Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de servicios IoT mediante IFTT. Aplicando los principios del Fog Computing al proyecto ecoMobility

Trabajo Fin de Master

Master en Ingeniería y Tecnologías de Sistemas Software

Autor: Fabián Martínez González

Tutor: Joan Fons i Cors

Julio 2018

Resumen

En la actualidad los sistemas IoT están en auge. Esto provocará un incrementará drástico del número total de dispositivos conectados a internet. Esa masiva cantidad de dispositivos conectados vislumbra futuros problemas de conectividad en entornos masificados. Para esto surge una solución para optimizar los sistemas IoT, La computación en Fog.

En este proyecto se estudiará, diseñará e implementará una plataforma con la que desplegar soluciones en entornos de computación en Fog. El objetivo es avanzar en la investigación del modelo de computación en Fog y crear una plataforma con la que desplegar soluciones de una forma similar a la que se haría en casos reales.

Primero se plantea una plataforma funcional pero con un entorno de ejecución simplificado y alejado de las necesidades reales de uso del Fog Computing.

Seguidamente se plantea una nueva iteración sobre el diseño inicial que utiliza las tecnologías más recientes para aproximarse a casos de uso reales.

Finalmente se aplica la plataforma creada sobre un caso de estudio del proyecto ecoMobility para mejorar ciertas partes y sustituirlas por un enfoque basado en la computación en Fog.

Palabras clave: IoT, Fog Computing, IFTT, Docker.

Abstract

Nowadays popularity of IoT systems is growing. Because of this the total number of connected devices will increase dramaticaly. This massive amount of connected devices distinguish future connectivity problems in crowded environments. Fog Computing comes up to optimize IoT systems.

In this Project will be studied, designed and implemented a platform for Fog Computing environments. The target is to progress the research of Fog Computing and to create a platform with which to deploy solutions in a way similar to what would be done in real cases.

First it's presented a simplified but functional platform wich isn´t useful for real use cases of Fog Computing.

Then it's presented a new versión using new technologies. This versión is meant to be useful for real use cases.

Finally the created platform is applied to the ecoMobility project. Some parts of ecoMobility were redesigned with a Fog Computing approach.

Keywords: IoT, Fog Computing, IFTT, Docker.

Tabla de contenidos

Tabla de contenidos 5
Índice de figuras 7
1 Introducción 9
1.1 Motivación 9
1.2 Objetivos 11
1.3 Relación con el proyecto ecoMobility 11
1.4 Metodología y plan de trabajo 11
2 Contexto tecnológico 12
2.1.1 Entorno de desarrollo 13
2.1.2 Lenguaje de programación: Java 14
2.1.3 Framework de desarrollo: Spring Boot 14
2.1.4 Gestor de dependencias: Maven 14
2.1.5 Base de datos: MongoDB 15
2.1.6 Docker 15
2.1.7 Spotify Docker Client 16
2.1.8 JGit 16
3 Computación en Fog 17
3.1 Historia del Fog Computing 17
3.2 Conceptos básicos 19
3.3 Problemas identificados 21
4 Diseño de la solución 22
4.1 Planteamiento general del diseño de la plataforma 24
4.1.1 Red Fog 25
4.1.2 Fog Node 25
4.1.3 Fog Manager 26
4.2 Propuesta de diseño de la plataforma 27
4.2.1 Fog Node 27
4.2.2 Fog Manager 27
4.2.3 Ciclo de vida del Fog Node 28
4.2.4 Despliegue de software en la plataforma Fog 29
4.3 Diseño de la plataforma con IFTT 30
4.4 Rediseño de la plataforma 33
4.5 Diseño de la plataforma con contenedores 35
5 Implementación 38

5.1 Implementación del Fog Node 38
5.1.1 Versión con IFTT 39
5.1.2 Versión con contenedores Docker 41
5.2 Implementación del Fog Manager 44
5.2.1 Red Fog 45
6 Caso de Estudio: ecoMobiliy 47
6.1 Descripción del problema 47
6.2 Solución propuesta 47
6.2.1 Detalles de implementación 50
6.2.2 Puesta en marcha 52
7 Conclusiones 61
8 Referencias 62

Índice de figuras

Figura 1 Crecimiento estimado del total de dispositivos IoT conectados hasta 2025 9
Figura 2 Entorno de desarrollo: Eclipse Oxygen.....13
Figura 3 Spring Boot.....14
Figura 4 Ejemplo de pom.xml en proyecto multimódulo 15
Figura 5 Docker 16
Figura 6 Repositorio GitHub de Spotify Docker Client.....16
Figura 7 Pilares de la arquitectura de referencia de OpenFog 18
Figura 8 Ejemplo Fog Computing 19
Figura 9 Ejemplo estructura plataforma Fog 24
Figura 10 Ejemplo red Fog con tres capas..... 25
Figura 11 Ejemplo intercambio de mensajes en el ciclo de vida de un Fog Node 29
Figura 12 Ejemplo estructura interna del Fog Node basado en contenedores Docker .. 35
Figura 13 Componentes internos Fog Node versión IFTT 39
Figura 14 Código ejecución script en "nashorn" con contexto configurado..... 40
Figura 15 Ejemplo servicios Docker en un Fog Node.....41
Figura 16 Código despliegue servicio en Docker 42
Figura 17 Código construcción de imagen Docker a partir de repositorio Git 43
Figura 18 Muestra código "geospatial query" y algoritmo red Fog 45
Figura 19 Diagrama con ejemplo de despliegue de la solución propuesta..... 48
Figura 20 Muestra máquinas virtuales en la web de gestión de las máquinas virtuales del DSIC..... 52
Figura 21 Representación red Fog utilizada en la prueba de la solución propuesta..... 53
Figura 22 Muestra FogNodeManager arrancando..... 53
Figura 23 Ejemplo consulta de nodos registrados 54
Figura 24 Consulta de nodos registrados en la prueba de la solución propuesta.....55
Figura 25 Muestra petición para guardar configuración del proyecto en la prueba de la solución propuesta..... 56
Figura 26 Respuesta a la petición de guardado de la configuración del proyecto 57
Figura 27 Muestra petición y respuesta para el despliegue del proyecto de la solución propuesta.....57
Figura 28 Muestra petición registro controlador de tráfico en un ctlcs-manager..... 58
Figura 29 Prueba consulta estado controlador de tráfico, semáforo verde 58
Figura 30 Prueba consulta estado controlador de tráfico, semáforo rojo..... 58
Figura 31 Muestra petición con lectura contaminación al pollution-controller 59
Figura 32 Muestra logs pollution-controller tras procesar lectura de contaminación .. 59
Figura 33 Muestra del estado final del controlador de tráfico tras la prueba realizada, semáforo corta el tráfico..... 59
Figura 34 Muestra petición al pollution-controller con lectura de contaminación dentro de los niveles permitidos 60
Figura 35 Muestra logs pollution-controller tras procesar lectura de contaminación dentro de los niveles permitidos 60
Figura 36 Estado final del controlador de tráfico tras procesado de lectura dentro de los niveles permitidos, semáforo verde funcionando de forma normal 60

1 Introducción

1.1 Motivación

En los próximos años el entorno IoT se verá potenciado drásticamente ya que este tipo de soluciones tecnológicas se popularizarán llegando a aplicarse en todos los aspectos de nuestras vidas. Lo que algunos llaman el internet de todas las cosas traerá consigo una cantidad masiva de dispositivos conectados jamás vista. En pocos años el número de dispositivos conectados a internet incrementará hasta superar los 50 billones [1]. Estos dispositivos formarán parte de sistemas de toda índole como por ejemplo sistemas críticos, sistemas de tiempo real, sistemas que manejan una cantidad de datos masiva,... Todos estos sistemas implementados en el entorno IoT traerán consigo incremento masivo en el uso de las comunicaciones a través de internet y de la computación en general. Los requerimientos tecnológicos que generará esta situación han planteado muchos retos que ya llevan unos años siendo investigados.

En la siguiente imagen se muestra el crecimiento estimado desde 2015 hasta 2025 de el número total de dispositivos conectados en total por todo el mundo en billones.

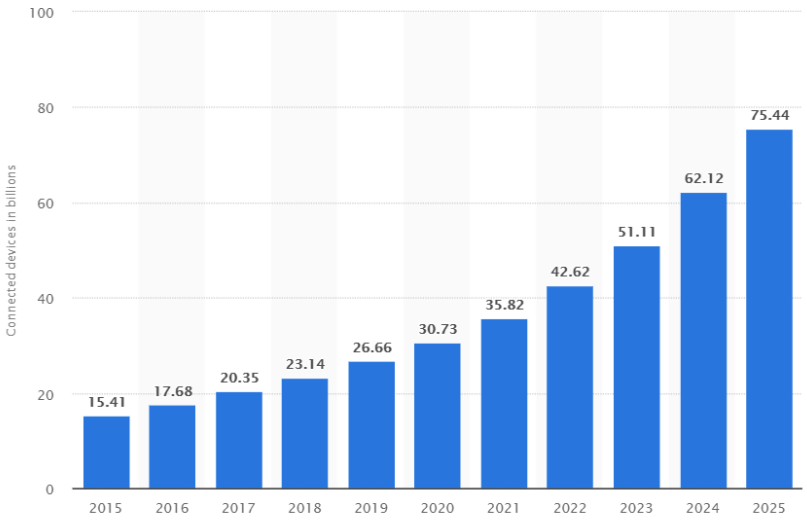

Figura 1 Crecimiento estimado del total de dispositivos IoT conectados hasta 2025

De uno de estos retos surgió la denominada 4º plataforma o Fog Computing, esta nueva plataforma surge para suplir las carencias y problemas que tiene el modelo actual de computación basado en el Cloud. El modelo de computación basado en el Cloud en escenarios como el comentado anteriormente tiene serios problemas para ofrecer tiempos de respuesta aceptables en un sistema de tiempo real o simplemente es totalmente ineficiente para un sistema que genera terabytes de datos cada día de forma remota y que necesitan ser procesados.

Como decía, hablamos de sistemas del entorno IoT, de forma general se puede decir que estos sistemas se basan en captar datos mediante sensores y procesarlos para o bien obtener información de valor o bien realizar alguna acción en el mundo físico mediante el uso de actuadores.

En un sistema IoT relativamente simple la carga de procesos no es lo suficientemente elevada y por lo tanto el sistema puede funcionar apoyándose 100% en el Cloud para la ejecución de sus procesos. Pero si pensamos en miles de sistemas IoT, simples y complejos, conviviendo en un determinado espacio físico, por ejemplo una Smart City, es fácil comprender que la capacidad de la red de ese lugar puede verse afectada seriamente si todos estos sistemas intentan conectarse con servicios Cloud que no necesariamente están cerca(físicamente hablando).

Los principales problemas por los que comenzó la investigación sobre el Fog Computing son: futuras redes saturadas y fuerte dependencia con servicios remotos que están lejos (físicamente). Derivado de esto último surgen elevados tiempos respuesta y elevados tiempos de transferencia para grandes volúmenes de datos, además de la ineficiencia que esto supone.

A grandes rasgos los sistemas basados en la computación en Fog buscan utilizar de forma más eficiente la red. Esto lo consiguen creando sistemas de computación distribuidos, acercando la capacidad de procesos a la fuente de los datos y por ende ejecutando los procesos cerca (físicamente) de la fuente de los datos. La conocida filosofía de divide y vencerás encaja perfectamente en la idea de computación en Fog, pues la computación en Fog con su naturaleza distribuida busca hacer un uso eficiente de la red y hacer más manejable la carga de procesos. Sin embargo esto no significa que el Cloud tenga que desaparecer, estos sistemas pueden delegar ciertos procesos, menos críticos, a servicios en el Cloud.

El principal reto que plantea este nuevo modelo de computación y lo que lo diferencia principalmente del Cloud, es su naturaleza distribuida. En un entorno Cloud el hardware donde se ejecuta el software está perfectamente localizado y controlado, los cambios en la topología de red no son comunes y las herramientas de distribución de software están más que probadas. Sin embargo en un entorno Fog el hardware está distribuido, la topología de red puede cambiar con relativa frecuencia y no hay herramientas ni probadas ni populares para la distribución de software por el entorno.

