Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

# Pasarela sobre Raspberry-Pi entre un servidor CoAP y una plataforma DBaaS

Autor: Sergio Martín Sánchez

Tutor: Germán Madinabeitia Luque

**Dep. De Ingeniería Telemática**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2019

Trabajo fin de grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

# Pasarela sobre Raspberry-Pi entre un servidor CoAP y una plataforma DBaaS

Autor:
Sergio Martín Sánchez

Tutor:
German Madinabeitia Luque
Profesor colaborador doctor

Dep. De Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo fin de grado: Pasarela sobre Raspberry-Pi entre un servidor CoAP y una plataforma DBaaS

Autor: Sergio Martín Sánchez

Tutor: German Madinabeitia Luque

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

# Agradecimientos

A mi familia.

*Sergio Martín Sánchez*

*Sevilla, 2019*

# Resumen

El internet de las cosas ha cambiado por completo muchos de los conceptos que rodean nuestra vida cotidiana. Ha cambiado la forma en la que alquilamos un coche o en la que controlamos una casa.

El auge de esta tecnología no habría sido posible sin el uso de los servicios en la nube. La nube, cuyas características más destacadas son la reducción de costes y su agilidad, ofrece soluciones informáticas adaptadas a los requisitos del usuario.

El Objetivo de este proyecto es implementar una pasarela entre nodos sensores que usan el protocolo CoAp y una base de datos en la nube.

# Abstract

The Internet of things has completely changed many concepts in our life. It has changed the way in which we rent a car or how we can keep the control of our home.

The development of this technology would have not been able without cloud computing services. Cloud services, which main characteristics are agility and cost reductiom, offer technology solutions adapted to costumer requirments.

The aim of this Project is to devolope a Gateway between sensors using CoAp protocol and a cloud database.

# ÍNDICE

# ÍNDICE DE FIGURAS

.

# 1 INTRODUCCIÓN

*Like air and drinking water, being digital will be noticed only by its absence, not its presence.*

Nicholas Negroponte

Hoy en día el uso de internet y su utilización resulta indispensable para casi todas las actividades, desde el funcionamiento en el día a día de una empresa hasta leer la prensa para la mayoría de las personas.

Internet se ha convertido en algo tan fundamental e imprescindible como lo es la electricidad, ha transformando el mundo tal y como era cambiando la forma de vivir de la gente.

Cada día que pasa es un día en el que vivimos más interconectados y en el que cada vez se generan más datos. Cada sesenta segundos se visualizan unos 4 millones de vídeos en YouTube, se escuchan unas 750 mil canciones en Spotify o se visualizan cerca de 98 mil horas de video en Netfix.

No solo son más personas las que cada día acceden a servicios en la red sino que también aumentan sustancialmente la cantidad de objetos conectados entre sí. Smart TV, Wearables y luces inteligentes constituyen todo un nuevo concepto que va a volver a cambiar el mundo tal y como lo conocemos.

El internet de las cosas o IOT - por sus siglas en inglés - es un concepto que se basa en la interconexión de cualquier objeto con cualquier otro de su alrededor desde un frigorífico hasta una televisión. El objetivo es hacer que estos objetos se comuniquen entre sí y utilicen esta información para realizar nuevas funciones. Son numerosos los ejemplos de empresas que han hecho del internet de las cosas su principal producto como Car 2 go, Muving o recientemente IKEA con un set de luces inteligentes para colocar en casa.

El desarrollo del internet de las cosas propone un nuevo contexto ante el que deben responderse ciertas preguntas esenciales para su implementación. ¿Qué protocolo es el más adecuado para la comunicación entre sí de estos objetos? ¿Cuál es la manera más eficiente –en términos energéticos- de hacerlo?

Los servicios en la nube son otro pilar fundamental para el internet de las cosas. Este nuevo tipo de servicios implican serias disyuntivas que deben ser analizadas. ¿Qué tipo de servicios se ofrecen? ¿Cuál es el más apropiado para mi aplicación?

El avance de los servicios en la nube es un desarrollo paralelo al de internet de las cosas debido a que actúa sirviendo como soporte para su implementación. Este servicio consiste fundamentalmente en ofrecer soluciones informáticas (servidores, software, bases de datos, análisis de datos, redes...) a través de internet (“la nube”).

## 1.1 Objetivos

Como hemos comentado en la introducción, el uso de IoT así como los servicios en la nube es una tecnología en auge que todavía está por pulir y desarrollar.

En concreto, en este proyecto vamos a implementar un escenario IoT doméstico. Una serie de nodos simulados enviarán datos a una Raspberry-Pi haciendo uso del protocolo CoAP. La Raspberry-Pi se encargará de tratar los datos recibidos y los almacenará en una base datos en la nube, que dependiendo del proveedor se hará de una forma u otra.

En nuestro proyecto se distinguen 2 fases principales:

- Documentación y exposición sobre IoT, el protocolo CoAP y los servicios en la nube.
- Implementación de la solución: Programación de los nodos sensores y de los servicios en la nube.

## 1.2 Escenario del sistema

El sistema implementado (figura 1-1) consiste en una vivienda con dos habitaciones, un servicio, una cocina, un baño y un salón. En concreto, y describiendo un poco más los sensores implantados:

- Un sensor que mide si la luz está o no encendida en cada una de las habitaciones. El consumo energético es una de las principales preocupaciones de los dueños de una casa. Poder conocer el estado de todas las luces de una casa en cualquier momento y desde cualquier lugar es esencial para evitar un consumo indeseado.
- Un sensor de temperatura en el exterior. A través de este sensor se puede monitorizar la temperatura en el exterior en todo momento.
- Un sensor de temperatura en un frigorífico. A través de este sensor somos capaces de saber qué temperatura exacta tiene nuestro congelador o frigorífico. No son pocas las veces que uno se da cuenta de que su congelador ha dejado de funcionar demasiado tarde. Midiendo la temperatura en su interior podemos conocer cualquier comportamiento anómalo.
- Una Raspberry-Pi que actúa como pasarela entre los sensores antes descritos y una base de datos en la nube. La Raspberry-Pi recopila los datos generados por los sensores, los trata y los almacena en la nube.

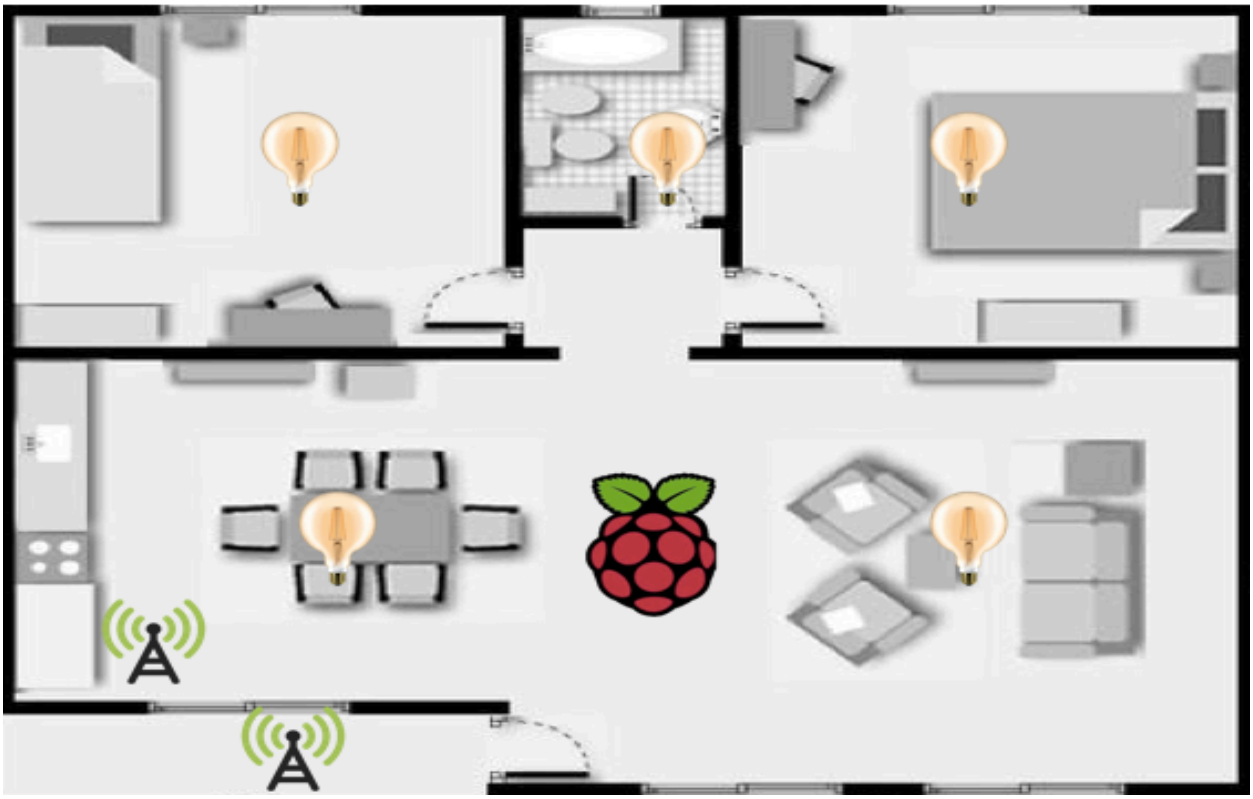

Figura 1-1 Escenario vivienda

En la figura 1-2 se ilustra la implementación lógica del sistema. Los nodos sensores se comunican con la Raspberry-Pi haciendo uso del protocolo CoAP. La Rapsberry-Pi se comunica con estos sensores haciendo uso del mismo protocolo, recopila datos y los publica en la nube. Los datos se publican en la nube con el objetivo de que sean accesibles para cualquier otra aplicación que los necesite, como por ejemplo una App o una web. Para la implementación de esto último se han elegido los tres principales proveedores de servicios en la nube: Google, Microsoft y Amazon.

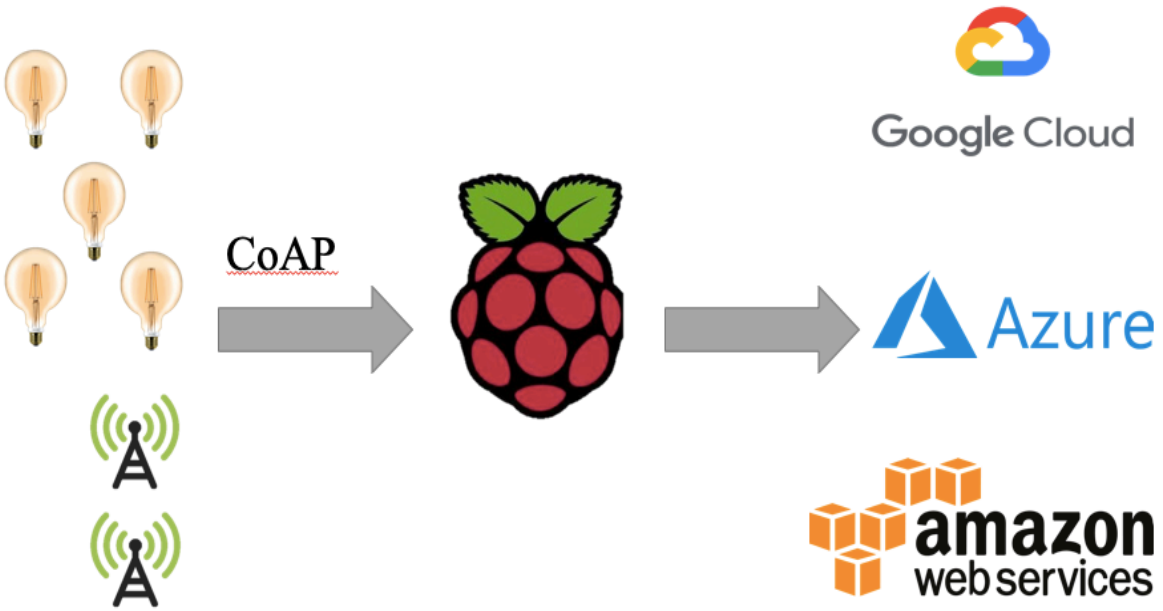


Figura 1-2 Escenario Lógico

# 2 INTERNET DE LAS COSAS

*The advance of technology is based on making it fit in so that you do not really even notice it, so it is part of everyday life.*

Bill Gates

El internet de las cosas es la interconexión de objetos digitales que son cotidianos mediante internet.

Esto permite el intercambio automático de información entre los propios dispositivos o con terceros sin necesidad de actividad humana.

Según un estudio recientemente publicado por Gartner, se prevé que en 2020 haya más de 20 billones de dispositivos conectados. Estos dispositivos no tienen que ser necesariamente ordenadores, sino que pueden ser dispositivos funcionales como por ejemplo un frigorífico, un coche o el motor de un avión. Con esta previsión, es seguro que el internet de las cosas generará un enorme impacto en la economía, transformando industrias enteras al mundo digital, creando nuevas oportunidades de negocio o cambiando para siempre la relación con los consumidores.

## 2.1 Características

Las aplicaciones para estos dispositivos conectados a internet son muchas y van en aumento. Según Gartner hoy en día solo un 35% de las empresas hacen uso de algún producto que emplee el internet de las cosas. Para 2020 un 65% de las empresas habrá ya implementado de alguna manera esta tecnología.

Para hacernos una idea de cuales son hoy en día las aplicaciones más notables del internet de las cosas recurrimos a un estudio publicado por IoT Analytics. En dicho estudio se valora el impacto en Google, Twitter y Linkedin de las áreas en cuestión. Las más destacadas son:

- Smart Home: Hace referencia a todas las tareas que se pueden automatizar en una casa mediante la interconexión entre sí de los distintos objetos. Aportando distintos servicios como sistemas de gestión energética o de seguridad. Saber si te has dejado alguna luz encendida, si falta algún alimento en tu frigorífico son algunos de los servicios que ofrece este campo.
- Wearables: hace referencia a los dispositivos que llevamos encima de forma cotidiana; relojes y gafas inteligentes son algunos de los productos que se ofertan hoy en día, y van en aumento.
- Smart City: Se define como el uso de dispositivos interconectados para gestionar y administrar los diferentes servicios de una ciudad, como pueden ser transporte público o privado, eficiencia energética, protección civil o el uso del espacio público. Una ciudad inteligente es capaz de captar las demandas de los ciudadanos y generar una solución como por ejemplo gestionar el tráfico en tiempo real.
- Smart Grid: También se denomina redes eléctricas inteligentes. Consiste en aplicar el internet de las cosas a la generación, almacenamiento, distribución o consumo de energía. El fin de la Smart grid es conseguir un suministro eléctrico más fiable, eficiente y sostenible.

- Industrial internet: Se define como el uso del internet de las cosas en la empresa. El impacto económico así como la transformación digital de industrias enteras resultará incuestionable. La experiencia de usuario cambiará para siempre.
- Connected car: El uso del internet de las cosas es ya una realidad en este campo y son muchos los servicios que hoy día se ofrecen, como por ejemplo el pago por el uso, coche compartido o el coche completamente autónomo.
- Connected Health: Se define como el uso del internet de las cosas para ofrecer servicios relacionados con la salud de forma remota. Alertar a los servicios de urgencias automáticamente en caso de infarto, avisarle de que tiene la tensión alta o de que necesita beber agua pueden ser alguno de los usos que veamos en el futuro.
- Smart retail: Se define como el uso del internet de las cosas en el ámbito del comercio. La experiencia de usuario así como la interacción con el mismo son los roles fundamentales. Claro ejemplo es la nueva tienda física de amazon en la que no hay cajeros, solo coger el producto y salir andando por la puerta.
- Smart supply chain: Se define como el uso del internet de las cosas para mejorar las operaciones logísticas de una empresa. Son muchos los campos de aplicación en este apartado; Previsión de inventarios, seguimiento exhaustivo del proceso, relación con el vendedor o prever operaciones de mantenimiento son algunas de ellas.
- Smart farming: Se define como el uso del internet de las cosas en la agricultura o la ganadería. Poder predecir las condiciones meteorológicas o poder saber en tiempo real si una vaca embarazada está a punto de parir son algunos ejemplos.

Estas son (figura 2-1) algunas de las aplicaciones más famosas. No son todas y probablemente si en 5 años volvemos a leer este capítulo es muy probable que hayan cambiado por completo.

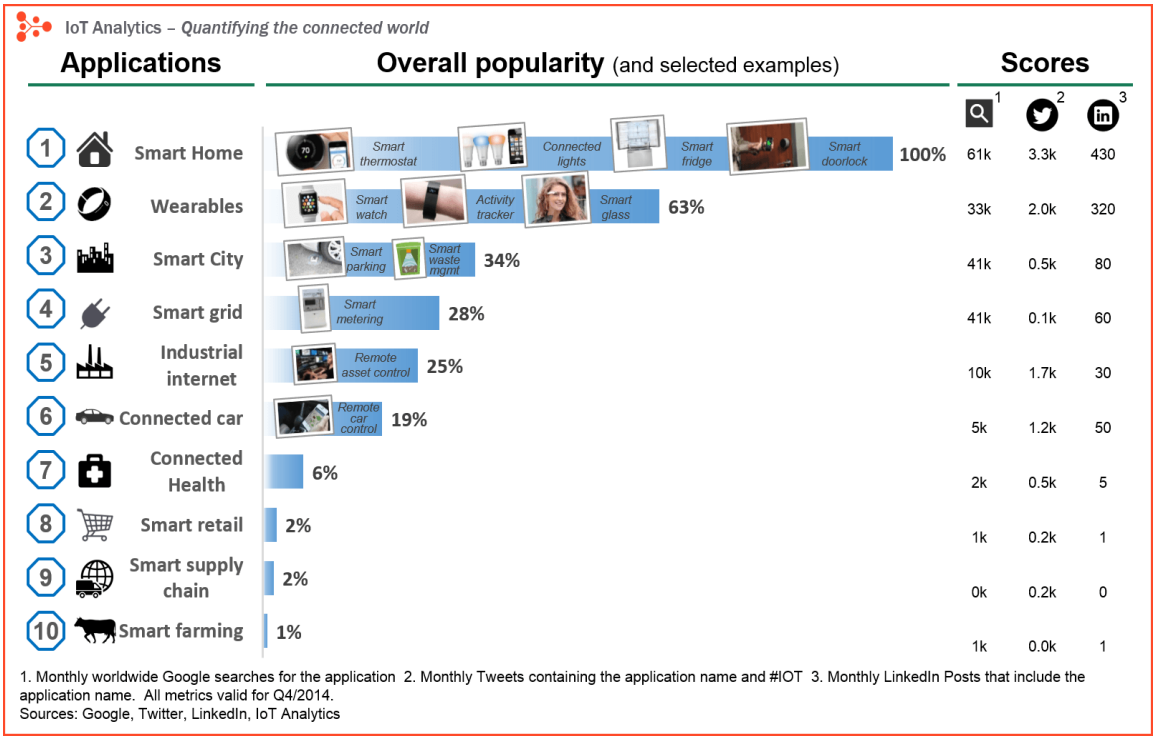


Figura 2-1 Informe aplicaciones IoT más conocidas

## 2.2 Desarrollo

Llegados a este punto, hemos definido el internet de las cosas y hablado de sus principales aplicaciones pero, ¿Cómo se implementan? ¿Cuál es la situación actual de las industrias?

Según un estudio publicado por Hewlett Packard, en el que se realizan una serie de preguntas a los líderes de las principales empresas e industrias que han adoptado el internet de las cosas se pueden sacar las siguientes conclusiones:

- Un 82% ha visto un incremento en la eficiencia del negocio desde que apostó por el internet de las cosas.
- Un 81% creen que su organización es ahora más eficiente.
- Un 73% ha experimentado un ahorro en costes.
- Un 78% dice haber mejorado la experiencia de sus consumidores.
- Un 72% dice haber aumentado sus beneficios.
- Un 72% ha visto mejorada la visibilidad de los procesos en la organización.

No es casualidad que con estos datos sobre la mesa sean cada vez más las compañías que apuesten por implementar el internet de las cosas. Se espera que el número de empresas que incorporen esta tecnología alcance el 85% en 2019 y el 40% cree que el uso del internet de las cosas les hará expandir su mercado. La transformación es imparable y día tras día cada vez son más las empresas en sumarse a la ola.

La implementación del internet de las cosas en las empresas tampoco es arbitrario y a menudo no resulta una tarea fácil. Requiere un estudio pausado y coherente con los objetivos de la empresa y el contexto de la industria. Este proceso se puede resumir en una serie de subprocesos (figura 2-2) como los que siguen:

- Identificar los objetivos del negocio.
- Desarrollar los casos de usos para el internet de las cosas.
- Desarrollar un mapa de los pasos a seguir para su implementación.
- Implementar provisionalmente la solución propuesta.
- Con los resultados del paso anterior, actualizar el mapa de pasos a seguir en la implementación.
- Por último, implementar la solución final.

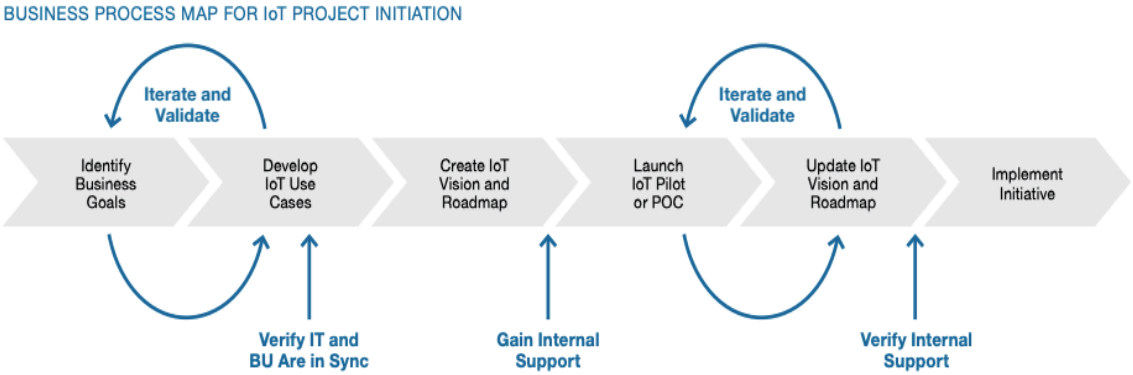


Figura 2-2 Proceso desarrollo del producto para IoT

## 2.3 Principales barreras en su implementación

Cada vez que a lo largo de la historia ha irrumpido una nueva tecnología siempre han surgido barreras para su implementación. Falta de medios, de accesibilidad o quizás poca rentabilidad son algunas de las barreras con las que más de una idea o tecnología se ha encontrado a lo largo de la historia. Pero, ¿cuáles son las principales barreras a las que se enfrenta el internet de las cosas?

A la hora de implementar el internet de las cosas suelen ser 5 los componentes funcionales que se estudian:

- Seguridad.
- Privacidad.
- Costes.
- Eficiencia.
- Facilidad de uso.

Es evidente que con 25 billones de dispositivos interconectados y con la capacidad de procesamiento actual va a haber pocos datos que no podamos conocer. Una persona en un día genera más datos de los que se imagina: cuánto tiempo pasa delante de un escaparate, cuánto tiempo pasa conduciendo, lo que hace en su tiempo libre… Todos estos datos que hasta ahora no se podían o eran muy difíciles de medir, en el futuro no lo serán. Será posible conocer sus gustos, sus preferencias o qué piensan. Será posible, con la suficiente cantidad de datos, predecir lo que hará una persona en un día normal.

La privacidad siempre ha sido una de las preocupaciones del ser humano y nunca ha estado tan en peligro como con el internet de las cosas. Es sin duda una de las grandes barreras a las que se enfrenta esta tecnología y uno de los grandes temas de debate.

Si la privacidad es una de las principales preocupaciones a la hora de implementar el internet de las cosas, otra es la seguridad. Con la llegada de internet el ser humano ha ido paulatinamente trasladando su vida a internet. Las tarjetas del banco, la base de datos de clientes de una empresa o el navegador en tiempo real que le indica al coche el camino más corto en función del tráfico son algunos de los servicios que hoy se dan por fundamentales.

Según un estudio publicado por Gartner, la seguridad jugará un papel fundamental en la implementación del internet de las cosas y es unas de las principales preocupaciones de las empresas. Según este estudio un 32% de los líderes en empresas IT consideran la seguridad como la principal barrera para su desarrollo. En 2020 se prevé que el 25% de todos los ataques se concentren en el internet de las cosas. Esto preocupa aún más si se le suma que sólo se dedicará el 10% del presupuesto a la seguridad.

Poniendo ambas cuestiones en perspectiva es seguro decir que el internet de las cosas jugará un papel importante en la sociedad, pero cómo se resolverá el problema de la privacidad y de la seguridad no está tan claro. No son pocos los casos que hoy en día han tenido problemas de seguridad haciendo que millones de datos se hagan públicos.

El internet de las cosas es y será una realidad pero aún queda camino por hacer, debates por tener y soluciones que plantear en cuanto a la privacidad y a la seguridad se refiere.

# 3 CLOUD COMPUTING

*I don't need a hard disk in my computer if I can get to the server faster… carrying around these non-connected computers is byzantine by comparison.*

Steve Jobs

La informática en la nube es, dicho de manera sencilla, la oferta de soluciones informáticas (Servidores, software, bases de datos, análisis de datos, redes…) a través de internet ("la nube"). Normalmente el servicio suele ser bajo demanda pagando solo por los recursos que se necesitan. De esta forma se reducen costes operativos, los servicios se pueden escalar según las necesidades del negocio y permite agilizar la implementación de la infraestructura.

El término de "la nube" hace tiempo que dejó ser un mero fenómeno meteorológico para convertirse en otro bien distinto. Lejos de ser un futuro lejano es ya un presente que está transformado el mundo. Según un estudio realizado por RightScale, en el que se consulta a un gran número de expertos y líderes en el tema, el 26% de las empresas gastan de media unos 6 millones de dólares al año mientras que el 52% gastan al menos 1.2 millones de dólares (figura 3-1).

