto3)	22
4. Descripción de la solución propuesta	22
4.1. Solución propuesta	22
4.2. Desarrollo de la solución	26
4.2.1. Configuración inicial. Usuario terraform y credenciales de AWS	26
4.2.2. Código de la infraestructura	27
4.2.3. Código del simulador	29
4.2.4. Código auxiliar	30
4.2.5. Modelado de la base de datos (Elasticsearch)	30
4.3. Despliegue de la solución	31
5. Pruebas realizadas	36
5.1. Tiempo de despliegue y destrucción de la infraestructura	36
5.2. Comunicación hacia los dispositivos	37
5.3. Representación de los datos	38
5.4. Prueba de concepto	39
6. Presupuesto	41
7. Conclusiones y trabajos futuros	43
Referencias	45

Lista de Acrónimos

A continuación, se incluye una tabla con la lista de acrónimos que serán utilizados a lo largo del desarrollo de este documento.

Nº	Acrónimo	Significado
1	AWS	Amazon Web Services
2	IaC	<i>Infrastructure as Code</i> o Infraestructura como Código
3	IoT	<i>Internet of Things</i> o Internet de las Cosas
4	GCP	Google Cloud Platform
5	IaaS	<i>Infrastructure as a Service</i>
6	PaaS	<i>Platform as a Service</i>
7	SaaS	<i>Software as a Service</i>
8	EC2	<i>Elastic Compute Cloud</i>
9	CLI	<i>Command Line Interface</i>
10	MVP	<i>Minimum Viable Product</i>
11	NB-IoT	<i>Narrow Band IoT</i>

Tabla 1: Lista de acrónimos

1. Introducción

En este trabajo de investigación, se va a tratar de proponer una solución al siguiente problema:

Necesidad de un sistema económicamente viable para llevar a cabo las diferentes tareas requeridas para una agricultura de precisión eficiente tanto en recursos como en presupuesto y que a su vez permita su utilización de manera sencilla por los operarios, los cuales no necesariamente dispondrán de un perfil técnico que conozca las diferentes tecnologías empleadas en el sistema.

Para dar una solución a esta problemática se va a tratar de integrar el paradigma *IoT* junto con la computación en la nube o *Cloud Computing* en el dominio de aplicación de la agricultura. También se van a estudiar posibles soluciones para los elementos hardware de la solución, aunque no es el objetivo principal de este trabajo.

Como objetivo o meta principal se va a tratar de dar una solución basada en computación en la nube que encaje con el paradigma *IoT*. Se diseñará una infraestructura base que soporte el sistema, así como una arquitectura software que sirva para procesar y almacenar los datos recogidos de los dispositivos *IoT*.

Antes de poder presentar una solución que dé respuesta a la necesidad planteada, es necesario realizar un estudio sobre la actualidad de las tecnologías que se pretenden usar. Es decir, estudiar el estado del arte e investigar sobre ejemplos reales que implementen dichas tecnologías. Este paso será presentado en el siguiente apartado de este proyecto.

Es importante recalcar que el objetivo de este trabajo reside en el diseño y la implementación de una solución básica, que represente un *Minimum Viable Product (MVP)* o producto viable mínimo para la parte del sistema encargada de recoger e indexar los datos de los dispositivos *IoT*. Es por ello que al final del trabajo se expondrán posibles líneas de investigación o trabajos futuros, así como una descripción general de lo que debería ser un sistema completo y final.

A continuación, en el capítulo 2. Antecedentes y Marco Tecnológico se va a proceder a explicar todo el contexto en el cual se basará este trabajo de investigación y desarrollo. Se tratarán todos los aspectos tecnológicos relacionados y se describirán algunos ejemplos que soporten la idea que se pretende conseguir con el *MVP* mencionado.

Tras analizar el marco tecnológico, se detallarán en el capítulo 3. Especificaciones y restricciones de diseño las tecnologías que se van a utilizar en el proyecto, otorgando un razonamiento a cada una de las elecciones que se harán para el desarrollo.

Una vez descritas las tecnologías a utilizar, se explicará la solución propuesta en el capítulo 4. Descripción de la solución propuesta. En ese capítulo se mostrará con todo detalle el diseño, desarrollo y despliegue de la solución.

Tras el despliegue de la solución propuesta, se realizarán sobre el sistema diferentes pruebas documentadas en el capítulo 5. Pruebas realizadas, que demuestren que el *MVP* funciona como se espera. Tras estas pruebas, se detallará en el capítulo 6. Presupuesto un desglose del presupuesto de la infraestructura desplegada.

Finalmente en el capítulo 7. Conclusiones y trabajos futuros, se analizará en qué lugar se encuentra la idea inicial y se sentarán diferentes bases para establecer una línea de continuidad sobre la misma.

2. Antecedentes y Marco Tecnológico

Para el desarrollo de una solución que responda al problema anteriormente planteado en el apartado de Introducción, se ha realizado una investigación del marco actual en el que se encuentran diversas tecnologías y paradigmas para el momento del desarrollo de este trabajo.

Tras un tiempo de investigación, se ha decidido dividir este apartado en la descripción del estado actual de las siguientes tecnologías, paradigmas o soluciones que han resultado interesantes para el caso de uso para el cual se va a desarrollar una parte de la solución:

1. Agricultura de Precisión o *Smart Farming*
2. Internet de las Cosas o *Internet of Things (IoT)*
3. Computación en la Nube o *Cloud Computing*
4. Hardware de dispositivos conectados a IoT
5. Protocolos de comunicación de dispositivos conectados a IoT

Adicionalmente se van a presentar algunos ejemplos reales de implementación de soluciones para el desarrollo de la agricultura de precisión utilizando IoT.

2.1. Agricultura de Precisión

La agricultura de precisión es el proceso mediante el cual se monitorizan las posibles variables que puedan existir en el proceso de cultivo de un campo agrícola (como por ejemplo la humedad del aire, la temperatura del suelo, etc.) para poder llevar un control sobre estas variables y también poder decidir una futura actuación en función de dichas métricas.

La agricultura de precisión se puede dividir en cuatro fases distintas que estarán activas durante todo el proceso de cultivo y una fase más que se realizará de manera periódica (basado en [1]):

1. Recogida de datos: en esta fase los dispositivos elegidos para la monitorización de las diferentes métricas, variables o elementos del campo de cultivo se encargan de recoger datos. Es la primera fase y de la que nace todo el proceso de la agricultura de precisión. Es la fase esencial ya que la precisión de la recogida de datos es la que va a marcar la calidad final del sistema. Si se tiene una precisión demasiado baja esta no va a poder solucionarse con el resto del sistema a implementar, ya que hará de cuello de botella para la calidad final de la solución. Es por esto que es imprescindible contar con un equipo hardware suficientemente preciso para más adelante poder realizar el resto de las fases de una manera eficiente y que realmente aporte un valor a la agricultura actual.
2. Almacenamiento y pre procesamiento de los datos: una vez recogidos y preprocesados (si fuese necesario) los datos que miden los dispositivos que actúan como sensores, es necesario disponer de un sistema de almacenamiento dedicado a guardar todos estos datos. Este punto es crucial para el futuro desarrollo del sistema, ya que elegir un método de almacenamiento poco eficiente, caro o no adecuado para el caso de uso, va a condicionar las fases de análisis. Es necesario ajustar este almacenamiento de los datos teniendo en cuenta el tipo de datos a

recoger, la forma de indexado que se va a utilizar y el tamaño de cada unidad de datos que se defina. En el capítulo dedicado a la solución propuesta, se argumentará el almacenamiento escogido teniendo en cuenta, sobre todo, el posible escalado que pueda sufrir.

3. **Análisis de los datos:** la fase de análisis de datos es la que va a decidir las actuaciones y la que va a permitir aportar el valor añadido a la agricultura. En esta fase de análisis es crucial entender las necesidades que pueda tener el agricultor, el ingeniero agrónomo o cualquier persona implicada en la explotación agrícola y el trabajo que realicen. Es necesario comprender qué aspectos son esenciales para una agricultura eficiente, en qué es necesario enfocar los esfuerzos del sistema y cómo se detectan posibles problemas mediante estos análisis de la información obtenida de las medidas de los sensores.
4. **Toma de decisiones:** esta fase se ejecuta justo después y utiliza las salidas o conclusiones de la fase anterior para automatizar procesos llevados a cabo en la agricultura. En esta fase la lógica implementada debe ser mínima, ya que el grueso del procesado de datos se encuentra en la anterior. Al igual que en la anterior fase, es necesario conocer las necesidades de los distintos actores implicados en la explotación agrícola, y los procesos que llevan a cabo dependiendo de su experiencia y observaciones. El objetivo es conseguir la máxima automatización posible, por lo tanto, es imprescindible entender los procesos de la agricultura y plantear soluciones que mejoren dichos procesos.
5. **Evaluación del rendimiento:** al contrario que las cuatro primeras fases que se encuentran activas el 100 % del tiempo, esta fase se debe realizar de manera periódica para estudiar cómo de buena es la solución, qué problemas mejora y cuáles no, y qué remedios hay que estudiar en cada momento para que realmente el sistema implementado no sólo automatice las labores llevadas a cabo de manera manual, sino que maximice los resultados de la agricultura. En esta fase es importante contar de nuevo con la ayuda y opinión de las personas encargadas de trabajar los campos de cultivo, que aportarán una visión diferente a la visión técnica del sistema y adaptada a la agricultura en sí misma y las labores que realizan.

Para este trabajo, se va a hacer hincapié en las tres primeras fases, teniendo en cuenta que para la primera fase únicamente se realizarán simulaciones de los dispositivos. Es objetivo de este trabajo el presentar una solución que dé respuesta a la problemática que presentan las fases dos y tres, con especial foco en la segunda fase. Una vez presentada e implementada la solución, se elaborará una lista de pasos a seguir con ideas de implementación de las siguientes fases.

No es objetivo de este trabajo la problemática que presenta la gestión de los dispositivos hardware: energía limitada, ancho de banda limitado, adaptación a las condiciones atmosféricas y meteorológicas, tipos de sensores, etc., pero si se van a exponer ejemplos de sensores y tecnologías que podrían encajar en la primera fase de la agricultura de precisión aquí mencionada.

2.2. Internet de las Cosas (IoT)

Internet de las Cosas es un paradigma que se basa en la interconexión de dispositivos u objetos de uso cotidiano mediante Internet [2]. Esto permite interactuar con dichos objetos de una manera completamente distinta para la que fueron pensados en un primer momento, permitiendo aportar

valor añadido a la función de estos objetos. Este paradigma permite el control remoto, la monitorización, la automatización de tareas, la identificación, etc. de diferentes aspectos relacionados con estos objetos.

Internet de las Cosas es un concepto extremadamente amplio, por lo tanto, se va a hacer foco en cómo va a ser una parte imprescindible para la solución que se quiere plantear, y cómo se utiliza actualmente para aplicaciones similares.

En resumen, este paradigma encaja a la perfección en el caso de uso que aplica: monitorización de campos de cultivo. Mediante dispositivos o sensores conectados a través de Internet, va a ser posible implementar dicha monitorización de manera remota, lo que va a mejorar la eficiencia de la agricultura al permitir cubrir amplios rangos de tierras de cultivo sin tener que emplear tiempo y recursos en el desplazamiento físico del operario o agricultor hacia dichas tierras.

Internet de las Cosas es la pieza central del puzzle que representa el sistema completo dedicado a la agricultura de precisión. Es el medio por el cual se pasa del mundo analógico de los datos de una tierra a la parte digital que almacena y procesa esos datos.

Para este trabajo es necesario centrarse en aquellos dispositivos pertenecientes al paradigma de Internet de las Cosas que han sido concebidos para consumir un mínimo de recursos y energía, ya que el hardware desplegado deberá residir en los propios campos de cultivo, con poco o ningún acceso a fuentes de energía externa. Este problema puede subdividirse en tres escenarios diferentes:

1. Los dispositivos tendrán acceso a fuentes de energía: este escenario es prácticamente imposible que se presente en un campo de cultivo abierto. Sí podría ser una opción en un campo de cultivo cerrado, como un invernadero.
2. Los dispositivos no tendrán acceso a fuentes de energía pero se podrá tener un *Gateway* central cercano que sí tiene acceso a fuentes de energía: en este escenario los dispositivos no van a enviar información directamente a Internet, sino que la enviarán a un dispositivo común a todos ellos que es el que se encargará de retransmitir esa información hacia Internet. Este escenario sería ideal para un campo de cultivo abierto, ya que permitiría que los dispositivos sensores y actuadores ahorrasen la energía necesaria para poder enviar a Internet, pudiendo enviar la información a un dispositivo cercano, a un rango de distancia menor. Este escenario posibilita el uso de protocolos pensados únicamente para este caso de uso como 6LowPan [3] o Zigbee [4].
3. Los dispositivos no tendrán acceso a fuentes de energía y no dispondrán de un *Gateway* cercano al que enviarle información en un rango de distancia corto: este escenario es el más complejo en cuanto a la gestión de energía de los dispositivos. Estos deberán enviar directamente a Internet la información recogida o utilizar un protocolo de mayor alcance para transmitir hacia el *Gateway*, como LoRa [5], lo que supondrá un gasto extra de energía. En esta clase de escenario es importante conocer tecnologías diseñadas para trabajar en estas condiciones. Además de LoRa, mencionada anteriormente, NB-IoT [6] y LTE-M [7] conforman dos tecnologías de comunicación que serán de gran utilidad en la parte hardware y de comunicaciones de un sistema planteado dentro de este escenario.

No es objetivo de este trabajo dar una solución a la problemática del hardware para este escenario, pero es necesario conocer las opciones y alternativas que existen actualmente para ello. Esta parte de la investigación soporta la viabilidad de este trabajo y de líneas de investigación futuras posteriores al desarrollo de este caso de uso.

Sin embargo, sí es objetivo de este trabajo proponer una solución compatible con cualquiera de los escenarios aquí presentados. El objetivo es proponer una solución altamente compatible para cualquier tipo de escenario desplegado, para así poder adaptarse a las diferentes problemáticas específicas de cada campo de cultivo en el que pueda implantarse esta solución.

Más adelante en este mismo apartado, se estudiarán algunas opciones o alternativas para los elementos del sistema aquí mencionados: el hardware de dispositivos conectados *IoT* y los protocolos de comunicación necesarios.

2.3. Computación en la Nube

La computación en la nube es un paradigma o modelo que se basa en ofrecer servicios de computación que pueden ser accedidos de manera ubicua y bajo demanda [8]. Estos servicios pueden componerse de diferentes elementos informáticos configurables, como por ejemplo redes, servidores, almacenamiento, servicios, etc. Estos recursos deben de poder crearse y destruirse de manera rápida con la menor gestión posible por parte del usuario final.