Teniendo en cuenta todo lo dicho se hace palpable la necesidad de una plataforma de computación distribuida que permita desplegar y gestionar software para ejecutar en entornos Fog.

1.2 Objetivos

Los objetivos planteados para este trabajo de fin de máster son:

- Crear una plataforma para desplegar de forma simple soluciones IoT en entornos de computación Fog.
- Diseñar dicha plataforma con un enfoque de compartición de recursos para posibilitar ofrecer el acceso a la plataforma como un servicio.
- Implementar y desplegar un sistema IoT en un entorno Fog con dicha plataforma.

1.3 Relación con el proyecto ecoMobility

Este proyecto surge, entre otras cosas, para realizar una de las posibles mejoras de diseño sobre el proyecto ecoMobility. En concreto se ha elegido rehacer el sistema de gestión y respuesta ante alarmas de contaminación. Dicho sistema utilizaba un servicio centralizado para realizar toda la gestión y respuesta, abrir y cerrar semáforos, ante alarmas de tráfico. Debido a que la tarea de controlar los semáforos consiste básicamente en acceder a controladores alojados en los semáforos. Esta es una tarea perfecta para aplicar el concepto de división y distribución del trabajo que encaja a la perfección para aplicar en un entorno de computación Fog.

1.4 Metodología y plan de trabajo

La metodología seguida para la realización de este trabajo ha sido similar al modelo iterativo de desarrollo de software. Concretamente se han realizado dos iteraciones dicha metodología de desarrollo de software. Las etapas de cada una de estas iteraciones ha sido: análisis, diseño, implementación y pruebas.

Respecto al plan de trabajo se prevé realizarlo siguiendo los siguientes pasos:

- Estudio de los conceptos clave del trabajo y de la documentación disponible
- Análisis y propuesta de una solución
- Diseño detallado de la solución
- Implementación de la solución
- Prueba de la solución con su aplicación en un determinado caso de estudio.
- Redacción de la memoria

2 Contexto tecnológico

En este capítulo introduciré el contexto tecnológico relacionado con el trabajo realizado con el fin de dar una visión, desde el punto de vista tecnológico, de los conceptos clave de cómo surgió el Fog Computing. Finalmente aprovecharé para comentar las diferentes tecnologías y herramientas que han sido empleadas para realizar este trabajo

Este trabajo está relacionado con el entorno del Internet de las Cosas (IoT) porque en él se estudia un nuevo modelo de computación, el Fog Computing, que como indican sus precursores surge para dar respuesta los requerimientos tecnológicos del IoT.

Cuando se dice que el Fog Computing surge para dar respuesta a los requerimientos tecnológicos del IoT nos referimos a que el Fog Computing es una nuevo modelo de computación que sobre el papel soluciona los problemas que tiene el IoT en ciertos casos de uso cuando se aplica en conjunto con el modelo de computación usado actualmente, el Cloud Computing.

Surgen varios problemas al desarrollar grandes soluciones IoT apoyándose únicamente en el Cloud Computing.

Uno de los problemas más acusados por los precursores del Fog Computing es la mala eficacia en sistemas de tiempo real que requieren de respuestas rápidas, esta mala eficacia se debe a la latencia de las conexiones de red con servicios Cloud.

Del que tiene el IoT con el Cloud Computing se pueden interpretar otros, la eficiencia y la seguridad. Los sistemas IoT diseñados con una dependencia del 100% en servicios Cloud se pueden ver como sistemas centralizados, ya que todos sus procesos de negocio dependen de un único punto centralizado y localizado.

Digo que un sistema centralizado de este tipo tiene problemas de seguridad porque al ser centralizado hay un único punto de fallo, y ante caídas de dicho punto el sistema entero estaría inoperativo. Por otro lado hay un problema de eficiencia porque si las conexiones con los servicios Cloud son constantes y hay un movimiento de datos considerable habría una latencia notable en las comunicaciones de red y el sistema en general no funcionaría con la rapidez que se esperaría para sistemas de tiempo real o simplemente no sería efectivo en un sistema crítico como pueden ser muchos de los sistemas desarrollados en el ámbito del IoT.

Pero el surgimiento de este nuevo modelo de computación no significa que la computación en Cloud deba desaparecer. El Fog Computing tampoco es perfecto, los precursores del Fog Computing abogan por una convivencia de estos dos modelos de computación para que se complementen entre ellos y así conseguir mejores sistemas IoT que ofrezcan mejores niveles de servicio en general.

Por esta razón las tecnologías empleadas en el ámbito de la ingeniería del software no tienen necesariamente porque cambiar. Me refiero a que la tecnología actual como las herramientas para el desarrollo de software, los lenguajes de programación empleados, los frameworks utilizados para desarrollar, … no es estrictamente necesario sustituirlas por nuevas tecnologías de este tipo que sea necesario desarrollar. Las tecnologías de

desarrollo actuales son válidas para la transición al desarrollo de software para Fog Computing.

Es importante marcar que si bien las tecnologías para el desarrollo de software no van a ser sustituidas, las tecnologías como plataformas para el despliegue de software o tecnologías de administración de sistemas no van a tener la misma suerte con la llegada de la computación en Fog. Este tipo de tecnologías mencionadas en el ámbito de la gestión de sistemas son las que necesariamente han de ser creadas para hacer realidad la computación en Fog, ya que el cambio drástico respecto al Cloud Computing es el cambio generalizado al uso de sistemas distribuidos y por lo tanto serán necesarias nuevas herramientas y tecnologías para la gestión de sistemas distribuidos del ámbito del Fog Computing.

A continuación haré un breve repaso de las tecnologías del ámbito del desarrollo de software que se han utilizado para realizar este trabajo.

2.1.1 Entorno de desarrollo

El entorno de desarrollo o IDE (Integrated Development Environment) utilizado para desarrollar todas las aplicaciones necesarias para este proyecto ha sido Eclipse, concretamente se ha utilizado la versión estable Oxygen 4.7.2 [6].

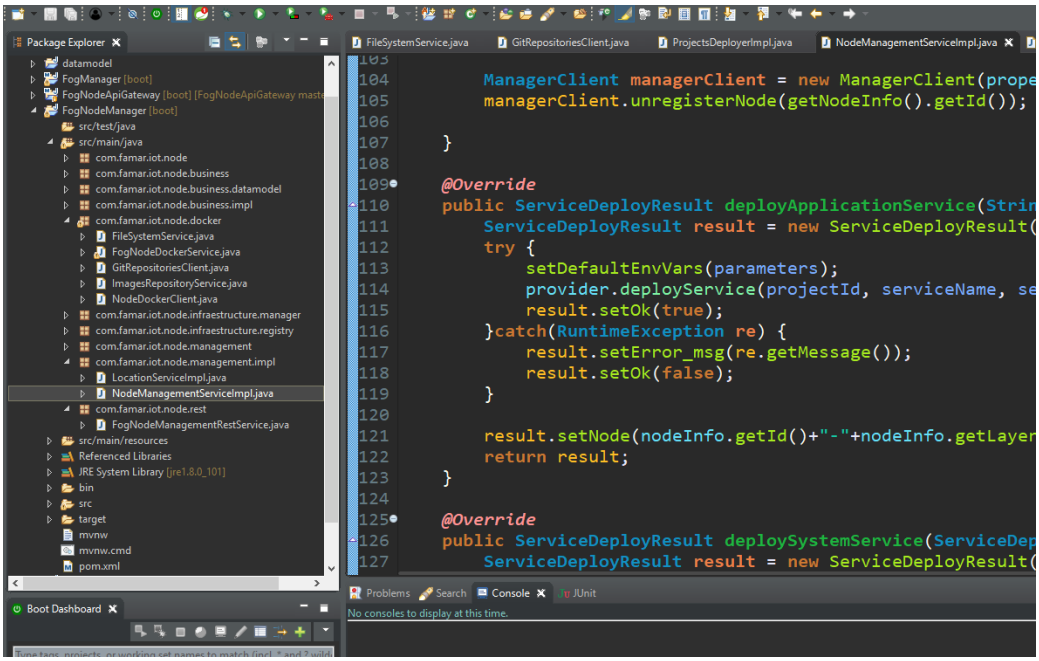

Figura 2 Entorno de desarrollo: Eclipse Oxygen

2.1.2 Lenguaje de programación: Java

Java [7] ha sido el lenguaje de programación utilizado para desarrollar todas las aplicaciones necesarias para este trabajo. Java es un lenguaje de programación muy popular, es un lenguaje de programación de propósito general, orientado a objetos y concurrente. Java es un lenguaje multiplataforma porque es un lenguaje que se compila a bytecode, que puede ser ejecutado desde cualquier máquina virtual Java (JVM) sin importar la arquitectura del computador en la que se ejecute. En la actualidad el lenguaje de programación Java es propiedad de la empresa Oracle.

2.1.3 Framework de desarrollo: Spring Boot

Spring Boot ha sido el framework utilizado para desarrollar todas las aplicaciones Java necesarias para este proyecto. Spring Boot [8] es un framework para crear de forma rápida y sencilla aplicaciones Java basadas en el famoso framework Spring. Spring y todos los componentes que forman el framework son de código abierto.

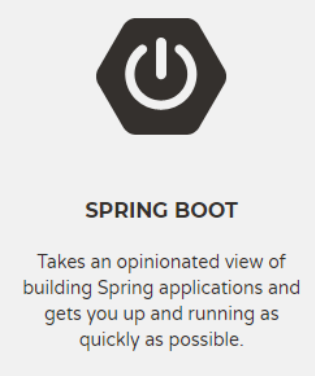

Figura 3 Spring Boot

2.1.4 Gestor de dependencias: Maven

En todos los desarrollos realizados para este proyecto se ha utilizado el gestor de dependencias Maven [9], que está perfectamente integrado con las aplicaciones Spring Boot y el entorno de desarrollo eclipse.

En la siguiente imagen se puede ver un ejemplo del pom.xml que se ha utilizado para gestionar las dependencias comunes entre todos los proyectos desarrollados. Ya que como todos los proyectos compartían el mismo modelo de datos se decidió sacar a un proyecto independiente (FogDatamodel) todo el modelo de datos común de la plataforma y después usar este proyecto como dependencia en el resto de proyectos.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" x
    xsi:schemaLocation="http://maven.apache.org/POM/
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.famar</groupId>
    <artifactId>FogPlatform</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>

    <modules>
        <module>FogDatamodel</module>
        <module>FogManager</module>
        <module>FogNodeManager</module>
        <module>FogNodeServiceRegistry</module>
        <module>FogNodeNetworkService</module>
    </modules>