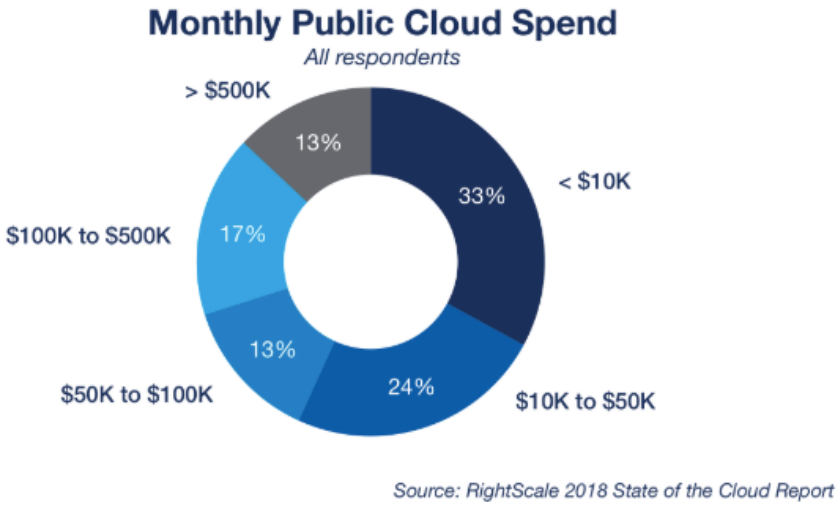


Figura 3-1 Estadística gasto mensual nube pública

Pero fijarnos solo en el dinero que emplean las empresas en la nube no es suficiente para comprender la magnitud del concepto del que hablamos. No es, ni será importante solo por el dinero que genera, sino por la adopción universal hacia la que se dirige este servicio. Aparte del volumen de negocio debemos también fijarnos a la velocidad a la que cambia. ¿Cuál es su expectativa de futuro? En el mismo estudio citado anteriormente se cuestionan algo parecido y los resultados son llamativos. Las empresas no solo hacen ya uso de la nube sino que planean rápidamente incrementar su uso (figura 3-2). El 20% espera doblar su presupuesto en este tipo de servicios, mientras que el 71% planea incrementar el gasto en al menos un 20% (figura 3-3). Lo más llamativo es quizás que el 82% de las empresas planean  de una u otra manera aumentar su posición en la nube.

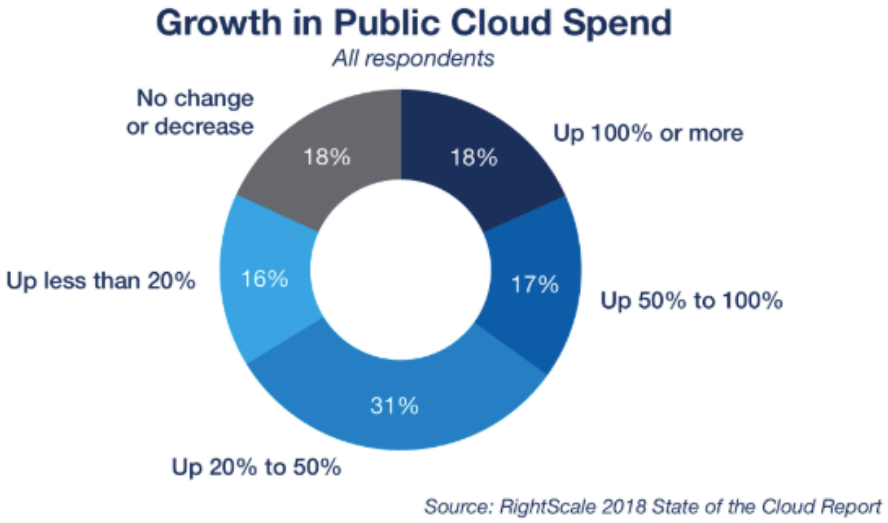


Figura 3-2 Estadística crecimiento nube pública

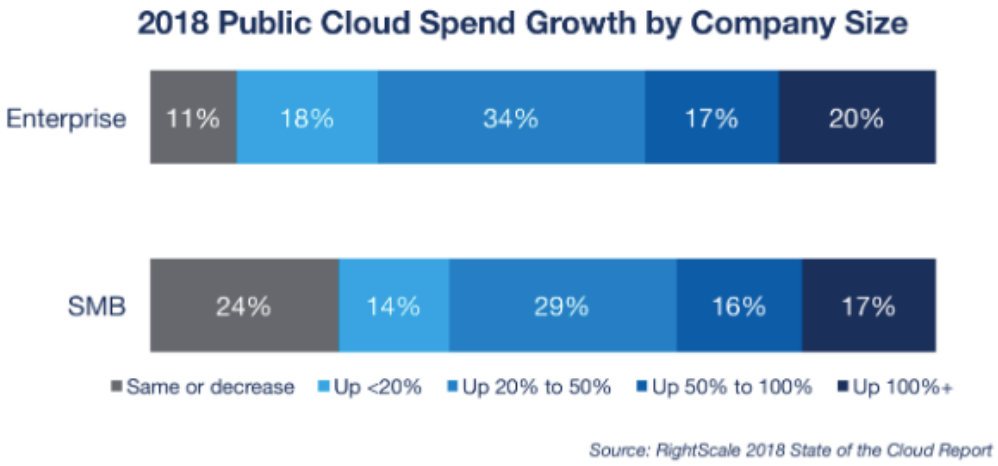


Figura 3-3 Estadística crecimiento del gasto según empresas

Para desarrollar aún más estos conceptos cabe preguntarnos como se traduce lo anteriormente expuesto en oportunidades de negocios o de trabajo. Por lo pronto podemos sacar algunas conclusiones a simple vista. Por ejemplo, hemos observado que tanto el gasto como las empresas que van a usar servicios en la nube en un futuro van a aumentar. Es fácil concluir de aquí que esto se va a traducir en un incremento en la oferta de trabajo en este campo. Esta conclusión resulta trivial, pero no resulta tan trivial la decisión de las empresas a la hora de usar esta oferta laboral. ¿Se crearán puestos específicos relacionados? ¿Se sumará a la lista de directivos el experto en cloud computing? O por el contrario ¿Se externalizará el servicio a empresas dedicadas a ello? La experiencia en la historia nos dice que ambas pasarán, de hecho ya ocurre. Según Microsoft, en 2015 había unos 14 millones de empleos solo relacionados con la nube. Según el foro económico mundial será uno de los 8 empleos que una empresa contratará en 2020.

Si analizamos los datos que nos aporta RightScale no hay lugar a dudas. El 44% de las grandes empresas ya tienen un equipo que se dedica a la nube y el 25% planea tener uno (figura 3-4).

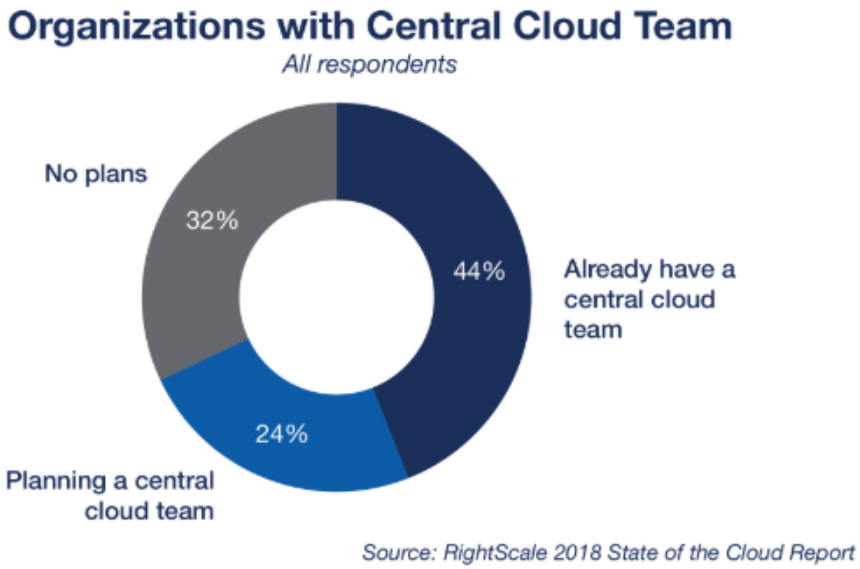


Figura 3-4 Estadística empresas con equipo especializado en la nube

De acuerdo con el estudio publicado por SpiceWorks la opción que lidera las razones por las que las compañías optan por este tipo de servicios es la necesidad de actualizar su infraestructura tecnológica (figura 3-5). Otras razones; que aunque no las primeras también son mayoritarias, son la necesidad de priorizar proyectos IT o la preocupación por la seguridad.

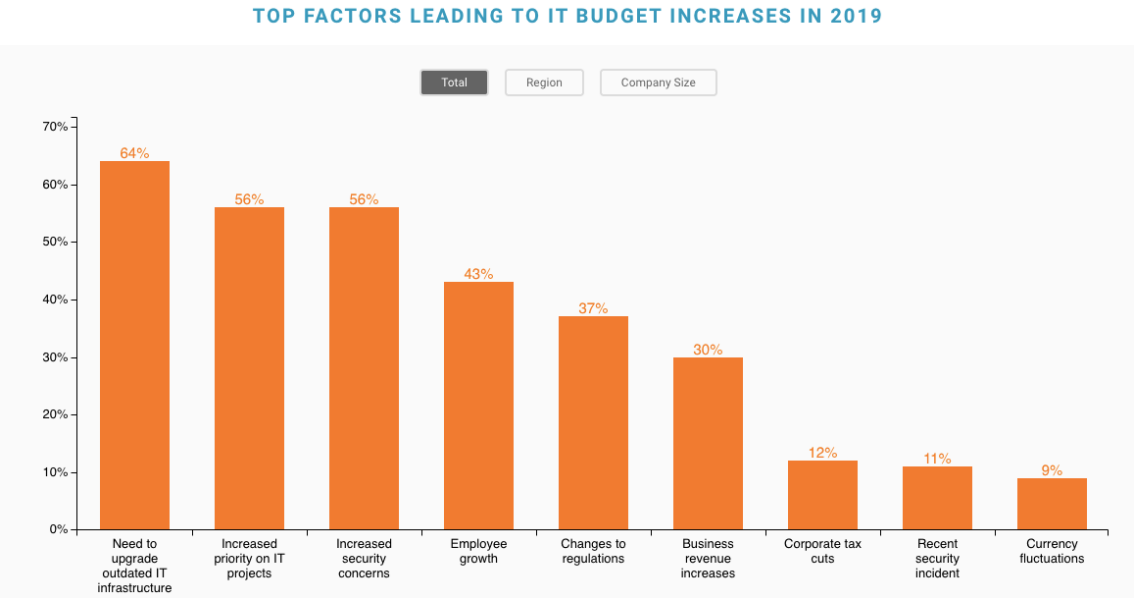


Figura 3-5 Principales motivos adopción de la nube

Por último, para terminar de describir la situación actual: ¿Quiénes son los principales actores en la nube?

Amazon Web Services lidera el mercado con el 64% de aplicaciones funcionando en este entorno, Microsoft y Google le persiguen con un 45% y 18% respectivamente (figura 3-6). Están también pero en una posición menos protagonista IBM, VMware, Oracle o Alibaba.

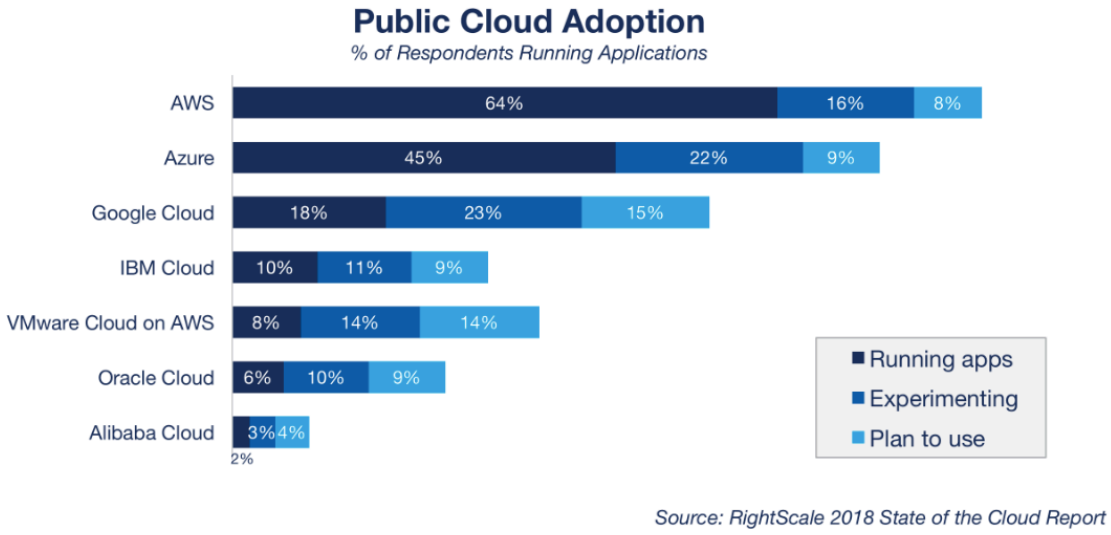


Figura 3-6 Reparto del mercado de la nube

## 3.1 Tipos de nube

Para implementar nuestra aplicación debemos conocer que hay distnitos modelos a la hora de implementar la nube. Son definidos fundamentalmente por su tamaño o accesibilidad. En primer lugar, deberemos definir el tipo de desarrollo en la nube o la arquitectura en la que se implementaran nuestros servicios. Hay 3 tipos de nube:

- Nube pública.
- Nube privada.
- Nube híbrida.

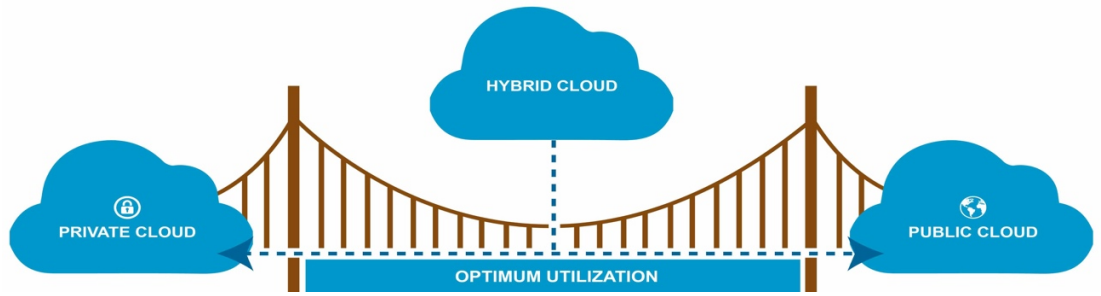


Figura 3-7 Tipos de nube

### 3.1.1 Nube pública

La nube pública es la forma más común en este servicio. La principal característica de la nube pública es que es propiedad de otro proveedor de servicios, que la administra y la gestiona ofreciendo los recursos a través de internet.

En la nube pública, todo el hardware así como la infraestructura y el software son propiedad del proveedor y los recursos normalmente son compartidos por otros usuarios.

Algunos ejemplos de proveedores de estos servicios son Amazon o Microsoft.

### 3.1.2 Nube privada

Una nube privada hace referencia a recursos en la nube que utiliza exclusivamente una empresa o una organización. La implementación de una nube privada puede ser por ejemplo el centro de datos de cualquier empresa. Una nube es privada si se mantiene dentro de una red privada.

Este es el tipo de servicio implementantado por empresas que por diversas políticas no quieren compartir la infraestructura con más usuarios, bien por seguridad, privacidad u otro tipo de política interna.

Las empresas que optan por este tipo de servicio, al ser los únicos usuarios, podrán tener gran control sobre la implementación adaptándola perfectamente a sus requisitos. Las empresas cuyo componente crítico sea controlar el entorno lo máximo posible son las principales interesadas en este servicio.

### 3.1.3 Nube híbrida

La nube híbrida es aquella que comparte una parte pública y una privada enlazadas que permite compartir datos y aplicaciones.

Al compartir ambas nubes una empresa podrá gozar de la flexibilidad y escalabilidad que aporta la parte pública y de la posibilidad de optimizar la infraestructura que aporta la parte privada.

### 3.1.4 Aplicaciones en la nube

Hemos definido los tres principales tipos de nubes y entornos que se implementan hoy en día. La realidad es que la mayoría de empresas requieren de soluciones complejas, que no son triviales y que de ellas dependen en buena parte el correcto funcionamiento del negocio.

En la mayoría de los casos las soluciones no son únicas y normalmente se suelen adoptar distintos tipos de nube. Según un estudio realizado por RightScale, el 81% de las empresas optan por una solución con distintas nubes. En algunos casos públicas, en otros  privadas y en el 51% de los casos soluciones híbridas (figura 3-8).

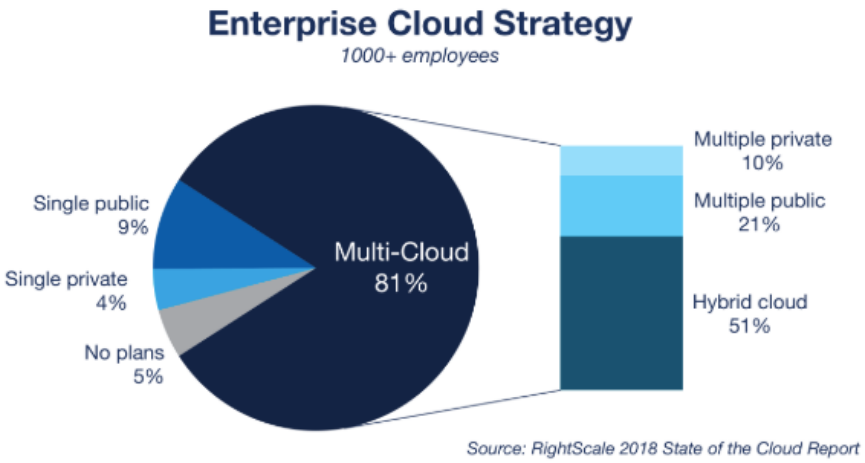


Figura 3-8 Estrategia empresarial en la nube

Con los datos ya presentados, sabemos que cada vez van a ser más y más las empresas que implementen este servicio. ¿Qué solución van a adoptar? Con los datos del estudio antes presentado y contrastándolos con los de años anteriores podemos sacar como conclusión que cada vez son más las empresas que apuestan por una solución hibrida. En 2017 esta era la opción elegida por el 51% de las empresas mientras que en 2018 se espera que sea el 58% (figura 3-9).

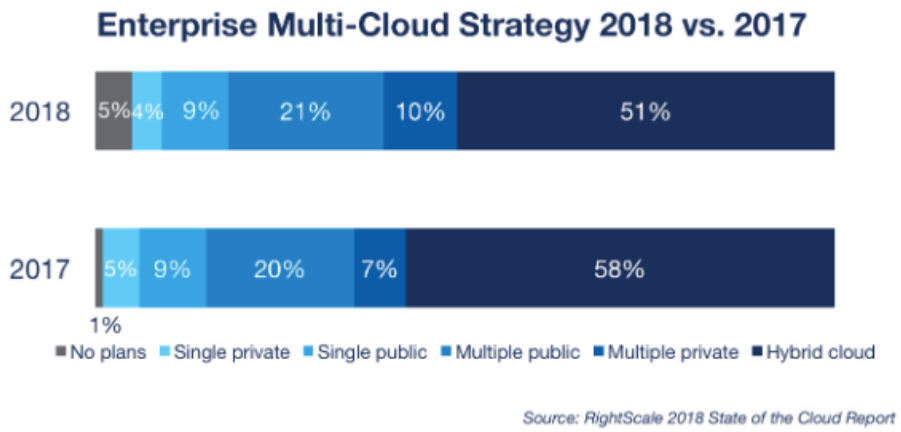


Figura 3-9 Comparativa estrategia empresarial en la nube

## 3.2 Características y ventajas de la nube

### 3.2.1 Servicio bajo demanda

Se refiere al servicio que provee el propietario de la nube en el que habilita una serie de recursos a petición del usuario, en el momento que desee y de la forma solicitada. Un usuario puede así aumentar su capacidad de almacenamiento y el tiempo de procesamiento cuando lo necesite.

La nube permite que implementar este tipo de servicios sea algo sencillo y flexible. Normalmente el usuario es capaz de hacer todo esto a través de un simple navegador.

El servicio bajo demanda es considerado una de las características claves del éxito de la nube. Implica que un usuario sea capaz de decidir constantemente que recursos necesita y de que manera.

De la forma tradicional una empresa tenía que hacer una planificación de los recursos que iba a necesitar y estar continuamente haciendo una valoración sobre si lo que ya tiene es suficiente. Una empresa ahora por ejemplo podría contratar recursos en función de variables, aportándole una sencillez y una elasticidad que antes no tenía.

### 3.2.2 Recursos compartidos

Los recursos informáticos del proveedor son compartidos para servir a distintos usuarios en un modelo de multi propiedad en el que se usan recursos tanto virtuales como físicos bajo demanda, como se observa en la figura 3-10. Hay una independencia entre donde se encuentran los recursos y el usuario; este último no conoce donde están, salvo a un nivel de abstracción alto como puede ser elegir el país o la zona geográfica. Algunos recursos compartidos son por ejemplo el procesamiento, almacenamiento etc…

A la hora de implementar y de configurar los distintos entornos en este proyecto, como ya se explicará más adelante, los distintos proveedores nos ofrecían la posibilidad de elegir el país en el que se iban a situar nuestros recursos.

Compartir recursos es una característica esencial en la nube pública. Juega un papel fundamental en la eficiencia y el alto rendimiento de este tipo de servicios.

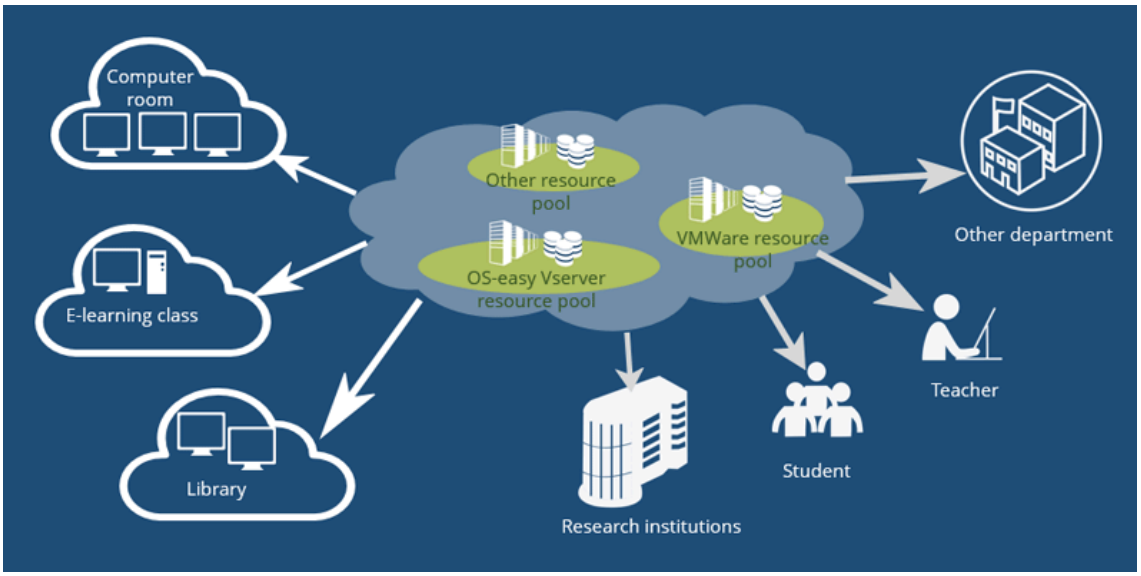


Figura 3-10 Recursos compartidos

### 3.2.3 Acceso en la red

Los recursos son accesibles a través de la red haciendo uso de clientes sencillos como pueden ser por ejemplo un navegador, una aplicación Android…

Esta propiedad es una consecuencia directa de que el cliente no conozca ni tenga control sobre la localización exacta de los recursos. Que sea accesible desde cualquier parte, de una manera rápida y sencilla constituye una poderosa herramienta que descentraliza en una empresa el acceso a los recursos (figura 3-11).

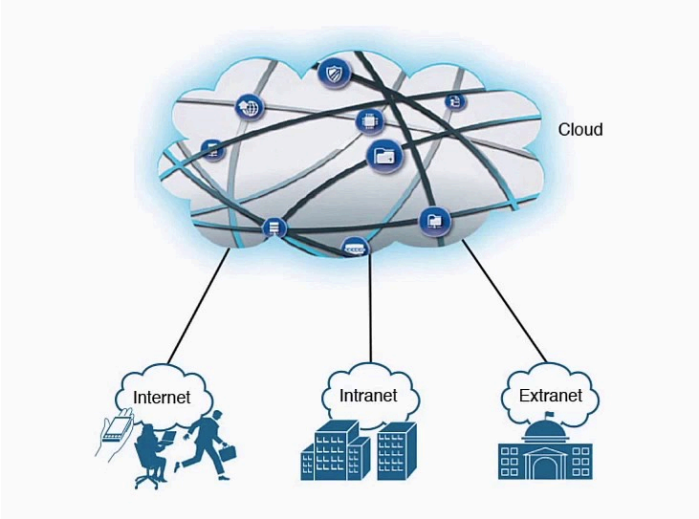


Figura 3-11 Acceso en la red

### 3.2.4 Elasticidad

Los recursos pueden cambiar en el tiempo según las necesidades. Una aplicación escalable puede demandar más recursos o liberarlos en función de sus necesidades (figura 3-12).

La elasticidad hace referencia a la capacidad de expansión y contracción de los recursos.

Esta característica es fundamental para el rendimiento de muchas aplicaciones implementadas en la nube. Imaginen, por ejemplo, que tienen una aplicación web y que cuando ésta recibe un número de visitas imprevistas los recursos se escalan para poder abastecer al número de usuarios. Una vez más le da el poder al usuario y a la empresa de poder planificar sus recursos bajo demanda, con una situación y contexto determinados.

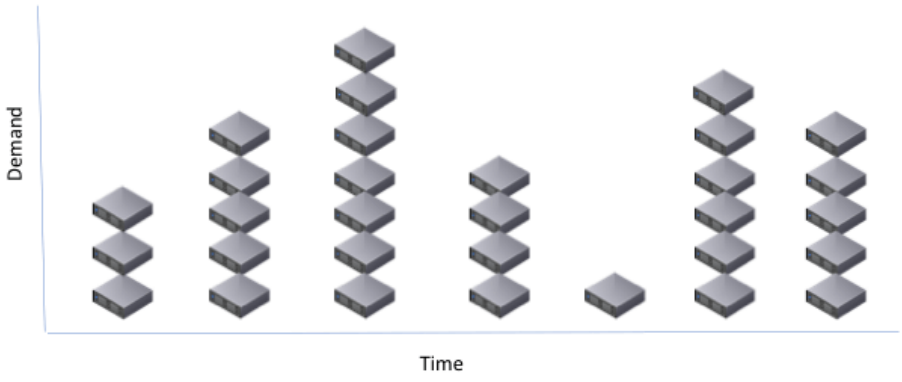


Figura 3-12 Elasticidad

### 3.2.5 Medición y gestión del servicio

Los sistemas en la nube controlan y gestionan automáticamente los recursos, optimizando las operaciónes en base a distintos tipos de métricas basadas en tiempos de procesamiento, almacenamiento… (Figura 3-13).

Poder hacer mediciones de los servicios que tu aplicación implementa es una herramienta que indiscutiblemente es imprescindible para optimizar recursos y hacerlo de una forma eficiente. En función de estas métricas podemos por ejemplo tomar la decisión de escalar una serie de recursos, o simplemente de planificar que necesitamos otro tipo de recursos.