Dentro del modelo de computación en la nube, existen diferentes formas de ofrecer dichos servicios al usuario final, mediante diferentes modelos [9]:

- *IaaS* (*Infrastructure as a Service* o infraestructura como servicio): este modelo ofrece alternativas de infraestructura basadas en *Cloud*. Dentro de este modelo se encuentran los servidores y las máquinas virtuales, servicios de almacenamiento de ficheros u objetos, elementos de red como cortafuegos o tablas de ruta, etc.
- *PaaS* (*Platform as a Service* o plataforma como servicio): mediante este modelo los proveedores *Cloud* ofrecen una plataforma normalmente orientada al desarrollo, en la cual todas las dependencias o herramientas que pueda necesitar dicho desarrollo estén preinstaladas y preparadas para su uso, lo que permite ahorrar tiempo y dinero que se dedicaría a las instalaciones y configuraciones de dichas herramientas o dependencias. La característica principal de este modelo es que puede ser utilizado sin necesidad de poseer conocimiento técnico en la administración de sistemas.
- *SaaS* (*Software as a Service* o Software como servicio): este modelo ofrece un servicio situado capas por encima de los anteriores mencionados. Se caracteriza por ofrecer un software disponible para su uso de manera inmediata, que ofrece un servicio o responde a una necesidad específica. Dentro de este modelo se encuentran los proveedores de email, proveedores de almacenamiento en nube, aplicaciones de gestión de usuarios, etc.

Comparar ventajas y desventajas de un modelo frente a otro no tendría sentido alguno, ya que cada modelo responde a una necesidad diferente, y es el caso de uso específico en cada momento lo que determinará qué modelo deberá ser usado teniendo en cuenta restricciones, costes, operativa, etc.

Por otro lado, es necesario mencionar que este tipo de computación trae consigo innumerables ventajas, sin embargo, hay que tener en cuenta los aspectos negativos de este modelo. Es imprescindible entender que la computación en la nube no es un modelo que resulte ser el más eficiente en todos los aspectos para todos los casos de uso. Es importante también, adaptar la manera de realizar los diseños y las arquitecturas al modo de funcionar de la computación en la nube. Algunas características de la computación en la nube que pueden ser a su vez ventajas y desventajas, dependiendo de la forma en la que este modelo se utilice, son:

- Costes: siempre que los diseños y las implementaciones estén realizadas de manera eficiente, los costes de la infraestructura en la nube serán menores que los costes de los servidores físicos. La nube permite generar infraestructura con un coste muy bajo con respecto a lo que supondría comprar los recursos hardware para realizar el mismo despliegue *on-premises* (en las instalaciones físicas del usuario). Sin embargo, si a modo de ejemplo, se desea migrar una infraestructura que está desplegada *on-premises* directamente a los servidores de algún proveedor de computación en la nube, existe una gran probabilidad de que los costes sean incluso mayores a los que se tendrían en ese momento sin migrar los sistemas. Por tanto, es completamente imprescindible, si se quiere aprovechar al máximo las ventajas respecto a los costes que ofrece la computación en la nube, adaptar los sistemas que se quieran desplegar, diseñar y/o migrar a la forma de operar que tiene dicho paradigma. Es necesario estudiar primero los diferentes proveedores *Cloud*, para después de haber elegido uno de ellos, estudiar los diferentes servicios que ofrece y sus precios asociados para así poder determinar cuál va a ser la mejor solución, tanto en rendimiento como en costes.
- Escalabilidad: el hecho de no tener que adquirir físicamente infraestructura, convierte a la computación en la nube el mejor modelo para ofrecer una escalabilidad que se adapte a cualquier momento o entorno. Una gran ventaja de la computación en la nube es que es posible crear y destruir infraestructura, escalar (tanto hacia arriba como hacia abajo y tanto en vertical como en horizontal) dicha infraestructura libremente y automatizar este escalado para que se adapte en cada momento a la demanda que le exijan. En las infraestructuras *on-premises*, es necesario conocer de antemano cual va a ser la demanda de tráfico hacia ese sistema, y cuáles pueden ser sus posibles picos, para no dejar de ofrecer servicio en ningún momento. Fallar en estas estimaciones puede suponer grandes pérdidas de dinero invertido en recursos que no soporten dichos picos de tráfico, y además, puede derivar en la necesidad de realizar una nueva compra de recursos. Sin embargo, en el modelo de computación en la nube, esto no ocurrirá debido a la naturaleza de los recursos que se utilizan. Estos recursos pueden crearse, destruirse y escalarse en todo momento para responder eficientemente a cualquier demanda en cualquier momento.
- Seguridad: existen opiniones diversas sobre la seguridad en la nube. Este aspecto es decisivo para una gran cantidad de compañías que se niegan a adoptar el modelo de computación en la nube pensando que no es un entorno seguro. Sin embargo, la seguridad en la nube necesita de una gestión similar a la seguridad de los sistemas *on-premises*. Es necesario crear y proveer una seguridad en todo tipo de entornos y modelos. El hecho de utilizar la nube no significa que un sistema esté más o menos seguro, ya que esto va a depender de la forma en la que dicha seguridad sea implementada y gestionada. Por otra parte, existe el inconveniente de que utilizar los servicios en nube lleva consigo implícita una necesidad de confianza hacia el proveedor de dichos servicios. Es por esto que deben estudiarse a fondo los diferentes proveedores y elegir aquellos que generen una gran confianza tanto en seguridad como en

fiabilidad del servicio. Otro aspecto a tener en cuenta sobre la seguridad es la privacidad de los datos y la información. Es necesario utilizar los servicios de aquellos proveedores que ofrezcan absoluta confidencialidad de los datos que se trasladen a su infraestructura. Este aspecto puede medirse investigando qué certificaciones de seguridad (ISO 27001, cumplimiento con HIPAA, etc.) acreditan al proveedor. Es necesario investigar cada proveedor *Cloud* para conocer si sus servicios encajan o no con el caso de uso que se quiera aplicar.

- *Vendor Lock-in*: quizás sea uno de los contras más importantes en el modelo de computación en la nube. Utilizar los servicios de un proveedor *Cloud* exige establecer un compromiso con dicho proveedor. Esto puede generar problemas si en el futuro quiere migrarse a otro proveedor distinto o quiere dejar de utilizarse el modelo de computación en nube. Por este motivo es importante realizar buenas estimaciones sobre costes, estudiar en profundidad qué ofrece cada proveedor y cómo puede adaptarse a las necesidades en cada momento. Si este aspecto es un gran inconveniente, se pueden adoptar estrategias como por ejemplo el uso de nube híbrida (parte del sistema en nube pública y parte en nube privada u *on-premises*) o el uso de estrategias *Multi-Cloud* (uso de diferentes proveedores *Cloud*), teniendo en cuenta que estas estrategias aumentan exponencialmente los costes.
- Aspectos técnicos: es necesario tener en cuenta también distintas características del uso de la computación en la nube como por ejemplo la necesidad de contar con una conexión a Internet en todo momento para poder acceder a dichos servicios. En la actualidad, esto no debería suponer un problema para prácticamente todo tipo de escenario o caso de uso, pero es un aspecto que también debe tenerse en cuenta. Además, es imprescindible tener en cuenta que, al no existir acceso físico a los servidores o elementos del sistema, no es posible pararlos cortando su fuente de energía, o no pueden realizarse reparaciones de forma física sobre ellos.

Para el caso de uso que se va a estudiar en este trabajo, únicamente se planteará la utilización de un proveedor de nube pública (es decir, los servidores o servicios utilizados serán compartidos con el resto de los clientes del proveedor, aislados únicamente de manera lógica unos de otros y no de manera física).

Otro aspecto interesante de la computación en la nube es que (dependiendo del proveedor) la infraestructura puede ser gestionada utilizando el modelo de infraestructura como código, del cual se hablará más adelante en este trabajo, en el apartado Descripción de la Solución Propuesta. En ese mismo apartado se tratarán más a fondo diferentes temas aquí expuestos, adaptados al proveedor elegido para el despliegue del sistema.

2.4. Hardware de dispositivos conectados a IoT

Para la realización de este proyecto, como ya se ha mencionado anteriormente, el desarrollo se va a enfocar en el diseño de la infraestructura *Cloud* que soporte el sistema. Sin embargo, es necesario conocer que existen soluciones hardware actualmente en el mercado que dan respuesta a la parte física del sistema, en la que se desplegarían los diferentes sensores y/o actuadores IoT.

Existe una gran cantidad de fabricantes y dispositivos que podrían usarse para un sistema como el que se presenta en este proyecto, sin embargo, se han recopilado aquí dos tipos de dispositivos de

dos fabricantes diferentes con una amplia cuota de mercado, que pueden dar respuesta al sistema de forma distinta:

- Raspberry Pi Zero W y similares [10]: Raspberry Pi es un ordenador en miniatura, montado sobre una placa base única que posee una serie de pines de entrada y salida, aparte de conexiones USB y HDMI. Posee una gran comunidad como soporte que ayuda a los desarrolladores a compartir sus soluciones y resolver los errores que encuentren en el desarrollo. La Raspberry Pi es capaz incluso de ejecutar un sistema operativo (Raspbian) con un entorno gráfico. Existen numerosas alternativas a Raspberry con la misma meta: una placa de propósito general.
- Dispositivos de Libelium [11]: Libelium es un fabricante especializado en IoT. Sus dispositivos están mucho más orientados a propósitos bien definidos. Es por ello que, para un sistema en fase de producción, dispositivos como estos son la mejor alternativa. Están diseñados y preparados para una función específica.

Aparte de las mencionadas, existe una gran cantidad de soluciones de hardware para IoT de diferentes fabricantes que tienen diferentes propósitos a la hora de fabricar sus dispositivos, aunque las descritas anteriormente son las que cuentan con mayor aceptación. En un proceso de desarrollo de un proyecto como este, es importante conocer todo tipo de dispositivos, ya que cada uno puede ser útil en distintas fases del desarrollo e investigación. Por ello, una posibilidad es conseguir un dispositivo de propósito general, asequible, para poder realizar diferentes pruebas de concepto y comprobar que la infraestructura (en este caso en nube pública) que soporta el sistema completo está diseñada correctamente y se adecúa a los requerimientos. Una vez bien definido el sistema e implementada la parte independiente del hardware, sería interesante tener a disposición dispositivos reales de producción, para poder realizar los diferentes desarrollos a nivel software que estos requieran, así como pruebas de conectividad y gasto energético.

En la última parte de este proyecto, en el apartado en el cual se expongan posible trabajos futuros y próximos pasos, se detallará un ejemplo de posible desarrollo futuro que involucre un dispositivo hardware real y no simulado. Para el desarrollo de este proyecto se pretende únicamente simular estos dispositivos con un pequeño desarrollo software que ayude a realizar los tests necesarios para comprobar el funcionamiento de la infraestructura desplegada.

2.5. Protocolos de comunicación de dispositivos conectados a IoT

De nuevo es necesario mencionar que, aunque este proyecto no esté enfocado a la implementación del hardware IoT que realice las mediciones y actuaciones, hay que tener en cuenta que el proyecto pueda ser implementado de manera completa en un entorno real. Es por ello por lo que en este apartado se van a tratar diferentes protocolos de comunicación enfocados a IoT y al bajo consumo que pueden ser de aplicación para este proyecto.

A continuación se describen diferentes protocolos y/o tecnologías de comunicación que son importantes dentro del paradigma de Internet de las Cosas:

- LoRa: es una tecnología inalámbrica con alta tolerancia a las interferencias, largo alcance y bajo consumo. Está gestionada actualmente por la LoRa Alliance. El protocolo que implementa dicha tecnología es LoRaWAN [5] [12].

- SigFox: es una solución para conectar dispositivos IoT. Es una solución propietaria y su precio se basa en el número de dispositivos conectados [13].
- WiMax: IEEE 802.16, es un estándar de comunicación inalámbrica definido para redes de área metropolitana(MAN). [14]
- Zigbee: IEEE 802.15.4 es un protocolo utilizado para redes de área local de bajo consumo orientado a dispositivos que funcionan mediante batería [4].
- LTE-M: pertenece a la tecnología *Low Power Wide Area*. Las redes LTE-M (*Long Term Evolution for Machines*) pueden coexistir con las redes actuales 2G, 3G y 4G permitiendo mayor interoperabilidad y compatibilidad, pudiendo reutilizar la base de LTE y con un gasto energético significativamente optimizado [7].
- NB-IoT: pertenece a la misma familia de tecnologías que LTE-M aunque se comporta mejor cuando las comunicaciones no requieren una tasa de refresco alta, lo que hace que sea una buena elección para el presente trabajo [15] [6].
- MQTT: es un protocolo de conectividad máquina a máquina diseñado para implementar un modelo de publicador/suscriptor. [16]

Los aquí expuestos son solo un pequeño porcentaje de las alternativas que existen en la actualidad. En una posible línea de investigación futura, sería necesario realizar un estudio mucho más extenso sobre estos protocolos, así como diferentes pruebas de calidad, conectividad y consumo energético utilizando diversos protocolos y tecnologías. En este proyecto, únicamente se exponen algunos de ellos para tener una idea general de las alternativas, pero no se profundiza más porque ello podría componer un nuevo trabajo de la misma envergadura que este. Es una de las líneas de investigación futuras más disruptivas con la forma de realizar este proyecto, basado al 100 % en implementación de una solución para la parte software.

2.6. Ejemplos de Agricultura de Precisión

A continuación se van a exponer ejemplos reales de agricultura de precisión que muestran casos de éxito o implementaciones reales de la utilización de la tecnología IoT en la agricultura.

El primero de los ejemplos viene de la mano de Agrotech y utiliza los dispositivos Libelium [17]. En este caso de uso se ha desplegado un sistema de monitorización sobre un viñedo basado en IoT haciendo uso de dispositivos Libelium y la nube pública Microsoft Azure. Tiene como fin proveer de gran cantidad de datos para su posterior análisis y ayudar al sector de la agricultura en su digitalización.

Otro ejemplo es el documentado en [18]. En este artículo, se ha implementado una pequeña granja conectada y se ha desarrollado una aplicación móvil que consulta los datos accediendo a una API REST. Se trata de un escenario real que incluye todo lo necesario para realizar una monitorización de una tierra de cultivo, aunque se ha realizado en un entorno cerrado y controlado.

El último ejemplo que se presenta es uno de los mencionados en [19] y que utiliza la solución Sensing4Farming de Vodafone. En este caso se ha implantado un sistema de monitorización basado en sensores que utilizan *NarrowBand IoT* [20]. Se ha utilizado para un viñedo mediante una alianza con Emilio Moro [21].

Como se ha podido comprobar, hay multitud de formas de implementar un sistema de monitorización para tierras de cultivo utilizando IoT. Aquí únicamente se han presentado tres ejemplos de estas implementaciones, que son los más parecidos al sistema a desarrollar en este trabajo. La mayoría de los sistemas se centran en medidas como la temperatura, la humedad del terreno, la presión atmosférica, etc. La recopilación de estas medidas en diferentes partes de la tierra de cultivo permitiría una actuación diferenciada por cada una de estas partes, dependiendo de la necesidad de las mismas en cuanto a riego, tratamiento de las plantas, etc.

2.7. Infraestructura como Código

Por último, para finalizar este capítulo, se detallarán las características de la base software de este proyecto, la Infraestructura como Código o *IaC*.

En los ejemplos descritos anteriormente se ha podido comprobar cómo uniendo la tecnología IoT junto con la agricultura se han podido desplegar y probar sistemas que conforman un ecosistema de agricultura de precisión. Sin embargo, todos estos ejemplos son puntuales, desarrollados para la prueba o diseñados *ad-hoc* para un caso de uso concreto. La diferencia principal que se va a presentar en este trabajo es hacer uso de *IaC* para desarrollar y desplegar la infraestructura que soporta un sistema de estas características.