</project>
```

Figura 4 Ejemplo de pom.xml en proyecto multimódulo

2.1.5 Base de datos: MongoDB

En las aplicaciones desarrolladas que requerían de un sistema de persistencia se ha utilizado la base de datos MongoDB [10]. No hay motivos especiales para su uso. Se podría decir que la razón principal para haberla elegido es la experiencia previa que tengo utilizando esta base de datos en aplicaciones Spring Boot, ya que gracias al proyecto Spring Data es muy fácil utilizar casi cualquier base de datos en un proyecto Spring

MongoDB es una base de datos documental escalable y flexible que almacena documentos en formato JSON. Es adecuada para clusterización ya que ha sido desarrollada para poder trabajar de forma distribuida. MongoDB forma parte de la nueva familia de sistemas de base de datos No SQL. Ha sido creada con una licencia de código abierto GNU.

2.1.6 Docker

Docker [11]es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux.

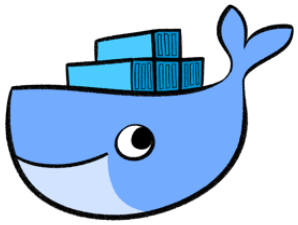

Figura 5 Docker

En este proyecto se ha usado concretamente Docker configurado en un modo de funcionamiento especial el cual ofrece ciertas funcionalidades muy útiles. El modo swarm [12]es un modo de funcionamiento que habilita funcionalidades para orquestación de servicios, entre otras. Estas funcionalidades de alto nivel para orquestar servicios permiten realizar despliegues con configuraciones de replicación y escalado, balanceo de carga, service discovery a través de un DNS dinámico,...

2.1.7 Spotify Docker Client

Para utilizar Docker y poder realizar acciones sobre el de forma programática era necesario utilizar un cliente para Docker. Entre varias alternativas de código abierto finalmente se escogió un cliente escrito en java por la empresa Spotify [13].

Figura 6 Repositorio GitHub de Spotify Docker Client

2.1.8 JGit

JGit [14] es un proyecto open-source de la fundación eclipse. Este proyecto ofrece un cliente escrito en Java para acceder a repositorios Git. De un modo similar al cliente de Docker en el proyecto esta librería es utilizada para acceder y descargar de forma programática el software de un repositorio.

3 Computación en Fog

En este capítulo haré una revisión de la breve historia del Fog Computing en conjunto con su estado actual y usos en el mercado. Seguidamente describiré que es el Fog Computing e introduciré los conceptos básicos que lo definen para así conocer más detalles sobre el Fog Computing y poder comprender el trabajo realizado que se describe en siguientes capítulos.

3.1 Historia del Fog Computing

El concepto de computación en Fog surgió por primera vez en 2012 con el artículo *"Fog Computing and Its Role in the Internet of Things"* publicado por trabajadores de Cisco Systems Inc [2] . Esta era la primera vez que se utilizaba el término *"fog computing"*. En dicho artículo se habla que la computación en Fog sería necesaria para dar solución a los requerimientos tecnológicos que el Internet de las Cosas demanda.

Tres años después, en noviembre de 2015, se creó el OpenFog Consortium [3] formado por grandes empresas de la industria de las TIC entre ellas Cisco, Intel y Microsoft entre otras. Este consorcio ha permitido acelerar el avance en el desarrollo e investigación de la computación en Fog. La principal responsabilidad del consorcio es definir una arquitectura de referencia con la que intentar sentar las bases del Fog Computing y crear una arquitectura estándar y abierta a todo el mundo, lo que es necesario para conseguir plataformas interoperables y escalables.

En febrero de 2016 se publicó el libro blanco con una versión inicial de la arquitectura de referencia. Finalmente un año después se publicó la primera versión oficial de *"OpenFog Reference Architecture"* [4]. En la arquitectura de referencia publicada se disemina la computación en Fog desde el punto de vista de las diferentes partes involucradas. Las partes involucradas van desde desarrolladores de software hasta fabricantes de hardware, pasando por proveedores de servicios hardware o administradores de sistemas. La arquitectura de referencia define ocho pilares básicos del Fog Computing que los utiliza como aspectos a cumplir por una plataforma Fog. Estos pilares son: seguridad, escalabilidad, apertura, autonomía, programabilidad, RAS (Fiabilidad, disponibilidad y facilidad de servicio), agilidad y jerarquía. Es importante destacar que la arquitectura de referencia propuesta por OpenFog Consortium es la documentación más importante que hay en la actualidad sobre los conceptos que engloban al Fog Computing y la referencia principal sobre la que se apoya el trabajo realizado.

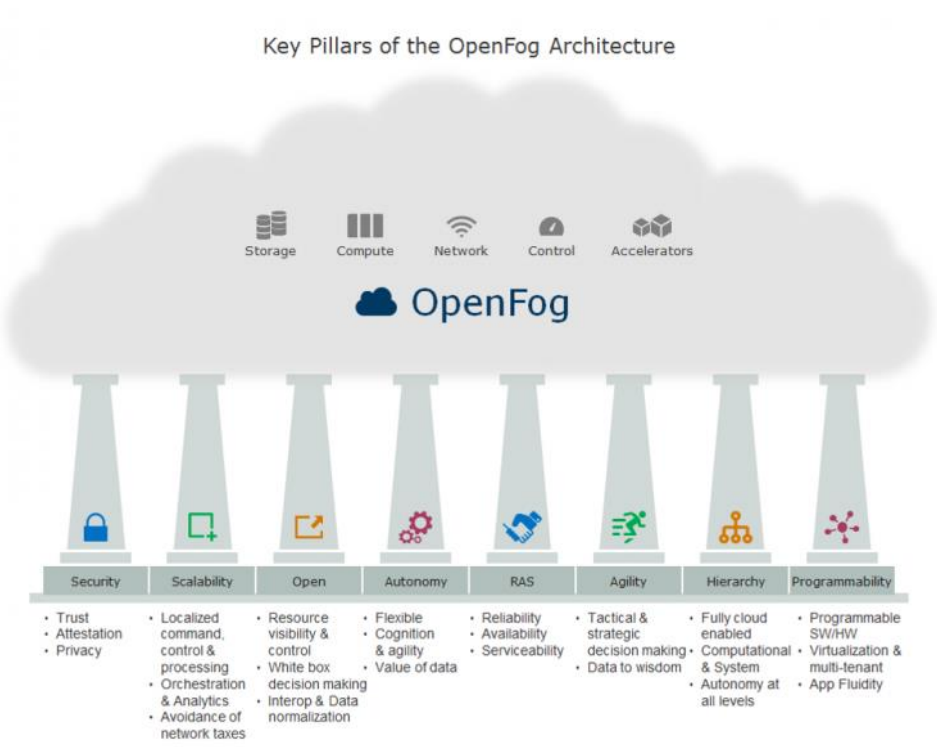

Figura 7 Pilares de la arquitectura de referencia de OpenFog

OpenFog Consortium es el principal referente del panorama de la computación en Fog aunque actualmente ya es posible encontrar empresas que utilizan el término *"Fog Computing"* para acuñar sus productos o servicios.

Quisiera destacar el caso de la empresa SONM [5] que se vende como una plataforma de computación Fog descentralizada. Esta empresa, partiendo del concepto de nodos de computación (Fog Node), que define la arquitectura de referencia ha creado una red de computación distribuida. Su modelo de negocio se basa en clientes que pagan por utilizar capacidad de cómputo y en proveedores que ganan dinero poniendo su hardware a disposición de la red. El producto creado por esta empresa está más orientado al uso en minería de criptomonedas, de hecho el hardware de los proveedores se utiliza para minería en los tiempos muertos en los que no se requiere de uso por parte de clientes. Esta empresa se ha unido al Fog Consortium en enero de 2018.

3.2 Conceptos básicos

El Fog Computing, o computación en Fog, es un nuevo modelo de computación que fue planteado originalmente por la empresa Cisco como una nueva arquitectura de desarrollo software que daba respuesta a las carencias que tiene el uso del Cloud Computing para soluciones IoT.

Sus máximos precursores lo definen como una arquitectura que ofrece recursos de almacenamiento y cómputo en el camino entre el Cloud y los dispositivos conectados. En otras palabras el Fog es una capa intermedia entre los dispositivos (sensores, actuadores, dispositivos conectados en general,...) y el Cloud que ofrece, al igual que ya lo hace el Cloud, capacidad de cómputo y almacenamiento pero aprovechando su proximidad a los dispositivos y permitiendo así ejecutar procesos sensibles a la latencia de red como por ejemplo procesos de sistemas críticos o de sistemas de tiempo real [15]. Otra forma de describirlo, aunque menos formal, es que el concepto de Fog promueve traer la mayoría de carga de procesos en los sistemas IoT al lugar más cercano a donde residen los dispositivos y donde, por lo tanto, se generan los datos.

Uno de los conceptos más importantes del Fog Computing es el de Fog Node. La computación en Fog está movida por una red de nodos Fog Node. Estos nodos tienen capacidad de procesamiento y almacenamiento. Los Fog Node pueden ser desde pequeños computadores embebidos hasta *datacenters* de tamaño medio ubicados en puntos estratégicos. En la arquitectura de computación en Fog estos nodos se estructuran en redes que pueden ser simplemente redes a nivel lógico o redes jerárquicas que aprovechan la distribución de la red para crear enlaces eficientes desde los dispositivos conectados hasta el Cloud [4].

En la figura 7 se muestra un ejemplo del concepto clave descrito anteriormente, la computación en Fog se produce en una capa intermedia entre los dispositivos finales y el Cloud.

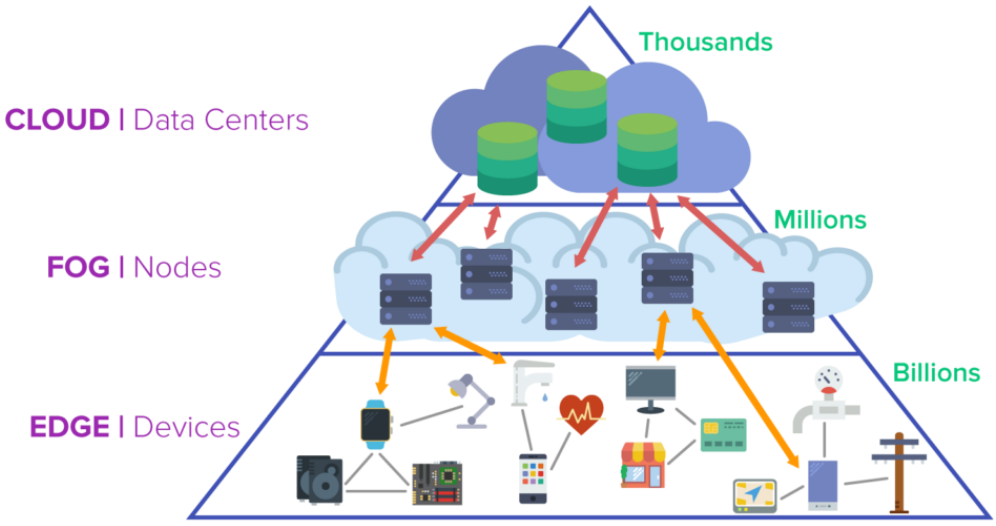

Figura 8 Ejemplo Fog Computing

De entre las diferentes ventajas que se atribuyen al Fog Computing me gustaría destacar:

- Este modelo de computación permite crear sistemas con baja latencia de red, lo que habilita sistemas críticos y de tiempo real que no son factibles si se usa únicamente el Cloud.
- Se hacen posibles sistemas basados en Big Data para aplicar a soluciones IoT. Estos sistemas harían análisis de datos en tiempo real utilizando los nodos Fog Node y aprovechando la baja latencia de las comunicaciones de red con ellos.
- La arquitectura de sistemas que promueve el Fog Computing es una arquitectura eficiente que reduce el ancho de banda y la carga de la red, lo que a la vez es una solución para posibles futuros problemas en ciudades inteligentes o en grandes sistemas IoT.

Por otro lado, respecto a desventajas, quisiera destacar la respuesta oficial de OpenFog Consortium a la pregunta de si esta nueva arquitectura es más compleja y costosa que el modelo basado en el Cloud. OpenFog Consortium reconoce que añadir esta nueva arquitectura a sistemas IoT incrementa la complejidad de los sistemas, pero argumenta que en ocasiones esa complejidad será necesaria simplemente porque a pesar de la complejidad añadida la arquitectura Fog permite que dichos sistemas funcionen de forma efectiva [15]. Un ejemplo son los ya comentados sistemas para análisis de datos en tiempo real, sistemas críticos,…

Un concepto muy interesante que se utiliza en este contexto es el Fog as a Service (FaaS). Esta idea la propone OpenFog Consortium por la misma razón por la que aboga por crear una arquitectura estándar para sistemas Fog que permita la interoperabilidad y escalabilidad. El Fog as a Service pretende ser un símil a lo que es Cloud as a Service.

Para conseguir hacer FaaS se identifica el reto de conseguir un control sobre la capacidad de cómputo y de la capacidad de almacenamiento similares a los que tenemos en entornos de computación en Cloud. Esto es un reto porque en este ámbito no existen herramientas o sistemas de gestión de entornos Fog, además que esto lo corrobora el trabajo realizado por OpenFog Consortium para intentar estandarizar el concepto de Fog Computing y facilitar la introducción al mercado de todas las tecnologías relacionadas que tienen que ser desarrolladas.

3.3 Problemas identificados

Este modelo de computación descrito en la arquitectura de referencia plantea varios retos para desarrollar las tecnologías y herramientas necesarias que en su conjunto harán posible la computación en Fog en sí y su uso en proyectos reales.

La arquitectura de referencia solo describe como debe funcionar un entorno Fog en aspectos generales y crea pautas a seguir para desarrollar sistemas que apliquen este modelo de computación.

Por otra parte actualmente no hay apenas soluciones de computación en Fog en el mercado, están comenzando a aparecer empresas que utilizan el término Fog Computing en sus productos. Pero respecto a soluciones que se hayan creado siguiendo el diseño de la arquitectura de referencia y estén orientadas al concepto de FaaS no hay nada en el mercado.

Para crear un servicio de FaaS primero es necesario disponer de una plataforma con la que tener el control sobre un entorno Fog y así ofrecer su uso como un servicio. Como ya se ha comentado previamente, esto es uno de los retos que deben investigarse para hacer posible la computación en Fog.

Por lo tanto se identifica el problema de crear una plataforma de computación Fog siguiendo los conceptos de la arquitectura de referencia y que permita ofrecer FaaS.

4 Diseño de la solución

En este capítulo se va a abordar la etapa del proyecto en la que se estudió y diseñó la solución con la que alcanzar los objetivos planteados en este trabajo.

Ya se han indicado los objetivos planteados para este proyecto pero se pueden englobar en uno solo, diseñar y crear una plataforma Fog para ofrecer FaaS. Pues se ha identificado, y los trabajos de OpenFog Consortium lo corroboran, que habrá una necesidad en el mercado de soluciones que ofrezcan Fog as a Service.

Partiendo del objetivo general de este proyecto y con los estudios realizados identifico que para poder ofrecer la computación en Fog como un servicio se necesita de un sistema, concretamente una plataforma, que permita administrar y controlar dicho entorno al igual que ya se hace en plataformas Cloud.

La realización para este proyecto ha seguido un proceso evolutivo que describiré brevemente a continuación.