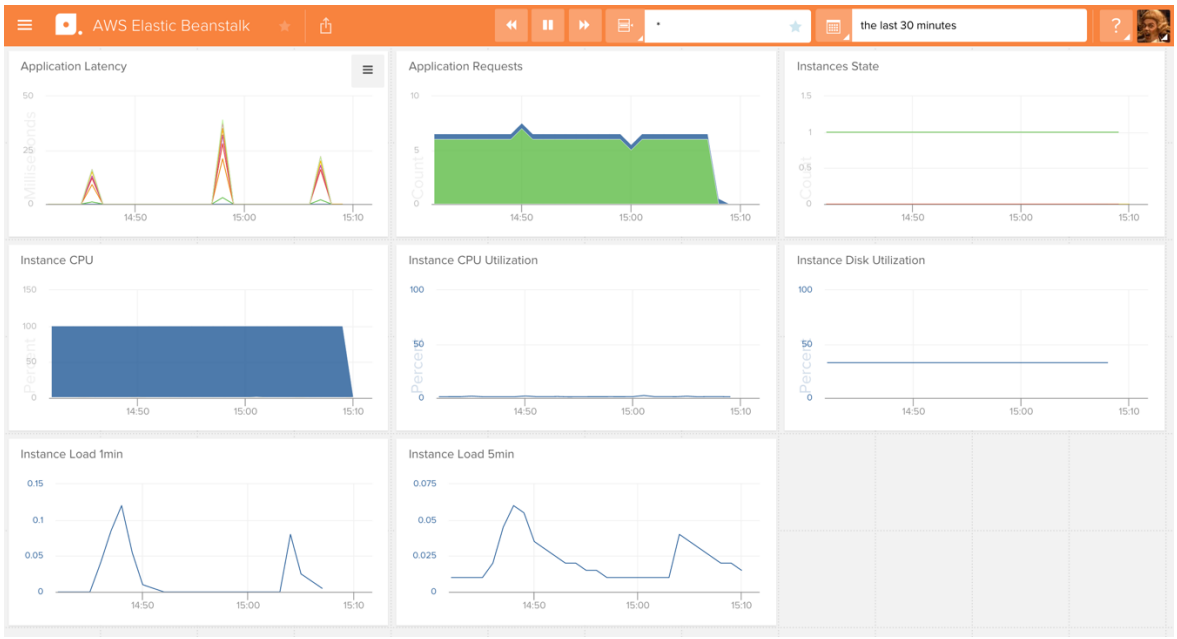


Figura 3-13 Medición del servicio

## 3.3 Principales barreras en su implementación

Lo primero que debemos preguntarnos a la hora de analizar una tecnología novedosa es cuales son los principales retos a los que se enfrenta y qué barreras debe superar para poder tener éxito.

En un estudio recientemente publicado por logic monitor (figura 3-14) se pueden extraer, a grandes rasgos, varias conclusiones:

- La seguridad es lo que más preocupa. La incógnita de cómo de seguros están mis datos es una constante en la mayoría de las empresas encuestadas.
- La privacidad es otro de los grandes desafíos, no solo es ya la seguridad sino el derecho a la privacidad lo que cada vez inquieta a más gente.
- Falta de personal experto en el tema. La tecnología crece más deprisa que el tiempo en el que se forma capital humano y esto puede ser un factor de ralentización en el crecimiento de la tecnología.

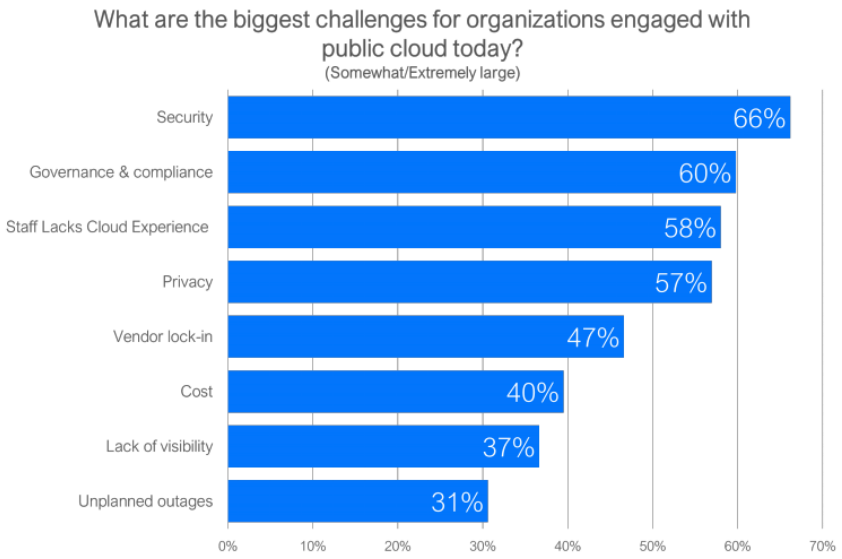


Figura 3-14 Barreras de implementación

## 3.4 Servicios en la nube

La nube prevé la implementación de todo tipo de aplicaciones. Incluso para cada tipo de aplicación puede que haya diversas formas de implementarlas.

Son tres los principales servicios que ofrece la nube (figura 3-15), y se caracterizan por el nivel de recursos que deseemos gestionar.

- **IaaS o infrasestructura como servicio** se caracteriza porque el proveedor de servicios se encarga de controlar hasta la capa del sistema operativo y el usuario se encargará de implementar su aplicación a partir de aquí.
- **PaaS o plataforma como servicio** se caracteriza porque el usuario ya puede usar algún tipo de aplicación establecida previamente por el proveedor para implementar su aplicación.
- **SaaS o software como servicio** se caracteriza porque el usuario se beneficia de una aplicación ya implementada por el proveedor.

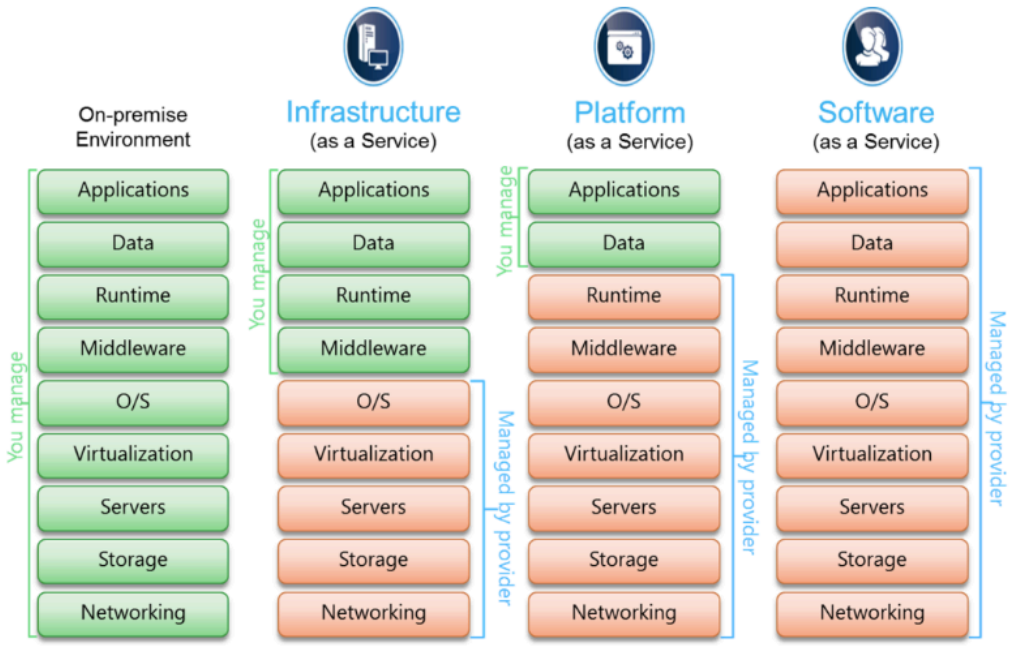


Figura 3-15 Tipos de servicio

Cabe destacar que cada modelo de servicio tiene sus propias ventajas e inconvenientes que destacaremos más adelante. Por ejemplo el usuario que usa la infraestructura como servicio tendrá que tener un control más minucioso sobre su aplicación que el que usa software como servicio, no obstante le permitirá tener un mayor grado de libertad y flexibilidad.

### 3.4.1 Infraestructura como servicio


Figura 3-16 Infraestructura como servicio

Es un tipo de servicio en la nube en el que el proveedor oferta a los usuarios recursos como servidores, almacenamiento o de red. El usuario usa su propia plataforma o aplicación haciendo uso de la infraestructura del proveedor. (Figura 3-16).

Sus principales características son:

- Pago por el uso de recursos bajo demanda.
- La estructura es escalable en función de la necesidad de almacenamiento o procesamiento.
- Ahorro en compra y mantenimiento de una infraestructura propia.
- Habilita la virtualización de labores de administración.
- Permite innovar con rápidez debido a su flexibilidad.

Los usos típicos para este servicio son:

- Hospedaje web: puede resultar más económico que de una forma tradicional.
- Informática de alto rendimiento que necesite procesar gran cantidad de datos.
- Desarrollo y pruebas de aplicaciones que se implementan y cambien rápidamente.

.

### 3.4.2 Plataforma como servicio

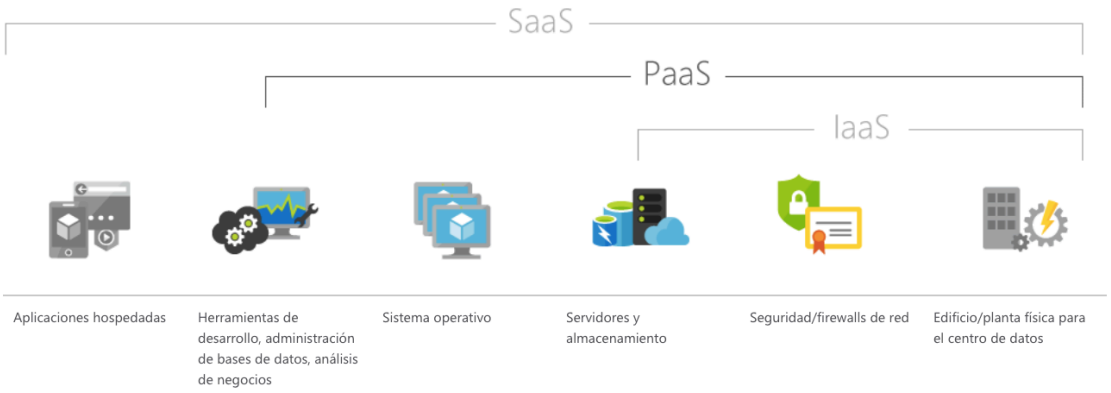


Figura 3-17 Plataforma como servicio

Este tipo de servicio se basa en el que el proveedor oferta un entorno completo en el que un usuario puede desarrollar, gestionar e implementar su plataforma o aplicación. Además de servidores, almacenamiento y sistema operativo el usuario puede hacer uso de algunas herramientas pre-configuradas por el proveedor. (Figura 3-17).

Sus principales características son:

- Reducir el tiempo de programación.
- Desarrollar para más de una plataforma con más facilidad.
- Colaboración en equipos de desarrollo distribuidos geográficamente.
- Uso de herramientas sofisticadas a un precio asequible.
- Agregar más funcionalidades sin necesidad de contratar a más personal.

Los usos típicos para este servicio son:

- **Marco de desarrollo:** PaaS permite a los desarrolladores crear aplicaciones haciendo uso de componentes de software integrados.
- **Análisis empresarial:** Las empresas tienen a su alcance métricas y datos generados por las aplicaciones que pueden usar en su beneficio.
- Otro tipo de servicios como seguridad, flujos de trabajo…

### 3.4.3 Software como servicio


Figura 3-18 Software como servicio

Este tipo de servicio se caracteriza porque el proveedor oferta a los usuarios un software basado implementado en la nube. Los usuarios no tienen que instalar absolutamente nada, solo hacer uso del software ofrecido por el proveedor a través una interfaz web normalmente. (Figura 3-18).

SaaS ofrece una solución de software integral donde normalmente se paga por el uso. Todos los recursos implicados como servidores u otros equipos son gestionados por el proveedor.

Sus principales características son:

- Obtener acceso a aplicaciones sofisticadas. Algunas de estas pueden ser el correo electrónico o el uso de editores de texto en la nube.
- Pagar solo por lo que se usa.
- Software de cliente gratuito. En la mayoría de los casos puede ser un navegador web o una simple app.
- Facilitar el uso a los empleados.
- Obtener datos de la aplicación desde cualquier parte.

Los usos típicos para este servicio son:

- Correo electrónico como por ejemplo Gmail o Outlook 365.
- Editores de textos como google drive.
- Calendarios, almacenamiento de fotos…

Según un estudio publicado por Gartner (figura 3-19) la mayor parte del aumento del gasto se producirá en la oferta de infraestructuras como servicio con un 23.31%. Aplicación y software como servicios también aumentaran sus cuotas un 18,24% y un 15,65% respectivamente.

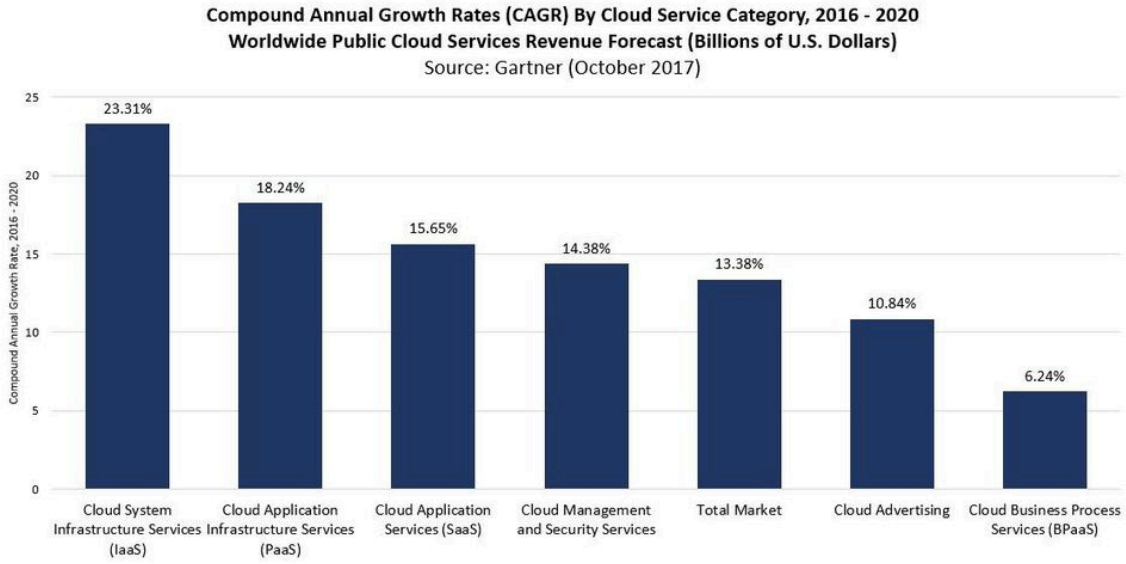


Figura 3-19 Crecimiento de los distintos servicios

# 4 CoAP: PROTOCOLO DE APLICACIÓN RESTRINGIDA

*By giving people the power to share, we are making the world more transparent*

Mark Zuckerberg

CoAP es un protocolo de aplicación que se especializa en la gestión de nodos sensores. Está diseñado para sensores de baja potencia. Suele implementarse en aplicaciones máquina a máquina M2M – por sus siglas en inglés - siendo de especial utilidad en el internet de las cosas, en adelante IOT.

Al igual que en HTTP, la interacción se compone de un modelo cliente/servidor y está basado en el uso del modelo REST: el servidor habilita una serie de recursos a través de una URL y los clientes pueden acceder a estos recursos usando métodos implementados como POST, GET, PUT y DELETE. Una de sus mayores diferencias con HTTP es que utiliza UDP en vez de TCP.

No cabe duda que el IOT jugará un papel determinante en el futuro donde habrá billones de nodos interconectados; que CoAP y HTTP compartan el modelo REST hace que sea muy fácil interconectarlos usando una pasarela o proxy; de este modo un cliente web puede acceder a un sensor sin ni siquiera saberlo (figura 4-1).

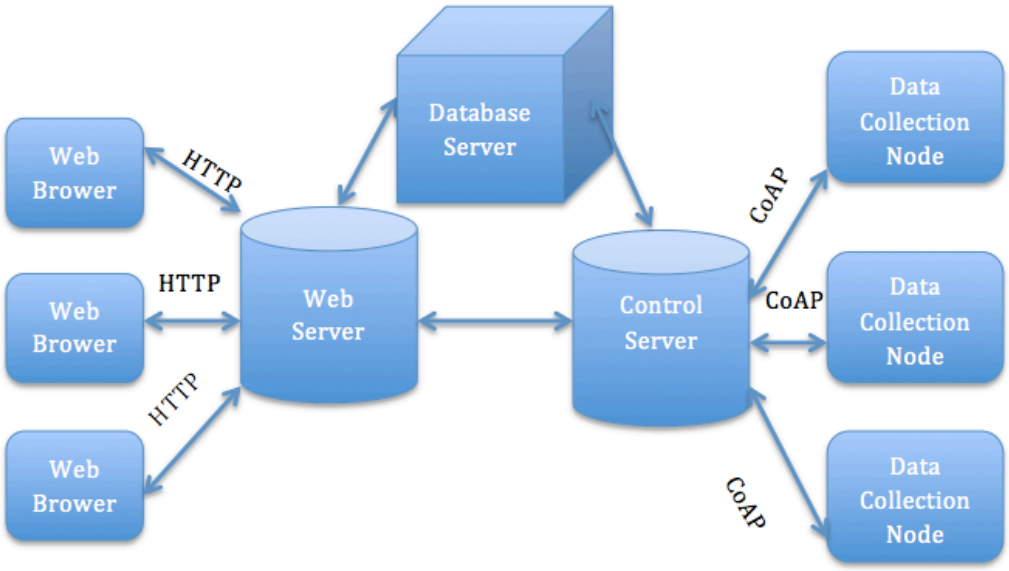


Figura 4-1 Esquema general

Dada la enorme cantidad de sensores que habrá en el futuro la eficiencia con la que compartan la información resulta crucial. CoAP está diseñado de cara a nodos-sensores que usan microcontroladores de 8 bits con pequeñas cantidades de ROM Y RAM.

Uno de los objetivos de CoAP es el diseño de un protocolo genérico para redes de sensores restringidos y optimizado para aplicaciones M2M (machine-to-machine).

Podríamos pensar en CoAP como una aproximación de dos capas: una donde se procesan las peticiones o respuestas y otra para gestionar los mensajes asíncronos de UDP. Sin embargo CoAP es un protocolo de una sola capa y estas dos características son definidas en la cabecera del mensaje.

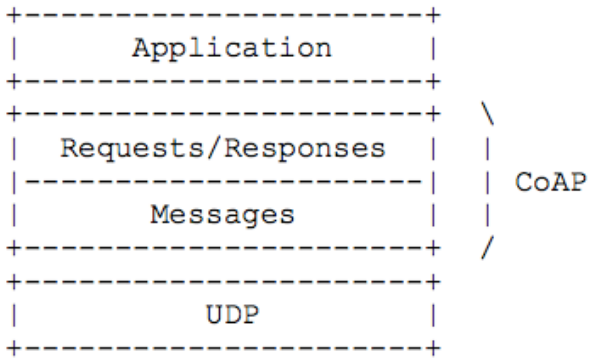


Figura 4-2 Modelo de capas

.

## 4.1 Características

CoAP tiene las siguientes características:

- Protocolo de aplicación que satisface las necesidades de nodos sensores haciendo uso de UDP con opciones de fiabilidad, implementando peticiones unicast y multicast.
- Intercambio de mensajes asíncronos.
- Baja sobrecarga en la cabecera para simplificar el mensaje.
- Opción de transporte seguro mediante DTLS (Datagram transport layer security).
- Similitud con HTTP, al estar basado en REST favorece la integración en la web.
- Admite la posibilidad de caching para aumentar la eficiencia.
- Se apoya en URI y Content-type.

## 4.2 Mensajes Coap

### 4.2.1 Modelo de mensajes

CoAP usa una cabecera binaria de 4 octetos que viene sucedida por un conjunto de opciones y la carga útil del mensaje. Este formato es así tanto para peticiones como para respuestas.

Cada mensaje contiene un ID que se usa para detectar duplicidades o para enviar mensajes confirmables.

La confiabilidad se consigue marcando un mensaje como confirmable, que es retransmitido cada cierto tiempo hasta que el receptor envía un ACK que contenga el mismo ID del mensaje. Cuando el receptor no es capaz de

procesar el mensaje envía un mensaje de reset en vez de un ACK. (Figura 4-4).

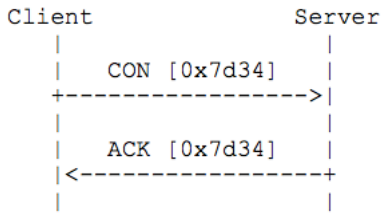


Figura 4-4 Modelo de mensaje confirmable

Si un mensaje no requiere de transferencia fiable se marca como no confirmable. Estos mensajes no requieren de una confirmación o ACK pero aun así tendrán su correspondiente ID para evitar duplicidades. (Figura 4-5).

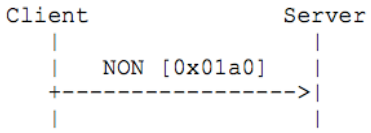


Figura 4-5 Modelo de mensaje confirmable

### 4.2.2 Formato del mensaje

Un mensaje CoAP está compuesto por las siguientes partes:

- Una cabecera compuesta por 4 octetos dividida en cinco:
  - Versión (ver): indica la versión de CoAP; los mensajes cuya versión se desconozca serán ignorados.
  - Type (T): indica si el mensaje es confirmable, no confirmable, ACK o reset.
  - Token length (TKL): indica la longitud de la variable Token.
  - Código: indica si el mensaje es una petición o una respuesta.
  - ID del mensaje: asigna un ID a cada mensaje para evitar duplicidades o en caso de transferencia fiable.
- Token: el valor del token se usará para relacionar peticiones y respuestas.
- Opciones: describe las distintas opciones deseadas en el mensaje.
- Carga útil o Payload.

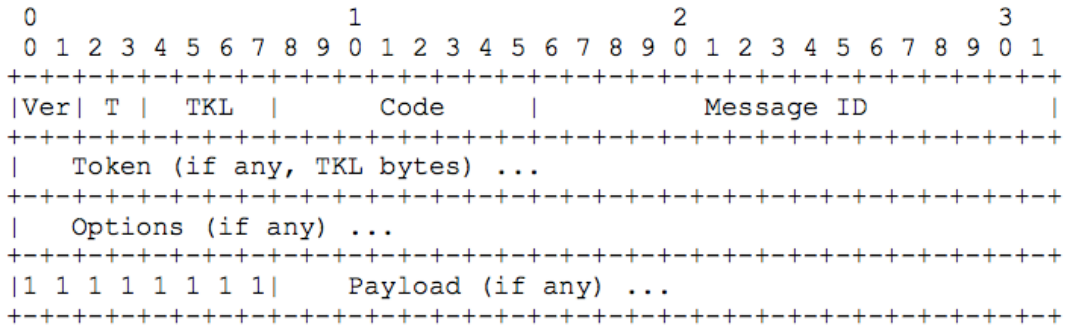


Figura 4-3 Modelo de cabecera CoAP

### 4.2.3 Opciones

Los tipos de opciones se pueden diferenciar en dos clases, críticas y opcionales. Su principal diferencia consiste en cómo son tratadas por un nodo cuando este no la reconoce.

- Si una opción de clase opcional no es reconocida por el nodo debe ser ignorada.
- Si una opción critica de una petición confirmable no es reconocida por el nodo se deberá generar una respuesta con el código 4.02 (bad option) y en el cuerpo de mensaje un diagnostico describiendo dicha opción.
- Si una opción critica de una respuesta confirmable, o "piggy-backed", no es reconocida debe ser descartada.
- Si una opción crítica de un mensaje no confirmable no es reconocida será descartada.

Tanto las peticiones como las respuestas pueden incluir una o más opciones.

CoAP define una serie de opciones:

- **Content-Format**: indica la representación del formato del mensaje.
- **Etag**: funciona como identificador local de recursos que varían en el tiempo.
- **Location-Path**: indica la ubicación de un recurso.
- **Location-Query**: indica la ubicación de un recurso.
- **Max-Age**: indica el tiempo máximo en el que una respuesta almacenada en caché es considerada "fresh". Una respuesta es fresh si no ha pasado demasiado tiempo para considerarla desactualizada y por tanto puede considerarse valida como respuesta a una petición.
- **Proxy-Uri**: se usa como petición de reenvío a un proxy
- **Uri-Host**: solicita un host específico de internet.
- **Uri-Path**: especifica la ruta del recurso.
- **Uri-Port**: especifica el número del puerto del recurso.
- **Uri-Query**: es una cadena de consulta.
- **Accept**: se usa para especificar el formato del mensaje aceptado.
- **If-Match**: puede ser utilizada para hacer una petición condicional en función de la opción Etag.
- **If-None-Match**: puede ser utilizada para hacer una petición en función de la existencia del nodo de destino.

Algunas de estas opciones pueden ser configuradas con un valor por defecto; en este caso la opción no debe ser incluida en el mensaje. Se asumirá el valor por defecto.

### 4.2.4 Transmisión de mensajes

Los mensajes en CoAP son transmitidos de forma asíncrona. Los mensajes pueden llegar al nodo destino desordenados, duplicados o sin avisar.

CoAP implementa un mecanismo de fiabilidad para tratar este tipo de problemas, pero sin llegar a la complejidad de todas las características que tiene TCP.

Este sistema consiste en:

- Detectar duplicidades tanto para mensajes confirmables como no confirmables.
- Esperar y retransmitir dejando un tiempo exponencial para mensajes confirmables.

#### 4.2.4.1 Identificación Nodo destino

Un nodo destino consiste en aquel al que van dirigidos los mensajes. Regularmente el nodo destino será identificado de una manera u otra dependiendo de la seguridad implementada.

Si no se toman medidas de seguridad el nodo será identificado mediante una dirección IP y un puerto UDP.

Si se definen medidas de seguridad el nodo será identificado acorde a lo definido en dicho modo, mediante un sistema de clave compartida o certificado.

#### 4.2.4.2 Mensaje confirmable

La fiabilidad en la transmisión de un mensaje comienza marcando dicho mensaje como confirmable en su cabecera. Un mensaje confirmable siempre se lleva a cabo con una petición o una respuesta salvo que su uso sea para un mensaje de reset en cuyo caso estará vacío.

El nodo destino debe tomar una de estas dos acciones:

- Responder al mensaje con un ACK.
- Descartar el mensaje si procede.

Descartar un mensaje conlleva la emisión de un mensaje de reset e ignorarlo.

Cuando se responde al mensaje este llevará el mismo ID y podrá incluir una respuesta o ir vacío. En el caso en el que el mensaje descartado sea el ACK simplemente se ignorará.

El nodo emisor reenviará el mensaje confirmable hasta recibir un ACK o hasta agotar el número de intentos de retransmisión. La retransmisión está controlada por dos factores:

- Un temporizador exponencial que se irá duplicando en cada retransmisión.
- Una variable MAX_RETRANSMIT que determina el número de veces que un mensaje puede ser retransmitido.