La Infraestructura como Código es una tecnología que permite poder tratar una infraestructura de la misma manera que se puede tratar un programa software. Esto es, mediante código, versionable y reutilizable. Poder tratar la infraestructura como cualquier otro software aumentará la rapidez y eficiencia en los despliegues de la misma, reducirá la posibilidad de errores al automatizar tareas y ayudará a entender una infraestructura simplemente mirando el código que la define [22][23][24].

Hacer uso de esta tecnología evita los problemas de falta de documentación sobre los recursos que existen desplegados en un determinado escenario. El código por sí mismo sirve como documentación. Permite además poder adaptar la infraestructura a las necesidades en cada momento de manera rápida y sencilla. Es además la mejor forma de gestionar los servicios que ofrece un proveedor de nube pública, pudiendo aprovechar al máximo las ventajas mencionadas anteriormente en este mismo capítulo. Además, es una gran ayuda a la hora de no olvidar destruir recursos accidentalmente que generen gasto y repercuta negativamente en el coste.

Utilizar esta tecnología va a permitir replicar y adaptar el sistema a innumerables situaciones o escenarios que se planteen en diferentes campos de cultivo con diferentes necesidades. Es la utilización de *IaC* lo que va a permitir que sin demasiado esfuerzo tanto de trabajo como económico se pueda desplegar un sistema que soporte un escenario real de agricultura de precisión.

3. Especificaciones y restricciones de diseño

Para la realización de este proyecto se persigue dar respuesta a los siguientes requisitos generales:

1. Económicamente viable: se propondrá una solución que sea económicamente viable, cuyo presupuesto y gasto estimado se adapten a la magnitud del sistema a desplegar para cada caso de uso. Es por ello por lo que el sistema deberá de poder ajustarse en presupuesto a cualquier rango de gasto establecido, cumpliendo una serie de requisitos de rendimiento mínimos. Es por ello que se realizará un pequeño análisis del presupuesto en uno de los apartados de este proyecto.
2. Completamente escalable: derivado del punto anterior, el sistema debe de ser definido de tal manera que escale de manera automática, o al menos, que permita este escalado, para cualquier tipo de caso de uso independientemente de la magnitud del sistema y la cantidad de datos enviados y/o recibidos y procesados. De esta forma el sistema podrá ser utilizado tanto con fuentes de datos pequeñas, como por ejemplo una única tierra de cultivo, como con grandes cantidades de nodos de envío de datos, como podría ser una empresa de producción y cultivo de tierras en masa.
3. Adaptable y compatible a todo tipo de casos de uso similares: en este proyecto se describirá un caso de uso muy bien definido y cerrado, lo que no tiene que significar que la solución o el sistema definido no sea capaz de adaptarse a otros casos de uso similares pero diferentes de base. Se propondrá un sistema que sea capaz de adaptarse a cualquier situación de monitorización basada en IoT similar al caso de uso de este proyecto. Para ello se tratará de implementar el sistema de la manera más compatible posible con cualquier forma de enviar los datos a la nube. Se perseguirá el uso de *endpoints* a modo de API que permitan la comunicación desde cualquier punto con acceso a Internet. De esta forma, será posible la integración con cualquier fabricante de hardware y no se cerrará la solución a un único tipo de dispositivos o fabricantes.
4. Reutilizable: utilizar infraestructura como código hará que el sistema no solo sea más sencillo de actualizar y mantener, sino que será completamente reutilizable. Esto permitirá poder desarrollar el sistema una vez, pero desplegarlo las veces que se requiera, lo que supondrá un ahorro de costes una vez que el sistema esté definido. Además, la infraestructura como código permite versionarla y actualizarla de una manera muy simple y rápida sin ningún coste adicional al de los recursos de computación en *Cloud* que se utilicen. Adicionalmente, cualquier desarrollo que se realice para este proyecto será distribuido de manera libre utilizando una licencia *Open Source* [25].
5. Producto final de fácil utilización por todo tipo de perfiles: para que el sistema sea aceptado por parte de los clientes finales potenciales, deberá ser intuitivo y fácil de utilizar, que no requiera de mantenimiento por parte de dicho cliente final. Por su naturaleza, el sistema será utilizado por personas encargadas de labrar y mantener tierras de cultivo. El sistema tiene que adaptarse a la utilización por parte de cualquier tipo de perfil, sea o no técnico, y no debe suponer un tiempo extra de mantenimiento para los clientes del producto. El producto final deberá de adaptarse a los clientes y no al contrario. Ya que este proyecto se enfoca en la infraestructura base, no se realizarán desarrollos de ningún tipo de interfaz que dé respuesta a esta restricción y/o especificación. Este desarrollo se marcará como trabajos futuros al de este proyecto.

Aparte de las restricciones anteriormente mencionadas, también se tendrá en cuenta que:

1. Para el desarrollo de este proyecto y la implementación del *MVP* indicado en el capítulo de introducción, se pretende mantener a coste 0 la infraestructura que se despliegue para dar soporte al sistema.
2. Una de las intenciones principales es mantener el coste lo más bajo posible incluso para un sistema en producción real. Para ello, como se ha mencionado anteriormente, se realizará un estudio sobre el presupuesto de la infraestructura propuesta para el sistema.

Por ello, el *MVP* implementado tendrá una funcionalidad mínima pero nunca podrá ser utilizado en un sistema en producción conservando las mismas especificaciones técnicas y de rendimiento, ya que será implementado para realizar una pequeña prueba de concepto que muestre el funcionamiento de la infraestructura con una cantidad de datos transferidos ínfima que nunca podrá dar solución a una problemática real de una tierra de cultivo. Sin embargo, sí se detallarán diferentes especificaciones técnicas de la infraestructura que deberán de modificarse para que el sistema pueda adaptarse correctamente a distintas necesidades de entornos productivos.

Por todo esto, se han estudiado las tres nubes públicas principales que existen actualmente: AWS, Azure y GCP. Tras una breve investigación sobre ellas, se ha llegado a la conclusión de utilizar AWS debido a que la capa gratuita de 1 año que ofrece posee los elementos clave que se necesitan para este sistema. Además, al ser la nube con mayor uso actualmente, posee también una comunidad de usuarios mayor que ayudará a la realización de este proyecto. Sin embargo, el hecho de implementar el sistema utilizando infraestructura como código junto con una herramienta que se agnóstica del proveedor *Cloud* a utilizar permitirá migrar de una nube a otra de manera más sencilla, aunque no de manera trivial. Por el contrario, al elegir una única nube se va a generar un *vendor lock-in* con dicha nube. Es por ello por lo que, como trabajos futuros a este mismo proyecto, se propondrá el estudio del mismo sistema, pero basado en las otras dos nubes mencionadas, Microsoft Azure y GCP.

3.1. Descripción de las tecnologías y servicios necesarios

Antes de empezar a exponer la solución propuesta, es necesario describir las diferentes tecnologías y herramientas que se pretenden utilizar para dicha solución. Para ello se van a describir tanto AWS como los diferentes servicios necesarios que posee, así como la herramienta Terraform de *IaC*, de la empresa Hashicorp. Para los diferentes desarrollos de código se va a utilizar el lenguaje Python, al ser el más utilizado para los desarrollos que tengan que comunicarse con AWS mediante el uso de su API y librerías. Asimismo, se realizarán pequeñas herramientas auxiliares escritas en Bash.

3.1.1. Amazon Web Services (AWS)

Para describir todos los servicios de AWS, la documentación de referencia ha sido la propia documentación oficial de AWS. [26]

- **Identity and Access Management (IAM)**

Bajo este servicio AWS ofrece la posibilidad de crear usuarios, roles y políticas. Estos usuarios son propios de AWS y sirven para modificar la infraestructura que se despliega en AWS pero no son usuarios del nivel de aplicación.

- **IAM Users, IAM Policies, IAM Roles**

Los IAM Users que se pueden crear pueden ser de dos tipos: usuarios normales de consola de AWS o usuario para acceso programático. Normalmente, el primer tipo se crea para personas que vayan a acceder a la consola web de AWS y el segundo tipo se utiliza para accesos mediante la CLI (Command Line Interface) de AWS o para agentes de despliegue como puede ser Terraform, el cual se explica más adelante en este apartado. Las IAM Policies sirven para administrar los diferentes tipos de permisos que se pueden asignar a un usuario, mientras que los IAM Role se utilizan para conceder permisos a servicios de AWS sobre otros servicios de AWS, o para estrategias multi cuenta en las que se conceden permisos entre cuentas. Los IAM Roles tienen asignadas IAM Policies. Existen numerosos servicios aparte de IAM en AWS para administrar usuarios y accesos, sin embargo, para el caso de uso que supone este proyecto, no serán necesarios servicios adicionales.

- **Networking**

El apartado de redes dentro de AWS se basa en redes y subredes privadas las cuales comienzan sin acceso a Internet. En este subapartado se expondrán los servicios más relevantes que aplican a este proyecto.

- **Virtual Private Cloud (VPC) y Subnets**

Las VPC conforman una red de tamaño máximo /16. Por lo tanto habrá un máximo 65534 posibles direcciones. Como ejemplo, si la red es 10.101.0.0/16, la IP más baja será 10.101.0.1 y la más alta 10.101.255.254, siendo 10.101.0.0 la dirección de la red y 10.101.255.255 la dirección de *broadcast*. Como puede observarse, al establecer la máscara de red en /16 los dos primeros dígitos de la dirección quedarán fijos y los dos últimos serán los que varíen generando las diferentes direcciones IP.

Estas VPC pueden dividir en subredes (Subnets) en las cuales se podrán colocar los diferentes recursos que se quieran desplegar.

- **Security Groups y Network Access Controls Lists (NACLs)**

Conforman los cortafuegos de AWS. Los Security Groups se asocian a instancias u otros servicios y contienen una política restrictiva por defecto. Deniegan el tráfico a todo lo que no esté contemplado en sus reglas. Por el contrario, los NACLs poseen una política permisiva por defecto y mediante sus reglas se puede denegar o permitir el tráfico de manera explícita a conveniencia.

- **Route Tables**

Las Route Tables son tablas de ruta normales sin ninguna peculiaridad específica. Por defecto las VPC contienen las rutas para poder intercomunicar los recursos desplegados en ellas.

- **Internet Gateway y NAT Gateway**

El Internet Gateway actúa como un punto de acceso que comunica las VPC con Internet. Sin desplegar este elemento, no existirá la conexión con Internet desde una VPC.

El NAT Gateway sirve para comunicar subredes privadas con las subredes públicas y para exponer servicios que se encuentren en subredes privadas, entre otros usos. Son el elemento que permite dar acceso a Internet a subredes privadas.

- **Storage**

En AWS, el almacenamiento se ofrece mediante diferentes servicios que dan respuesta a diferentes tipos de almacenamiento como puede ser almacenamiento de objetos, almacenamiento en bases de datos o almacenamiento a nivel de bloque.

- **Simple Storage Service (S3)**

S3 es el servicio que ofrece almacenamiento de objetos de AWS. Ofrece una gran disponibilidad y fiabilidad. Es posible utilizarlo para almacenar cualquier objeto de hasta 5TB de tamaño, y ofrece un tamaño total sin restricciones. Es posible utilizar este servicio para servir páginas web estáticas y como base de datos para logs de monitorización. Permite versionar los objetos, cifrarlos, controlar el acceso a los mismos, etc.

- **Elastic Block Storage (EBS)**

Este servicio ofrece almacenamiento a nivel de bloques utilizado para ser los volúmenes que tienen las instancias. Permite de manera sencilla hacer *snapshots* y restaurarlos, y es el servicio que ofrece almacenamiento persistente para las instancias. Sino se selecciona un volumen EBS al lanzar una instancia, los datos que almacene se perderán al apagar la instancia.

- **Bases de datos**

AWS ofrece una gran cantidad de bases de datos, tanto relacionales como no relacionales, así como servicios propios de AWS como Amazon Aurora. Ejemplos de estas bases de datos son RDS, Redshift, DynamoDB, etc.

- **Computing**

- **Elastic Compute Cloud (EC2)**

Es el servicio de computación de AWS que permite lanzar instancias con las distribuciones de Windows o Linux que se requiera. No solo engloba las instancias, sino que el servicio EC2 agrupa también otro tipo de recursos como los anteriormente mencionados Security Groups, balanceadores de carga, etc.

- **Lambda**

Este servicio ofrece una computación *serverless*. Permite ejecutar código simplemente subiendo dicho código a AWS y seleccionando el entorno de ejecución necesario (Python, NodeJS, Golang, etc.). Su facturación se basa en el uso y permite una gran versatilidad al poder ejecutar piezas de código sin tener que aprovisionar servidores para ello. Permite ejecutarse de multitud de formas como por ejemplo llamándola desde otro servicio de AWS, a través de una API, de manera manual, etc.

- **IoT Services**

AWS posee un servicio denominado IoT Core bajo el cual se agrupan todos los recursos necesarios para gestionar una flota de dispositivos IoT. Bajo el IoT Core se pueden registrar dispositivos que se identificarán en AWS mediante certificados.

- **Otros servicios**

- **Elasticsearch Service**

Este servicio ofrece un motor de Elasticsearch junto con Kibana como un servicio mediante el cual no es necesario mantener la computación que lo soporta. Esta clase de

servicios son especialmente útiles en pruebas de concepto como la que se va a presentar en este proyecto.

3.1.2. Terraform

Toda la información sobre Terraform presentada en este apartado tiene como fuente la página oficial del producto [27].

Terraform es una herramienta codificada en el lenguaje Golang que sirve para crear, desplegar y versionar infraestructura. Utilizando *IaC* es posible mantener una infraestructura mediante Terraform. Esta herramienta pertenece al fabricante Hashicorp y es una herramienta *Open Source*, gratuita y agnóstica del proveedor *Cloud* a utilizar.

Para AWS, Terraform no es más que un agente de despliegue representado como un IAM User destinado al acceso programático. Terraform es declarativo, lo que significa que la infraestructura que se despliegue será exactamente la que se indique en el código. Esto supone que, si en el código se crean 3 recursos del mismo servicio, siempre habrá 3 recursos, independientemente de las veces que se ejecute.

Terraform por sí mismo no puede crear recursos para AWS. Necesita de un *provider* que exponga los recursos de AWS y entienda las interacciones necesarias a realizar contra la API de AWS.

Terraform dispone de varios comandos básicos:

- **init**: mediante la ejecución de este comando, Terraform inicializa y descarga los recursos que necesite para la ejecución del código. En este paso Terraform descarga el *provider* (en este caso de AWS) y los recursos ubicados en las fuentes que se indiquen mediante el código (como, por ejemplo, el código de algún repositorio).
- **plan**: mediante este comando se ejecuta un plan que muestra las acciones que se van a realizar sobre la infraestructura.
- **apply**: este comando sirve para aplicar los cambios a la infraestructura. Al ejecutarle, Terraform realizará un plan y pedirá confirmación al usuario de si desea o no aplicar los cambios indicados en el plan.
- **destroy**: mediante este comando se puede destruir la infraestructura que figure en el archivo de estado de Terraform.