Se comenzó este proyecto diseñando una plataforma Fog prototípica que fuera relativamente simple. Para esta plataforma Fog inicial se siguieron unos parámetros de diseño más laxos respecto a la especificación del Fog Computing que propone el OpenFog Consortium. Con el objetivo de hacer simple esta plataforma se optó por crear un entorno de ejecución de JavaScript inspirado en IFTT, más adelante se explicará que es IFTT y que principios sigue.

El objetivo de crear esta plataforma relativamente simple se alcanzó en un breve periodo de tiempo, tan breve fue que se planteó un nuevo objetivo para este proyecto. El nuevo objetivo era crear una plataforma Fog prototípica pero válida para un caso de uso real. Se decidió replantear el diseño e implementación que se habían hecho para crear una plataforma algo más compleja y robusta que siguiera más fielmente los diseños de la arquitectura de referencia propuesta por OpenFog Consortium.

Sin embargo, a pesar de estas dos versiones que se hicieron, ambos diseños comparten la mayoría de conceptos básicos que son parte del núcleo de la plataforma. Las diferencias entre las dos versiones se centran en como se ejecuta el software en los Fog Node, estas diferencias se explicarán más adelante junto con más detalles.

Respecto al FaaS, este se consigue diseñando una plataforma que abstraiga al usuario, en este caso administradores de sistemas, de los detalles del despliegue de software en entornos Fog. También es requisito indispensable que el software se ejecute de forma aislada y no tenga acceso ni pueda afectar ni a los recursos utilizados por el software de otros usuarios. De este modo dicha plataforma puede ser utilizada a la vez por varios usuarios, al igual que se hace en el Cloud, y se puede ofrecer recursos de computación en Fog como un servicio. Estos parámetros de diseño fueron aplicados en ambas versiones de la plataforma.

Dicho esto, a continuación, primero se describirá el diseño de los conceptos básicos y comunes que definen la plataforma Fog propuesta. Primero se dará una introducción general de los conceptos de la plataforma y después se describirá el diseño dando más detalles.

Seguidamente se describirán los detalles de diseño específicos de la primera versión junto con las razones para su elección. A continuación se repasaran las reflexiones sobre la primera versión que causaron la decisión del rediseño de la plataforma y la posterior implementación de la nueva versión. Y por último se describirán los cambios introducidos a la plataforma tras el rediseño. He de decir que los dos capítulos de diseño correspondientes a las dos versiones de la plataforma utilizan detalles técnicos para describir el diseño, pues lo que las diferencia principalmente es la implementación y las tecnologías utilizadas, lo que ha influido en ciertos aspectos en el diseño seguido.

4.1 Planteamiento general del diseño de la plataforma

En este apartado voy a hacer una revisión general de las piezas que componen la plataforma Fog para ir introduciendo el diseño realizado desde un punto de vista teórico más abstracto.

La plataforma Fog que se propone en este trabajo se forma por dos componentes los cuales han sido denominados Fog Node y Fog Manager. En la siguiente imagen se pueden distinguir estos dos componentes.

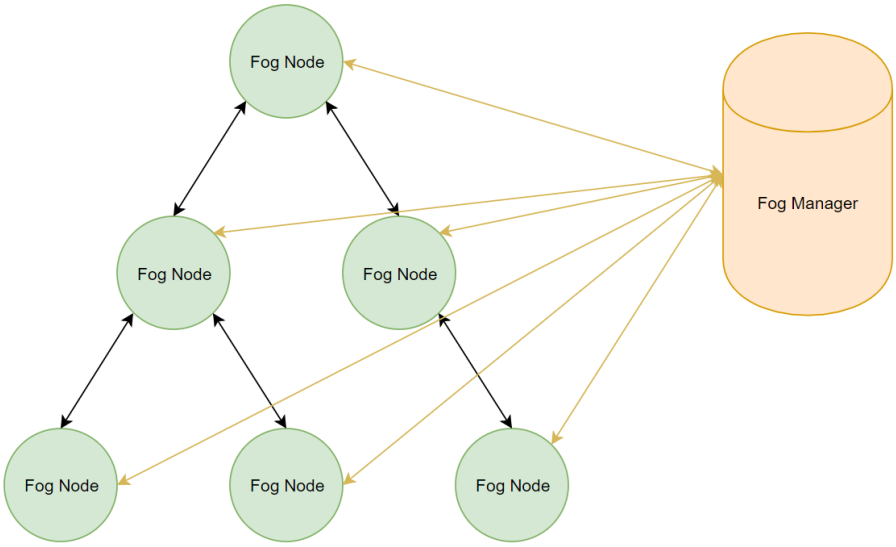

Figura 9 Ejemplo estructura plataforma Fog

En la imagen se puede ver una red con forma jerárquica compuesta por nodos Fog Node los cuales están todos conectados con el Fog Manager.

El Fog Node está basado en los mismos conceptos que se definen en la arquitectura de referencia. Es una unidad básica de computación y almacenamiento y en conjunto con más nodos Fog Node forman una red Fog, que es en esencia un entorno Fog.

Esta red de nodos no sirve de nada si no podemos ejecutar nuestro software en ella, por esta razón se añadió un rol más en este escenario de computación Fog, el Fog Manager.

El Fog Manager es el componente que tiene la responsabilidad de gestionar toda la plataforma. Este está conectado con todos los Fog Node para proveerles de software que ejecutar, para organizar la estructura de la red Fog y para administrar toda la plataforma en general.

Teóricamente en una estructura Fog así la comunicación con sensores y actuadores debe de hacerse solo a través de los nodos hoja del árbol que define la red, y la comunicación con servicios Cloud debe hacerse solo a través del nodo padre del árbol.

Respecto a las comunicaciones internas entre los Fog Node estas deberían hacerse solo entre nodos que estén directamente relacionados en la imagen mostrada previamente.

4.1.1 Red Fog

En los conceptos de redes Fog que se pueden encontrar no se especifica cómo se debe de formar exactamente una red Fog. En el concepto de red Fog que se utiliza en la plataforma propuesta una red Fog se entiende como una jerarquía de nodos, representado gráficamente se podría ver como un árbol. Para la correcta definición de la red de nodos son necesarios la ubicación de cada nodo, y la "capa" en la que se encuentra cada uno. Con estos datos es posible crear automáticamente la organización lógica y formar la jerarquía comentada anteriormente.

El concepto de capas se utiliza para indicar el nivel en el que están los nodos dentro del árbol, entiendo que la primera capa son los nodos hoja del árbol.

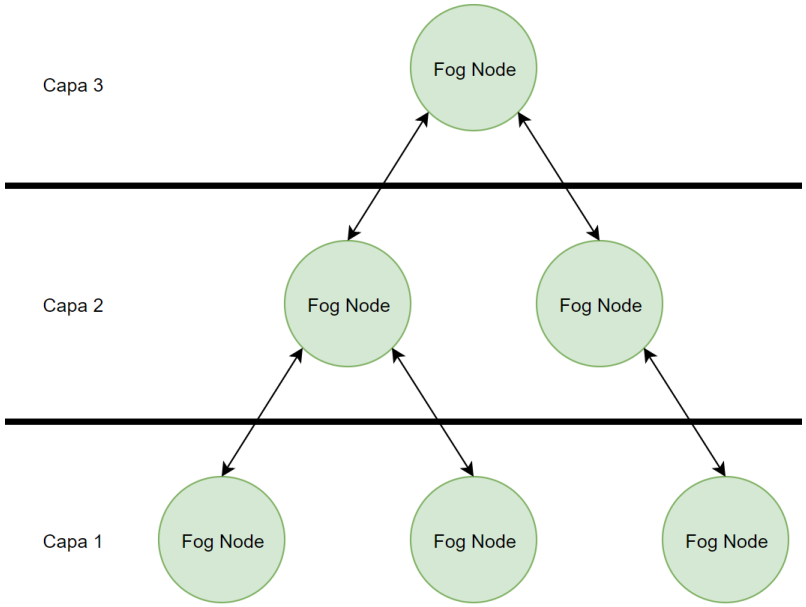

Figura 10 Ejemplo red Fog con tres capas

Las capas que tiene una red Fog dependen de la configuración que se realice en cada nodo de la red, es decir no hay un requerimiento específico que establece el número de capas ni la estructura que debe de tener. La red se forma a partir de los nodos que hay registrados en la plataforma. Por ejemplo, se podría conseguir una red de una única capa en la que todos los nodos están conectados entre sí configurando y registrando todos los nodos en la misma capa.

4.1.2 Fog Node

En la plataforma propuesta los Fog Node además de las características básicas de capacidad de cómputo y almacenamiento, tienen una serie de características que les permiten comunicarse con los nodos más cercanos de la red Fog.

En la plataforma presentada en este trabajo los Fog Node conocen todos los datos necesarios para comunicarse con los nodos más próximos dentro de su posición en la jerarquía, un nodo conoce la ubicación y la dirección ip de su nodo padre, de los nodos que están en su misma capa y de sus nodos descendientes. Esto último lo he acuñado como la visión de la red que tiene un Fog Node, ya que por concepto un Fog Node solo puede "ver" los nodos de la red más cercanos.

4.1.3 Fog Manager

El Fog Manager es el componente que tiene la capacidad de orquestar la red Fog en general, es decir, controlar el conjunto de nodos que hay en la red, definir la red y gestionar y distribuir el software que se ejecuta en los Fog Node.

El Fog Manager es un componente diseñado para gestionar la plataforma desde un punto centralizado, es una pieza muy importante de la plataforma Fog presentada, pues sus responsabilidades son imprescindibles para poder crear una red manejable y permitir que el conjunto de la plataforma en general funcione.

Como el Fog Manager es el componente que gestiona y distribuye el software por la red Fog se han definido una serie de conceptos sobre el software que se ejecuta en la plataforma. El software para esta plataforma Fog se organiza en proyectos y servicios. Estos conceptos, que serán explicados con más detalle en próximos capítulos, hacen posible que esta plataforma pueda ser utilizada por diferentes usuarios. Llegando así al nivel de ofrecer el acceso a la capacidad de cómputo de la plataforma Fog como un servicio.

4.2 Propuesta de diseño de la plataforma

En esta sección se va a describir un planteamiento de diseño más concreto sobre el diseño descrito de forma general en el capítulo anterior.

4.2.1 Fog Node

El diseño interno de un Fog Node está dividido en dos capas, la capa de gestión y la capa de servicio. Ambas capas son importantes, pero la capa de gestión tiene unas responsabilidades más claras, y por lo tanto se pudo definir mejor en la etapa de diseño.

La capa de gestión de un Fog Node es el software que realiza la interacción con el Fog Manager, como su nombre indica esta interacción es para la gestión del Fog Node en general. En esta capa se realizan tareas y acciones internas de la plataforma, como puede ser registrarse en la red Fog o consultar la visión de la red del nodo. Esta tarea de consulta de la red del nodo se hará al Fog Manager, ya que él se encargará de la gestión de la red, él será quien conozca la visión de la red de cada nodo.

Por otro lado la capa de servicio es un término que utilizo para abarcar todo lo referente a como se ejecuta el software dentro de un Fog Node, como he comentado previamente, un Fog Node tiene como meta principal ofrecer capacidad de cómputo. Esto no es más que ofrecer mecanismos para ejecutar software dentro del Fog Node, y como los mecanismos para ejecutar software es algo que puede variar según interpretaciones lo mantengo abstracto dentro de esta capa interna del Fog Node.

4.2.2 Fog Manager

El diseño del Fog Manager está muy acotado. Su responsabilidad es gestionar la red Fog y proporcionar un punto de acceso centralizado con el que utilizar la plataforma, desplegar software,...

Para tal propósito el Fog Manager mantiene un registro de todos los nodos activos que forman la red Fog. Para todo nodo registrado se comprueba periódicamente la disponibilidad del nodo para eliminar de la red los nodos que estén caídos o ya no estén disponibles.

Utilizando los datos del registro de nodos activos el Fog Manager forma automáticamente la estructura de la jerarquía de la red Fog, la cual puede ser consultada por los Fog Node, que solo tendrán acceso a los nodos colindantes dentro en la jerarquía de la red.

Como se ha indicado previamente se ha definido una organización del software por proyectos para permitir el acceso a varios usuarios a la plataforma y así, al igual que se hace en el Cloud, compartir los recursos de computación.

4.2.2.1 Proyectos y Servicios

Un proyecto es un conjunto de uno o más servicios. Un proyecto sirve como medio de organización para un conjunto de artefactos software relacionados con el mismo sistema o aplicación, los cuales son denominados servicios.

En la plataforma propuesta, un servicio es la unidad mínima de medida en cuanto a artefactos software. Para un servicio se define exactamente que software ejecuta y propiedades para identificarlo y definir su ejecución. En estas primeras versiones esta información será es el nombre del servicio, variables de entorno y en que capa de la red ha de desplegarse. Esta última propiedad es muy importante, con ella se pueden distribuir los distintos servicios de un proyecto por las distintas capas de la red y así aprovechar la organización de la red, siempre y cuando la casuística que se esté implementando se vea beneficiada de ello.

Con la definición de un proyecto y sus servicios el Fog Manager puede realizar el despliegue de dicho proyecto en la red. Este proceso se lleva a cabo mediante colaboración entre el Fog Manager y todos los Fog Node involucrados. En siguientes capítulos se explicará este proceso con más detalles.

4.2.3 Ciclo de vida del Fog Node

A modo aclaratorio y para comprender más cómo funciona la plataforma en general, voy a describir, añadiendo más detalles, cómo sería el ciclo de vida de un Fog Node en la plataforma Fog.

La primera fase en el ciclo de vida de un Fog Node es su inclusión en la red Fog. Es decir, cuando un Fog Node se inicia lo primero que hace es conectarse con el Fog Manager para registrarse e introducirse en la red Fog. Un Fog Node se registra en el Fog Manager con las coordenadas de su ubicación y la capa de la red Fog a la que pertenece. En las versiones iniciales de la plataforma solo se trabaja con coordenadas fijas, pero se contempla a futuro permitir que la ubicación de un Fog Node sea dinámica, y así habilitar la plataforma para entornos en movimientos como pueden ser los automóviles conectados.

En el registro del Fog Node el Fog Manager le asigna un identificador con el que posteriormente podrá identificarse ante el Fog Manager para consultar su visión de la red Fog.

El siguiente paso, una vez registrado un nodo, es el despliegue de software. Esto es todo un proceso que será descrito en el siguiente apartado.