Si pasado un tiempo se alcanza el número máximo de retransmisiones o se recibe un reset el emisor desistirá el envío. Por otra parte, si el emisor recibe un ACK en tiempo la comunicación habrá tenido éxito.

#### 4.2.4.3 Mensaje no confirmable

Algunos mensajes son enviados sin necesidad de ser confirmados. Por ejemplo en aplicaciones que emiten mensajes regularmente como la lectura de un sensor, donde una lectura eventual es suficiente.

Un mensaje no confirmable conlleva una petición o respuesta y no puede estar vacío.

Un mensaje no confirmable no será respondido con un ACK y en el caso de que eventualmente se rechace se podrá emitir un mensaje de reset que el emisor ignorará.

A nivel de CoAP no hay manera de saber si un mensaje marcado como no confirmable es recibido o no. El emisor puede optar por transmitir el mensaje varias veces dentro de un máximo marcado por la variable MAX_TRANSMIT_SPAN. Los mensajes no confirmables llevan un ID, así aunque el mensaje sea retransmitido varias veces el nodo destino solo responderá una vez.

### 4.2.5 Correlación de mensajes

Un mensaje de tipo ACK o de reset se relacionara con un mensaje confirmable o no confirmable a través de un ID. El campo ID está compuesto por 16 bits sin signo en la cabecera del mensaje.

### 4.2.6 Tamaño del mensaje

La especificación de CoAP solo limita con un máximo el tamaño del mensaje. Un mensaje debe caber en un solo paquete para evitar la fragmentación y tener que enviar más de un paquete. Un mensaje CoAP correctamente encapsulado debería caber en un solo paquete IP, si la MTU de destino es desconocida se asumirá una MTU de 1280 bytes.

Si además no se conoce el tamaño de las cabeceras una buena estimación es considerar un máximo de 1152 bytes para el tamaño del mensaje y 1024 bytes para la carga útil.

## 4.3 Semántica de petición y respuesta

El protocolo CoAP opera bajo un modelo de petición respuesta similar a HTTP.

### 4.3.1 Petición

Una petición se basa en aplicar solicitar una acción mediante un método al recurso deseado. En la petición se incluye un identificador de dicho recurso y una carga útil además de alguna opción.

### 4.3.2 Respuestas

El tipo de respuesta se especifica mediante un código en la cabecera del mensaje; hay tres tipos de códigos:

- (2) éxito: la comunicación se ha ejecutado correctamente.
- (4) error en el cliente.
- (5) error en el servidor.

En el caso más básico la respuesta se incluirá en el mismo ACK (lo que requiere que la petición provenga de un mensaje confirmable). A este tipo de respuesta se le denomina “piggy-backed” y no será necesario un mensaje adicional con la respuesta.

Puede que no sea posible responder usando “piggy-backed”. Por ejemplo la respuesta de un mensaje no confirmable siempre requiere de un mensaje independiente. Dos ejemplos de una petición GET en las que el servidor responde de forma “piggy-backed”; en una hay éxito y en la otra no. (Figura 4-6).

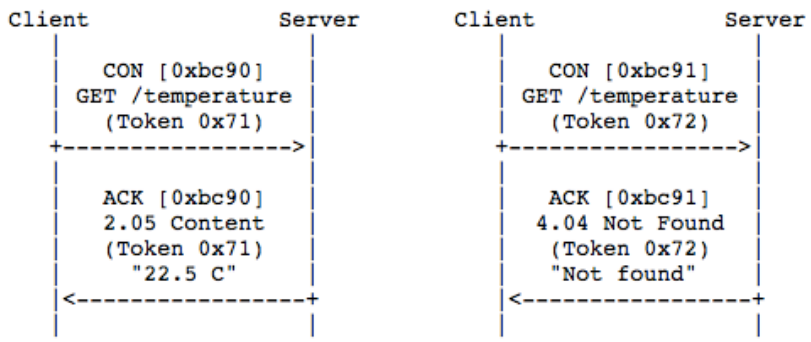


Figura 4-6 Modelo de mensajes “piggy-backed”

En cambio cuando el servidor responda de manera independiente primero enviará el ACK y luego la respuesta cuando esté lista. (Figura 4-7).

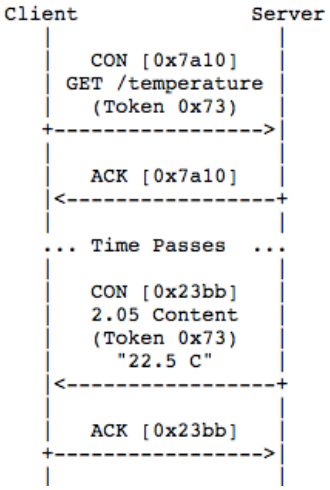


Figura 4-7 Modelo de mensajes con respuesta independiente

Como se puede observar algunas veces se utiliza un token, compuesto de 8 octetos para ligar una petición con una respuesta. Toda respuesta que lleve un token generado en el cliente será respondida por un servidor haciendo referencia al mismo.

Los tokens serán generados por el cliente de tal manera que sean únicos en el contexto cliente-servidor.

Las reglas exactas para ligar una petición a una respuesta son:

- El destinatario en la respuesta debe ser el mismo el nodo emisor de la petición.
- En una respuesta "piggybacked" tanto el ID del mensaje como el token deben coincidir. En una respuesta separada solo el token coincide.

.

## 4.4 Caching

Caching hace referencia al proceso mediante el cual un nodo almacena una respuesta obtenida en la memoria caché. Si recibe una petición similar puede hacer uso de esta respuesta ahorrando tiempo de respuesta y uso de ancho de banda.

El objetivo de caching es reutilizar una respuesta antigua para una petición similar nueva, en algunos casos esto se puede hacer sin necesidad de preguntar a la red, reduciendo así la latencia. Hay dos métodos mediante los cuales el nodo podrá reutilizar la respuesta.

### 4.4.1 Modelo freshness

Una respuesta es fresh si no ha pasado demasiado tiempo para considerarla desactualizada y por tanto puede considerarse valida. La información es almacenada en caché. Si la respuesta es "fresh" se usa como respuesta sin necesidad de tener que contactar con el servidor, mejorando el tiempo de respuesta. Si no es fresh se contacta con el servidor y se actualiza la información en caché.

El mecanismo para determinar si una respuesta es "fresh" viene impuesto por parte del servidor que proporciona en la respuesta un tiempo de expiración usando la opción Max-Age. Esta opción indica que una respuesta no será "fresh" después de los segundos indicados; el valor por defecto de esta opción son 60 segundos.

### 4.4.2 Modelo de validación

Cuando un nodo tiene almacenado en cache varias respuestas obtenidas mediante GET que no puede usar porque no son “fresh”, puede usar la opción Etag del método GET para que el servidor evalúe si es posible renovar o no su condición de “fresh”. A esto se le denomina proceso de validación.

# 5 IMPLEMENTACIÓN DEL SISTEMA

*We have no idea how far it is going to go.*

Steve Jobs

Una vez que detallados los distintos elementos de nuestro proyecto procedemos a explicar la implementación y la solución elegida.

Nuestro sistema será un entorno simulado en el que se programarán una serie de sensores.

Estos sensores son los siguientes:

- Un sensor que mide si la luz está o no encendida en cada una de las habitaciones. El consumo energético es una de las principales preocupaciones de los dueños de una casa. Poder conocer el estado de todas las luces de una casa en cualquier momento y desde cualquier lugar es esencial para evitar un consumo indeseado.
- Un sensor de temperatura en el exterior. A través de este sensor se puede monitorizar la temperatura en el exterior en todo momento.
- Un sensor de temperatura en un frigorífico. A través de este sensor somos capaces de saber que temperatura exacta tiene nuestro congelador o frigorífico. No son pocas las veces que uno se da cuenta de que su congelador ha dejado de funcionar demasiado tarde. Midiendo la temperatura en su interior podemos conocer cualquier comportamiento anómalo.

Estos sensores, que generan tipos de datos distintos, se comunican con una Raspberry-Pi haciendo uso del protocolo de aplicación restringida CoAP. A continuación, la Raspberry-Pi almacenará estos datos en una base datos en la nube.

Con el fin de aumentar el grado de compatibilidad se ha hecho uso de los 3 mayores proveedores del mercado:

- Amazon Web Services.
- Google Cloud.
- Microsoft Azure.

Hay dos partes lógicas diferenciadas (Figura 5-1) que giran en torno a la Raspberry-Pi:

- Una primera parte, basada en el internet de las cosas haciendo uso del protocolo CoAP.
- Una segunda parte, basada en CoAP,l a nube y el uso de sus servicios.

Las distintas partes del proyecto a nivel de programación son:

- Programación del cliente CoAP.
- Programación del servidor CoAP.
- Programación de la transferencia de datos a la nube.
- Crear cuentas en los distintos proveedores de servicios cloud.

- Programación de las bases de datos en la nube.
- Conexión a la nube y transferencia de datos.

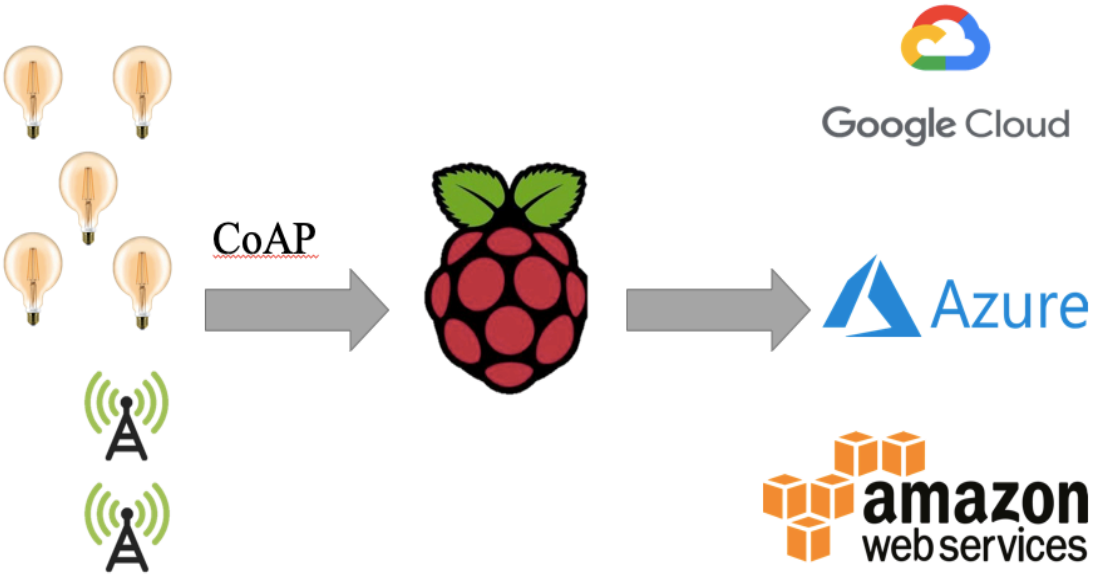


Figura 5-1 Implementación lógica

## 5.1 Programación de nodos sensores

Para programar los distintos sensores hemos decidido hacerlo en python3 haciendo uso de la librería Aiocoap.

### 5.1.1 Libreria Aiocoap

Aiocoap es una librería escrita en Python que soporta los distintos estándares y RFCs que definen el protocolo CoAP.

Su última versión puede encontrarse en https://github.com/chrysn/aiocoap que a su vez sirve como sitio de desarrollo.

Sus principales módulos son:

#### 5.1.1.1 Aicoap

Este módulo es el que implementa el protocolo de aplicación restringida CoAP.

#### 5.1.1.2 Aicoap.protocol

Este módulo define las clases que son responsables del seguimiento de los mensajes:

- Context: se encarga de gestionar las terminaciones (sockets UDP) haciendo que se pueda tanto emitir mensajes como responderlos.
- Requests: se genera cada vez que se produce una petición para poder seguir la respuesta.

#### 5.1.1.3 Aicoap.message

En este módulo se definen todos los atributos y los métodos para manipular un mensaje CoAP. Algunos atributos son:

- Payload.
- Mtype (CON, ACK...)
- Code (si es respuesta o petición).
- Opt (opciones).

#### 5.1.1.4 Aicoap.options

Se define cómo implementar los diferentes tipos de opciones de la cabecera en un mensaje. Algunos métodos son:

- Add_option(option)
- Delete_option(number)
- Get_option(number)
- Option_list()

Aunque no son todos, estos son algunos de los principales módulos que contiene la librería. Algunos más se irán explicando conforme avancemos en la programación de los sensores.

## 5.2 Implementación de los sensores

Python3 viene ya instalado en nuestra Raspberry-Pi por lo que en cuanto a esto respecta no tenemos que hacer nada.

Lo primero que tenemos que hacer es instalar Aiocoap (Figuras 5-2 y 5-3).

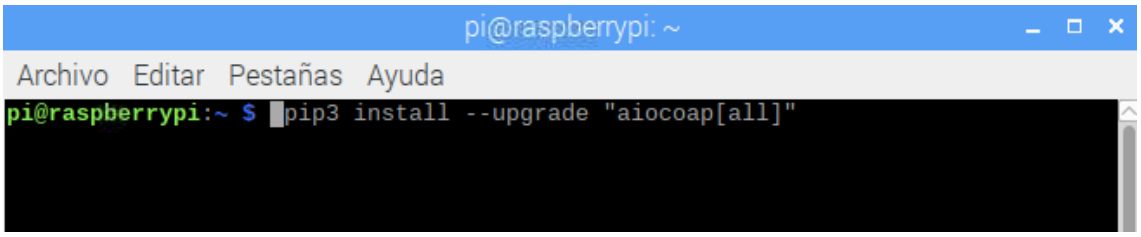


Figura 5-2 Instalar Aiocoap

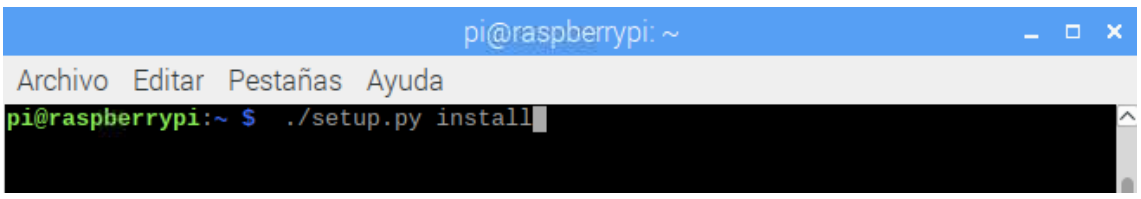


Figura 5-3 Instalar Aiocoap

Si no disponemos de pip3 podemos descargar directamente los archivos de Aiocoap y descomprimirlos.

A continuación, importamos todas las librerías que vamos a usar (Figura 5-4). *Datetime* y *ramdom* se usarán para la obtener la fecha del sistema y generar números al azar respectivamente.

*Asyncio* es una librería de python3 que permite programar procesos que responden ante un evento. Por ejemplo, cuando un servidor o en este caso un sensor tiene que responder a una petición.

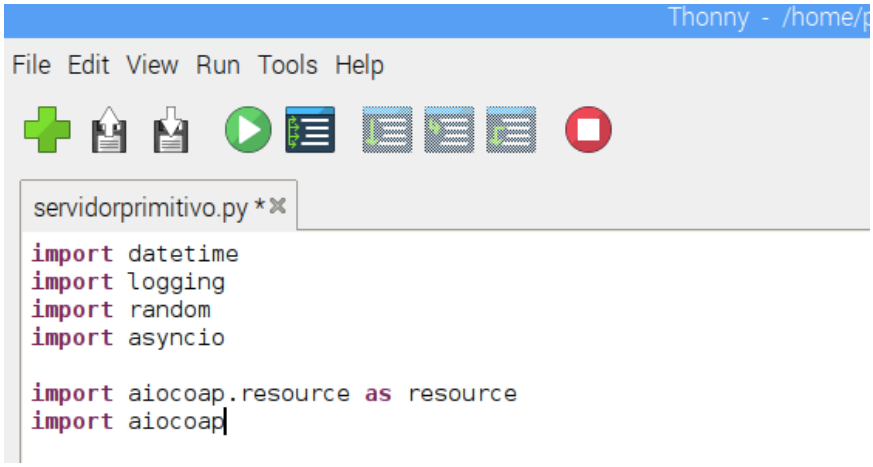


Figura 5-4 Programación de sensor

A continuación se muestra cÓmo se programa un sensor que distingue si una luz está encendida o apagada. Habrá uno en cada estancia de la vivienda. Programamos una clase de tipo resource que defina nuestro sensor y como responde ante los distintos tipos de peticiones, como por ejemplo una petición GET (Figura 5-5).

Inicializamos nuestro sensor y luego definimos el método que se encargará de responder al a petición. Cuando llega una petición GET a este recurso se genera un payload con el estado en el qué se encuentra la luz en el momento de la petición. Puede estar encendida o apagada. Por último para simular cambios en el sensor se fuerza el cambio de estado tras cada petición.

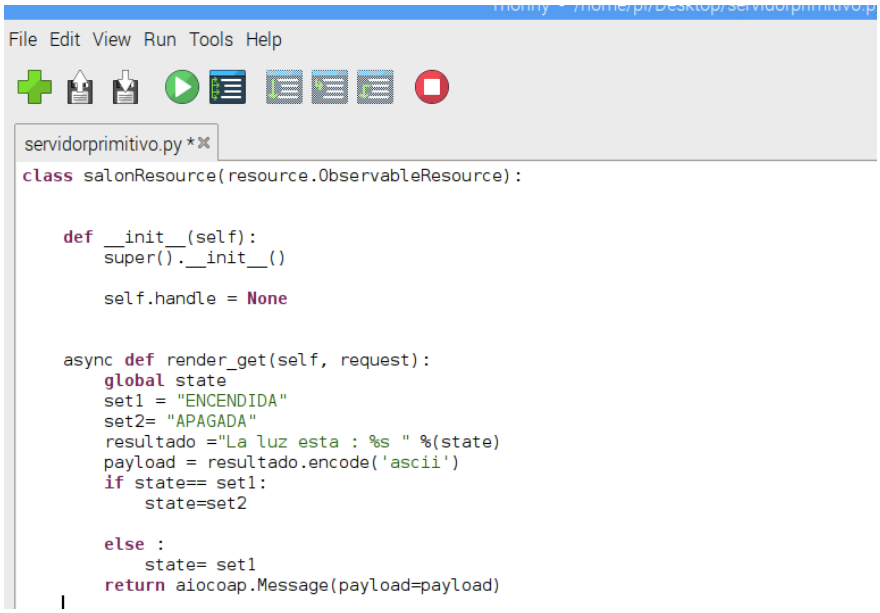


Figura 5-5 Código sensor de luz

Para programar el sensor de temperatura se procede de la misma manera (Figura 5-6). La única diferencia es la forma en la que se genera el payload del mensaje. Se genera una temperatura al azar haciendo uso de la librería ramdom. Esta temperatura simulada será distinta cada vez que este sensor reciba una petición.

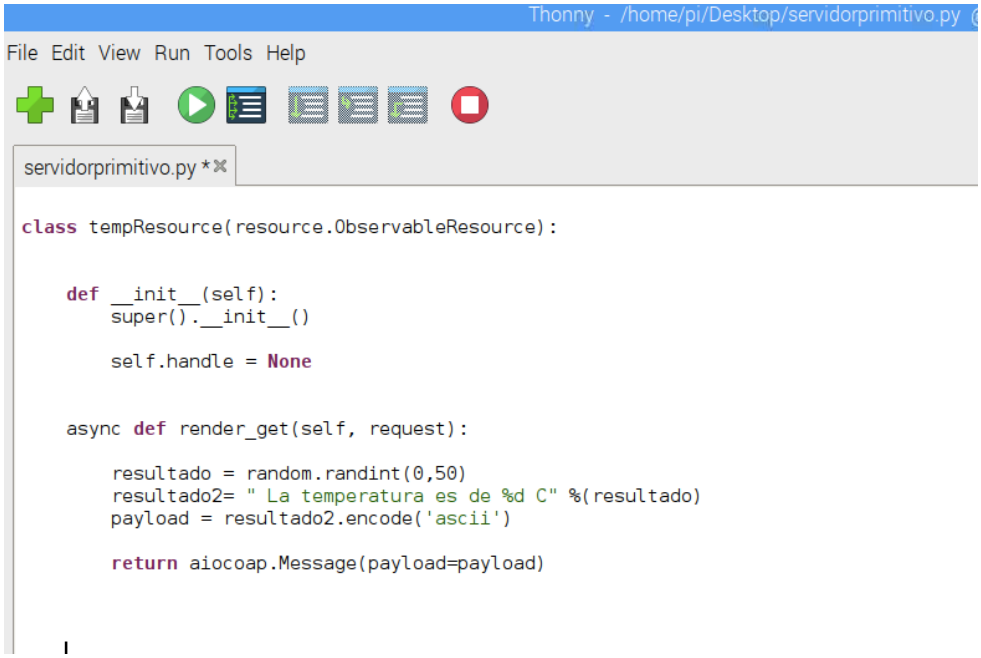


Figura 5-6 código sensor de temperatura

El último sensor que nos queda nos permite saber si el congelador de nuestro frigorífico sigue funcionando o no. Para ello se genera una temperatura simulada que oscila entre 5 y -8 grados. Cuando la temperatura está por encima de 0 grados quiere decir que el congelador por cualquier razón ha dejado de funcionar (Figura 5-7).

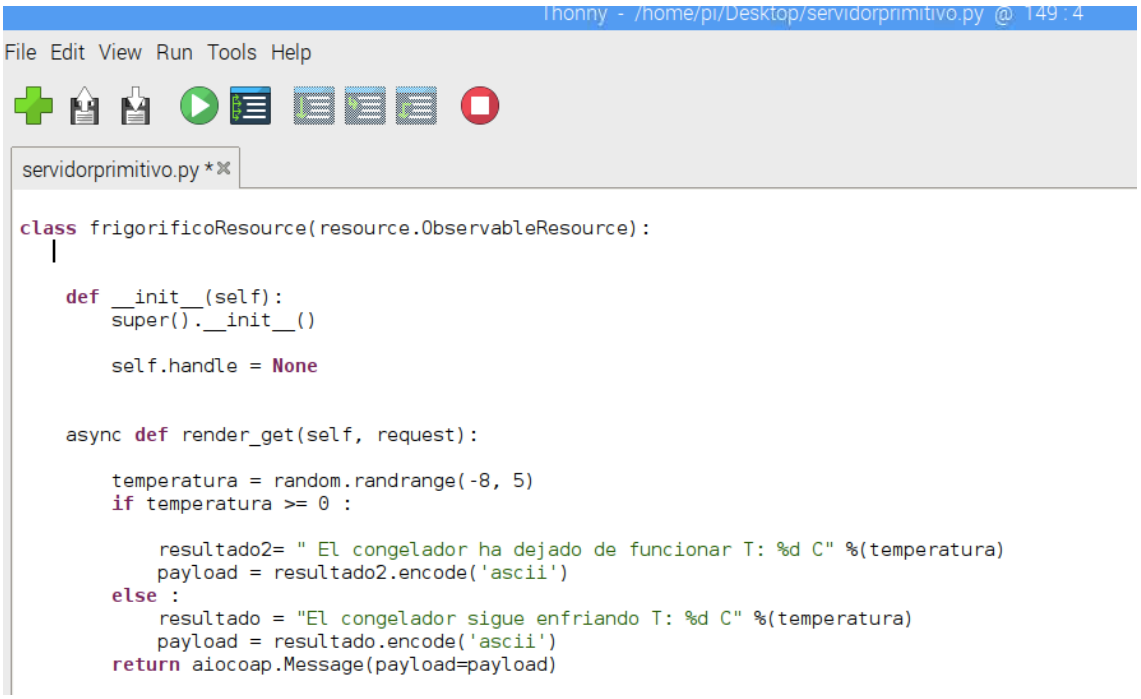


Figura 5-7 código sensor de temperatura congelador

Ahora que ya tenemos los diferentes sensores definidos con sus atributos y sus métodos nos queda hacerlos visibles en nuestros servidor.

En el principal del programa deberemos incluir cada sensor en la lista de recursos para que estos sean accesibles a través de una URI. Creamos el contexto de conexión de red en el servidor y se inicializa para escuchar posibles peticiones (Figura 5-8).

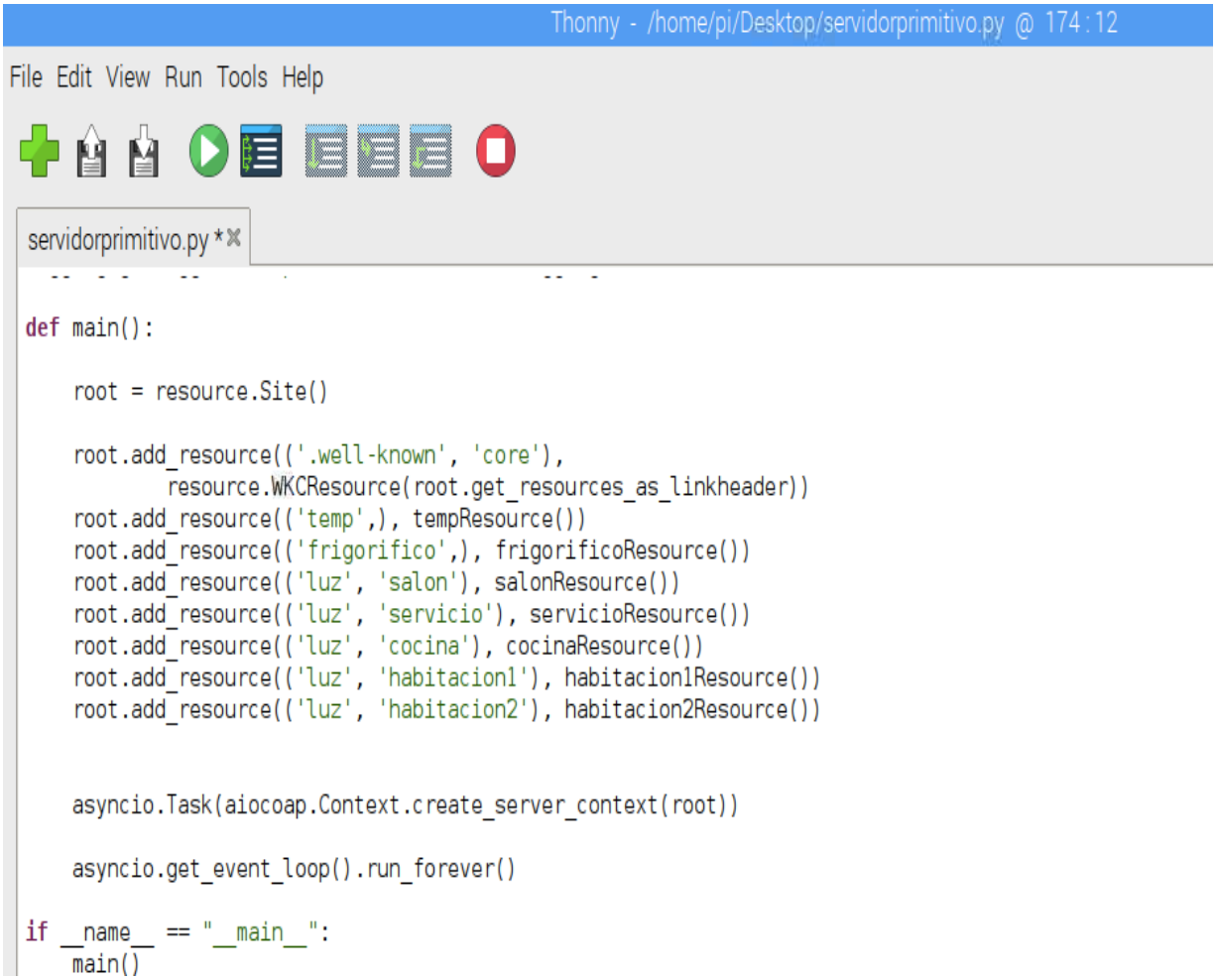


Figura 5-8 Código programación sensor

## 5.3 Implementacion del cliente

En la programación del cliente en la Raspberry-Pi se han hecho uso de las mismas librerías usadas en la programación de los sensores.