La infraestructura se gestiona mediante el archivo de estado de Terraform (*tfstate*). Esto significa que, si se realiza un cambio manual sobre una infraestructura desplegada mediante Terraform, al no estar reflejado en el *tfstate*, Terraform no detectará este cambio y pueden surgir errores. Lo ideal es que una infraestructura desplegada con Terraform se gestione única y exclusivamente desde la herramienta para no generar conflictos. Este archivo de estado se puede almacenar de manera local o de manera remota para compartirlo entre diferentes usuarios o no perderlo por un posible fallo en una máquina local.

Se ha decidido utilizar Terraform en este proyecto al ser la herramienta más extendida actualmente para este cometido. Además, utilizando Terraform el código puede ser compartido fácilmente, y la infraestructura puede ser modificada de manera rápida y sencilla. Terraform se instala como un único binario en cualquier dispositivo Windows, Linux o Mac. También es compatible con FreeBSD, OpenBSD y Solaris.

3.1.3. Python y AWS SDK para Python (boto3)

Como se ha mencionado al inicio de este apartado, para el resto de herramientas o aplicaciones necesarias para la realización del proyecto, se empleará el lenguaje Python. Elegir este lenguaje está motivado por la librería existente para trabajar con AWS: boto3 [28]. Esta librería representa el SDK para trabajar con AWS para Python. Es posible utilizar otros lenguajes como NodeJS o Java, pero normalmente resulta más sencillo utilizar Python ya que es el lenguaje que más se utiliza para trabajar contra la API de AWS.

Mediante el uso de los servicios y las herramientas aquí mencionados, se tratará de ofrecer diferentes respuestas a un sistema IoT sobre AWS. La idea principal es crear un sistema base que posea todos los elementos necesarios para un funcionamiento mínimo y ofrecer alternativas añadiendo o cambiando dichos elementos para crear sistemas más complejos que se adecúen mejor a diferentes casos de usos más definidos.

4. Descripción de la solución propuesta

A continuación se va a proceder a explicar la solución propuesta. Todo el código que se menciona se encuentra adjunto como anexo a este documento y también puede consultarse en el repositorio correspondiente de Github [25]

4.1. Solución propuesta

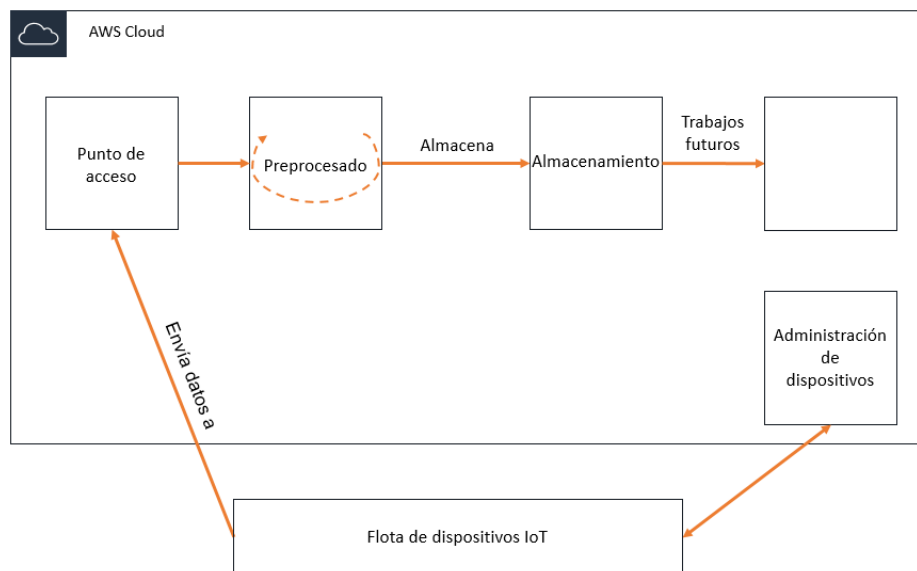


Figura 1: Esqueleto del escenario básico

Como puede observarse en la Figura 1, la idea principal es dar una solución para los elementos *Punto de acceso*, *Preprocesado* y *Almacenamiento*, que esté implementada de tal manera que sea compatible con cualquier elemento que se establezca como la flota de dispositivos IoT y *Trabajos futuros*. Asimismo, establecer un sistema de *Administración de dispositivos* con una comunicación bidireccional para trabajos de actualización o monitorización del propio dispositivo IoT.

Un ejemplo de implementación para la flota de dispositivos IoT sería lo que se puede apreciar en la siguiente imagen:

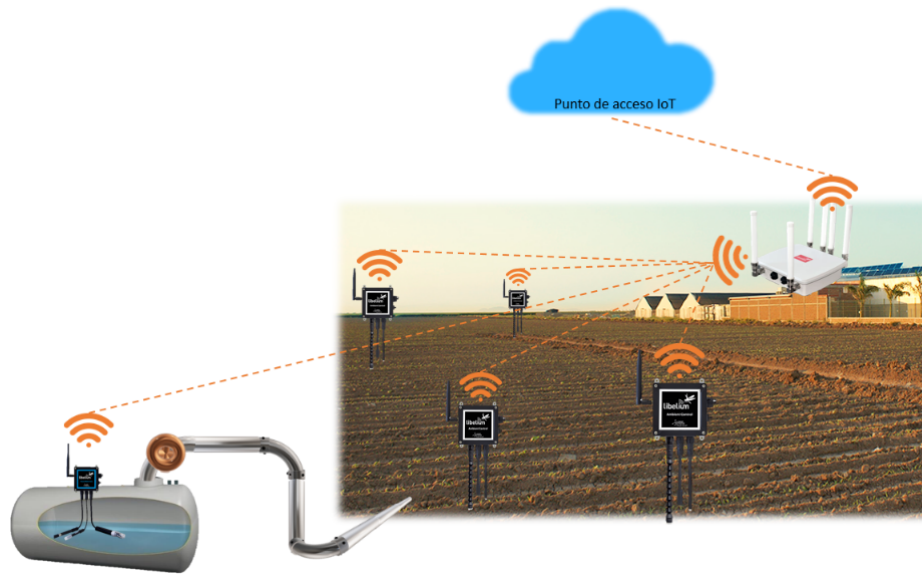


Figura 2: Ejemplo de flota de dispositivos IoT para una tierra de cultivo.

En la Figura 2, pueden verse diferentes dispositivos del fabricante Libelium. En concreto están ubicados como sensores diferentes modelos Plug & Sense! de Libelium: Ambient Control en la tierra y Smart Water en el depósito de agua. También, como punto centralizado de los sensores y punto de conexión con Internet, se encuentra en las naves instalado un dispositivo Meshlium Xtreme que actúa como IoT Gateway [11]. Como punto de entrada al sistema en *Cloud*, el Meshlium Xtreme comunicaría los datos recogidos al *Punto de acceso*, el cual correspondería con el mostrado en la Figura 1. Esto es simplemente un ejemplo teórico de una solución que sería compatible con el sistema que se pretende implementar con este proyecto.

Por lo tanto y como primer sistema se va a desarrollar el siguiente escenario, utilizando el IoT Core de AWS, el servicio de computación AWS Lambda y el Elasticsearch Service.

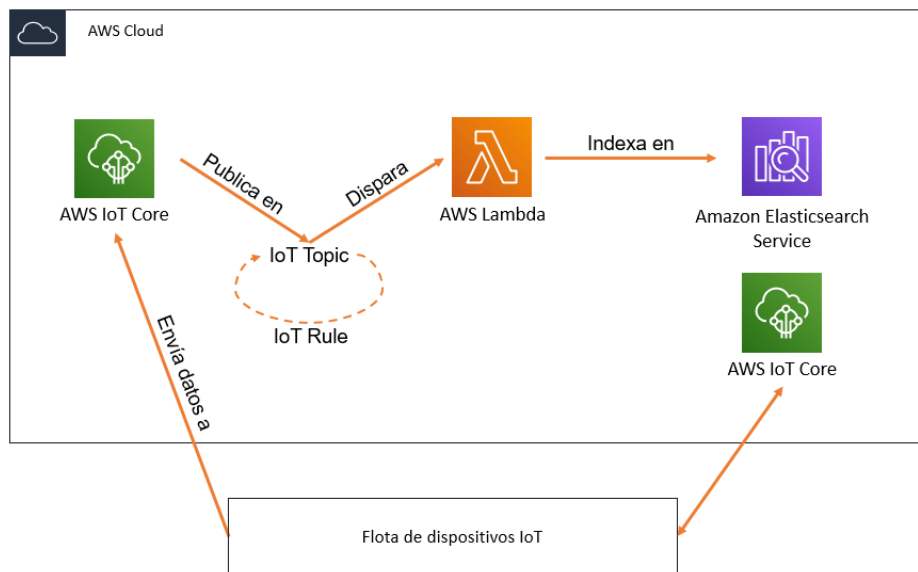


Figura 3: Ejemplo del escenario base del sistema.

En la Figura 3 se pueden ver los diferentes elementos del sistema a los que se les ha dado una respuesta utilizando diferentes servicios de AWS. A grandes rasgos, el funcionamiento del sistema es el siguiente:

1. La flota de dispositivos IoT envía información a un punto de acceso situado dentro del servicio IoT Core de AWS. En concreto, se enviará la información a un IoT Topic de AWS que actúa como punto de acceso MQTT[16]. Por tanto, el dispositivo que envíe datos hacia el Topic únicamente tendrá que implementar un cliente MQTT (más adelante se hablará sobre cómo se implementa de manera segura esta conexión).
2. Al recibir un mensaje en el IoT Topic, una IoT Rule del servicio IoT Core de AWS se encargará de realizar una acción con dicho mensaje. En este ejemplo, ese mensaje se envía como entrada a una función desplegada en el servicio AWS Lambda (más adelante se explicará la razón de utilizar una función como intermediario y no utilizar la acción por defecto que ofrece la IoT Rule para indexar en Elasticsearch Service).
3. Mediante la función Lambda, se realiza un preprocesado de los datos recibidos y se indexa en el servicio Elasticsearch Service, que actúa como base de datos.
4. Una vez almacenados los datos en la base de datos de Elasticsearch Service, se pueden lanzar las consultas que sean necesarias para obtener los datos que se requieran, o puede configurarse el motor de visualización que viene incluido por defecto en el servicio, Kibana, para representar los datos indexados como desee el usuario. Este último punto pertenece al elemento *Trabajos futuros* representado en el esqueleto del sistema de la Figura 1.

A continuación, se presenta un análisis detallado del sistema y de las razones de usar los servicios mencionados:

- En primer lugar, con respecto al IoT Topic del AWS IoT Core, se ha utilizado ya que se necesita únicamente un cliente de MQTT desde la parte hardware del sistema. Es un protocolo

altamente extendido y muy liviano, que encaja a la perfección en un sistema IoT de estas características. Por lo tanto, este IoT Topic representa una cola MQTT a la que se publica apuntando hacia un *endpoint*, que es global para toda la cuenta en AWS, y especificando el nombre asociado al IoT Topic concreto. Además, para que esta conexión sea segura, desde el IoT Core, se pueden registrar dispositivos identificados mediante certificados digitales. Para registrar un dispositivo, se generará un certificado mediante el cual se identificará dicho dispositivo para enviar los datos. De esta manera se realiza la comunicación de manera segura, y el *Punto de acceso* no estará habilitado a cualquier fuente, sino que, mediante una política de seguridad, únicamente se podrán enviar datos desde los dispositivos registrados.

- En segundo lugar, haciendo alusión al tercer punto del funcionamiento del sistema anteriormente presentado, se utiliza un intermediario entre el IoT Core y el Elasticsearch Service. Desde una IoT Rule, es posible indexar de manera directa en un Elasticsearch Service desplegado, pero se ha decidido introducir una función Lambda que haga de intermediario debido a la necesidad de compatibilidad con todo tipo de sistemas. De este modo, se produce un desacoplamiento entre ambas capas del sistema, lo que permitiría la adaptación a otros requerimientos modificando únicamente la función Lambda y no parte del servicio de IoT Core. Operacionalmente es mucho más sencillo redespigar una Lambda mediante Terraform, que modificar la configuración interna del IoT Core y de la IoT Rule. Además, una función Lambda permite un grado de libertad sobre el procesamiento e indexado de los datos mucho mayor que permitirá adecuarse mucho más a las especificaciones concretas del sistema. Además, si por ejemplo surgiese la necesidad de cambiar Elasticsearch Service por una instalación del Stack de Elastic sobre instancias EC2, obligaría a cambiar la configuración del IoT Core mientras que usando esta Lambda únicamente sería necesario cambiar el punto de acceso a Elasticsearch.
- Por último, este sistema está enteramente basado en la computación *serverless*. La computación *serverless* se basa en utilizar servicios gestionados por un proveedor para que el usuario no tenga que desplegar y administrar sistemas o servidores. Normalmente se ofrecen como servicios con un precio mayor al que supondría montar el mismo sistema utilizando máquinas virtuales o instancias, pero se eliminan los costes operacionales de dicha instalación [29]. Para un proyecto de este tipo, este tipo de computación ofrece la posibilidad de utilizar servicios difíciles de instalar y mantener, de manera sencilla por cualquier usuario no experimentado. Es por ello que para este proyecto se ha optado por seguir esta filosofía para la totalidad del proyecto, pudiendo intercambiar los diferentes elementos mencionados en la Figura 1 por componentes más tradicionales implementados sobre una máquina virtual o un servidor. Elasticsearch Service se ofrece como un servicio gestionado, dentro de la computación *serverless*. Utilizar un servicio gestionado permite no tener que administrar ningún tipo de servidor. Además, usar Elasticsearch permitirá en los trabajos futuros utilizar su propio motor de búsqueda para lanzar consultas a su base de datos, por lo que no será necesario un motor de búsqueda adicional. Esto sigue en la línea de buscar la mayor compatibilidad posible con otros elementos que se añadan al sistema. A modo de ejemplo, podría instalarse el Stack de Elastic sobre instancias EC2 y llevar a cabo la instalación de este en la duración de este proyecto, sin embargo, se ha considerado más interesante dedicar mayor cantidad de tiempo al sistema en general que a la instalación de sus diferentes elementos. Es por ello por lo que se ha optado por usar este tipo de computación *serverless*.

4.2. Desarrollo de la solución

4.2.1. Configuración inicial. Usuario terraform y credenciales de AWS

Una vez definida la solución, se va a desarrollar e implementar de manera completa. El primer paso es crear una cuenta en AWS. Al ser una cuenta nueva, permitirá usar la capa gratuita de AWS [30]. Esta capa gratuita incluye lo siguiente, entre otros muchos servicios adicionales:

- 750h al mes de Elasticsearch Service (instancia tipo t2.small.elasticsearch y una sola zona de disponibilidad) y 10Gb de almacenamiento EBS.
- 250K mensajes publicados o enviados al mes para IoT.
- 1M de peticiones al mes para AWS Lambda.

Por tanto, para realizar un *MVP* de la solución, encaja a la perfección.

Una vez creada la cuenta en AWS, el siguiente paso es crear un IAM User para Terraform, que tendrá como nombre terraform.