Por último, cuando un Fog Node se va a apagar, antes de apagarse la capa de gestión indica al Fog Manager que elimine del registro el Fog Node que está a punto de dejar de estar disponible.

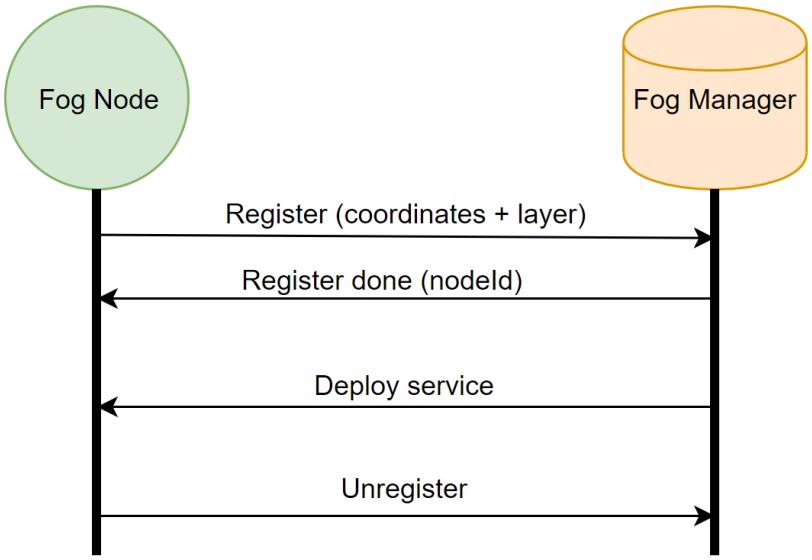

Figura 11 Ejemplo intercambio de mensajes en el ciclo de vida de un Fog Node

4.2.4 Despliegue de software en la plataforma Fog

El proceso de despliegue de un proyecto comienza en el Fog Manager que es donde está toda la información de los nodos de la red y la definición de los servicios que se tienen que desplegar. La tarea real de realizar la instalación y el despliegue del software se realiza en los Fog Node, concretamente en la capa interna de gestión de los Fog Node.

Por lo tanto el Fog Manager simplemente dada la definición de cada uno de los servicios que componen un proyecto envía la tarea de instalación y despliegue a todos los Fog Node que sean de la misma capa que hay especificada en la definición del servicio.

Quiero remarcar que se ha tomado la decisión de diseño de en las versiones desarrolladas para este trabajo solo permitir que la instalación de software sea un proceso que deba solicitarse manualmente por los usuarios de la plataforma. También se planteó que el software se instalara con un proceso proactivo que instala el software de los proyectos de forma automática cuando un nuevo nodo se une a la red. Pero se ha decidido dejar este planteamiento para trabajos futuros por no añadir más complejidad al proyecto.

En la capa de gestión que implementa un Fog Node se ejecutan las tareas de instalación y despliegue de software. En próximos apartados se explicará cómo funciona exactamente este despliegue, pues es algo íntimamente relacionado con la tecnología con la que se implementa.

4.3 Diseño de la plataforma con IFTT

Una de las primeras motivaciones que me surgieron para comenzar con este proyecto fue la de poder crear sistemas software que se ejecuten en entornos Fog y poder hacerlo de forma sencilla e intuitiva. Los entornos Fog son entornos complejos de por sí, como ya se ha comentado, son entornos de computación distribuida y esto en general es una dificultad añadida para crear todo tipo de sistemas software.

La computación en Fog es algo íntimamente relacionado con el IoT y para el diseño de esta plataforma Fog, a pesar de ser una versión simplificada, se siguieron unos parámetros de diseño orientados al uso en aplicaciones del IoT. En este sentido he intentado llevar el concepto de IoT a su mínima expresión, para mí en el IoT todo se reduce al envío de mensajes con información básica que el receptor o receptores pueden analizar y procesar. Esta visión del IoT es válida dentro de un entorno Fog, pues un modelo de paso de mensajes encaja en un entorno formado por un conjunto de nodos de computación distribuidos que se comunican entre sí y con el exterior.

Con todo esto me motivé e inspiré para crear una plataforma con la que crear sistemas software para el procesado de mensajes de un sistema IoT dentro de un entorno de computación Fog.

Para alcanzar el objetivo de poder crear sistemas de forma sencilla e intuitiva para entornos Fog se decidió inspirarse en el conocido proyecto IFTT.

IFTT [16] son las siglas de “If this, then that” que en castellano significa “Si ocurre esto, haz esto otro”. IFTT es el nombre de una plataforma de internet que surgió a finales de 2010. Con esta plataforma es posible automatizar diferentes tareas y acciones sobre dispositivos conectados a internet y sobre aplicaciones. Hay una gran variedad de aplicaciones y dispositivos compatibles con la plataforma, algunos de los más conocidos son Facebook, Spotify, Amazon Alexa, Hue,…. IFTT ofrece sus servicios a través de un sitio web y una aplicación móvil para IOS y Android. Ofrece una gran variedad de acciones automáticas predefinidas además de la posibilidad de construir tus propias acciones personalizadas, conocidas como recetas. La personalización en las recetas se puede hacer utilizando servicios web, lo que crea un sinfín de posibilidades para crear funcionalidades.

He de destacar respecto al diseño de la plataforma Fog planteada que no se buscaba en ningún momento crear una experiencia igual a la que ofrece la plataforma IFTT original. Como he indicado quería inspirarme en IFTT para la creación de sistemas software y como tal, siguiendo la filosofía de IFTT, se propuso una versión de la plataforma Fog en la que un sistema software se compusiera de “recetas” del estilo de IFTT. Concretamente un sistema software en la plataforma Fog sería lo que se ha descrito previamente como un “proyecto”, y las “recetas” IFTT del sistema serían los “servicios” que componen un proyecto de los que ya he hablado previamente.

Esa sería la correlación entre la plataforma Fog y un sistema implementado con filosofía IFTT. Pero como ya he indicado, esto no era una implementación exacta de

IFTT. Para entender mejor las diferencias describiré primero las características de una "receta" IFTT y seguidamente explicaré como es un servicio Fog con la filosofía IFTT.

Una "receta" IFTT está formada por una condición y una acción, se puede ver como la unidad mínima para ofrecer funcionalidad. En el IFTT original no hay que escribir código para crear las "recetas", las funcionalidades se crean mediante interfaces de usuario en las que se enlazan condiciones y acciones haciendo "drag and drop". La condición de una "receta" es un evento que se dispara automáticamente dentro de una plataforma IFTT y que evalúa si una "receta" ha de ejecutarse, si la condición se evalúa satisfactoriamente se procede a ejecutar la acción definida en la receta. En el IFTT original hay una serie de acciones predefinidas como por ejemplo: publicar en Facebook o Twitter, pero también se pueden crear acciones personalizadas con llamadas a servicios REST.

Por otro lado, un servicio en la plataforma Fog propuesta con filosofía IFTT está formado por un script escrito en JavaScript. A diferencia del IFTT original en la versión planteada de IFTT para entornos Fog solo hay un evento que dispara la ejecución de los servicios, este es la recepción de un nuevo mensaje en el Fog Node. Comparándolo con el IFTT original, el script que forma un servicio hace la tarea de la condición y la acción. Con este script se deja libertad al desarrollador para crear su lógica de negocio para analizar y procesar mensajes.

A modo recapitulativo. Esta versión de la plataforma Fog se basa en un modelo de paso de mensajes para soportar aplicaciones en IoT. Por lo tanto en esta versión se emplean los Fog Node de la red para el paso de mensajes. Al mismo se mantiene la plataforma lo más intuitiva posible simplificando el software que se distribuye por la plataforma, permitiendo únicamente scripts en JavaScript que se ejecutan cada vez que se recibe un mensaje.

Este paso de mensajes se hace en los scripts, que implementan la lógica de negocio oportuna para el procesado de cada mensaje. Un ejemplo de script podría ser, un script que comprueba que el mensaje recibido es una lectura de contaminación, y en caso afirmativo enviara una alarma a un su Fog Node padre si el nivel de contaminación supera ciertos límites preestablecidos.

Con el fin de poder crear estos scripts de forma más simple se ofrece un api con funcionalidades básicas para poder interactuar con la red Fog.

La interacción que se puede hacer desde estos scripts con la red Fog es un punto muy importante, pues el aspecto más potente de esta versión de IFTT es la creación de sistemas software mediante composición de pequeños scripts con lógica de negocio que se ejecutan de manera distribuida. Este es un punto muy importante porque permite beneficiarse de la computación distribuida y aprovechar la jerarquía que tiene la red Fog para hacer una distribución física y lógica de los servicios que componen un proyecto.

Se quiere ofrecer un api que principalmente permita el envío de mensajes estándar entre nodos de la red Fog. Sin entrar en detalles específicos, la definición de este api es la indicada a continuación.

La api definida está compuesta por tres funciones, todas como parámetro de entrada el mensaje a enviar y no retornan ningún valor.

forward(message)	Esta función permitirá enviar el mensaje pasado como parámetro al nodo superior, es decir al nodo padre, del nodo actual.
share(message)	Esta función enviará el mensaje pasado como parámetro a todos los nodos que estén al mismo nivel que el nodo actual.
delegate(message)	Esta función permitirá enviar el mensaje pasado como parámetro a todos los nodos que cuelguen en la jerarquía de la red Fog del nodo actual.

4.4 Rediseño de la plataforma

Se alcanzó el objetivo de crear una plataforma de computación Fog inspirada en IFTT pero aun así seguía interesado en ver qué podía conseguir con esta plataforma. Fue en este punto cuando siendo crítico fui realmente consciente de que una plataforma como la creada no tiene apenas utilidad en un entorno real. Esto es así por diferentes motivos:

- Los scripts se ejecutan embebidos en el propio software del Fog Node, esto no tiene un buen rendimiento, y en entornos con mucha carga no sería eficiente y el propio Fog Node podría quedar fuera de servicio.
- Aunque se creó un api para usar en los scripts no se dispone de la misma versatilidad que se dispone si se programa en cualquier otro lenguaje de programación que es ejecutado de una forma nativa.
- Sería muy difícil convencer a un tercero de escribir todo el software de sus sistemas en un lenguaje de scripting con un api que solo se puede usar en una determinada plataforma.

Por motivos como los recién mencionados se siguió investigando en este ámbito de la computación en Fog. La arquitectura de referencia propuesta por el Fog Consortium fue de mucha ayuda y sirvió de gran inspiración para la siguiente iteración que se produjo en el diseño y desarrollo de la plataforma.

Referente al software para entornos Fog, la arquitectura de referencia dice lo siguiente:

“A robust fog deployment requires that the relationship between a fog node, fog platform, and fog software are seamless” [4]

De esta frase se puede entender que los distintos componentes que forman el sistema Fog no pueden estar fuertemente acoplados entre sí.

Esta idea ha sido muy importante para comprender como sería la forma más adecuada de crear una plataforma Fog, porque crear un entorno novedoso como es este es complicado.

Si se crea una plataforma con un entorno de ejecución que solo puede ejecutar software en un lenguaje específico va a ser muy difícil que esta plataforma se haga popular. Visto desde la perspectiva del desarrollo software no se haría popular porque los desarrolladores no podrán trabajar con los lenguajes y tecnologías que prefieran o porque no sería posible utilizar la tecnología más adecuada para la tarea que debe realizar el software. Por otro lado desde el punto de vista empresarial no se haría popular porque generaría desconfianza e incertidumbre que el software que se desarrolla solo se pueda ejecutar en una determinada plataforma que depende de un tercero.

Dicho con otras palabras, la arquitectura de referencia fomenta la creación de un entorno en el que haya libertad para utilizar cualquier lenguaje de programación, al igual que sucede en el Cloud, y que no esté fuertemente acoplado con la plataforma en la que se va a ejecutar.

Esto es así por varias razones, por ejemplo, poder añadir software existente a un proyecto con facilidad o para poder usar software creado para la plataforma en otro entorno. Con esta premisa sería posible disponer de software que se puede ejecutar indistintamente en una plataforma Fog o Cloud.

Este nivel de acoplamiento tan bajo entre la plataforma, el hardware y el software, en definitiva, esta libertad para el uso de las tecnologías más adecuadas para cada tarea se consigue con la virtualización. La ejecución virtualizada del software en el hardware es lo único que permite ejecutar muchas tecnologías diferentes al mismo tiempo sobre el mismo hardware. Básicamente lo que la arquitectura de referencia intenta decir es esto, los Fog Node tiene que ofrecer su capacidad de cómputo a través de tecnologías de virtualización y así al igual que sucede en el Cloud se habilita el uso de casi cualquier tecnología software.