Se ha configurado el cliente para que cada cierto tiempo haga peticiones a los sensores y podamos comprobar los resultados esperados.

Como se puede ver en la Figura 5-9, debemos crear un contexto de conexión. Después configuramos cómo queremos que sea la petición, en este caso es de tipo GET, y por último indicamos la dirección del recurso del que queremos obtener la información.

En este caso concreto la dirección es localhost porque tanto cliente como sensor se están ejecutando en la raspberry-Pi.

Una vez que enviamos la petición el cliente quedará a la espera de una respuesta haciendo uso de la librería asyncio. Si el recurso existe y todo es correcto, se imprime el código de respuesta y el payload, de lo contrario imprime un error.

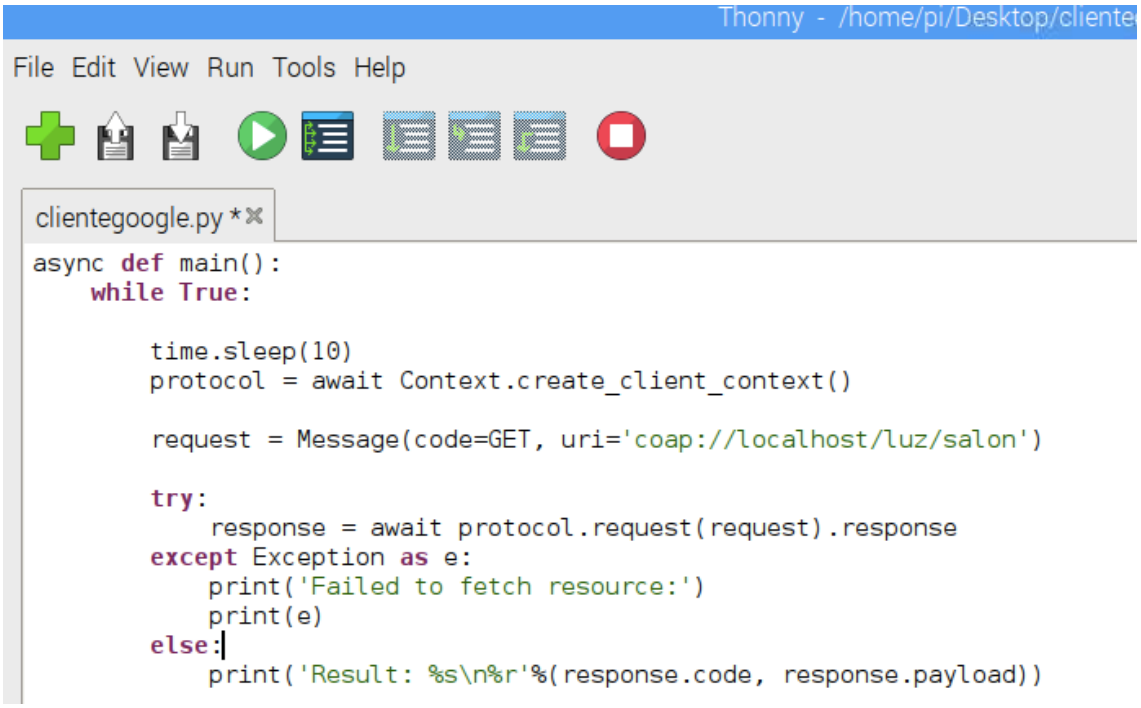


Figura 5-9 código cliente

## 5.4 Implementacion en Google Cloud

En este apartado vamos a explicar los pasos necesarios para implementar nuestra aplicación en una base de datos haciendo uso de google cloud.

Lo primero que debemos hacer es crear una cuenta gratuita haciendo uso de cualquier cuenta de Google (Figura 5-10).

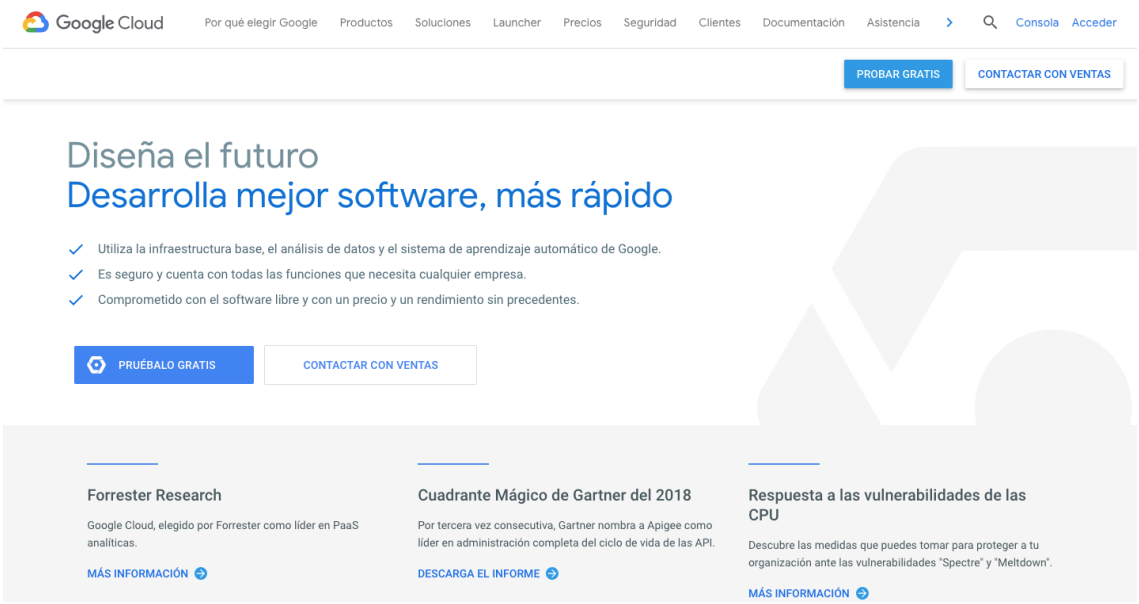


Figura 5-10 Crear cuenta Google Cloud

Una vez que hemos creado nuestra cuenta y accedemos a la consola, tenemos que acceder a la página para instancias SQL. Podemos acceder desde el panel que está a la izquierda (Figura 5-11).

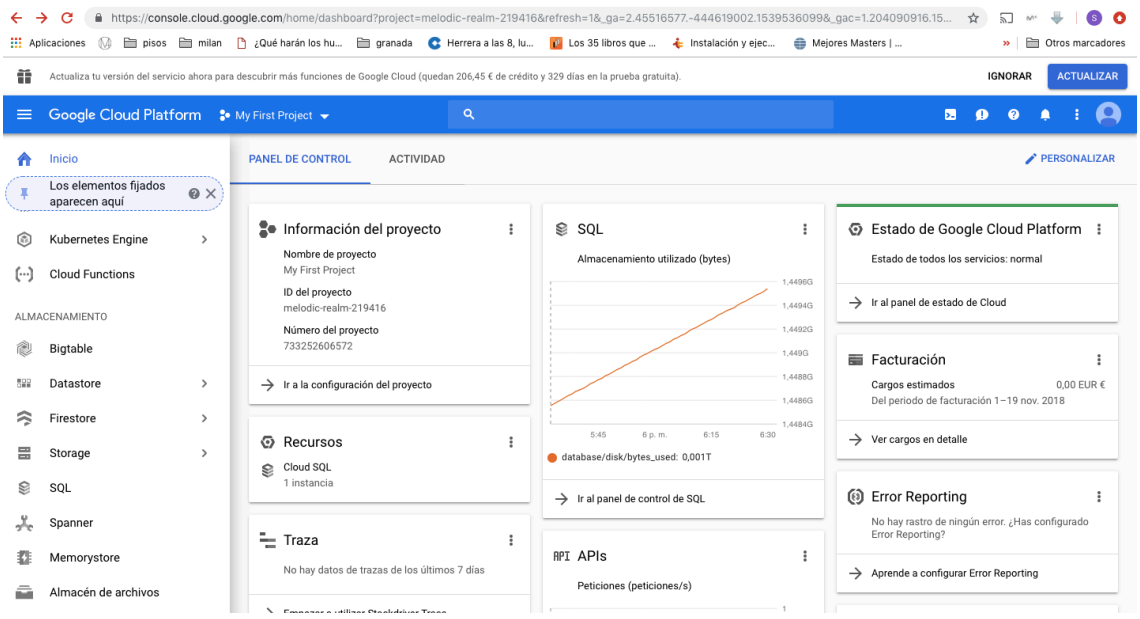


Figura 5-11 Crear instancia

Una vez en la página, pinchamos sobre crear una instancia, elegimos una base de datos de 2ª generación y nos aparecerá una pantalla para configurar la instancia a nuestro gusto (Figura 5-12).

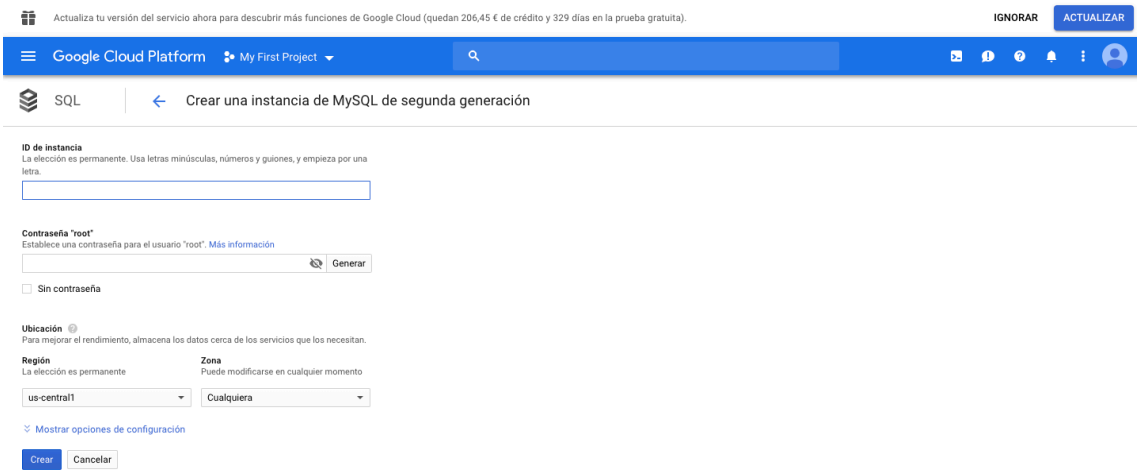


Figura 5-12 Crear instancia

Una vez hemos creado la instancia debería aparecernos algo como esto (Figura 5-13):

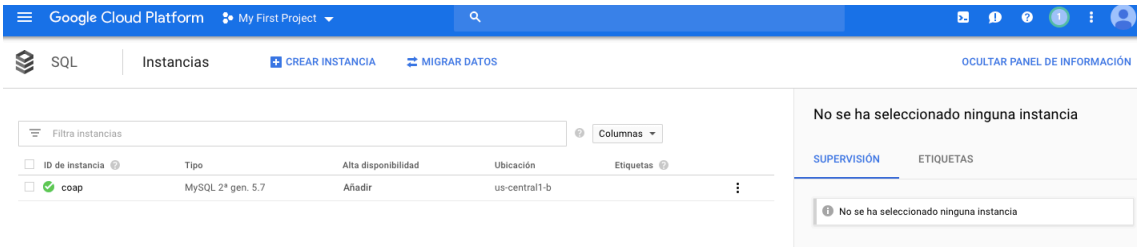


Figura 5-13 Panel de instancias

A continuación pinchamos sobre la instancia que hemos creado. Llegaremos al panel de control de nuestra instancia. De aquí hay varios datos importantes que tenemos que recordar como la dirección de ip pública para luego poder conectarnos. Por último debemos ir a la pestaña de conexiones (Figura 5-14).

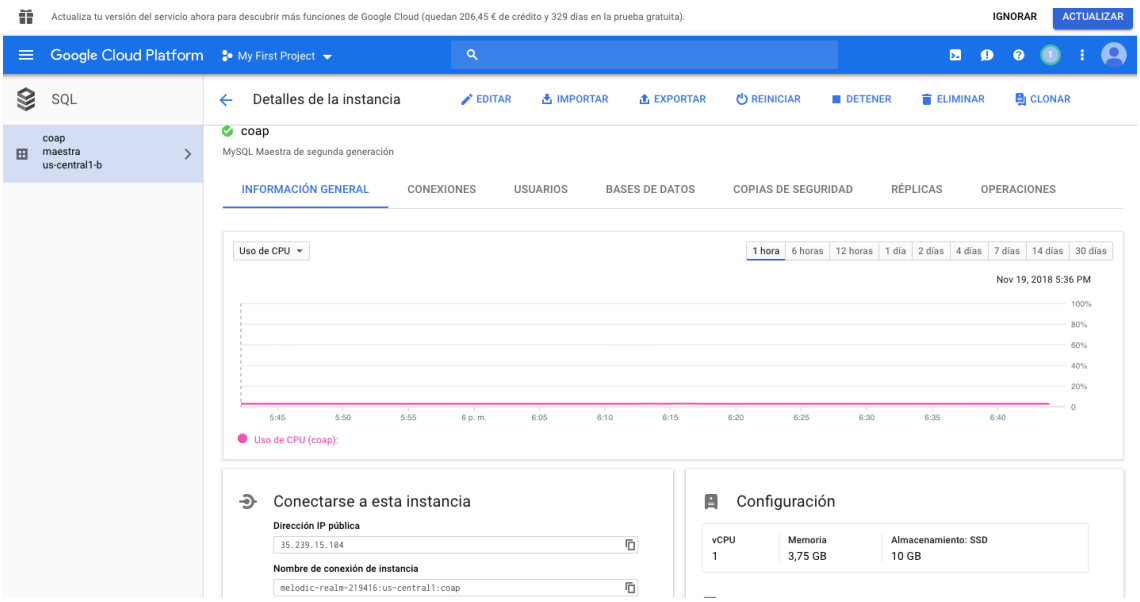


Figura 5-14 Panel de control de instancias

Todos los servicios cloud prohíben, por defecto, el tráfico entrante de redes no autorizadas. Por lo tanto, si queremos conectarnos a nuestra base de datos tenemos que autorizar la red desde la cual nos vamos a conectar (Figura 5-15).

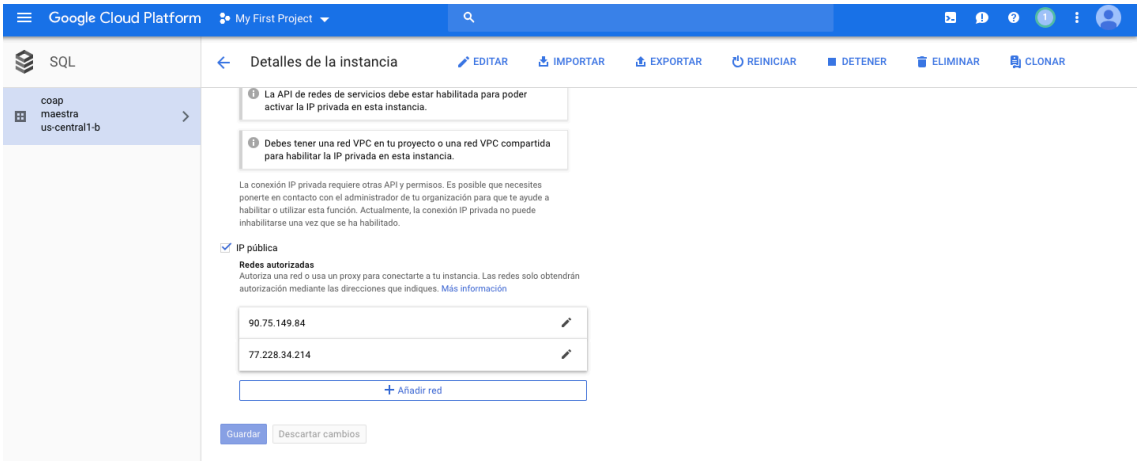


Figura 5-15 Panel de control seguridad

Ya está creada la instancia de la base de datos, ahora falta definir la base de datos y sus respectivas tablas desde la consola. Se accede pinchando en el icono de Shell arriba a la derecha y una vez en la consola nos conectamos a nuestra instancia de la siguiente manera (Figura 5-16).

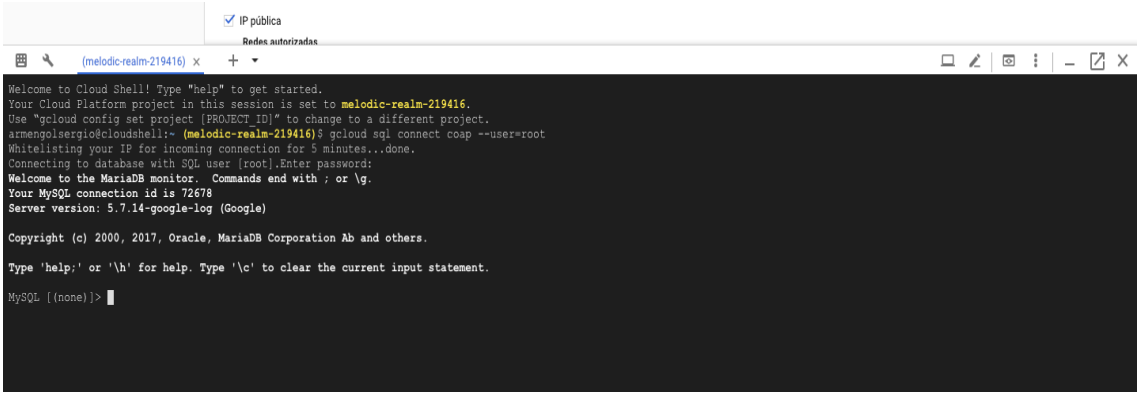


Figura 5-16 Shell Google Cloud

Una vez ya conectados, procedemos a crear una base de datos con sus respectivas tablas. En nuestro caso la base de datos se llamará TFG y tendrá tres tablas. Una tabla almacenará todos los datos que respectan a los sensores de luz, otra almacenará los datos de temperatura y la última almacenará los valores de si el congelador funciona o no.

Para crear una base de datos en MySQL se ejecuta el siguiente comando:

```
CREATE DATABASE guestbook;
```

Para crear una tabla de ejemplo se siguen los siguientes comandos:

```
CREATE TABLE entries (guestName VARCHAR(255), content VARCHAR(255),
    entryID INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(entryID))
```

En las figuras 5-17 y 5-18 se muestran la base de datos y las tablas creadas.

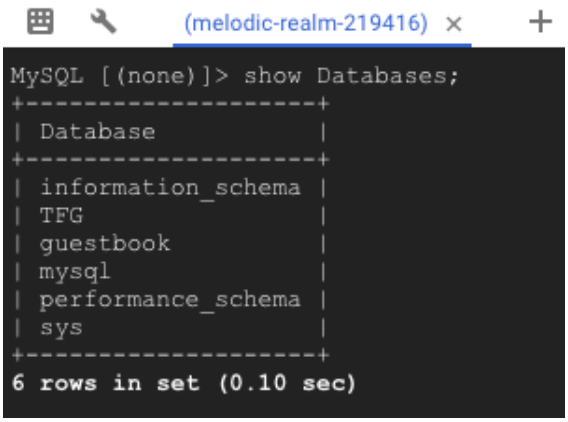


Figura 5-17 Bases de datos creadas

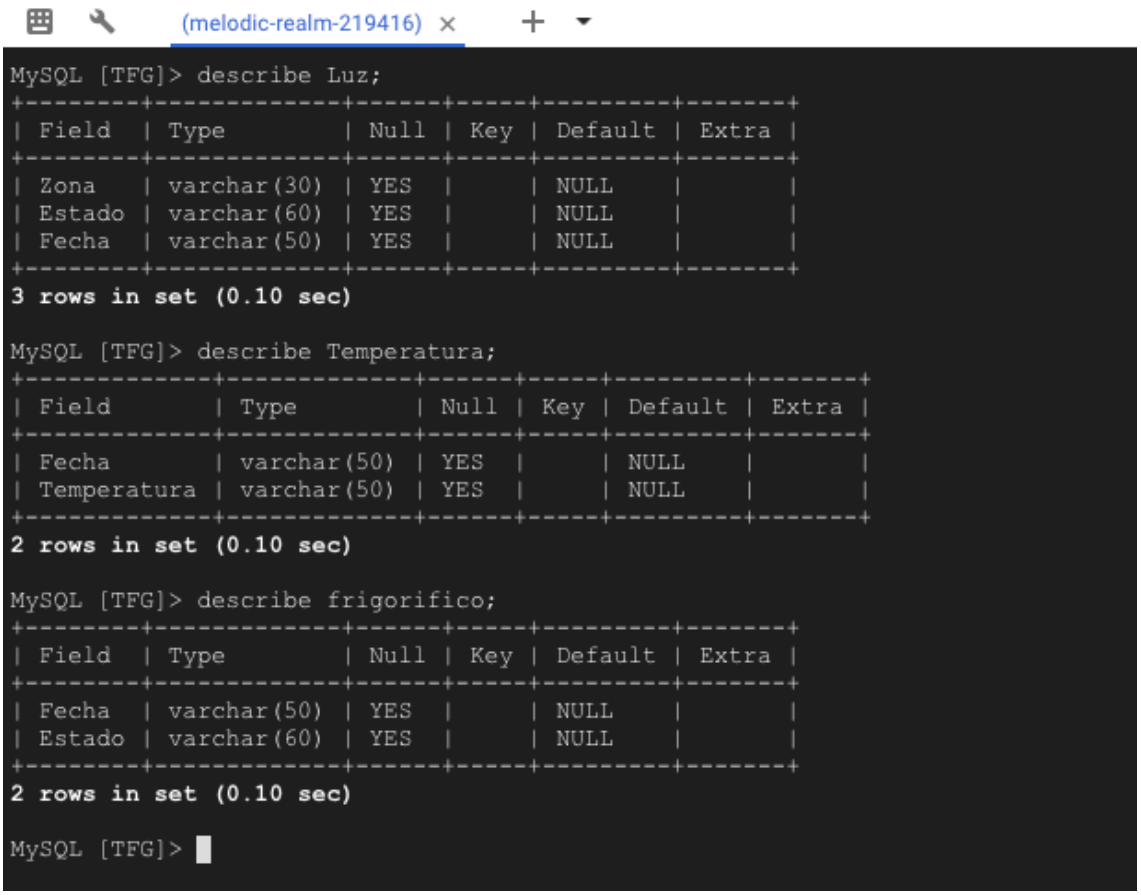


Figura 5-18 Tablas creadas

Ya tenemos nuestra base de datos creada y funcionando. Nos podemos olvidar por ahora de esto y volvemos a nuestra Raspberry-pi.

Para conectarnos a la base de datos debemos instalar el cliente de mysql en la Raspberry-Pi.

```
Sudo apt-get update
sudo apt-get install mysql-client
```

Para conectarnos desde nuestro cliente en la raspberry-Pi vamos a hacer uso de una librería que se llama pymysql.

```
pip3 install PyMySQL
```

Ahora ya lo tenemos todo listo para que desde el código que detallamos anteriormente podamos conectarnos a la base de datos y guardar los datos recibidos.

Importamos la librería que hemos instalado y definimos la cadena de conexión. En esta cadena se especifican todos los aspectos de la base de datos necesarios para conectarnos a la base de datos (Figura 5-19).

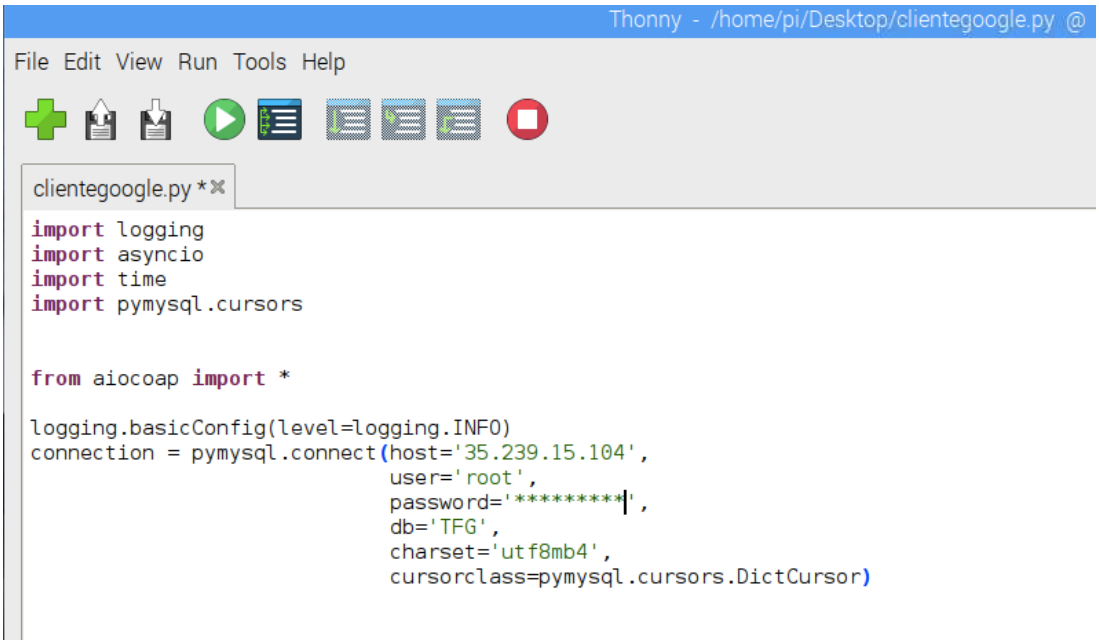


Figura 5-19 Conexión Google Cloud

Una vez que ya estamos conectados a la base de datos nuestro siguiente objetivo es insertar en ella los datos recibidos de los sensores. Cuando recibimos la respuesta, se crea la query con los datos recibidos y se ejecuta la operación. El proceso es el mismo para todos los sensores, lo único que varía son las tablas en las que se insertan los datos (Figura 5-20).