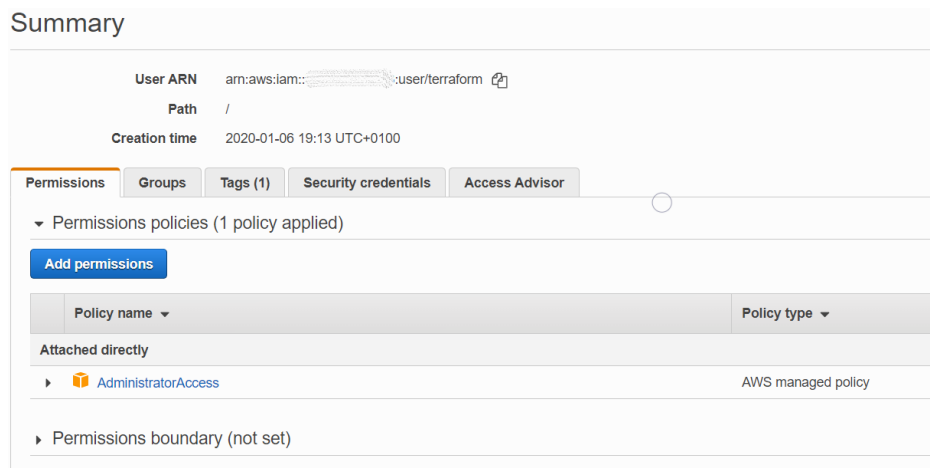


Figura 4: Política del usuario Terraform

Como puede observarse en la Figura 4, el usuario terraform necesita acceso a todos los servicios de AWS. Podría únicamente tener acceso a los servicios para los cuales se vaya a desplegar infraestructura, sin embargo, otorgarle política de administración evitará problemas cuando sea necesario interactuar con otros servicios no contemplados en su política.

Asimismo, en la siguiente imagen, la Figura 5, puede observarse que este usuario no cuenta con acceso a la consola web de AWS ya que únicamente necesita acceder de manera programática a través de la API de AWS:

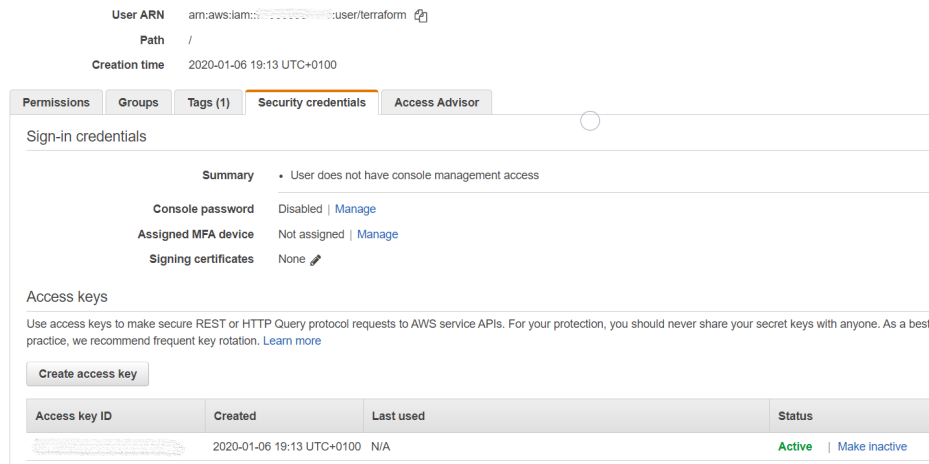


Figura 5: Security Credentials en el usuario Terraform

Como resultado de crear este usuario, AWS genera un *Access Key* y un *Secret Key* que serán las credenciales del usuario terraform, y que será necesario, de cualquiera de los métodos posibles, entregárselas al programa Terraform para realizar los despliegues de la infraestructura.

4.2.2. Código de la infraestructura

El siguiente paso es configurar con dichas credenciales el acceso a la consola de AWS. En este caso, se ha instalado la CLI de AWS [31] y se van a utilizar las credenciales por defecto de AWS pertenecientes en este caso al usuario terraform y configuradas mediante el comando *aws configure* en la máquina local desde la que se va a desplegar la infraestructura (censurados datos sensibles):

```
jorge@ThinkCentre:~$ cat .aws/credentials
[default]
aws_access_key_id = [Redacted]
aws_secret_access_key = [Redacted]
```

Figura 6: Captura del archivo .aws/credentials

Una vez realizada toda esta configuración, se ha creado un repositorio en Github que albergará el código utilizado para la realización de este proyecto[25]. Este código será entregado como anexo a este documento.

A continuación, se van a explicar partes del código que son interesantes de mencionar:

1. La configuración del *provider*, como se ha mencionado anteriormente, será AWS, se encuentra en las líneas de código que se muestran en la siguiente figura, dentro del archivo que tiene como nombre en el repositorio 00_main.tf:

```
provider "aws" {
  region = "eu-west-1"
}
```

Figura 7: Parte del código del fichero 00_main.tf del repositorio del proyecto

Como puede observarse en la Figura 7, el *provider* para AWS se ha configurado estableciendo la región (eu-west-1) que corresponde a la región de AWS de Irlanda. No es necesario aportar más configuración ya que el fichero de credenciales es el utilizado por defecto, al igual que el perfil de dicho archivo de credenciales.

2. La siguiente parte del código que es interesante mencionar es la referente al *backend*. Esta configuración se muestra en la siguiente imagen:

```
terraform {
  backend "s3" {
    shared_credentials_file = "~/.aws/credentials"
    profile                 = "personal"
    bucket                  = "iot-jdr-terraform-tfstate"
    key                     = "aws-iot-tfg.tfstate"
    workspace_key_prefix    = "workspaces"
    region                  = "eu-west-1"
    encrypt                  = true
  }
}
```

Figura 8: Configuración del backend en terraform

Esta configuración que se puede observar en la Figura 8 es referente al *tfstate* mencionado en el apartado en el cual se explica Terraform. Por defecto, el *tfstate* se guarda de manera local en el equipo desde el cual se lance la ejecución del comando *terraform apply*. Sin embargo, para este proyecto se ha decidido utilizar un *backend* remoto que permita guardar dicho *tfstate* en un Bucket S3. Esto permite realizar despliegues desde diferentes equipos recuperando dicho *tfstate*. En este caso se ha decidido guardar encriptado por razones de seguridad y se ha utilizado un *workspace* el cual permite tener diferentes *tfstate* para diferentes entornos de trabajo representados por el mismo código de Terraform.

3. Código de la infraestructura: el resto del código representa la infraestructura y está organizado en:
 - Archivos de configuración y variables fijas: 00_main.tf y 01_locals.tf: estos archivos configuran el entorno de Terraform y contienen variables que son fijas para todo el proyecto.
 - Archivos de infraestructura del sistema: 02_elasticsearch.tf y 03_lambda.tf: estos archivos contienen los recursos de infraestructura que son comunes a todo el sistema. Como bien indican los nombres de dichos archivos, los recursos son Elasticsearch Service y AWS

Lambda. Las funciones lambda se guardan en una carpeta separada llamada lambdas y están escritas en Python.

- Archivos de infraestructura para una sola tierra de cultivo: 10_poc_cropland.tf, 11_poc_cropland_variables.tf y 12_poc_cropland_outputs.tf: contienen la configuración para una tierra de cultivo. La idea es que puedan añadirse tantos archivos similares a estos como tierras de cultivo se desee monitorizar en el mismo sistema, distinguiéndolos por el nombre de la tierra el cual es en este caso poc. El primer archivo contiene los dispositivos IoT, las políticas y roles relacionados con ellos y las funciones lambda que son específicas para una tierra de cultivo. Es interesante mencionar que, para evitar la redundancia de código, se ha elaborado un módulo de terraform para un dispositivo IoT [32]. Este módulo se llama aws_iot_thing y se puede encontrar en el mismo repositorio.
- Archivos de variables y salida del sistema: 99_variables.tf y 99_outputs.tf: contienen las variables generales del sistema y la salida pertinente del mismo.
- Archivo de variables de entorno: aws-iot-tfg-dev.tfvars: contiene variables de un entorno. Esto está pensado teniendo en cuenta que el mismo código se usará en distintos entornos típicos (desarrollo, preproducción y producción). Esto facilita desarrollos en el sistema sin perder tiempo de servicio del mismo. Este archivo se le pasa a terraform antes de realizar un apply, mediante el uso de la opción -var-file. Cómo se lanza Terraform se explicará en el apartado despliegue de la solución.

4.2.3. Código del simulador

Se ha codificado un simulador para evitar tener que recurrir a dispositivos físicos que no solo aumentarían el presupuesto del *MVP* que se ha desarrollado en este proyecto, sino que implicaría tener que adquirir dichos dispositivos, configurarlos y testarlos antes de incluirlos en el *MVP*. El *MVP* desarrollado tiene como objetivo demostrar la viabilidad de los elementos *Punto de acceso*, *Preprocesado* y *Almacenamiento* de la Figura 1.

Este simulador se basa en un programa escrito en Python que implementa un cliente MQTT. Para ejecutarlo se ha decidido implementar en un contenedor Docker [33]. Docker es una tecnología de contenedores que permite virtualizar parte de un sistema operativo con objeto de ser más liviano que una máquina virtual pero que permita aislar por completo el entorno de ejecución de una aplicación. Esta tecnología no es objetivo de estudio en este proyecto por lo que no se va a profundizar en ella, sin embargo, se ha decidido ejecutar el simulador a través de ella ya que permite una portabilidad suficiente para poder ejecutarla en cualquier sitio que tenga instalado el motor Docker.

En resumen, el simulador implementa cuatro dispositivos ejecutando cuatro procesos en Python de manera simultánea, los cuales generan datos aleatorios de diferentes medidas relacionadas con una tierra de cultivo (temperatura ambiente, humedad, presión, etc.). En el caso del *MVP* se ha decidido implementar los dispositivos que harán la función de sensores ubicados en la tierra de cultivo virtual.

El código se compone de:

1. Un archivo llamado Dockerfile en el cual se encuentran las especificaciones del contenedor Docker a ejecutar.

2. Un archivo denominado `iot_thing_simulator.py` que implementa el simulador, junto a un archivo llamado `requirements.txt` que contiene las dependencias del simulador.
3. Un archivo `run.sh` que es el que ejecutará el contenedor Docker y contiene las instrucciones de ejecución de los cuatro procesos Python del simulador.

4.2.4. Código auxiliar

Con el fin de automatizar todo lo posible el despliegue de la solución, se ha decidido crear varios scripts que sirvan de ayuda para ello. Estos scripts son:

1. `launch_tfg.sh`: este script está escrito en Bash y se encarga de automatizar los comandos necesarios para el despliegue. Esto es:
 - Crea un *workspace* de Terraform y ejecuta el comando *terraform apply -auto-approve* que omite la confirmación del usuario antes de aplicar los cambios.
 - Ejecuta el script que se explica en el siguiente punto (`save_outputs_terraform.py`) para guardar las variables de salida de Terraform y poder usarlas tanto en el simulador como en el resto de herramientas que sean necesarias.
 - Recoge la IP de la instancia EC2 que se va a utilizar para ejecutar el simulador.
 - Copia mediante `scp` el código del simulador en la instancia. Ejecuta mediante `ssh` el comando *docker build* y el comando *docker run* para ejecutar el simulador.
2. `save_outputs_terraform.py`: este script recoge la salida de Terraform una vez desplegada la infraestructura. Guarda en una carpeta con nombre `config` los siguientes datos:
 - Certificados de los dispositivos IoT. Estos certificados se almacenan en el *tfstate* mencionado anteriormente. Dicho *tfstate* se encuentra cifrado dentro del Bucket S3 por lo que contiene la seguridad necesaria para el almacenaje de dichos certificados. Tal y como se describe en [34], se trata el *tfstate* como datos sensibles y por tanto se controla su acceso mediante políticas de IAM y cifrado desde el lado del servidor.
 - Punto de acceso del dominio de Elasticsearch: se guarda para poder acceder a la base de datos.
 - Punto de acceso IoT de AWS: se guarda para poder pasárselo como parámetro al simulador.
3. `destroy_tfg.sh`: este script simplemente ejecuta el comando *terraform destroy -auto-approve* para destruir toda la infraestructura creada sin tener que pedir confirmación al usuario.

4.2.5. Modelado de la base de datos (Elasticsearch)

Un aspecto importante del proyecto es el modelado de la base de datos. Para esta base de datos, como ya se ha indicado anteriormente, se ha elegido el motor Elasticsearch bajo el servicio *serverless* de AWS Elasticsearch Service.

Elasticsearch es un motor de búsqueda en tiempo real y un sistema distribuido de almacenamiento [35]. Si hubiera que identificar que tipo de base de datos es, se situaría dentro del grupo de bases de datos no relacionales (NoSQL) [36].

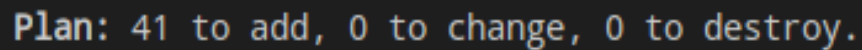
Elasticsearch almacena documentos en formato JSON. Estos documentos se almacenan dentro de un índice y cada uno de esos documentos tiene una serie de campos o valores. Dependiendo del tipo de campo (numérico, texto, etc.), Elasticsearch lo almacena de una manera u otra [37]. Esto es lo que permite que las búsquedas en Elasticsearch sean eficientes y rápidas.

Para este proyecto en particular, se ha decidido que cada una de las IoT Things que publiquen datos hacia AWS sean representadas por un índice en Elasticsearch. El nombre de este índice será `<campo-de-cultivo>-<IoT-Thing>` (A modo de ejemplo, los datos del sensor1 del campo de cultivo llamado poc serán indexados en un índice llamado poc-sensor1). De este modo las búsquedas serán más sencillas y cada índice contendrá los valores que proporcionen en cada momento los dispositivos. De este modo, si cada dispositivo es de un tipo diferente, no habrá una mezcla de campos en los documentos de cada índice, y todos los documentos de un índice tendrán los mismos campos.

4.3. Despliegue de la solución

Como se ha mencionado anteriormente, el despliegue de la solución se ha implementado a través de un script escrito en Bash (`launch_tfg.sh`). En este apartado serán descritos los resultados de los pasos que sigue el script.