Partiendo de todo lo dicho se tomó la decisión de trabajar en una plataforma Fog que utilizara tecnologías de virtualización. Concretamente se utilizó Docker y se aprovecharon las ventajas de los modernos sistemas de gestión de contenedores para la ejecución del software, es decir para la ejecución de los servicios que componen los proyectos que se pueden definir en la plataforma. La introducción del uso de sistemas de gestión de contenedores a la plataforma permitía muchas nuevas oportunidades para el diseño de la plataforma y le daba más utilidad en un entorno real.

4.5 Diseño de la plataforma con contenedores

El cambio al uso de contenedores Docker conllevaba remodelar la plataforma para cambiar la forma en que se ejecuta el software en los Fog Node. Esto, lógicamente, repercute al diseño e implementación del Fog Node. Como se ha comentado previamente el diseño de un Fog Node se divide en dos capas, la capa de gestión y la capa de servicio, y la capa de servicio fue totalmente rediseñada e implementada de nuevo.

Los cambios básicamente consistían en eliminar todo rastro de filosofía IFTT y ofrecer la capacidad de cómputo a través un sistema de gestión de contenedores que necesariamente debe convivir en el Fog Node. Es decir, en esta nueva versión, los servicios pueden estar escritos en cualquier lenguaje de programación mientras se puedan ejecutar dentro de contenedores Docker.

Una de las posibilidades que surgieron del uso de contenedores era poder ejecutar en contenedores, además de los servicios que forman los proyectos, ciertos servicios que proporcionan herramientas de gestión y control para uso interno del Fog Node.

Por esta razón se definieron dos tipos de servicios que se ejecutan dentro de la capa de servicio de un Fog Node, los servicios de aplicación y los servicios de sistema. Esta decisión al igual que el uso de contenedores fue motivada por la arquitectura de referencia, que también define dos tipos de servicios que se ejecutan dentro de un Fog Node.

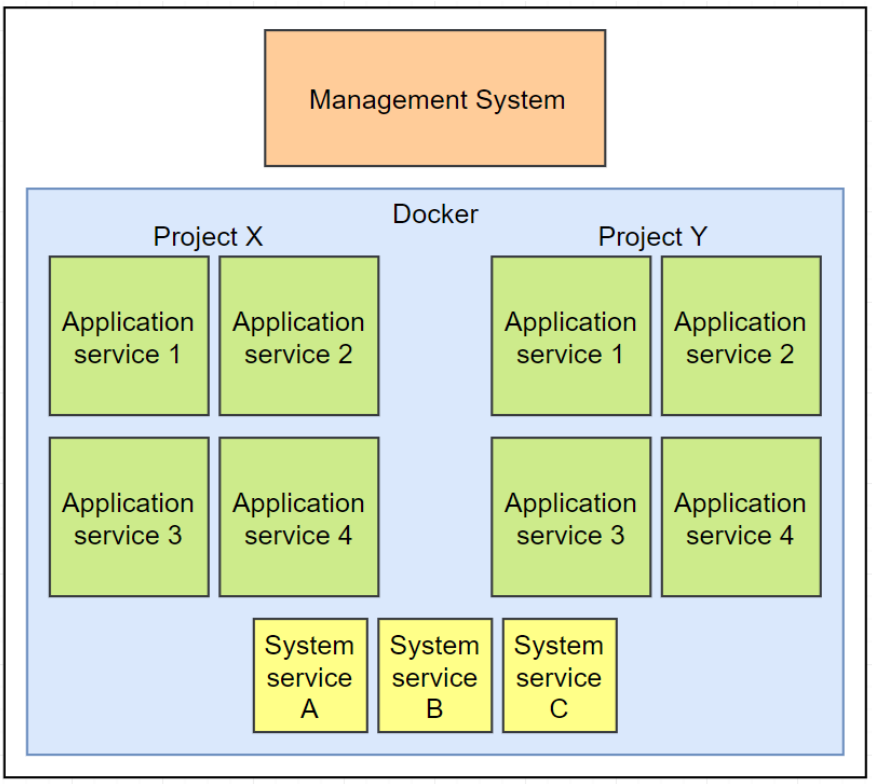

Figura 12 Ejemplo estructura interna del Fog Node basado en contenedores Docker

4.5.1.1 Servicios de aplicación

Son los servicios con lógica de negocio específica que han de ser implementados para cada proyecto y con funcionalidad determinada. Son el equivalente a los servicios que se implementaban en scripts en la primera versión de la plataforma. Al igual que en la primera versión, a estos servicios se les ofrece un api para poder interactuar con la red Fog, pero en este caso ese api se ofrece con los servicios de sistema.

4.5.1.2 Servicios de sistema

Los servicios de sistema son servicios necesarios en cada Fog Node para proporcionar la arquitectura y estructura necesaria para que el Fog Node pueda hacer sus funciones. Al ejecutarse los servicios de aplicación dentro de contenedores Docker se necesitan ciertos componentes para gestionar los servicios de aplicación y para proporcionar ciertos servicios de utilidad a los servicios de aplicación. Estos servicios de utilidad de los que hablo son el equivalente al api disponible en los scripts que había en la primera versión de la plataforma. Otra de las utilidades que ofrecen los servicios de sistema es de hacer de interfaz entre la capa de gestión y la capa de servicio.

Se diseñaron una serie de servicios de sistema básicos para cumplir unos requisitos mínimos con los que poder poner la plataforma Fog a funcionar. Estos servicios son:

- **Service registry**, un servicio con el mismo concepto de descubrimiento de servicios que se usa en entornos microservicios pero con una implementación específica que almacene y de acceso de forma aislada a la información de los servicios en ejecución organizada por proyectos, requisito indispensable para aislar entre sí el software de distintos proyectos. Al igual que en los sistemas de descubrimientos de servicios comunes deberá implementar una comprobación del estado de los servicios en ejecución. Este servicio será de utilidad tanto para la plataforma, para poder saber desde el exterior el estado de los servicios de un Fog Node, como para los propios servicios de aplicación, que podrán usar este servicio mediante un api REST para consultar los servicios disponibles en el Fog Node que están a su alcance(servicios de su propio proyecto).
- **Network service**. Este servicio ofrecerá un api REST para consultar las direcciones de red de los nodos más cercanos. Es decir, este servicio permitirá consultar la “visión de la red” del propio Fog Node. Este servicio podría usarse como un servicio interno y construir servicios de sistema sobre él. En dicho caso se añadiría una capa más de abstracción a través de un api que los servicios de aplicación utilizarían para interactuar con la red Fog, pero para no añadir más complejidad al proyecto solo se utilizará el Network service y será consultado directamente por los servicios de aplicación. Es una buena idea la de crear una serie de servicios de sistema que dependan del Network service para crear una capa de abstracción y que desde los servicios de aplicación sea más cómodo y seguro la interacción con el resto de la red Fog, sin embargo lo dejaré para futuras versiones de la plataforma.

Al igual que los servicios de abstracción para interactuar con la red Fog hay más servicios que aunque sea interesante su desarrollo no se han tenido en cuenta para este proyecto y se dejarán para trabajos futuros.

Uno de esos servicios de sistema, el cual considero de los más interesantes y útiles, es un Api Gateway específico para este caso de uso en el que se implementaran funciones de autenticación y limitación de acceso (máximo de peticiones superadas), auditoría, logging,... un servicio como este sería de utilidad para crear una capa entre el Fog Node y el exterior que proporcione seguridad al Fog Node y a la plataforma Fog en general.

Por otro lado en el rediseño de la plataforma tenemos el Fog Manager. El Fog Manager no se vio muy afectado por el cambio, pues se había creado usando conceptos generalizados de proyectos y servicios y no fueron necesarios cambios en el diseño.

Lo único destacable es la gestión de los servicios de sistema. Los servicios de sistema son software que se ejecuta en contenedores Docker en los Fog Node. El despliegue de los servicios de sistema se desencadena en el Fog Manager. Cuando un nuevo Fog Node se registra en la red el Fog Manager da la orden de despliegue de los servicios de sistema en dicho Fog Node.

5 Implementación

El proceso de implementación, al igual que el de diseño fue evolutivo. Primero se realizó la implementación del primer diseño, y tras acabar dicha implementación se rediseñaron e implementaron de nuevo ciertas partes pensando en utilizar otras tecnologías más modernas y con una visión más cercana a un escenario real.

En este capítulo se darán detalles sobre la implementación que se ha realizado de los diferentes componentes de la plataforma Fog utilizando las tecnologías comentadas en el capítulo dos de este texto.

5.1 Implementación del Fog Node

El software que compone un Fog Node es muy dependiente de la tecnología en la que se basa la capa de servicio. Se podría decir que todo se articula en torno a la tecnología escogida para la ejecución de los servicios, en este caso un motor de JavaScript o un sistema de virtualización basado en contenedores.

Conceptualmente un Fog Node está dividido en dos capas ya mencionadas, la capa de gestión y la capa de servicio. A pesar de esta separación de conceptos en la etapa de diseño, en el momento de la implementación ambas capas son dependientes de la tecnología utilizada para la ejecución de los servicios.

La capa de servicio está directamente relacionada con la tecnología empleada para la ejecución de los servicios, es en esta capa donde se interactúa directamente usando las interfaces disponibles para el uso y gestión de dicha tecnología, por ejemplo, en el caso de la tecnología de contenedores Docker la capa de servicio interactúa directamente con Docker utilizando un cliente open-source.

Por otro lado la capa de gestión también se ve influenciada por la tecnología con la que se ejecutan los servicios. Los cambios de tecnología pueden obligar a hacer cambios en el api que ofrece la capa de servicio, y la capa de gestión utiliza dicha api para las tareas de gestión del software que se ejecuta en el nodo.

Por lo tanto se puede decir que la implementación de la capa de gestión es algo que está acoplado al api que ofrezca la capa de servicio y a su vez, el api y la implementación de la capa de servicio es algo que depende de la tecnología utilizada para los servicios, para su instalación, despliegue y ejecución (según aplique en la tecnología utilizada).

Los detalles de implementación del Fog Node se pueden diferenciar en dos versiones, ya que como se ha indicado se han desarrollado dos versiones de la misma plataforma, la versión con IFTT y la versión con contenedores Docker.

Antes de continuar, es importante mencionar que no se ha hecho hincapié en aspectos de seguridad en este proyecto, por ejemplo no se han implementado mecanismo de seguridad para proteger las distintas api que ofrecen los distintos componentes de la

plataforma. Se ha tomado esta decisión por el carácter académico del trabajo y para no abordar demasiados temas diferentes en el proyecto.

5.1.1 Versión con IFTT

La versión con IFTT del Fog Node está formada por una sola aplicación escrita en Java y con el framework SpringBoot. En esta aplicación se implementan las funciones de la capa de gestión y de la capa de servicio. Esto es así ya que en esta versión los servicios, que son scripts JavaScript, se ejecutan dentro de la misma JVM que ejecuta la aplicación SpringBoot.

La siguiente imagen es una representación gráfica de los componentes que forman la versión con IFTT del Fog Node

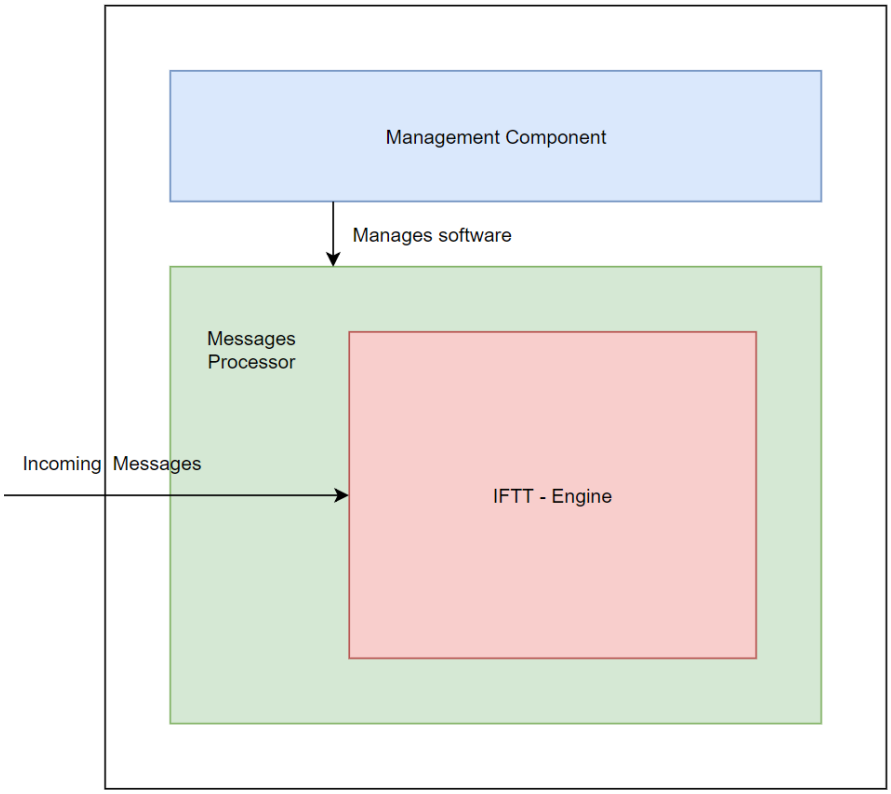

Figura 13 Componentes internos Fog Node versión IFTT

En la figura 13 se pueden ver los componentes principales que forman la versión con IFTT del Fog Node. Estos componentes son IFTT-Engine, Messages Processor y Management Component.

El componente IFTT-Engine en la práctica es principalmente un motor para JavaScript que se ejecuta embebido en la aplicación Java del Fog Node. Concretamente se ha usado el motor para JavaScript Nashorn [17], incluido con el lenguaje de programación Java a partir del JDK 8.

Este componente es de lo más importante de esta versión pues de él depende el objetivo principal de un Fog Node, ofrecer capacidad de cómputo.

Los motores JavaScript de este estilo permiten crear el contexto con el que se va a ejecutar un determinado script. Utilizando esto último para cada ejecución de un servicio se inicializa un contexto para esa ejecución. En ese contexto se carga el mensaje que se ha recibido, y que es la causa de la ejecución del script, y también se cargan las propiedades que se pueden configurar en el servicio en variables de contexto.

Se implementó un api para enviar mensajes con una estructura estándar a otros Fog Node y a otros servicios del mismo proyecto. Este api se inyecta en el contexto para poder utilizar sus funciones desde los scripts. Por la naturaleza de la tecnología utilizada, en esta versión los servicios no se despliegan, simplemente se guarda en memoria su script y sus propiedades asociadas y este se ejecuta con un determinado contexto cuando se recibe un nuevo mensaje en el nodo.

Los despliegues de servicios en esta versión son muy simples. En la solicitud de despliegue que hace el Fog Manager este envía el script y todos sus metadatos asociados tras esto el Fog Node almacena toda esa información en una colección en memoria, ese sería todo el proceso de despliegue.

Por otra parte respecto a la ejecución del servicio, cuando se recibe un mensaje en el nodo este entra por un api REST que implementa el componente Messages Processor este componente simplemente recorre toda la lista de scripts almacenados en la mencionada colección en memoria y los ejecuta con el componente IFTT-Engine.

En la siguiente imagen se muestra el código más destacable del componente IFTT-Engine. Se puede ver como se obtiene una instancia del motor nashorn y se cargan tres variables en el contexto de ejecución del motor para finalmente ejecutar el código del script usando el método eval de la interfaz ScriptEngine.