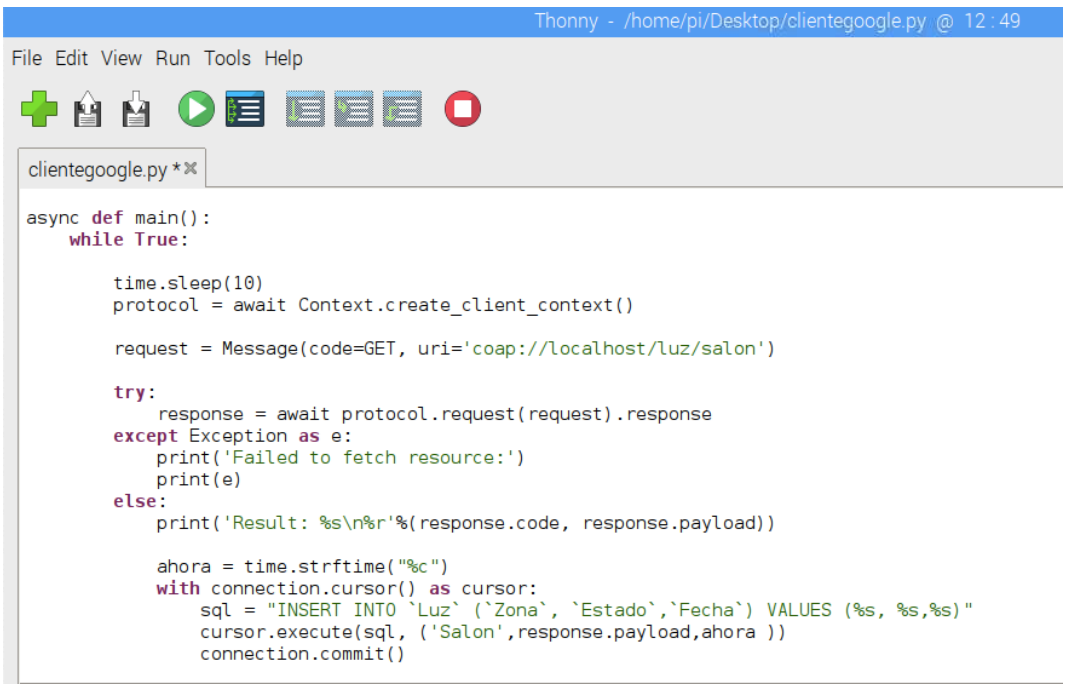


Figura 5-20 Insertando datos en Google Cloud

En la figura 5-21 se puede observar el comportamiento del sistema después de ejecutarlo.

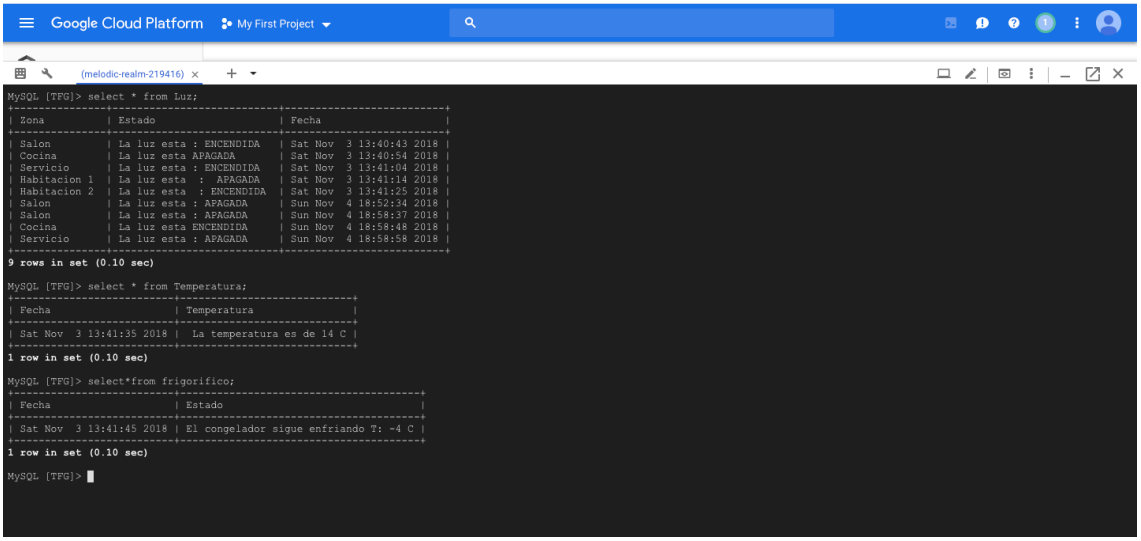


Figura 5-21 Resultado en Google Cloud tras simulación

## 5.5 Implementacion en Amazon Web Services

En este apartado vamos a explicar los pasos necesarios para crear e implementar nuestra aplicación en una base de datos haciendo uso de Amazon Web Services.

Lo primero que debemos hacer es crear una cuenta (Figura 5-22).

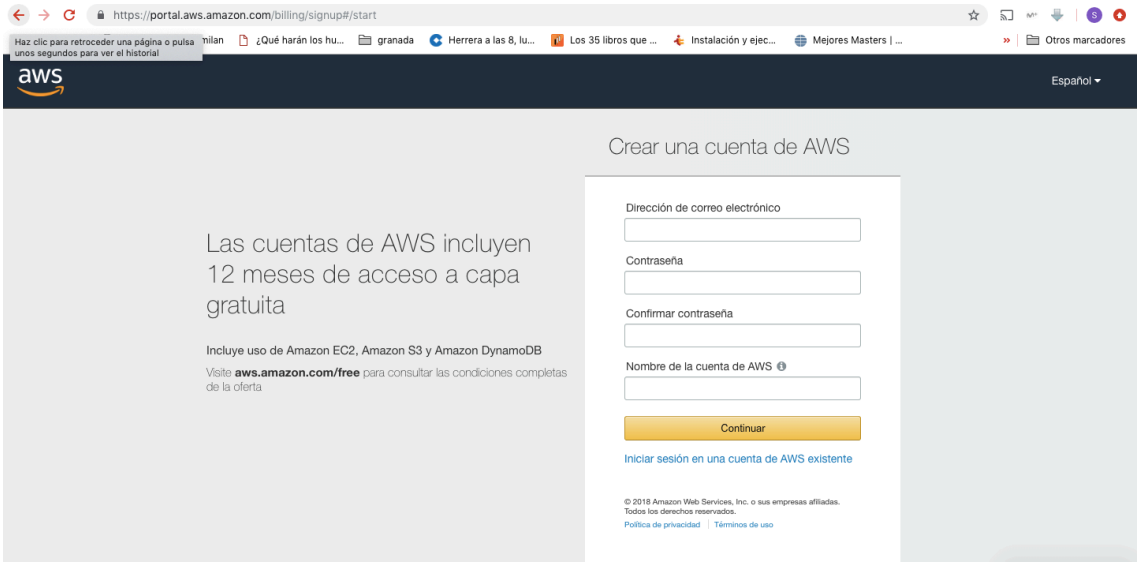


Figura 5-22 crear cuenta en AWS

Una vez que hemos creado nuestra cuenta tenemos que acceder a la consola de RDS (relational data base). Una vez accedemos a la consola le damos a crear base de datos (Figura 5-23).

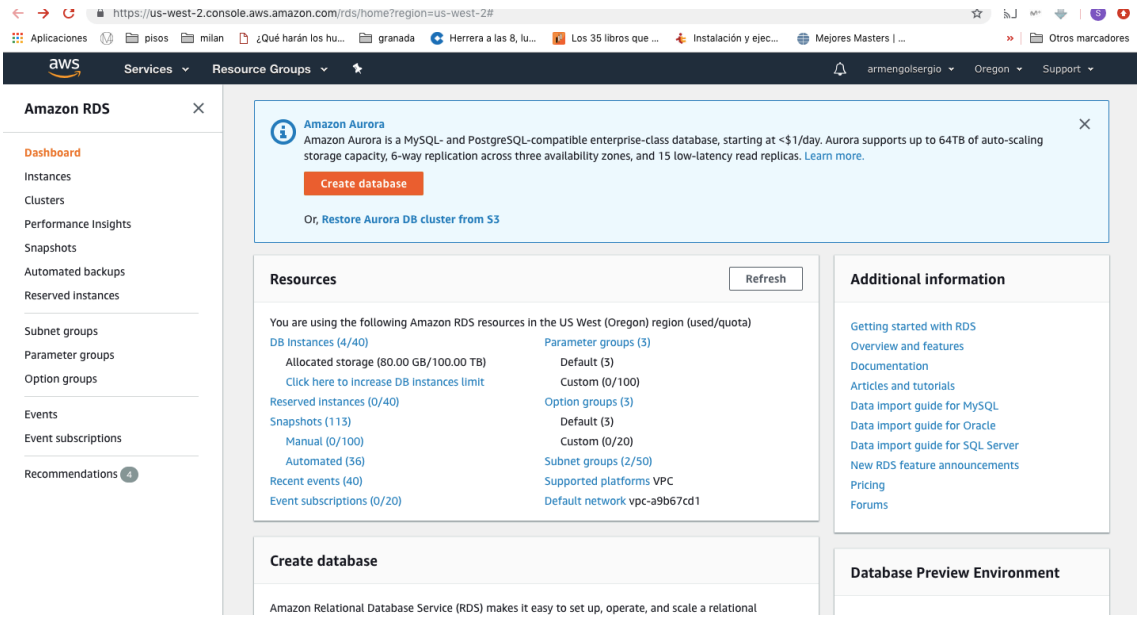


Figura 5-23 Panel RDS

Seleccionamos el tipo de base de datos (Figura 5-24) que deseamos y la configuramos igual que hicimos con google cloud.

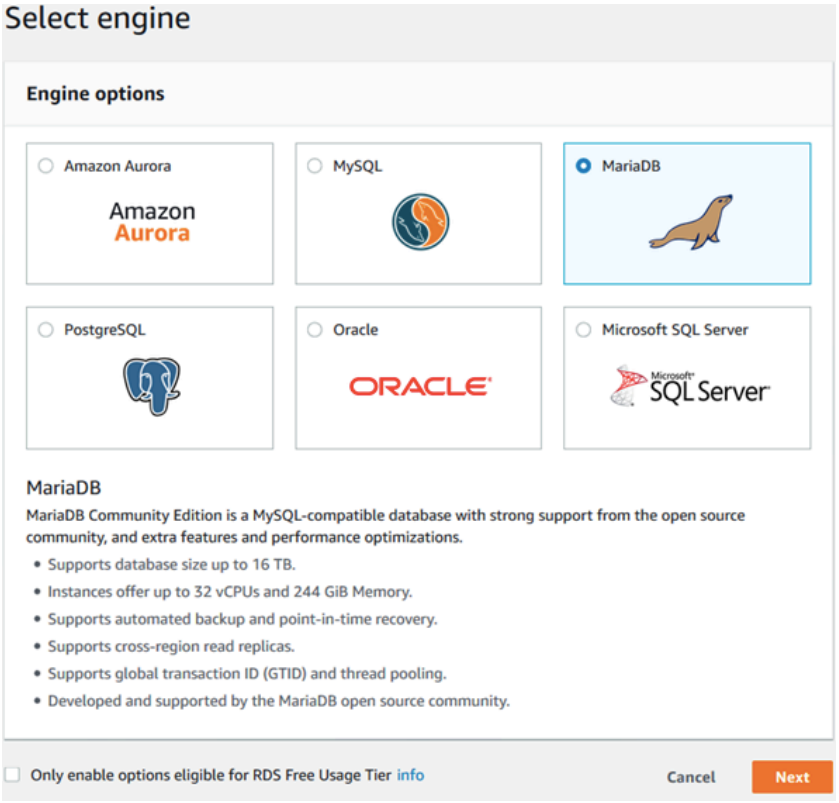


Figura 5-24 Creando base de datos en AWS

Una vez hemos creado la instancia debería aparecernos algo como en la figura 5-25:

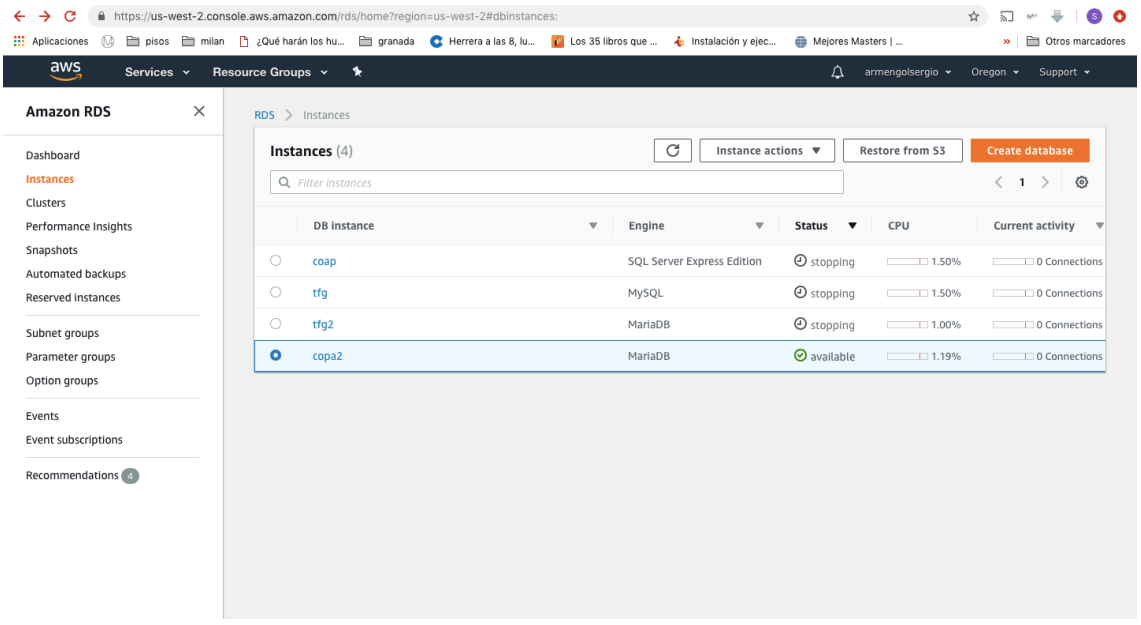


Figura 5-25 Panel de instancias en AWS

A continuación accedemos a la instancia que hemos creado (Figura 5-26). Del panel de control de la instancia hay varios datos importantes que tenemos que recordar, como el nombre de endpoint para luego poder conectarnos. Por ultimo debemos acceder a security groups.

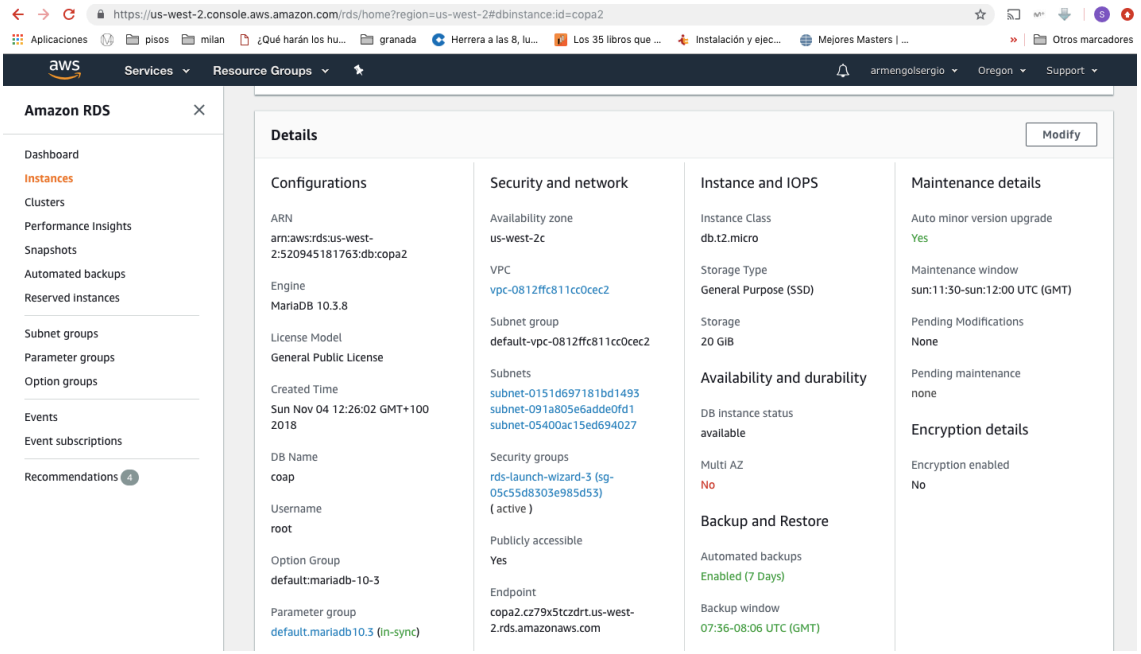


Figura 5-26 Panel gestión de instancia en AWS

Todos los servicios cloud prohíben, por defecto, el tráfico procedente de redes no autorizadas. Por lo tanto, si queremos conectarnos a nuestra base de datos tenemos que autorizar la red desde la cual nos vamos a conectar. Añadiendo esta regla (Figura 5-27) para las conexiones entrantes nos aseguramos poder conectarnos.

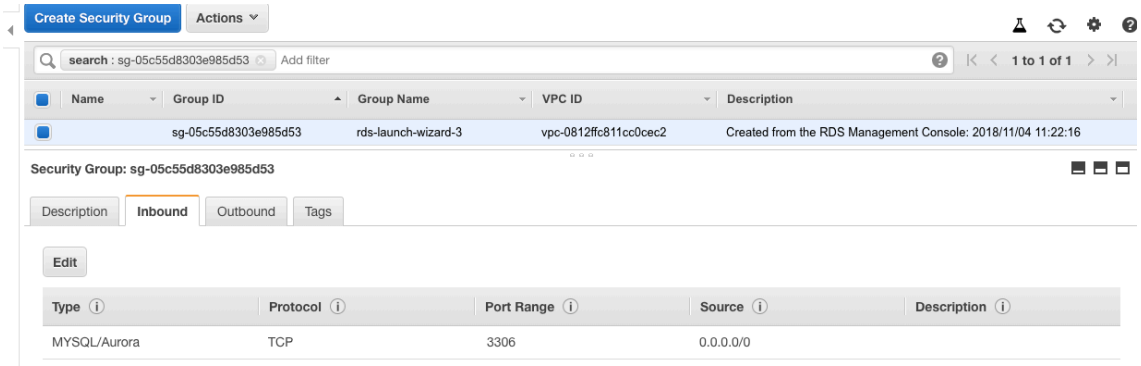


Figura 5-27 Panel de seguridad en AWS

Ya está creada la instancia, ahora falta crear la base de datos y sus respectivas tablas. El procedimiento para crear las tablas y la base de datos es exactamente el mismo que el empleado en google cloud. La base de datos creada y las tablas son exactamente iguales por lo que no tiene ningún interés detallar de nuevo el proceso.

Ya tenemos nuestra base de datos creada y funcionando, nos podemos olvidar por ahora de esto y volvemos a nuestra Raspberry-pi. Ahora ya lo tenemos todo listo para que desde el código que detallamos anteriormente podamos conectarnos a la base de datos y guardar los datos recibidos.

Importamos la librería pymysql que instalamos antes y definimos la cadena de conexión (Figura 5-28). En esta cadena se especifican todos los aspectos de la base de datos necesarios para la conexión.

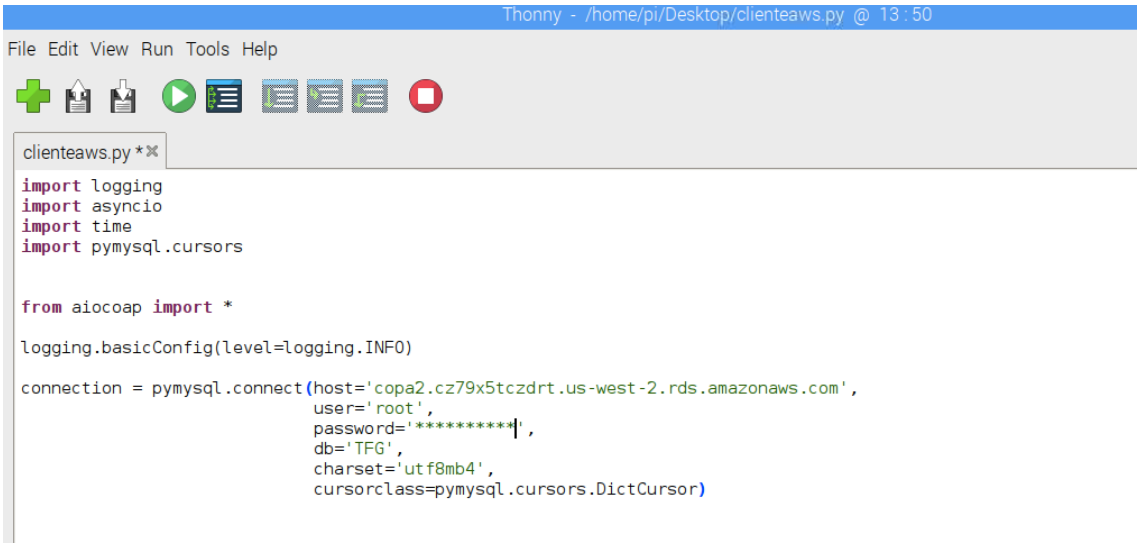


Figura 5-28 Conexión con AWS

Una vez ya definida la conexión a la base de datos insertamos en ella los mensajes recibidos. Cuando recibimos la respuesta se crea la query con los datos recibidos y ejecuta la operación. La operación es la misma para todos los sensores, lo único que varía son las tablas en las que se insertan los datos (Figura 5-29).

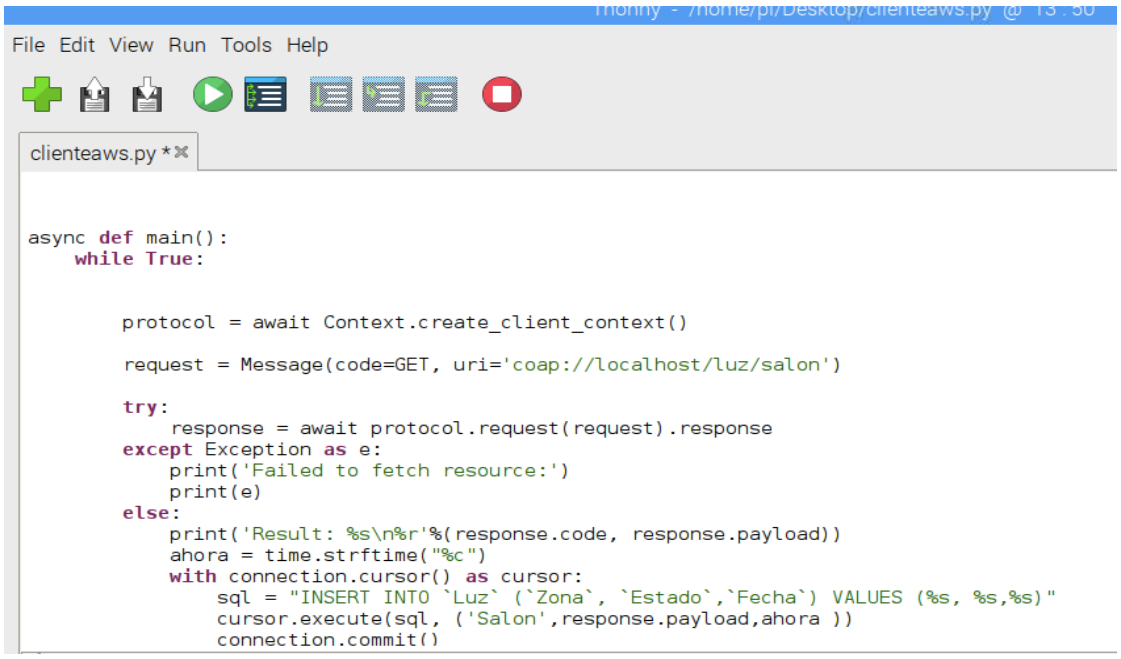


Figura 5-29 Insertando datos en AWS

En la figura 5-30 se puede observar el comportamiento del sistema después de ejecutarlo.

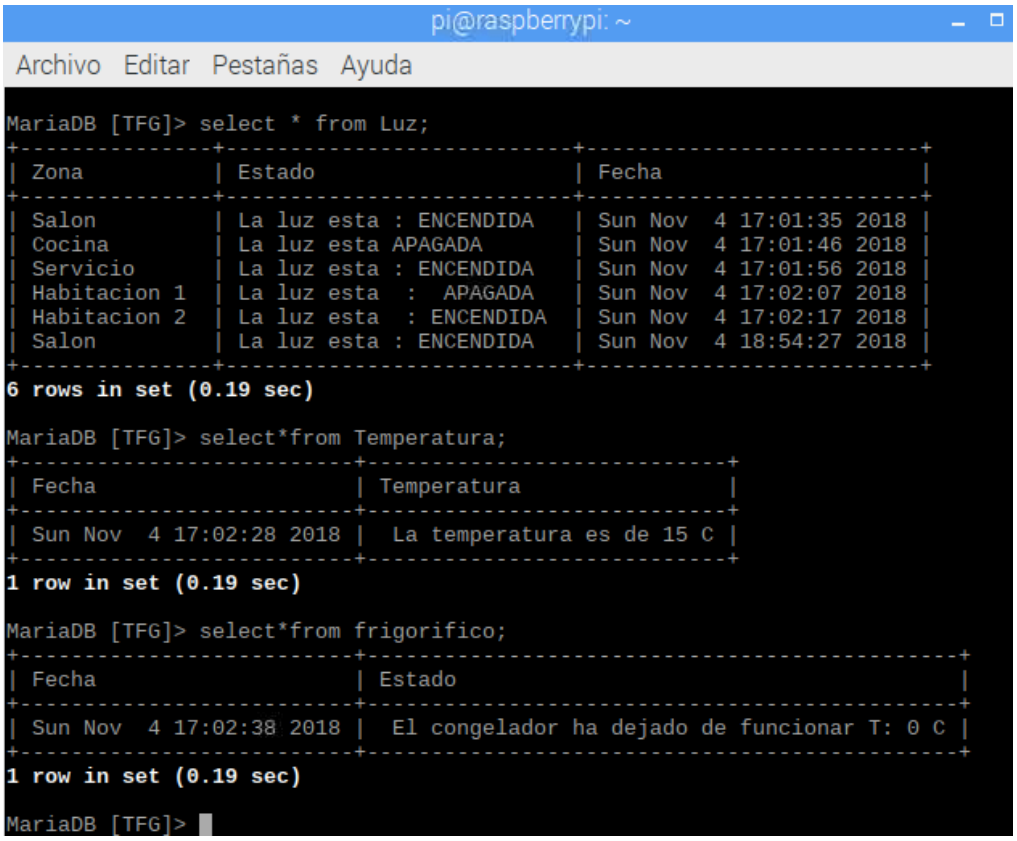


Figura 5-30 Resultado en AWS tras simulación

## 5.6 Implementacion en Microsoft Azure

En este apartado vamos a explicar los pasos necesarios para crear e implementar nuestra aplicación en una base de datos haciendo uso de Microsoft Azure.