1. El script está configurado para lanzar el comando `terraform apply -auto-approve`, lo que significa que no es necesaria una intervención manual que apruebe la aplicación de las acciones a realizar. Sin embargo, es interesante conocer qué muestra Terraform cuando se va a realizar el apply. Esto se consigue ejecutando `terraform plan`. A continuación, se muestran varias capturas del plan (el plan entero es demasiado extenso para documentarle en este espacio):



```
Plan: 41 to add, 0 to change, 0 to destroy.
```

Figura 9: Número de recursos a crear, destruir o modificar por Terraform

```

# aws_lambda_function.poc_lambda_index_to_es will be created
+ resource "aws_lambda_function" "poc_lambda_index_to_es" {
+   arn                = (known after apply)
+   filename            = ".terraform/lambda/poc_index_to_es.zip"
+   function_name       = "poc_index_to_es"
+   handler             = "poc_index_to_es.lambda_handler"
+   id                 = (known after apply)
+   invoke_arn          = (known after apply)
+   last_modified       = (known after apply)
+   layers              = (known after apply)
+   memory_size         = 128
+   publish             = false
+   qualified_arn        = (known after apply)
+   reserved_concurrent_executions = 4
+   role                = (known after apply)
+   runtime             = "python3.7"
+   source_code_hash     = "YbaBf3P/LYTRzsuhoV0UfQ7wZHL/tpwJWhgn7AXRJTU="
+   source_code_size     = (known after apply)
+   tags               = {
+     "environment" = "dev"
+     "owner"       = "jorge"
+     "project"     = "tfg"
+     "terraform"   = "true"
+   }
+   timeout             = 3
+   version             = (known after apply)

+   environment {
+     variables = (known after apply)
+   }

+   tracing_config {
+     mode = (known after apply)
+   }
+ }

```

Figura 10: Lambda que indexa en Elasticsearch

```

# module.sensor1.aws_iam_certificate.cert will be created
+ resource "aws_iam_certificate" "cert" {
+   active            = true
+   arn               = (known after apply)
+   certificate_pem    = (sensitive value)
+   id               = (known after apply)
+   private_key       = (sensitive value)
+   public_key        = (sensitive value)
+ }

# module.sensor1.aws_iam_policy_attachment.pol_attach will be created
+ resource "aws_iam_policy_attachment" "pol_attach" {
+   id      = (known after apply)
+   policy  = "poc-public-to-topic"
+   target  = (known after apply)
+ }

# module.sensor1.aws_iam_thing.thing will be created
+ resource "aws_iam_thing" "thing" {
+   arn                = (known after apply)
+   attributes         = {
+     "environment" = "dev"
+     "owner"       = "jorge"
+     "project"     = "tfg"
+     "terraform"   = "true"
+   }
+   default_client_id  = (known after apply)
+   id                 = (known after apply)
+   name               = "poc-iam-thing-1"
+   thing_type_name     = "poc-iam-type"
+   version            = (known after apply)
+ }

# module.sensor1.aws_iam_thing_principal_attachment.princ_attach will be created
+ resource "aws_iam_thing_principal_attachment" "princ_attach" {
+   id      = (known after apply)
+   principal = (known after apply)
+   thing    = "poc-iam-thing-1"
+ }

```

Figura 11: Recursos que se crearán para uno de los cuatro sensores IoT


```
# aws_elasticsearch_domain.poc will be created
+ resource "aws_elasticsearch_domain" "poc" {
  + access_policies      = (known after apply)
  + advanced_options     = (known after apply)
  + arn                  = (known after apply)
  + domain_id            = (known after apply)
  + domain_name          = "tfg-jorge"
  + elasticsearch_version = "7.1"
  + endpoint              = (known after apply)
  + id                   = (known after apply)
  + kibana_endpoint      = (known after apply)
  + tags                  = {
    + "environment" = "dev"
    + "owner"       = "jorge"
    + "project"     = "tfg"
    + "terraform"   = "true"
  }

  + cluster_config {
    + dedicated_master_enabled = false
    + instance_count           = 1
    + instance_type            = "t2.small.elasticsearch"
  }

  + domain_endpoint_options {
    + enforce_https = true
    + tls_security_policy = "Policy-Min-TLS-1-2-2019-07"
  }

  + ebs_options {
    + ebs_enabled = true
    + volume_size = 10
    + volume_type = (known after apply)
  }

  + encrypt_at_rest {
    + enabled      = (known after apply)
    + kms_key_id   = (known after apply)
  }

  + node_to_node_encryption {
    + enabled = (known after apply)
  }
}
```

Figura 12: Muestra del terraform plan para el recurso Elasticsearch Service

- Una vez validado y aplicado el plan de Terraform, se ejecutará el *terraform apply*. La salida de Terraform una vez terminado el proceso de aplicar los cambios se muestra en la siguiente figura:

```
Apply complete! Resources: 41 added, 0 changed, 0 destroyed.

Outputs:
elasticsearch_domain_arn = arn:aws:es:eu-west-1:██████████:domain/tfg-jorge
elasticsearch_domain_endpoint = search-tfg-jorge-██████████.eu-west-1.es.amazonaws.com
elasticsearch_domain_id = ██████████/tfg-jorge
iot_endpoint = ██████████.iot.eu-west-1.amazonaws.com
iot_thing_type_arn = arn:aws:iot:eu-west-1:██████████:thingtype/poc-iot-type
poc_iot_certificate_arn_sensor1 = arn:aws:iot:eu-west-1:██████████:cert/██████████027b8ce0d3a81f3bb9bb8c702214fb277bc2386ade7d30f3b2
poc_iot_certificate_arn_sensor2 = arn:aws:iot:eu-west-1:██████████:cert/██████████6a4f7db6262f5fd4bdcdc7af56c3d570892d007d92f7d5a4b8
poc_iot_certificate_arn_sensor3 = arn:aws:iot:eu-west-1:██████████:cert/██████████bd24ddf6c2693b80f999a3f8e7a96e0647412613bf711f8e39
poc_iot_certificate_arn_sensor4 = arn:aws:iot:eu-west-1:██████████:cert/██████████cd51058fbedf10e06b9e73766dab3107cea3a5e92395bcd90f
poc_iot_certificate_pem_sensor1 = -----BEGIN CERTIFICATE-----
MIIDTCCAKGAWIBAgIUvPwBMXHX1Ckx5etoXsOSRZCUqYwQYJKoZIhvcNAQEL
```

Figura 13: Resultado del comando terraform apply (Censurados los datos sensibles)

3. El siguiente paso es almacenar las variables de salida Terraform que se muestran en la figura anterior. Para ello, el script `launch.tfg.sh` invoca al script `save_outputs_terraform.py`. Este script guarda los *outputs* que son necesarios. Estos son (no se muestra el contenido de los archivos creados por seguridad):

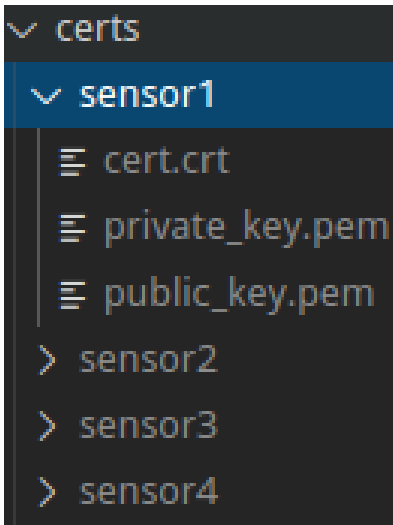


Figura 14: Certificados de las IoT Things desplegadas

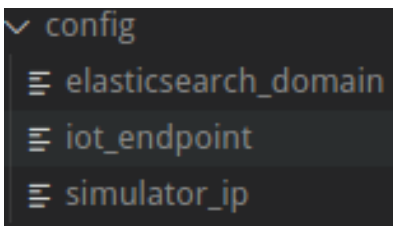


Figura 15: Parámetros de configuración

4. El último paso del despliegue es subir los archivos del simulador a la instancia que lo va a ejecutar, ejecutar el comando *Docker build* y ejecutar el comando *docker run*. La salida de dichos pasos puede observarse en las siguientes figuras:

AmazonRootCA1.pem	100%	1188	23.0KB/s	00:00
private_key.pem	100%	1675	38.2KB/s	00:00
cert.crt	100%	1224	25.8KB/s	00:00
public_key.pem	100%	451	10.2KB/s	00:00
private_key.pem	100%	1679	37.8KB/s	00:00
cert.crt	100%	1224	27.6KB/s	00:00
public_key.pem	100%	451	10.3KB/s	00:00
private_key.pem	100%	1675	38.3KB/s	00:00
cert.crt	100%	1220	27.2KB/s	00:00
public_key.pem	100%	451	10.0KB/s	00:00
private_key.pem	100%	1679	37.7KB/s	00:00
cert.crt	100%	1224	27.6KB/s	00:00
public_key.pem	100%	451	10.3KB/s	00:00
iot_thing_simulator.py	100%	4006	81.8KB/s	00:00
requeriments.txt	100%	23	0.3KB/s	00:00
run.sh	100%	1306	28.2KB/s	00:00
Dockerfile	100%	369	7.4KB/s	00:00

Figura 16: Archivos copiados mediante scp a la instancia que ejecutará el simulador

```

Sending build context to Docker daemon 41.98kB
Step 1/9 : FROM python:3.7-alpine
3.7-alpine: Pulling from library/python
cbdbe7a5bc2a: Pulling fs layer
26ebcd19a4e3: Pulling fs layer
f2a4afa74df8: Pulling fs layer
a3916b494c8f: Pulling fs layer
038cfaa9de84: Pulling fs layer
a3916b494c8f: Waiting
038cfaa9de84: Waiting
cbdbe7a5bc2a: Verifying Checksum
cbdbe7a5bc2a: Download complete
26ebcd19a4e3: Verifying Checksum
26ebcd19a4e3: Download complete
f2a4afa74df8: Verifying Checksum
f2a4afa74df8: Download complete
cbdbe7a5bc2a: Pull complete
a3916b494c8f: Verifying Checksum
a3916b494c8f: Download complete
038cfaa9de84: Verifying Checksum
038cfaa9de84: Download complete
26ebcd19a4e3: Pull complete
f2a4afa74df8: Pull complete
a3916b494c8f: Pull complete
038cfaa9de84: Pull complete
Digest: sha256:8acbf2568aaf64337a9ef2d4f15e0813566957b93fa6954c165335d1a54ca617

```

Figura 17: docker build 1/2

```

Status: Downloaded newer image for python:3.7-alpine
--> 61681f520204
Step 2/9 : COPY certs /iot-simulator/certs
--> 82680c97c5d9
Step 3/9 : COPY iot-simulator/requeriments.txt /iot-simulator/requeriments.txt
--> f145d1867a55
Step 4/9 : RUN pip3 install -r /iot-simulator/requeriments.txt
--> Running in 7b6811b73a9d
Collecting AWSIoTPythonSDK==1.4.8
  Downloading AWSIoTPythonSDK-1.4.8.tar.gz (80 kB)
Building wheels for collected packages: AWSIoTPythonSDK
  Building wheel for AWSIoTPythonSDK (setup.py): started
  Building wheel for AWSIoTPythonSDK (setup.py): finished with status 'done'
  Created wheel for AWSIoTPythonSDK: filename=AWSIoTPythonSDK-1.4.8-py3-none-any.whl
  Stored in directory: /root/.cache/pip/wheels/5a/f0/72/cf619382a05382dee601be0b5b06
Successfully built AWSIoTPythonSDK
Installing collected packages: AWSIoTPythonSDK
Successfully installed AWSIoTPythonSDK-1.4.8
Removing intermediate container 7b6811b73a9d
--> 98b31b9a3603
Step 5/9 : COPY iot-simulator/iot_thing_simulator.py /iot-simulator/iot_thing_simula
--> 2b8f45ce1dd3
Step 6/9 : COPY iot-simulator/run.sh /iot-simulator/run.sh
--> a2db217ce970
Step 7/9 : WORKDIR /iot-simulator/
--> Running in fd095ae90246
Removing intermediate container fd095ae90246
--> e776929aaecf
Step 8/9 : CMD ["/run.sh"]
--> Running in 3820e8e3b7af
Removing intermediate container 3820e8e3b7af
--> e040569ee848
Step 9/9 : ENTRYPOINT [ "sh" ]
--> Running in 1c01c62c2ed2
Removing intermediate container 1c01c62c2ed2
--> 475976286c97
Successfully built 475976286c97
Successfully tagged iot-simulator:latest
967b34d96c1fd59925ce74c79e45ceccc2a01a25e58258687ab25277210d25d1

```

Figura 18: docker build 2/2 y docker run

Al finalizar toda esta serie de pasos, la infraestructura final estará desplegada y el simulador comenzará a enviar datos con una frecuencia entre cinco y diez segundos. En una implementación

real, esta frecuencia sería mucho más baja ya que no son necesarios datos separados por tan poco tiempo en un entorno real. Se ha decidido implementar de esta forma para que sea ágil poder mostrar el funcionamiento de la solución.

5. Pruebas realizadas

Sobre el *MVP* propuesto para la solución, se han realizados varias pruebas.

Estas pruebas se han realizado desde un ordenador portátil con las siguientes características:

```
System:   Host: manjaro Kernel: 5.4.44-1-MANJARO x86_64 bits: 64 Desktop: Gnome 3.36.3 Distro: Manjaro Linux
Machine:  Type: Laptop System: Dell product: XPS 15 7590 v: N/A serial: 51WVN13
Mobo: Dell model: 0VYV0G v: A00 serial: /51WVN13/CNCMK0002K016C/ UEFI: Dell v: 1.5.0 date: 12/25/2019
```

Figura 19: Características del PC desde el cual se han ejecutado las pruebas

La versión de Grafana [38] utilizada en una de las pruebas es la 7.0.2.

A continuación, se detallan todas estas pruebas junto con sus resultados. Asimismo, se explicará una prueba de concepto de una hipotética situación a la que se podría enfrentar el sistema.

5.1. Tiempo de despliegue y destrucción de la infraestructura

La primera de estas pruebas ha consistido en utilizar el comando `time` [39] para poder medir el tiempo que tarda el *MVP* en desplegarse y comenzar a funcionar. Se ha ejecutado mediante el comando `time` el script `launch_tfg.sh` tres veces, así como sobre el script `destroy_tfg.sh` otras tres veces. De esta forma se puede comprobar cuánto tiempo tarda normalmente en desplegarse y cuánto en destruirse. Los resultados de estas pruebas han sido los siguientes:

	launch_tfg.sh	destroy_tfg.sh
Nº de prueba	Tiempo Empleado	Tiempo Empleado
1	14m 05s	5m 18s
2	13m 53s	5m 18s
3	15m 36s	5m 22s
Tiempo medio	14m 31s	5m 19s

Tabla 2: Prueba de despliegue y destrucción de la infraestructura

Como se puede observar en la Tabla 2, el sistema tarda de media catorce minutos y medio en desplegarse completamente. Este despliegue es de infraestructura y del simulador. A partir de ese momento, el simulador ya comienza a enviar datos al IoT Core.

También puede observarse que tarda de media algo más de cinco minutos y diecinueve segundos en destruirse por completo.

Por lo tanto, tras esta prueba se pueden sacar las siguientes conclusiones:

1. Desplegar un sistema en la nube (AWS en este caso) utilizando infraestructura como código es un proceso muy ágil. Se ha conseguido desplegar un sistema complejo, con una base de datos como es Elasticsearch en menos de 15 minutos.
2. Destruir el sistema creado es incluso más rápido. Además, utilizando *IaC*, no se corre el riesgo de olvidar algún recurso sin destruir que genere gasto de manera indeseada.
3. En este caso, con AWS y Terraform, los tiempos tanto de despliegue como de destrucción varían muy poco, por lo que se pueden asegurar rangos de tiempos para ambos procesos.
4. Usar *IaC* es la mejor manera de poder desplegar y destruir una infraestructura como esta, en un proveedor de nube pública. No utilizar *IaC* implicaría tener que realizar todos estos despliegues y destrucciones de manera manual. Esto se traduciría en una gran cantidad de tiempo invertida y en una gran posibilidad de error.