```
@Override
public ScriptExecutionResult executeScript(ScriptExecutionSpec spec) {
    ScriptExecutionResult result = new ScriptExecutionResult();
    result.setResult(Result.OK);

    try {
        ScriptEngineManager factory = new ScriptEngineManager();
        ScriptEngine engine = factory.getEngineByName("nashorn");
        setEngineContext(spec, engine);
        engine.eval(new String(spec.getScript()));
    } catch (Throwable e) {
        LOG.error("", e);
        result.setResult(Result.ERROR);
        result.setError_message(e.getMessage());
    }

    return result;
}

private void setEngineContext(ScriptExecutionSpec spec, ScriptEngine engine) throws JSONException {
    engine.put("message", spec.getMessage());
    engine.put("properties", spec.getProperties());
    engine.put("api", nodeMessagingApi);
}
```

Figura 14 Código ejecución script en "nashorn" con contexto configurado

5.1.2 Versión con contenedores Docker

En esta versión el Fog Node también está compuesto por una aplicación en Java con el framework SpringBoot, con el añadido de que esta aplicación depende de que haya un Docker host instalado y en ejecución en la máquina local para poder cumplir todas las responsabilidades del Fog Node.

Dicho Docker host es la tecnología que se usa en esta versión para la instalación, despliegue y ejecución de los servicios. La aplicación desarrollada en Java, en adelante FogNodeManager, se encarga de utilizar Docker con un cliente y realiza de forma automática, entre otras cosas, la administración y configuraciones oportunas sobre Docker, los despliegues de los servicios en Docker,...

De un modo similar a la versión anterior Docker es la parte más importante de este sistema y en la aplicación en Java la implementación de la capa de gestión y el api de la capa de servicio depende completamente de la tecnología en la que está basada la capa de servicio.

En este caso de uso el FogNodeManager administra Docker y realiza las configuraciones de red necesarias para ejecutar todos los servicios en un entorno aislado de los servicios que pertenezcan a proyectos diferentes. Esto es necesario para garantizar un uso seguro del Fog Node al ofrecer el acceso a la plataforma Fog como un servicio.

A continuación detallaré como utiliza Docker el FogNodeManager.

En el momento de inicialización del Fog Node se configura Docker en modo swarm para poder utilizar posteriormente las funcionalidades de Docker Swarm. Una de las funcionalidades que ofrece Docker para aislar los servicios entre sí es el uso de redes, esta técnica es muy usada por el FogNodeManager.

El FogNodeManager crea una red en Docker para cada proyecto del que se despliega algún servicio en el nodo. Con esto, usando la red de Docker, los servicios desplegados en un mismo nodo que sean del mismo proyecto se podrán acceder entre ellos directamente. Además el acceso local entre servicios es muy sencillo usando el sistema de DNS dinámico que ofrece Docker Swarm. Con dicho DNS un servicio está accesible en la red de Docker por el nombre con el que ha sido creado. Para evitar colisiones en los nombres de los servicios de un nodo, los nombres de los servicios utilizan como prefijo el identificador del proyecto al que pertenecen.