Lo primero que debemos hacer es crear una cuenta (Figura 5-31).

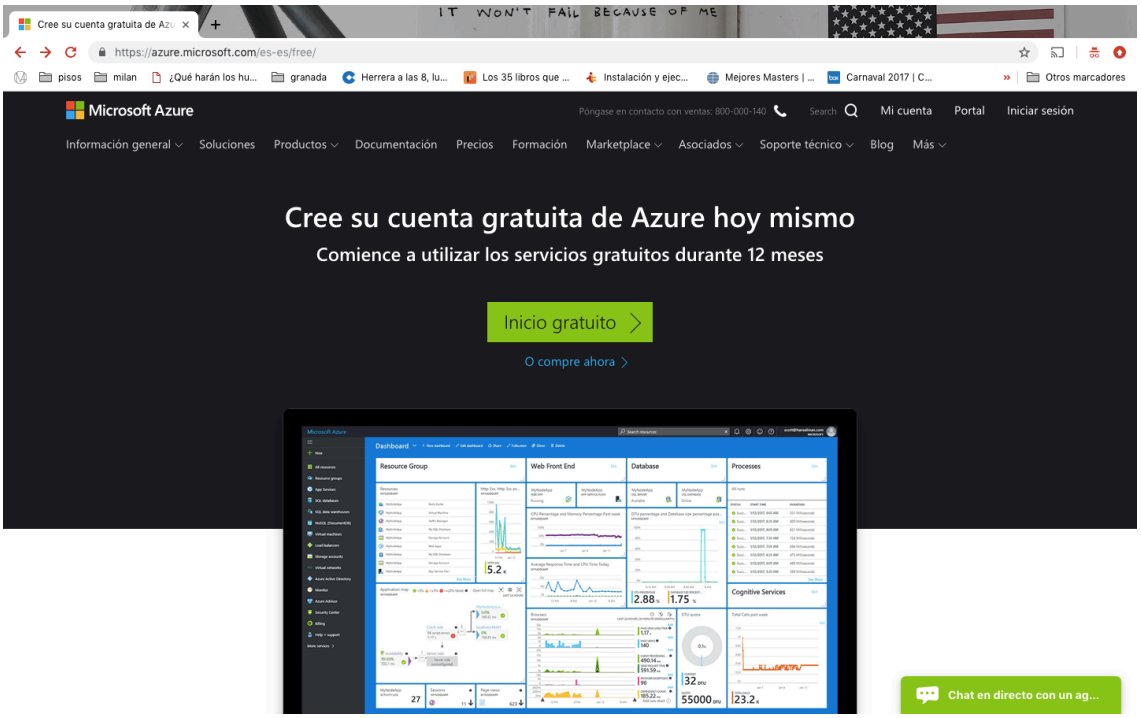


Figura 5-31 Crear cuenta Microsoft Azure

Una vez que hemos creado nuestra cuenta y accedemos al panel de recursos debemos darle a crear nuevo recurso y elegir SQL database. Una vez elegida esta opción rellenamos el formulario de la derecha (Figura 5-32).

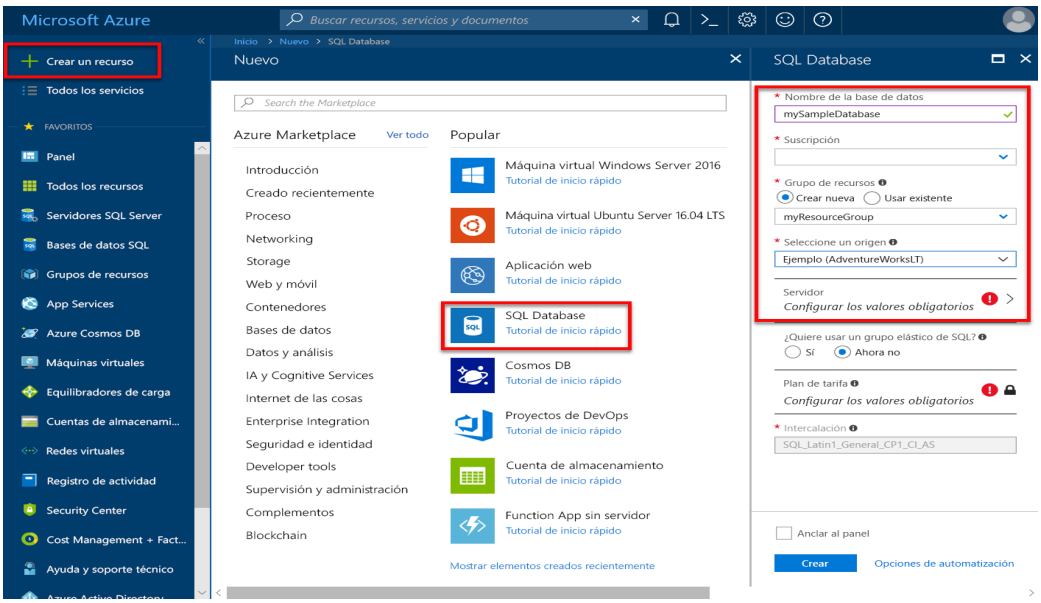


Figura 5-32 Panel de recursos de Microsoft

A Continuación, hay una peculiaridad que antes no hemos hecho. En Servidor, haga clic en Configurar los valores obligatorios y rellene el formulario de SQL Server (Figura 5-33).

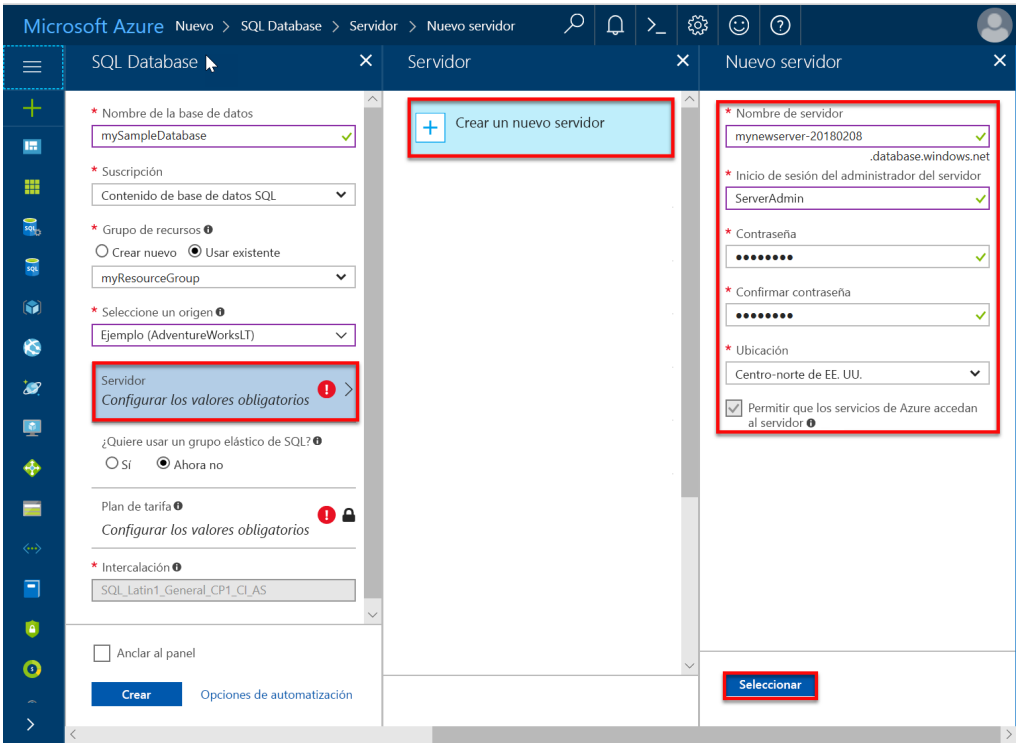


Figura 5-33 Crear base de datos en Microsoft

Una vez hemos creado la base de datos debería aparecernos algo como la figura 5-34:

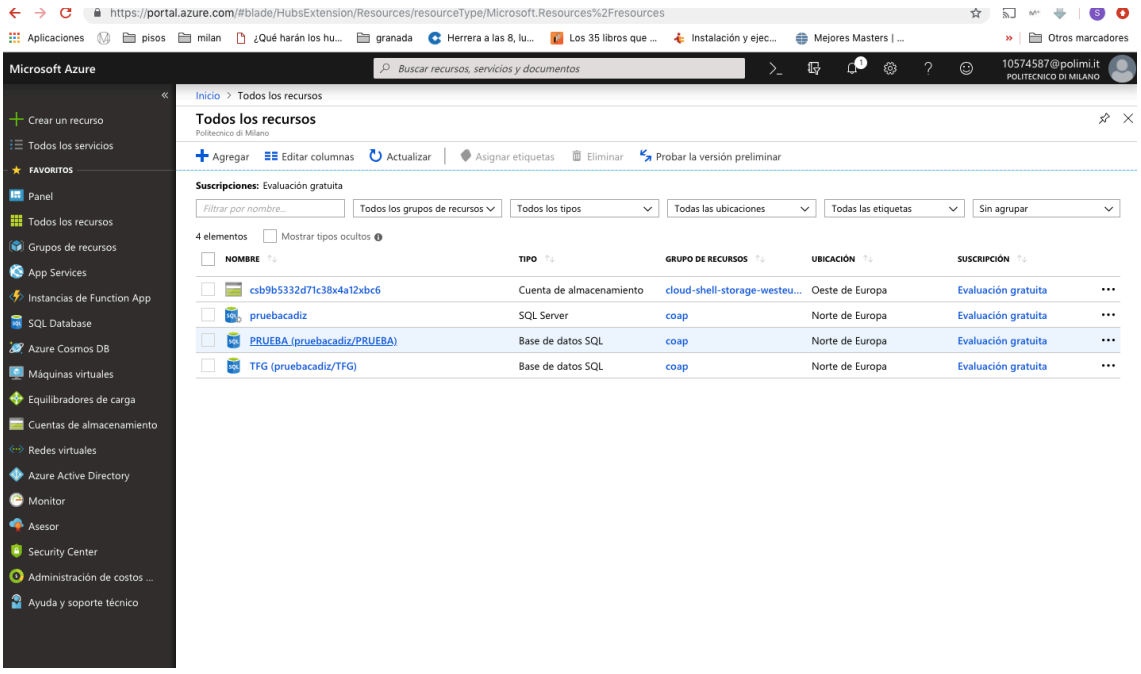


Figura 5-34 Panel de recursos creados en Microsoft

A continuación accedemos a la instancia que hemos creado. Llegaremos al panel de control de nuestra base de datos (figura 5-35). De aquí hay varios datos importantes que tenemos que recordar como el nombre del servidor para luego poder conectarnos. Por último procedemos establecer el firewall.

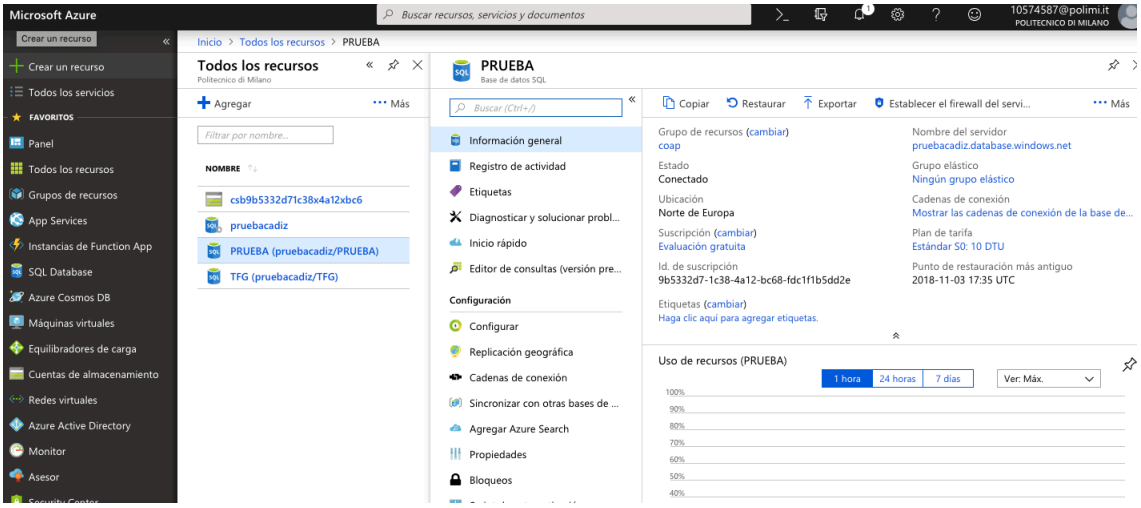


Figura 5-35 Panel de gestión de la base de datos en Microsoft

Todos los servicios cloud prohíben, por defecto, el tráfico procedente de redes no autorizadas. Por lo tanto, si queremos conectarnos a nuestra base de datos tenemos que autorizar la red desde la cual nos vamos a conectar (figura 5-36).

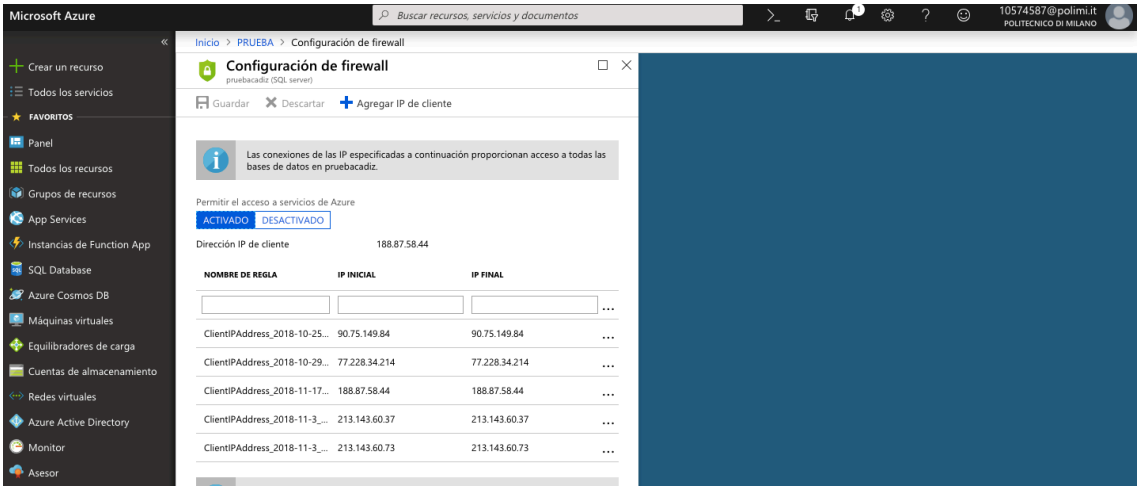


Figura 5-36 Panel de gestión de la seguridad en Microsoft

Ya está creada la instancia, ahora falta crear la base de datos y sus respectivas tablas. El procedimiento para crear las tablas y la base de datos es exactamente el mismo que el empleado en google cloud por lo que no tiene ningún interés detallar de nuevo el proceso.

Ya tenemos nuestra base de datos creada y funcionando. Nos podemos olvidar por ahora de esto y volvemos a nuestra Raspberry-pi. El proceso de conexión a esta base de datos usando python3 no es igual al seguido anteriormente por lo que se detalla a continuación.

Lo primero es instalar las siguientes librerías:

```
sudo apt-get install libffi-dev

sudo pip-3.2 install azure
```

Para comunicarnos con la base de datos vamos a utilizar pyodcb por lo que también debemos instalarlo.

```
sudo apt-get install unixodbc unixodbc-dev freetds-dev freetds-bin tdsodbc


sudo pip-3.2 install pyodbc
```

Una vez tenemos todo instalado ahora tenemos que configurar algunas dependencias. Modificamos el siguiente archivo y le añadimos las siguientes líneas al final:

```
sudo nano /etc/freetds/freetds.conf
```

```
[rpitestsqlserver]


    host = El nombre del servidor.


    port = 1433


    tds version = 8.0
```

El resultado debe ser como en la figura 5-37.

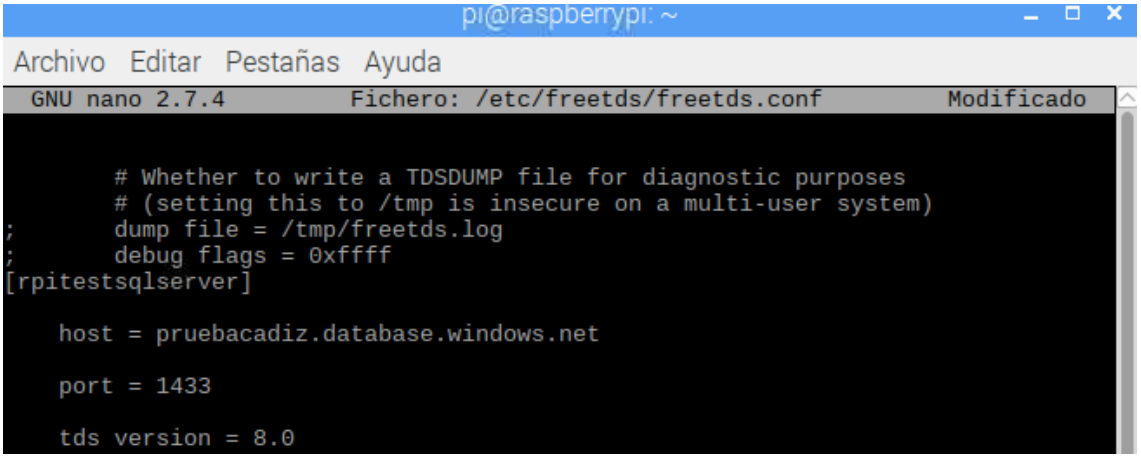


Figura 5-37 configurando ficheros.

Lo siguiente que tenemos que es averiguar la ubicación de los archivos:

```
sudo find / -name libtdsodbc.so


sudo find / -name libtdsS.so
```

Una vez comprobado que ambos archivos se encuentran en el mismo directorio procedemos a modificar el siguiente fichero al que también le añadiremos las líneas siguientes.

```
sudo nano /etc/odbcinst.ini
```

```
[FreeTDS]

Description = TDS driver (Sybase/MS SQL)

Driver = /usr/lib/arm-linux-gnueabihf/odbc/libtdsodbc.so

Setup = /usr/lib/arm-linux-gnueabihf/odbc/libtdsS.so

CPTimeout =

CPReuse =

FileUsage = 1
```

El resultado debe ser como en la figura 5-38.

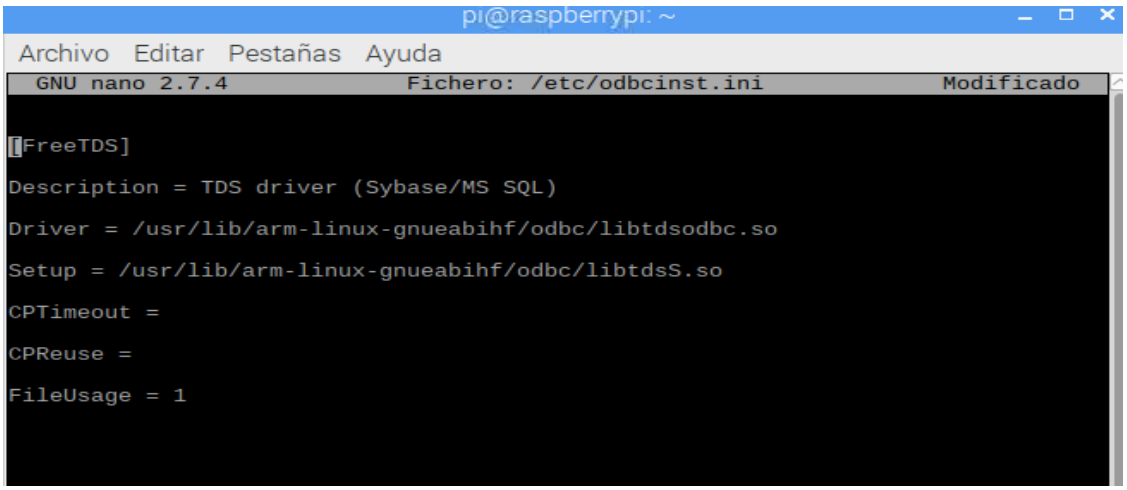


Figura 5-38 configurando ficheros 2.

Por último, modificamos el siguiente archivo para añadirle las líneas siguientes.

```
sudo nano /etc/odbc.ini
```

```
 [rpitestsqlserverdatasource]

Driver = FreeTDS

Description = ODBC connection via FreeTDS

Trace = No

Servername = nombre del servidor.

Database = nombre de la base de datos.

TDS_Version = 8.0
```

El resultado debe ser como en la figura 5-39.

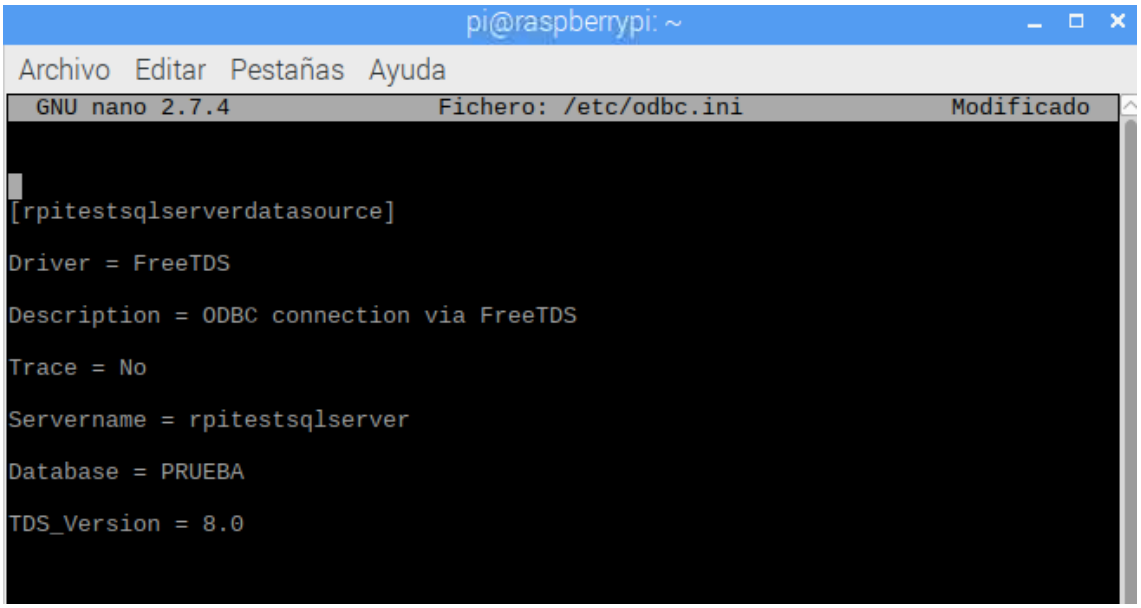


Figura 5-39 configurando ficheros 3.

Ahora ya lo tenemos todo listo para que desde el código que detallamos anteriormente podamos conectarnos a la base de datos y guardar los datos recibidos. Importamos la librería pyodbc que instalamos antes y definimos la cadena de conexión (figura 5-40). En esta cadena se especifican todos los aspectos de la base de datos necesarios para la conexión.

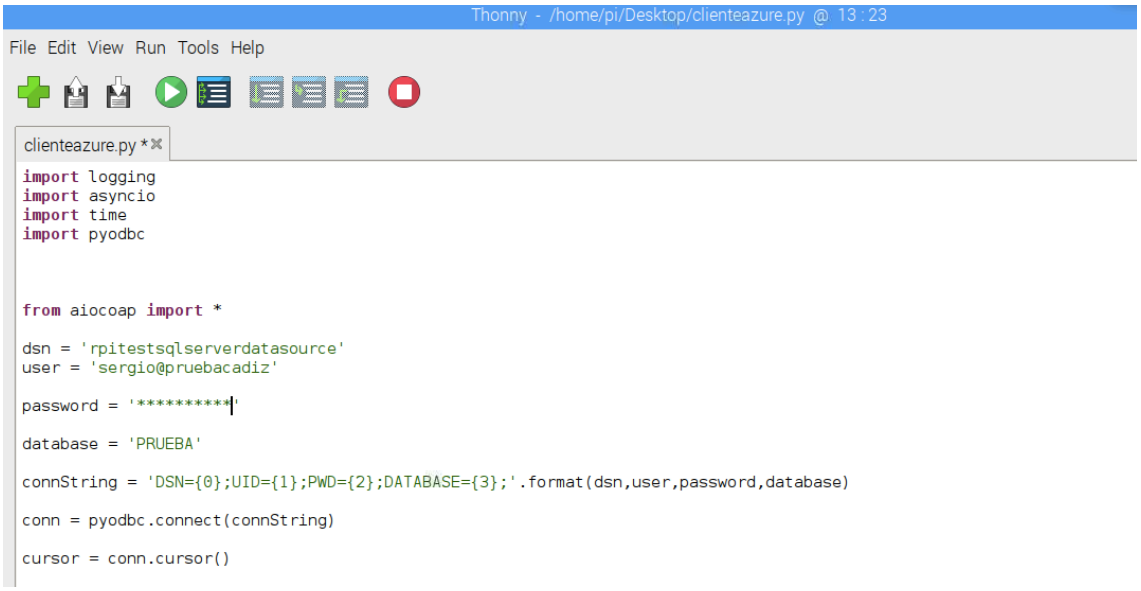


Figura 5-40 Conexión a Microsoft

Una vez ya definida la conexión a la base de datos insertamos en ella los mensajes recibidos. Cuando recibimos la respuesta se crea la query con los datos recibidos y ejecuta la operación. La operación es la misma para todos los sensores, lo único que varía son las tablas en las que se insertan los datos (Figura 5-41).