5.2. Comunicación hacia los dispositivos

Otra de las pruebas que se ha realizado contra el sistema es la comunicación en el sentido inverso, es decir, desde la nube hacia los dispositivos. Esta comunicación es importante ya que permite actuar sobre los propios dispositivos sin tener que estar en el mismo lugar físicamente. Esto permitiría actualizar los dispositivos, reiniciarlos, etc.

Para la realización de esta prueba se ha decidido implementa un pequeño script en Python que sirva como ejemplo, sin implementar un dispositivo o simulador real. El único objetivo de esta prueba es demostrar que es posible esta comunicación. Para ello, se va a hacer uso de las credenciales de uno de los cuatro dispositivos IoT que se despliegan.

Se ha decidido que el funcionamiento al recibir un mensaje desde el IoT Topic sea imprimirlo por pantalla y ejecutar el comando indicado en la máquina donde se lanza el suscriptor. En las siguientes imágenes podemos ver ejemplos de este funcionamiento:

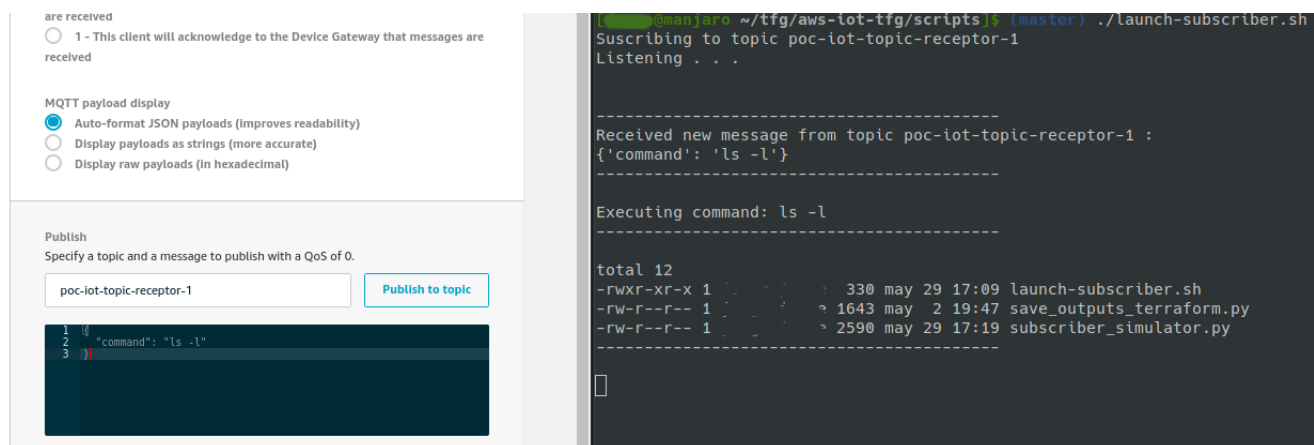


Figura 20: Prueba 1 de comunicación hacia el dispositivo IoT. Comando ls -l

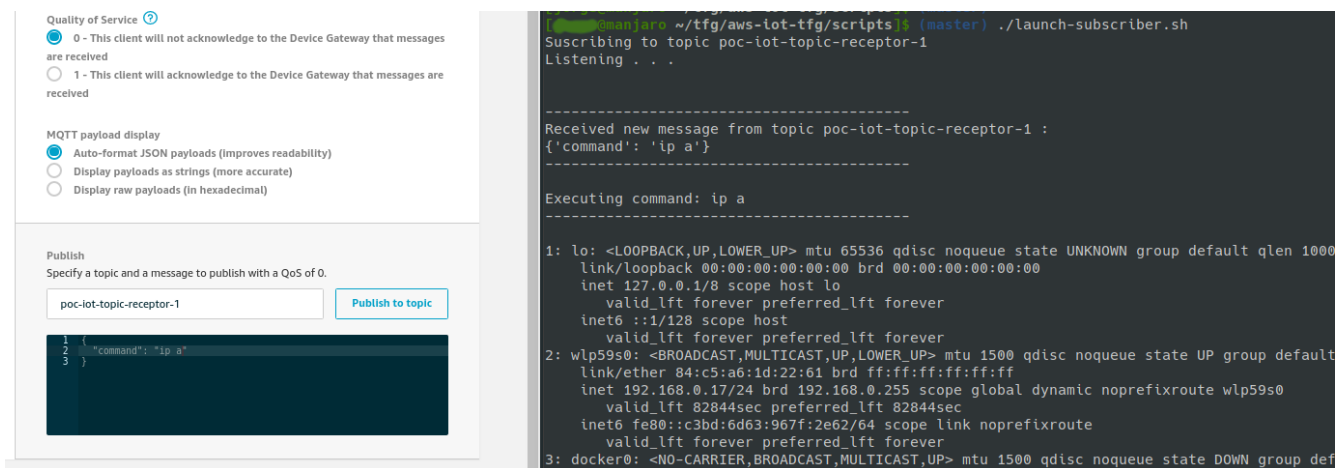


Figura 21: Prueba 1 de comunicación hacia el dispositivo IoT. Comando ip a

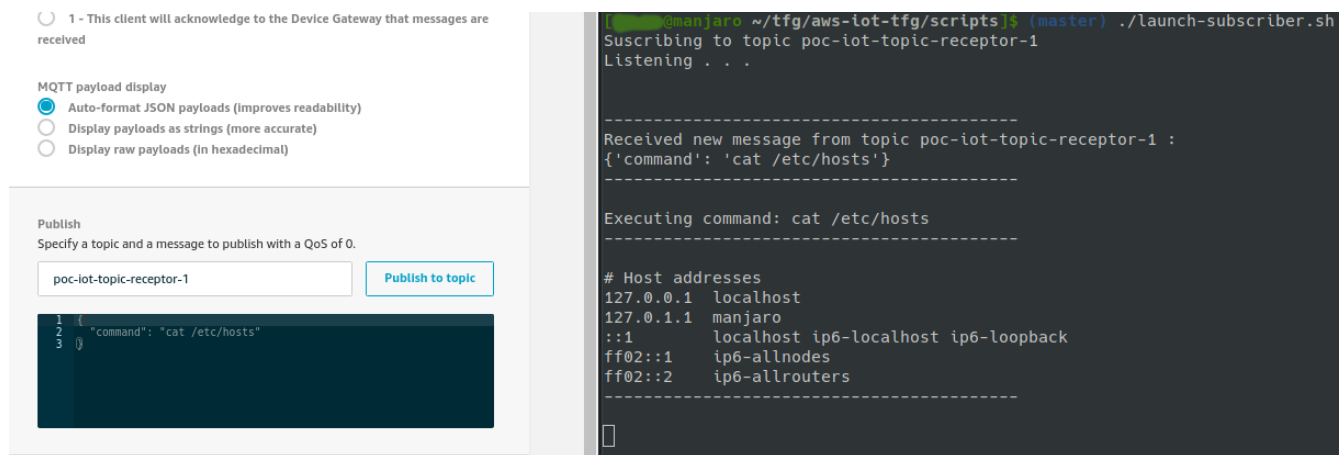


Figura 22: Prueba 1 de comunicación hacia el dispositivo IoT. Comando cat /etc/hosts

Como puede observarse, se pueden incluso ejecutar comandos de manera remota si el programa suscrito al topic está preparado para ello. De esta manera se puede actualizar el software, cambiar la configuración, etc. sin tener que estar físicamente conectado al dispositivo. En cuanto a la seguridad sobre esto, es necesario asegurar que el *topic* al cual se suscriben los dispositivos tiene restringida la escritura para que no haya compromisos de seguridad. Para poder realizar esta prueba, se ha tenido que modificar la política asociada a los dispositivos IoT desplegados para que tengan permiso para suscribirse y recibir. Anteriormente únicamente tenían permiso para publicar.

Con esta pequeña prueba se demuestra lo simple que sería poder actuar sobre los dispositivos IoT utilizando el mismo protocolo utilizado para publicar los mensajes hacia la nube.

5.3. Representación de los datos

La última prueba que se ha realizado sobre el sistema incluye una herramienta externa de visualización de datos: Grafana [38]. Grafana es una herramienta que puede integrarse con una gran cantidad de fuentes de datos como Elasticsearch. Posee también un motor de alertas que permitiría

ejecutar reglas o realizar acciones basándose en las métricas recolectadas. Es una herramienta gratuita y de código abierto.

Grafana se ejecutará en local, fuera del entorno del *MVP* por dos motivos: no perder la configuración cuando el entorno sea destruido y tener una herramienta conectada al sistema, para demostrar la compatibilidad del mismo.

Para el *MVP*, se van a generar cuatro cuadros de mando (*dashboards*) diferentes, uno para cada sensor y cada uno de ellos con diferentes visualizaciones de las métricas del mismo. Por simplificar, sólo se va a generar de manera completa el *dashboard* perteneciente al sensor1. El resto se prepararán pero no se completará su implementación en esta prueba. Esta herramienta sería una de las que, en un escenario real, sería operada por el encargado del campo de cultivo. Es por tanto una herramienta cuya configuración debe orientarse al usuario y debe ser fácil de utilizar.

A continuación, se incluye una captura de la interfaz de Grafana, configurada para recibir datos de Elasticsearch y representarlos en los diferentes *dashboards*. En el apartado de trabajos futuros, se detallarán posibles desarrollos a realizar utilizando esta herramienta.



Figura 23: Interfaz de Grafana mostrando las métricas del sensor1

La figura anterior no es más que un ejemplo de representación de las métricas de uno de los sensores IoT. Se ha configurado para mostrar los datos en tiempo real de las métricas que reportan los dispositivos hacia la nube. Se trata de un pequeño ejemplo de qué hacer con los datos una vez indexados en Elasticsearch.

5.4. Prueba de concepto

La última prueba realizada será una prueba de concepto ficticia sobre una hipotética situación que se podría dar en el mundo real al implementar un sistema como el del *MVP* desarrollado y a la

que se tratará de dar respuesta mediante una de las bases de este proyecto, la infraestructura como código.

Se tiene como supuesto el sistema aquí desarrollado ya desplegado y funcionando en un entorno real. En un momento dado, las necesidades del agricultor cambian y decide implementar un cambio en el sistema, añadiendo algún dispositivo más al mismo. Se obviará la operativa sobre el hardware del sistema ya que no es objetivo de esta prueba ni este proyecto.

Utilizando una metodología tradicional a demanda en el que el agricultor tiene que contactar con los responsables del servicio, el proceso sería el siguiente:

- El agricultor piensa el cambio y contacta con el responsable para explicárselo.
- Una vez llegado a un entendimiento, el responsable del sistema deberá realizar los cambios en el mismo.
- Una vez realizados los cambios, el responsable volverá a contactar con el agricultor indicándole que la petición está resuelta y ya es posible conectar esos dispositivos extra al sistema, enviándole la información que sea necesaria como salida de dichos cambios.

Como se puede observar, la automatización del proceso descrito anteriormente es nula y el tiempo empleado para llevarlo a cabo desde el primer paso hasta el último se puede extender en el tiempo varios días, dependiendo del volumen de este tipo de peticiones que el responsable gestione.

A continuación se va a proponer un modelo diferente orientado a la automatización y que es únicamente posible mediante la utilización de la infraestructura como código:

- El agricultor piensa en el cambio y, mediante algún tipo de aplicación instalada en su móvil o pc puede seleccionar qué cambio es el que quiere aplicar dentro de una lista de posibles cambios.
- Una vez seleccionado, de manera automática se lanzará un proceso de actualización de la infraestructura que mediante variables dentro del código Terraform, aumente el número de dispositivos y envíe al agricultor la salida necesaria de dicha actualización.

Con este último modelo, se evita así la necesidad de interacción entre el agricultor y el técnico, ahorrando una cantidad significativa de tiempo. Mientras que el tiempo del primer escenario se puede alargar en el tiempo horas o incluso días debido a los procesos de soporte tradicionales, el segundo escenario el proceso puede llevarse a cabo en minutos, de manera automatizada y evitando en la medida de lo posible la interacción humana disminuyendo las probabilidades de errores o malentendidos.

Es por ello que en la realización de este proyecto se ha decidido utilizar *IaC*. Hacer uso de esta tecnología permite conseguir un nivel de automatización prácticamente absoluto sobre cualquier infraestructura gestionada.

6. Presupuesto

Como se ha mencionado anteriormente en el apartado de desarrollo de la solución, el presupuesto para este proyecto es de \$0 al poder utilizar la capa gratuita de AWS [30]. Es interesante desglosar qué recursos se están utilizando para el *MVP* y cuánto costarían mensualmente dichos recursos fuera de la capa gratuita de AWS.

Los recursos más significativos son:

1. Para el sistema (excluidos recursos como por ejemplo IAM, ya que los que se crean son gratuitos siempre):
 - Elasticsearch Service:
 - t2.small.elasticsearch instance.
 - EBS Storage (10GB)
 - AWS Lambda Function
 - AWS Lambda Layer
 - 4 IoT Things
 - 4 IoT Topic Rules
2. Para el simulador (excluidos recursos como por ejemplo IAM o tablas de ruta, ya que los que se crean son gratuitos siempre):
 - EC2 Instance t2.micro
 - AWS VPC
 - AWS Subnet
 - AWS Elastic IP
 - AWS Internet Gateway

El precio de dichos recursos fuera de la capa gratuita es (para la región de Irlanda eu-west-1):

1. Para el sistema:
 - Elasticsearch Service [40]:
 - t2.small.elasticsearch instance: \$0.039/hora
 - EBS Storage (10GB): \$0.149 GB/month
 - AWS Lambda [41]
 - *Provisioned Concurrency*: \$0.000004646 por cada GB-segundo.
 - Peticiones \$0.20 por 1M de peticiones
 - Duration \$0.0000108407 por cada GB-segundo (depende de la cantidad de memoria reservada para la ejecución de la función)
 - AWS IoT [42]

- Conectividad: \$0.08 por cada millón de minutos conectado. (Es necesario enviar un mensaje *keep-alive* desde cada 20 minutos hasta cada 30 segundos para mantener conectividad, pero no es necesario ya que los mensajes de los dispositivos en el *MVP* envían desde cada 5 hasta cada 10 segundos)
- Mensajes: hasta 1000 millones de mensajes, \$1/millón de mensajes
- Reglas activadas: \$0.15 por cada millón de reglas activadas. En este caso, cada mensaje enviado activa una regla.

2. Para el simulador:

- AWS EC2 [43]
 - Instancia t2.micro (On-Demand): \$0.0126/hora
- AWS VPC, AWS Subnet, AWS Elastic IP, AWS Internet Gateway [44]
 - Todos estos recursos son gratuitos salvo la Elastic IP que únicamente se cobra el tiempo que no está asignada a una instancia. Para este caso, siempre que exista va a estar asignada a la instancia. Por lo tanto, \$0.

Teniendo en cuenta todos estos datos, y considerando:

- Un mes de 31 días. Por lo tanto, 744 horas.
- El sistema estará funcionando 24/7.
- Se crean 4 dispositivos que envían mensajes cada 5 segundos (frecuencia máxima de envío). Esto es 535680 mensajes al mes por cada dispositivo.
- Cada mensaje enviado, dispara una regla de AWS IoT. Esto es 535680 reglas disparadas al mes por cada dispositivo.
- Cada Lambda se va a ejecutar durante 0.5 segundos (se ejecuta una por cada regla disparada de AWS IoT) y se reserva 1GB de memoria por cada ejecución. Esto es 535680 ejecuciones al mes por cada dispositivo, lo que implica 267840 GB-segundos al mes.