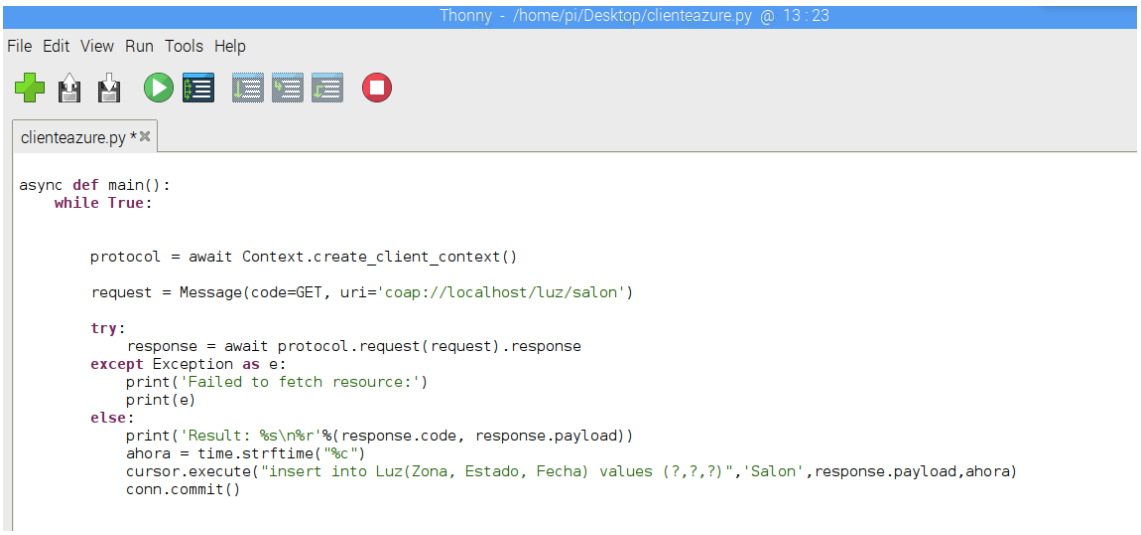


Figura 5-41 Insertando datos en Microsoft

En la figura 5-42 se puede observar el comportamiento del sistema después de ejecutarlo.

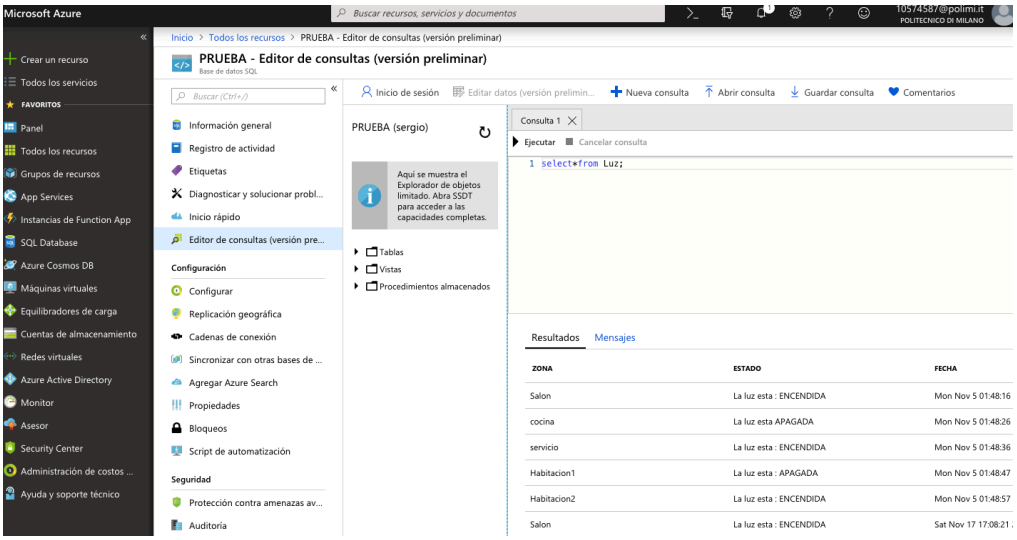


Figura 5-42 Resultado en Microsoft tras simulación

# 6 CONCLUSION

*Computing is not about computers any more. It is about living.*
*Negroponte*

Es indiscutible que el internet de las cosas ha llegado para quedarse. Cada vez son más los dispositivos que se conectan y conforme he avanzado en este proyecto seguro que la cifra ha aumentado en unos cuantos de miles. La transformación que va a causar en la vida de las personas es incuantificable y debemos quizás plantearnos cuál es el camino por delante antes de empezar a correr.

El internet de las cosas es un éxito que no se puede explicar sin la nube. La nube no solo ha supuesto un cambio en la forma de entender el almacenamiento sino también en los numerosos servicios que se han detallado en este proyecto. Antes, cuando las empresas tenían la mayoría de sus archivos en papel, el uso de ordenadores fue una auténtica revolución a la hora de gestionar la información. ¡Ahora ni siquiera hacen falta los ordenadores!

Por otra parte, el número de sensores que habrá en el mundo se contarán por billones y la forma de establecer una comunicación eficiente entre ellos jugará un papel determinante. El protocolo CoAP busca esa eficiencia en la conexión y adquiere cierta universalidad al basarse en el modelo rest.

El internet de las cosas contará con innumerables aplicaciones, algunas ya implementadas como el uso de vehículos a través de una app y otras aun por pulir como las relacionadas con la salud. Con el fin de recorrer el mejor camino posible deberemos definir que uso queremos hacer del internet de las cosas.

Una persona genera al día más datos de los que imagina, estos datos almacenados y debidamente procesados pueden servir por ejemplo para predecir el comportamiento de una persona en un día normal. Con billones de nodos sensores conectados entre sí captando todo tipo de datos no habrá ningún detalle de la vida de las personas que no se conozca. Los derechos a la intimidad y a la privacidad pueden verse expuestos si no se define una política correcta del uso de datos.

No cabe duda que el internet de las cosas junto con la nube cambiará la forma de hacer y entender la mayoría de las cosas. Debemos sin duda apostar por esta tecnología que nos traerá numerosas ventajas pero a su vez tenemos que definir que uso hacer de ella para evitar sus peores consecuencias.

# REFERENCIAS


1. Gartner,« Leading the IoT. Gartners insights on how to lead in a connected wordl », 2018 https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

2. Jacob Morgan para Forbes, «« A simple exlanation of the internet of things » Artículo, 2014 https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#59a382591d09

3. María Rueda Cruz para BBVA, «¿Qué son las Smart cities? » Artículo, 2017 https://www.bbva.com/es/las-smart-cities/

4. IoT Analytics, «The 10 most popular Internet of Things applications right now » Informe, 2016. https://iot-analytics.com/10-internet-of-things-applications/

5. Smart Grid, «Smart Grid» *Entrada en blog, 2018*. https://www.smartgridsinfo.es/smart-grid

6. Red eléctrica de España, « ¿Qué son las Smartgrid» Artículo, 2018. https://www.ree.es/es/red21/redes-inteligentes/que-son-las-smartgrid

7. Accenture, «Industrial internet of things, infographic» Informe, 2018. https://www.accenture.com/us-en/labs-insight-industrial-internet-of-things

8. Chad Morley para Medium, «7 Connected Car Trends Fueling the Future » Artículo, 2018. https://medium.com/iotforall/7-connected-car-trends-fueling-the-future-946b05325531

9. Margaria de Miguel Sánchez para Telefónica «Smart Retail Trends: una nueva experiencia digital en el punto de venta » Artículo, 2016 https://iot.telefonica.com/blog/smart-retail-trends-una-nueva-experiencia-digital-en-el-punto-de-venta

10. Daniel Newman para Forbes, « How IoT Will Impact The Supply Chain» Artículo, 2018 https://www.forbes.com/sites/danielnewman/2018/01/09/how-iot-will-impact-the-supply-chain/#3c2ee6923e37

11. Marie Aleksandrova para Eastern Peak, «IoT in Agriculture: 5 Technology Use Cases for Smart Farming (and 4 Challenges to Consider) » Artículo, 2018 https://easternpeak.com/blog/iot-in-agriculture-5-technology-use-cases-for-smart-farming-and-4-challenges-to-consider/

12. Hewlett Packard, «IoT Research » Informe, 2018 https://www.arubanetworks.com/assets/eo/HPE_Aruba_IoT_Research_Report.pdf

13. Brian Buntz para IoT World Today, « Top 10 Reasons People Aren't Embracing the IoT» Artículo,2016 https://www.iotworldtoday.com/2016/04/20/top-10-reasons-people-aren-t-embracing-iot/

14. CloudBolt, «IF IT ISN'T SELF-SERVICE, IT ISN'T A CLOUD» Entrada en blog, 2015 https://www.cloudbolt.io/blog/if-it-isnt-self-service-it-isnt-a-cloud/

15. Amazon, «¿Qué es la informática en la nube?» Página Web, 2018 https://aws.amazon.com/es/what-is-cloud-computing/

16. Joel Pérez para Oracle, «Transformando los modelos de IT a "Database as a Service (DBaaS)"» Artículo,2016 https://www.oracle.com/technetwork/es/articles/cloudcomp/oracle-cloud-dbaas-2877305-esa.html

17. Microsoft, «¿Qué es la informática en la nube?» Página Web, 2018 https://azure.microsoft.com/es-es/overview/what-is-cloud-computing/

18. IBM, «What is cloud computing?» Página Web, 2018 https://www.ibm.com/cloud/learn/what-is-cloud-computing

19. Louis Columbus para Forbes, «¿83% Of Enterprise Workloads Will Be In The Cloud By 2020? » Artículo,2018 https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/#37bf96496261

20. Logic Monitor, «Cloud Vision 2020: The Future of the Cloud A Survey of Industry Influencers » Informe,2018 https://www.logicmonitor.com/wp-content/uploads/2017/12/LogicMonitor-Cloud-2020-The-Future-of-the-Cloud.pdf

21. RightScale «cloud Computing Trends: 2018 State of the Cloud Survey» Informe, 2018 https://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2018-state-cloud-survey

22. Louis Columbus para Forbes, « Cloud Computing Market Projected To Reach $411B By 2020» Artículo,2017
https://www.forbes.com/sites/louiscolumbus/2017/10/18/cloud-computing-market-projected-to-reach-411b-by-2020/#8d3804278f29

23. Victoria Wilson para SysGroup, «10 Cloud Computing Statistics You Need To Know» Artículo 2018
https://www.sysgroup.com/resources/blog/10-cloud-computing-statistics-2018

24. SpiceWorks, «THE ANNUAL REPORT ON IT BUDGETS AND TECH TRENDS» Informe, 2018
https://www.spiceworks.com/marketing/state-of-it/report/

25. Louis Columbus para Forbes, « State Of Enterprise Cloud Computing, 2018»Artículo, 2018
https://www.forbes.com/sites/louiscolumbus/2018/08/30/state-of-enterprise-cloud-computing-2018/#2fe38354265e

26. IDG, «Executive Summary», 2018
https://cdn2.hubspot.net/hubfs/1624046/2018%20Cloud%20Computing%20Executive%20Summary.pdf?t=1541787621540

27. Jon Bashor, «Cloud Computing Saves L.A.-sized Power Bill » Estudio, 2013
http://cs.lbl.gov/news-media/news/2013/cloud-computing-energy-savings/

28. Cadie Thompson para el foro económico mundial, «8 jobs every company will be hiring for by 2020» Artículo, 2016
https://www.weforum.org/agenda/2016/01/8-jobs-every-company-will-be-hiring-for-by-2020/

29. Estorm, «Public Cloud vs Private Cloud vs Hybrid Cloud – Which Is Right For You? » Artículo,2018
https://estorm.com.au/news/public-private-hybrid-cloud-services/

30. Edwin Schouten para IBM, «Cloud computing defined: Characteristics & service levels » Entrada blog, 2014
https://www.ibm.com/blogs/cloud-computing/2014/01/31/cloud-computing-defined-characteristics-service-levels/

31. NIST, «The NIST Definition of Cloud Computing» NIST Special Publication 800-145, 2011.
http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

32. Prakhar Tyagi para Binary Informatics, « What is the Cloud Service Model?»Artículo, 2018
http://blog.binaryinformatics.com/technology/what-is-the-cloud-service-model/

33. IBM, «« IaaS, PaaS and SaaS – IBM Cloud service models» Página Web, 2018
https://www.ibm.com/cloud/learn/iaas-paas-saas

34. Microsoft, «¿Qué es IaaS?» Página Web, 2018
https://azure.microsoft.com/es-es/overview/what-is-iaas/

35. Microsoft, «¿Qué es SaaS?» Página Web, 2018
https://azure.microsoft.com/es-es/overview/what-is-saas/

36. Microsoft, «¿Qué es PaaS?» Página Web, 2018
https://azure.microsoft.com/es-es/overview/what-is-paas/

37. Google Cloud, « Quickstart for Cloud SQL for MySQL » Manual, 2018.
https://cloud.google.com/sql/docs/mysql/quickstart

38. Python, «Manual PyMySQL 0.9.2» Manual, 2018.
https://pypi.org/project/PyMySQL/

39. Google Cloud, « Connecting mysql Client Using Public IP» Manual, 2018.
https://cloud.google.com/sql/docs/mysql/connect-admin-ip

40. Chrysn para Github, «Aiocoap »Libreria, 2018
https://github.com/chrysn/aiocoap

41. Blog Python library, « Python 3 – An Intro to asyncio» Artículo, 2016
https://www.blog.pythonlibrary.org/2016/07/26/python-3-an-intro-to-asyncio/

42. Google Cloud, « Creating Instances» Manual, 2018.
https://cloud.google.com/sql/docs/mysql/create-instance

43. Google Cloud, « CLOUD SQL» Página Web, 2018.
https://cloud.google.com/sql/?hl=es

44. Amazon « Amazon RDS for MariaDB» Manual, 2018
https://aws.amazon.com/es/rds/mariadb/

45. Cloud Amazon « MariaDB en Amazon RDS» Manual, 2018
https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/CHAP_MariaDB.html

46. Amazon, «Conexión a una instancia de base de datos que ejecuta el motor de base de datos MariaDB» Manual, 2018
https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/USER_ConnectToMariaDBInstance.html

47. Amazon, « MariaDB 10.2 ya es compatible con Amazon RDS» Manual, 2018
https://aws.amazon.com/es/about-aws/whats-new/2018/01/mariadb-10-2-now-supported-on-amazon-rds/

48. Microsoft, «Guía de inicio rápido: Creación de una instancia de Azure SQL Database en Azure Portal» Manual, 2018
https://docs.microsoft.com/es-es/azure/sql-database/sql-database-get-started-portal

49. Michael Dupont, «Connecting Raspberry Pi to Microsoft Azure with Python3» Manual, 2018
http://mdupont.com/Blog/Raspberry-Pi/azure-python3.html

50. IETF, « The Constrained Application Protocol (CoAP)» RFC, 2016
https://datatracker.ietf.org/doc/rfc7252/

51. CoAP, RFC 7252 Constrained Application Protocol» 2018
http://coap.technology/

52. Eclipse Foundation, «MQTT and CoAP, IoT Protocols» Artículo, 2018
https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

53. Washington University, «Constrained Application Protocol for Internet of Things » Manual, 2018
https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html

# ANEXO A: CÓDIGO DE IMPLEMENTACIÓN

A continuación se detallan los códigos para la implementación del sistema:

- **Cliente Microsoft Azure:**

```
import logging
import asyncio
import time
import pyodbc



from aiocoap import *

#Configuracion de la conexion a la BBDD

dsn = 'rpitestsqlserverdatasource'
user = 'sergio@tfgcoap'

password = '*****'

database = 'coap'

connString                                                                    =
'DSN={0};UID={1};PWD={2};DATABASE={3};'.format(dsn,user,password,database)

conn = pyodbc.connect(connString)

cursor = conn.cursor()

#Configuracion ubicacion de recursos

salon='coap://localhost/luz/salon'
```

```
salon1='Salon'
cocina='coap://localhost/luz/cocina'
cocina1='Cocina'
servicio='coap://localhost/luz/servicio'
servicio1='Servicio'
room1='coap://localhost/luz/habitacion1'
rooma='Habitacion 1'
room2='coap://localhost/luz/habitacion2'
roomb='Habitacion 2'
temp='coap://localhost/temp'
frg='coap://localhost/frigorifico'

#Configurarion datos BBDD

SQL_luz= "INSERT INTO Luz(Zona, Estado,Fecha) VALUES (?, ?,?)"
SQL_TEMP= "INSERT INTO Temperatura(Fecha,Temperatura) VALUES (?, ?)"
SQL_FRG= "INSERT INTO frigorifico(Fecha, Estado) VALUES (?,?)"



#Funcion que recibe datos de los sensores y los guarda en la BBDD.

async def asking (ubicacion,zona,query,opcion):

#Crear contexto coap y peticion GET.

    protocol = await Context.create_client_context()

    request = Message(code=GET, uri=ubicacion)

    if opcion == 1:

# si los sensores son de Luz.

        try:
```

```
     #Realizamos peticion  y esperamos respuesta.
            response = await protocol.request(request).response



        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
        else:
    #Procesamos la respuesta y la almacenamos en la BBDD.
            print('Result: %s\n%r'%(response.code, response.payload))
            ahora = time.strftime("%c")
            with conn.cursor() as cursor:
                sql = query
                cursor.execute(sql,zona,response.payload,ahora)
                conn.commit()


                time.sleep(2)


    if opcion == 2:


#si los sensores son de temperatura.


        try:
            response = await protocol.request(request).response
        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
        else:
            print('Result: %s\n%r'%(response.code, response.payload))
            ahora = time.strftime("%c")
            with conn.cursor() as cursor:
                sql = query
                cursor.execute(sql,ahora,response.payload)
```

```
                conn.commit()
                time.sleep(2)




async def main():
    while True:
        await asking(salon,salon1,SQL_luz,1)

        await asking(cocina,cocina1,SQL_luz,1)

        await asking(servicio,servicio1,SQL_luz,1)

        await asking(room1,rooma,SQL_luz,1)

        await asking(room2,roomb,SQL_luz,1)

        await asking(temp,roomb,SQL_TEMP,2)

        await asking(frg,roomb,SQL_FRG,2)




if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
    conn.close()
    quit()
```

- **Cliente Google Cloud:**

```
import logging
import asyncio
import time
import pymysql.cursors



from aiocoap import *

#Configuracion de la conexion a la BBDD

logging.basicConfig(level=logging.INFO)
connection = pymysql.connect(host='***.***.161.55',
                             user='*****',
                             password='*********',
                             db='tfg',
                             charset='utf8mb4',
                             cursorclass=pymysql.cursors.DictCursor)

#Configuracion ubicacion de recursos

salon='coap://localhost/luz/salon'
salon1='Salon'
cocina='coap://localhost/luz/cocina'
cocina1='Cocina'
servicio='coap://localhost/luz/servicio'
servicio1='Servicio'
room1='coap://localhost/luz/habitacion1'
rooma='Habitacion 1'
room2='coap://localhost/luz/habitacion2'
roomb='Habitacion 2'
temp='coap://localhost/temp'
```

```
frg='coap://localhost/frigorifico'

#Configurarion datos BBDD

SQL_luz= "INSERT INTO `Luz` (`Zona`, `Estado`,`Fecha`) VALUES (%s, %s,%s)"
SQL_TEMP= "INSERT INTO `Temperatura` (`Fecha`, `Temperatura`) VALUES (%s, %s)"
SQL_FRG= "INSERT INTO `frigorifico` (`Fecha`, `Estado`) VALUES (%s, %s)"

#Funcion que recibe datos de los sensores y los guarda en la BBDD.

async def asking (ubicacion,zona,query,opcion):

    #Crear contexto coap y peticion GET.

    protocol = await Context.create_client_context()

    request = Message(code=GET, uri=ubicacion)

    if opcion == 1:

        # si los sensores son de Luz.

        try:
            #Realizamos peticion  y esperamos respuesta.
            response = await protocol.request(request).response
        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
        else:
            #Procesamos la respuesta y la almacenamos en la BBDD.
            print('Result: %s\n%r'%(response.code, response.payload))
            ahora = time.strftime("%c")
            with connection.cursor() as cursor:
                sql = query
```

```
                cursor.execute(sql, (zona,response.payload,ahora ))
                connection.commit()

                time.sleep(2)

    if opcion == 2:

        #si los sensores son de temperatura.

        try:
            response = await protocol.request(request).response
        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
        else:
            print('Result: %s\n%r'%(response.code, response.payload))
            ahora = time.strftime("%c")
            with connection.cursor() as cursor:
                sql = query
                cursor.execute(sql, (ahora,response.payload))
                connection.commit()
                time.sleep(2)



async def main():
    while True:




        await asking(salon,salon1,SQL_luz,1)

        await asking(cocina,cocina1,SQL_luz,1)
```

```
        await asking(servicio,servicio1,SQL_luz,1)

        await asking(room1,rooma,SQL_luz,1)

        await asking(room2,roomb,SQL_luz,1)

        await asking(temp,roomb,SQL_TEMP,2)

        await asking(frg,roomb,SQL_FRG,2)




if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
    connection.close()
```

- **Cliente Amazon Web Services:**

```
import logging
import asyncio
import time
import pymysql.cursors


from aiocoap import *

#Configuracion de la conexion a la BBDD


logging.basicConfig(level=logging.INFO)

connection         =         pymysql.connect(host='****.cz79x5tczdrt.us-west-
2.rds.amazonaws.com',
```

```
                            user='*******',
                            password='***********',
                            db='tfg2',
                            charset='utf8mb4',
                            cursorclass=pymysql.cursors.DictCursor)

#Configuracion ubicacion de recursos

salon='coap://localhost/luz/salon'
salon1='Salon'
cocina='coap://localhost/luz/cocina'
cocina1='Cocina'
servicio='coap://localhost/luz/servicio'
servicio1='Servicio'
room1='coap://localhost/luz/habitacion1'
rooma='Habitacion 1'
room2='coap://localhost/luz/habitacion2'
roomb='Habitacion 2'
temp='coap://localhost/temp'
frg='coap://localhost/frigorifico'

#Configurarion datos BBDD

SQL_luz= "INSERT INTO `Luz` (`Zona`, `Estado`,`Fecha`) VALUES (%s, %s,%s)"
SQL_TEMP= "INSERT INTO `Temperatura` (`Fecha`, `Temperatura`) VALUES (%s,
%s)"
SQL_FRG= "INSERT INTO `frigorifico` (`Fecha`, `Estado`) VALUES (%s, %s)"

#Funcion que recibe datos de los sensores y los guarda en la BBDD.
async def asking (ubicacion,zona,query,opcion):

    #Crear contexto coap y peticion GET.

    protocol = await Context.create_client_context()
```

```
    request = Message(code=GET, uri=ubicacion)

    if opcion == 1:

# si los sensores son de Luz.

        try:
            #Realizamos peticion  y esperamos respuesta.
            response = await protocol.request(request).response
        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
        else:
             #Procesamos la respuesta y la almacenamos en la BBDD.
            print('Result: %s\n%r'%(response.code, response.payload))
            ahora = time.strftime("%c")
            with connection.cursor() as cursor:
                sql = query
                cursor.execute(sql, (zona,response.payload,ahora ))
                connection.commit()

                time.sleep(2)

    if opcion == 2:

        #si los sensores son de temperatura.



        try:
            response = await protocol.request(request).response
        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
```

```
        else:
            print('Result: %s\n%r'%(response.code, response.payload))
            ahora = time.strftime("%c")
            with connection.cursor() as cursor:
                sql = query
                cursor.execute(sql, (ahora,response.payload))
                connection.commit()
                time.sleep(2)



async def main():
    while True:

        await asking(salon,salon1,SQL_luz,1)

        await asking(cocina,cocina1,SQL_luz,1)

        await asking(servicio,servicio1,SQL_luz,1)

        await asking(room1,rooma,SQL_luz,1)

        await asking(room2,roomb,SQL_luz,1)

        await asking(temp,roomb,SQL_TEMP,2)

        await asking(frg,roomb,SQL_FRG,2)



if __name__ == "__main__":
    asyncio.get_event_loop().run_until_complete(main())
    connection.close()
```

- **Servidor CoAp:**

```
        import datetime
import logging
import random
import asyncio

import aiocoap.resource as resource
import aiocoap

#Definicion de variables para simular estado de las luces.

state= "ENCENDIDA"

#Definicion de los tipos de sensores

class salonResource(resource.ObservableResource):



    def __init__(self):
        super().__init__()

        self.handle = None

#Definición metodo GET.

    async def render_get(self, request):
        global state
        set1 = "ENCENDIDA"
        set2= "APAGADA"
        resultado ="La luz esta : %s " %(state)  #Estado del sensor.
        payload = resultado.encode('ascii')
        if state== set1:
            state=set2
```

```
        else :
            state= set1
        #Devuelve el estado del sensor.
        return aiocoap.Message(payload=payload)


class tempResource(resource.ObservableResource):


    def __init__(self):
        super().__init__()

        self.handle = None


    async def render_get(self, request):

        resultado = random.randint(0,50) #Definimos una temperatura aleatoria
entre 0 y 50 grados
        resultado2= " La temperatura es de %d C" %(resultado)
        payload = resultado2.encode('ascii')

        return aiocoap.Message(payload=payload)

class frigorificoResource(resource.ObservableResource):


    def __init__(self):
        super().__init__()

        self.handle = None
```

```
    async def render_get(self, request):


        temperatura = random.randrange(-8, 5)
        if temperatura >= 0 :


            resultado2= " El congelador ha dejado de funcionar T: %d C"
%(temperatura)
            payload = resultado2.encode('ascii')
        else :
            resultado = "El congelador sigue enfriando T: %d C"
%(temperatura)
            payload = resultado.encode('ascii')
        return aiocoap.Message(payload=payload)




logging.basicConfig(level=logging.INFO)
logging.getLogger("coap-server").setLevel(logging.DEBUG)

def main():


    root = resource.Site()
#Configuracion de recursos en el servidor.


    root.add_resource(('.well-known', 'core'),
            resource.WKCResource(root.get_resources_as_linkheader))
    root.add_resource(('temp',), tempResource())
    root.add_resource(('frigorifico',), frigorificoResource())
    root.add_resource(('luz', 'salon'), salonResource())
    root.add_resource(('luz', 'servicio'), salonResource())
    root.add_resource(('luz', 'cocina'), salonResource())
    root.add_resource(('luz', 'habitacion1'), salonResource())
    root.add_resource(('luz', 'habitacion2'), salonResource())


#configuracion contexto del servidor
```

```
    asyncio.Task(aiocoap.Context.create_server_context(root))

#Bucle a la espera de peticiones.

    asyncio.get_event_loop().run_forever()

if __name__ == "__main__":
    main()
```

- **Aplicación de inicio:**

```
#!/bin/sh

echo""
echo""

echo "Pasarela protocolo CoAP sobre Raspberry Pi a DBaaS"
echo""
echo""

echo "Si usas:"
echo " Google cloud pulsa 1."
echo " AMAZON web services pulsa 2."
echo " Microsoft Azure pulsa 3."
echo""
echo""
read -p " Elija su opción :" numero

echo "Opcion elegida : $numero"
echo""
echo""

if  [ $numero -eq 1 ]
```

```
      then
            echo "Estas usando Google Cloud"
            python3 /home/pi/Desktop/tfg2/clientegoogle.py

      fi

if  [ $numero -eq 2 ]

      then
            echo "Estas usando AMAZON web services"
            python3 /home/pi/Desktop/tfg2/clienteaws.py

      fi

if  [ $numero -eq 3 ]

      then
            echo "Estas usando Microsoft Azure"
            python3 /home/pi/Desktop/tfg2/clienteazure.py

      fi
```