El precio del sistema desplegado para el *MVP* fuera de la capa gratuita de AWS sería:

- Para el sistema:
 - Elasticsearch: $\$0.039 * 744 \text{ horas} + \$0.149 * 10 \text{ GB-mes} = \30.51 al mes .
 - AWS Lambda:
 - *Provisioned Capacity* = $\$0.000004646 * 267840 \text{ GB-segundos} = \1.24 al mes
 - Peticiones = $\$0.20 * 535680/1\text{M} = \0.107136 al mes
 - Duración = $\$0.0000108407 * 267840 \text{ GB-segundos} = \2.9036 al mes
 - Total: 4 dispositivos * $(\$1.24 + \$0.107136 + \$2.9036) = \17 al mes
 - AWS IoT:
 - Conectividad: $\$0.08 * 44640/1\text{M} = \0.035712 al mes
 - Mensajes: $\$1 * 535680/1\text{M} = \0.53568 al mes

- Reglas: $\$0.15 * 535680/1M = \0.080352 al mes
- Total: $4 \text{ dispositivos} * (\$0.035712 + \$0.53568 + \$0.080352) = \$2.61$ al mes
- Total: $\$30.51 + \$17 + \$2.61 = \50.12
- Para el simulador:
 - AWS EC2: $\$0.0126 * 744 \text{ horas} = \9.37 al mes
 - Total: $\$9.37$ al mes

Por lo tanto, sumando los costes del sistema y del simulador, el gasto total al mes sería de aproximadamente $\$50.12 + \$9.37 = \$59.49$ al mes.

Sin embargo, este gasto no es real, ya que en un sistema real, los dispositivos no enviarían cada 5 segundos, sino cada 5 minutos, entre otras cosas. Por contra, no se han tenido en cuenta los posibles gastos que se generarían por el tráfico saliente de AWS, aunque serían prácticamente despreciables, ya que en un sistema IoT no se envían grandes volúmenes de datos.

La conclusión que se puede obtener de este presupuesto de ejemplo es que se puede conseguir un sistema muy complejo y elaborado con un presupuesto muy ajustado. Para ello es necesario conocer cómo se tarifica en la nube pública (AWS en este caso), y qué servicios son interesantes para utilizar teniendo en cuenta estas tarificaciones. Para trabajos futuros, si este sistema se despliega en un entorno de producción, sería necesario establecer otros procedimientos como copias de seguridad, apagado y encendido de recursos, reserva de instancias, etc.

7. Conclusiones y trabajos futuros

Una vez terminado este proyecto de investigación y desarrollo, se han obtenido las siguientes conclusiones:

1. El uso de infraestructura como código para la implementación de sistemas en la nube facilita en gran medida su gestión y reutilización. Usar *IaC* no solo permite gestionar la infraestructura de manera más eficaz, sino que evita fallos humanos, permite desplegar y evolucionar infraestructura de la misma forma que el software habitual y permite trabajar a varios equipos sobre la misma infraestructura sin problemas.
2. Es muy importante conocer los servicios que ofrece el proveedor de nube pública que se utilice, así como su modelo de facturación. Conocer bien estos servicios y cómo pueden dar respuesta a las diferentes necesidades hará que el sistema implementado no solo sea más eficiente sino también más económico.
3. Es importante conocer cuándo un servicio gestionado merece la pena teniendo en cuenta el tiempo y/o dinero que habría que invertir en preparar un sistema que realice el mismo trabajo y lo que cuesta dicho servicio gestionado. Hay veces que es más interesante desarrollar el sistema y hay veces que es mejor utilizar el servicio gestionado.
4. Utilizar la nube pública es la mejor manera de acceder a sistemas o servicios complejos de instalar (por ejemplo, Elasticsearch), de una manera sencilla y económica. Utilizando sistemas tradicionales y *on-premises*, la inversión para poder utilizar un sistema de estas características es mucho mayor que desplegarlo en la nube un tiempo determinado y después destruirlo.

5. Es necesario dedicar una gran cantidad de tiempo y esfuerzo a estudiar y entender cómo funciona el proveedor de nube pública para poder aprovechar al máximo lo que ofrece y poder ajustar el presupuesto al mínimo posible.

Asimismo, se han ido identificando posibles pasos futuros a seguir en una teórica continuación del desarrollo realizado hasta ahora en este proyecto:

1. Probar el mismo sistema pero sin usar un simulador, es decir, con dispositivos IoT reales como los mencionados en el apartado hardware IoT.
2. Aumentar la cantidad de datos recogidos de los sensores. Se pueden añadir datos como luminosidad, químicos presentes en el suelo, etc. que permitan realizar actuaciones o acciones más específicas y concretas teniendo en cuenta estos nuevos datos y métricas.
3. Aumentar el sistema aquí implementado añadiendo un segundo tipo de dispositivo IoT que sea un actuador. De esta manera, el sistema no solo dispondría de dispositivos de medida y monitorización, sino también de dispositivos que puedan alterar el entorno.
4. Al hilo de la idea anterior, sería necesario añadir al sistema un desarrollo de la gestión de las alertas, como el mencionado en la prueba con Grafana, sobre las métricas recogidas, que se comunique con los actuadores cuando se dispare una regla sobre una métrica concreta.
5. Implementar Elasticsearch Curator [45] en la base de datos. Se trata de una herramienta que permite eliminar los datos antiguos, a partir de los días que se establezcan. Esto hará que la base de datos no se llene con datos antiguos que no serán de utilidad para el sistema.
6. Siguiendo con la idea anterior, en lugar de eliminar los datos antiguos, se pueden guardar en un almacenamiento más barato, como S3, para posteriormente lanzar análisis sobre las métricas y poder sacar históricos de las mismas en las diferentes tierras de cultivo.
7. Investigar la posibilidad de desplegar el mismo sistema aquí expuesto en otros proveedores de *Cloud* mencionados, como Microsoft Azure o GCP. Quizás en alguno de estos proveedores el sistema puede ser más económico, más eficiente, etc. Esta línea de investigación puede abrir las posibilidades a un despliegue *Multi-Cloud*.
8. Investigar e identificar los parámetros que sería necesario modificar en el sistema para adecuarlo a un sistema productivo, como por ejemplo, la cantidad de almacenamiento de la que dispone Elasticsearch o el tipo de instancia elegido para ello.

Finalmente, se puede observar que las posibilidades de mejora son prácticamente infinitas y que un sistema de estas características puede estar en continuo desarrollo y evolución para ofrecer cada vez más servicios a petición de los usuarios del mismo. La mejor opción sería implementar el sistema en una tierra de cultivo real y tener una comunicación continua con el agricultor, el ingeniero agrónomo y otros actores existentes en una explotación agrícola, escuchando así las necesidades que puedan ir surgiendo para poder aplicarlas al sistema. Es necesario intentar aprovechar al máximo las posibilidades que ofrece la nube pública y su capa gratuita, sea cual sea el proveedor, ya que permite realizar desarrollos a coste 0 que luego pueden implantarse en un entorno real. Actualmente, con la tecnología *Cloud* e IoT en continuo crecimiento, es el mejor momento para sumergirse de lleno en las mismas y desarrollar ideas y pruebas que en el futuro pueden formar parte de un gran sistema, y así poder mejorar de forma continua los resultados y las operaciones en la agricultura.

Referencias

- [1] Qampo. *La agricultura de precisión*. 2020. URL: <https://qampo.es/la-agricultura-de-precision/> (visitado 26-04-2020).
- [2] Wikipedia. *Internet de las cosas — Wikipedia, La enciclopedia libre*. 2020. URL: https://es.wikipedia.org/w/index.php?title=Internet_de_las_cosas&oldid=125088224 (visitado 26-04-2020).
- [3] Thubert et al. *IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN)*. 2017. URL: <https://tools.ietf.org/html/rfc8138> (visitado 26-04-2020).
- [4] IPLocation. *How does ZigBee Work?* 2018. URL: <https://www.iplocation.net/zigbee> (visitado 26-04-2020).
- [5] LoRa Alliance. *What is the LoRaWAN® Specification?* 2020. URL: <https://loro-alliance.org/about-lorawan> (visitado 26-04-2020).
- [6] GSMA. *Narrowband – Internet of Things (NB-IoT)*. 2020. URL: <https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/> (visitado 01-07-2020).
- [7] GSMA. *Long Term Evolution for Machines: LTE-M*. 2020. URL: <https://www.gsma.com/iot/long-term-evolution-machine-type-communication-lte-mtc-cat-m1/> (visitado 01-07-2020).
- [8] Tim Grance (NIST) Peter Mell (NIST). *The NIST Definition of Cloud Computing*. 2011. URL: <https://csrc.nist.gov/publications/detail/sp/800-145/final> (visitado 26-04-2020).
- [9] Tony Hou. *IaaS vs PaaS vs SaaS Enter the Ecommerce Vernacular: What You Need to Know Examples & More*. 2020. URL: <https://www.bigcommerce.com/blog/saas-vs-paas-vs-iaas#the-key-differences-between-on-premise-saas-paas-iaas> (visitado 26-04-2020).
- [10] Raspberry PI Foundation. *Raspberry.org*. 2020. URL: <https://www.raspberrypi.org> (visitado 26-04-2020).
- [11] Libelium Comunicaciones Distribuidas S.L. *Libelium.com*. 2020. URL: <http://www.libelium.com/> (visitado 26-04-2020).
- [12] Lora Alliance. *A technical overview of LoRa® and LoRaWAN™*. 2015. URL: <https://loro-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf> (visitado 26-04-2020).
- [13] Sigfox. *What is Sigfox?* 2020. URL: <https://build.sigfox.com/sigfox> (visitado 26-04-2020).
- [14] P. Christensson. *WiMAX*. 2009. URL: <https://techterms.com/definition/wimax> (visitado 26-04-2020).
- [15] Arrigo Lupori. *NB-IoT vs LTE-M: Here's What The Cellular IoT Buzz Is All About*. 2020. URL: <https://ubidots.com/blog/nb-iot-vs-lte-m/> (visitado 01-07-2020).
- [16] *MQTT*. 2020. URL: <http://mqtt.org/> (visitado 26-04-2020).

- [17] Libelium. *New vineyard project developed with Libelium IoT platform on Agrotech, the app for crop management, powered by Efor and Ibercaja on Microsoft Azure*. 2019. URL: <http://www.libelium.com/new-vineyard-project-developed-with-libelium-iot-platform-on-agrotech-the-app-for-crop-management-powered-by-efor-and-ibercaja-on-microsoft-azure/> (visitado 26-04-2020).
- [18] Minwoo Ryu y col. «Design and implementation of a connected farm for smart farming system». En: nov. de 2015, págs. 1-4. DOI: 10.1109/ICSENS.2015.7370624.
- [19] James Blackman. *Four cases of Vodafone's local know-how in smart farming*. 2019. URL: <https://enterpriseiotinsights.com/20190522/channels/fundamentals/four-cases-of-vodafone-s-local-know-how-in-smart-farming> (visitado 25-05-2020).
- [20] Vodafone. *Internet of Things: NarrowBand IoT*. 2019. URL: <https://www.vodafone.es/c/empresas/grandes-clientes/es/soluciones/datos/internet-of-things/narrowband-iot/> (visitado 25-05-2020).
- [21] Bodegas Emilio Moro. 2019. URL: <https://www.emiliomoro.com/> (visitado 25-05-2020).
- [22] Hashicorp. *Infrastructure as Code: What Is It? Why Is It Important?* 2018. URL: <https://www.hashicorp.com/resources/what-is-infrastructure-as-code/> (visitado 01-07-2020).
- [23] Carlos Schults. *What Is Infrastructure as Code? How It Works, Best Practices, Tutorials*. 2019. URL: <https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/> (visitado 01-07-2020).
- [24] Tomas Fernandez. *What is Infrastructure as Code?* 2020. URL: <https://blog.stackpath.com/infrastructure-as-code-explainer/> (visitado 01-07-2020).
- [25] Jorge de Diego Rubio. *aws-iot-tfg*. 2020. URL: <https://github.com/j0rzsh/aws-iot-tfg> (visitado 01-07-2020).
- [26] AWS. *AWS Documentation*. 2020. URL: <https://docs.aws.amazon.com/> (visitado 26-04-2020).
- [27] Hashicorp. *Terraform.io*. 2020. URL: <https://www.terraform.io/> (visitado 26-04-2020).
- [28] AWS. *Boto 3 Documentation*. 2020. URL: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html> (visitado 26-04-2020).
- [29] AWS. *Serverless*. 2020. URL: <https://aws.amazon.com/serverless/> (visitado 26-04-2020).
- [30] AWS. *AWS Free Tier*. 2020. URL: <https://aws.amazon.com/free> (visitado 26-04-2020).
- [31] AWS. *AWS Command Line Interface*. 2020. URL: <https://aws.amazon.com/cli/> (visitado 26-04-2020).
- [32] Hashicorp. *Creating Modules*. 2020. URL: <https://www.terraform.io/docs/modules/index.html> (visitado 26-04-2020).
- [33] Docker Inc. *Docker.com*. 2020. URL: <https://www.docker.com/> (visitado 26-04-2020).
- [34] Hashicorp. *Sensitive Data in State*. 2020. URL: <https://www.terraform.io/docs/state/sensitive-data.html> (visitado 26-04-2020).
- [35] Elastic. *Getting Started with the Elastic Stack*. 2020. URL: <https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html> (visitado 04-05-2020).

- [36] MongoDB. *NoSQL Databases Explained*. 2020. URL: <https://www.mongodb.com/nosql-explained> (visitado 04-05-2020).
- [37] Elastic. *Data in: documents and indices*. 2020. URL: <https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html> (visitado 04-05-2020).
- [38] Grafana. 2020. URL: <https://grafana.com/> (visitado 17-05-2020).
- [39] *time - Linux User's Manual*. 2019. URL: <http://man7.org/linux/man-pages/man1/time.1.html> (visitado 17-05-2020).
- [40] AWS. *Amazon Elasticsearch Service pricing*. 2020. URL: <https://aws.amazon.com/elasticsearch-service/pricing/> (visitado 17-05-2020).
- [41] AWS. *AWS Lambda Pricing*. 2020. URL: <https://aws.amazon.com/lambda/pricing/> (visitado 17-05-2020).
- [42] AWS. *AWS IoT Core pricing*. 2020. URL: <https://aws.amazon.com/iot-core/pricing/> (visitado 17-05-2020).
- [43] AWS. *Amazon EC2 pricing*. 2020. URL: <https://aws.amazon.com/ec2/pricing/> (visitado 17-05-2020).
- [44] AWS. *Amazon VPC pricing*. 2020. URL: <https://aws.amazon.com/vpc/pricing/> (visitado 17-05-2020).
- [45] Elastic. *Curator Reference*. 2020. URL: <https://www.elastic.co/guide/en/elasticsearch/client/curator/5.8/index.html> (visitado 25-05-2020).