```
PS C:\Users\Fabian> docker service ls
ID                  NAME                                                MODE                REPLI
nqnfsbadde0j        5b0afde37de25b4408a10d41-pollution-controller       replicated          1/1
kweyarvl20r7        5b0da97c55a9811e5c0cf5d2-reports-service            replicated          1/1
eokbw6xax5zg        5b0da97c55a9811e5c0cf5d2-reports-service-database   replicated          1/1
l19yeugob8ct        network-service                                     replicated          1/1
qogv1gy2is1t        service-registry                                    replicated          1/1
```

Figura 15 Ejemplo servicios Docker en un Fog Node

Además de la configuración de red ya descrita se hace otra configuración más, la cual es muy necesaria. Los servicios de sistema son únicos en cada Fog Node y son usados por todos los servicios de aplicación desplegados en un Fog Node. Para poder ser accesibles desde los servicios de aplicación los servicios de sistema deben estar en la misma red que ellos, para ello se añaden dinámicamente los servicios de sistema a todas las redes de proyectos que se crean en el Fog Node, ya que por es posible configurar varias redes en un servicio de Docker Swarm y además también es posible hacerlo dinámicamente. Para los casos de comunicación entre servicios de sistema también se crea una red de sistema en la que solo se incluyen los servicios de sistema.

Respecto al despliegue de los servicios, se hace en dos etapas, primero se crea o se actualiza una imagen en Docker a partir del código fuente del servicio de aplicación que se va a instalar y seguidamente se crea o se actualiza un servicio de Docker Swarm con la imagen preparada anteriormente.

```
lic void deployDockerService(ServiceDeployParameters parameters) {
 try {
     String imageName = imagesRepository.getNewVersionImageName(param
     String imageId = imagesRepository.buildImage(imageName, paramete
     Service service = docker.findServiceByName(parameters.getService
     if(service==null) {
         createService(imageName, parameters);
         LOG.info("Service saved, imageId:"+imageId);
     }else {
         updateService(service, imageName, parameters);
         LOG.info("Service updated, imageId:"+imageId);
     }
 } catch (DockerException | InterruptedException | IOException | GitA
     LOG.error("",e);
     throw new ServiceDeployException(e);
 }
```

Figura 16 Código despliegue servicio en Docker

Respecto a la implementación de los servicios de sistema todos son pequeñas aplicaciones en Java utilizando el framework SpringBoot. Lo más destacable es que el NetworkService tiene una implementación muy simple. Lo único que hace es consultar la "visión de la red Fog" del nodo al Fog Manager llamando a una función del api REST del Fog Manager, puesto que como se contará a continuación esta funcionalidad está centralizada en el Fog Manager.

```
public String buildImage(String imageName, SoftwareRepositoryReference rep
    String localDirectory = getImage(repositoryReference);
    try {
        return docker.getClient().build(
                Paths.get(localDirectory),
                BuildParam.name(imageName));
    }finally {
        if(StringUtils.startsWithIgnoreCase(repositoryReference.getReposit
            cleanUp(localDirectory);
        }
    }
}

private String getImage(SoftwareRepositoryReference reference) throws IOE
    if(StringUtils.startsWithIgnoreCase(reference.getRepository(),"http"))
        return gitClient.getRepository(reference);
    }else {
        return reference.getRepository();
    }
}
```

Figura 17 Código construcción de imagen Docker a partir de repositorio Git

Respecto al código fuente de los servicios de aplicación. Este se obtiene de un repositorio Git en el que es requisito que haya un fichero Dockerfile para la creación de la imagen con la que se desplegará el servicio. Como ya se ha mencionado, no se ha presado atención en aspectos de seguridad en este proyecto. En el caso que nos ocupa no se ha implementado nada específico para la gestión de las credenciales con las que se accede al repositorio Git, se ha optado por pasar en variables de entorno el usuario y la contraseña con los que se autentica contra el repositorio Git, lógicamente esto limita a utilizar solo los repositorios Git válidos con esas credenciales. He de decir que esto es una solución temporal para no dedicar demasiado tiempo al problema de las credenciales para autenticarse contra el repositorio Git, por supuesto queda para trabajos futuros hacer un rediseño e implementación de los aspectos de seguridad de la plataforma.

5.2 Implementación del Fog Manager

El Fog Manager es el componente de la plataforma que se encarga de la gestión de la red de nodos y la configuración de los diferentes componentes de la plataforma, en las primeras versiones estos componentes solo son proyectos y servicios.

Respecto a la implementación del Fog Manager se podría pensar que dada la naturaleza distribuida de un entorno Fog sería lógico que los procesos de gestión de dicho entorno, como algunos de los implementados en el Fog Manager, se ejecutaran también de forma distribuida dentro de la red Fog. Dicho planteamiento es perfectamente válido y muy interesante para conseguir una plataforma Fog completamente distribuida y al mismo tiempo muy robusta. Esta idea para la implementación de los procesos de gestión de la plataforma, a pesar de ser muy interesante, se descartó rápidamente por la complejidad que conlleva y se dejó para trabajos futuros.

Para no hacer demasiado complejo el diseño del software y la implementación y poder conseguir en un tiempo de desarrollo relativamente corto un producto funcional con el que crear la plataforma diseñada se optó por implementar todos los procesos de configuración y gestión del entorno Fog en una aplicación centralizada.

A pesar de todo, la implementación centralizada del Fog Manager aporta interesantes beneficios a la plataforma. Este componente centralizado de la plataforma es perfecto para ser desplegado como un servicio Cloud, lo que aporta las ventajas de escalabilidad y disponibilidad de servicio típicas de los servicios Cloud. Otra ventaja es que ha sido la opción para desarrollar el Fog Manager menos compleja.

Sin embargo hay que ser consciente de que este tipo de implementación puede dar problemas a medio/largo plazo según el tamaño de la red que gestione.

La aplicación del Fog Manager al igual que el resto de aplicaciones en este proyecto está escrito en Java usando el framework SpringBoot.

Esta aplicación necesita de un medio de persistencia para las configuraciones que almacena, se ha utilizado una base de datos MongoDB. No hay motivos especiales para haber elegido MongoDB, es muy sencillo utilizarlo en una aplicación con SpringBoot. Lo más destacable respecto al uso de MongoDB es el uso de “geospatial query” para definir la jerarquía de la red Fog automáticamente.

5.2.1 Red Fog

La gestión de la red Fog es una de las responsabilidades más importantes del Fog Manager y a su vez uno de los conceptos más importantes de la plataforma en sí.

El concepto de red Fog es simplemente un término con el que acuño a un conjunto de nodos Fog Node que están organizados con una distribución lógica en forma de jerarquía, como un árbol.

A continuación describiré la implementación de la red Fog con la que se estructura la plataforma desarrollada.

Para crear una red de nodos primero es necesario disponer de un registro con todos los nodos con sus metadatos asociados que van a formar la red. Este registro se ha almacenado en una colección de MongoDB y mantenido por el Fog Manager, es en este registro donde se guardan y eliminan los datos de los Fog Node cuando se registran y se des-registran con el Fog Manager.

Como ya se ha mencionado, dos de los metadatos que definen un Fog Node son la capa de la red a la que pertenecen y las coordenadas del punto en el que se "encuentran". Respecto a las coordenadas de los Fog Node en las versiones desarrolladas solo se contemplan Fog Node con coordenadas fijas, el desarrollo de mecanismos para cambiar las coordenadas dinámicamente se ha dejado para trabajos futuros.

La implementación de esta red se ha hecho con un algoritmo relativamente simple. Se ha utilizado la información de las coordenadas de los nodos almacenadas en MongoDB en conjunto con la funcionalidad de MongoDB "geospatial query" para definir la red con un coste computacional bajo.

En la siguiente imagen se puede ver un fragmento del código que utiliza la "geospatial query" de MongoDB usando la forma *Circle* y con un radio de búsqueda de 1 kilometro.

```
List<DTOFogNode> nodes = nodeRepository.findByLocationWithin(
        new Circle(current.getLocation(), new Distance(1D, Metrics.KILOMETERS)));
FogNetworkView view = new FogNetworkView();
for(DTOFogNode node : nodes) {
    if(view.getParent()==null && IotLayers.isUpperLayer(current.getLayer(), node.g
        view.setParent(toNetworkDetails(node));
    }else if(StringUtils.equals(current.getLayer(), node.getLayer()) && !StringUti
        view.getBrothers().add(toNetworkDetails(node));
    }else if(IotLayers.isBelowLayer(current.getLayer(), node.getLayer())) {
        view.getDescendents().add(toNetworkDetails(node));
    }
```

Figura 18 Muestra código "geospatial query" y algoritmo red Fog

La idea fundamental de dicho algoritmo es que realmente nunca se calcula la definición de toda la red con un proceso, es decir, todos los datos que definen el conjunto de toda la red son los que están en la colección de MongoDB pero el sistema desarrollado no recorre en un solo proceso todo el conjunto de estos datos. La clave está en la forma en la que un nodo consulta la definición de la red. Es aquí donde sale a colación el concepto descrito en la fase de diseño de la "visión de la red" que tienen los Fog Node.

Conceptualmente ningún componente necesita saber la estructura completa de la red, un nodo solo necesita saber de los nodos que están en la misma capa que él, del nodo del que cuelga y de los nodos que cuelgan de él. Estos conceptos son los que se han seguido para la implementación.

Con todo lo dicho, cuando un nodo consulta su "visión de la red" el proceso busca en la colección de MongoDB utilizando "geospatial query" todos los nodos dentro de un círculo cuyo centro son las coordenadas del nodo que está consultando su "visión". La query utilizando un círculo retorna los registros ordenados por cercanía al centro del mismo. Finalmente se recorre dicha lista y se guardan en una estructura de datos los nodos de interés, los cuales ya han sido mencionados, para el nodo que hace la consulta y se retorna dicha estructura de datos. En el caso de no encontrar un nodo padre dentro del círculo de búsqueda se hace posteriormente una búsqueda más concreta de los nodos de la capa superior al nodo que hace consulta y, en caso de encontrarse alguno, se utiliza el más cercano como padre del nodo que hace la consulta.

6 Caso de Estudio: ecoMobiliy

Uno de los objetivos de este proyecto era realizar un desarrollo aplicando los principios del Fog Computing en el proyecto ecoMobility con el fin de introducir los principios de este modelo de computación a ciertas partes de dicho proyecto desarrollado en la UPV y así aprovechar algunas de las ventajas que este modelo de computación ofrece.

En este capítulo se va a mostrar la solución propuesta para reemplazar el actual sistema de gestión de los controladores de tráfico por un sistema que cumpla esas funciones con una arquitectura distribuida aplicando los principios del Fog Computing. Concretamente se va a describir una solución desarrollada utilizando la plataforma Fog desarrollada en este trabajo.

6.1 Descripción del problema

De forma muy resumida se podría decir que el proyecto ecoMobility es un proyecto de investigación sobre cómo se podría gestionar la movilidad en una Smart City. Una de las diferentes funciones que están desarrollados para este proyecto relacionado con el entorno IoT es el control automatizado del tráfico en función de la contaminación de la ciudad. Esta función está implementada con un sistema centralizado que controla y gestiona una red de controladores de luces de tráfico en función de eventos que recibe con lecturas de contaminación en diferentes puntos de la ciudad. Este sistema puede controlar una gran área de espacio desde un solo punto centralizado.

6.2 Solución propuesta

Se ha desarrollado un sistema que ofrece una funcionalidad equivalente al sistema automático de control de las luces de tráfico mencionado anteriormente. Este sistema ha sido diseñado como un sistema distribuido pensando en la eficiencia de las comunicaciones de red y en la eficiencia en cuanto a la carga computacional, es decir se ha intentado repartir la carga computacional por todo el sistema que al ser distribuido se encuentra repartido por toda el área sobre la que el sistema actúa.

Como el desarrollo realizado se hizo para la plataforma creada se creó un proyecto en la plataforma en el que se crearon tres servicios de aplicación diseñados para formar este sistema.

El sistema diseñado se ha estructurado con tres capas de la jerarquía de la red Fog. En la siguiente imagen se muestra la solución propuesta representada sobre una Smart City y desplegada en una determinada estructura Fog.

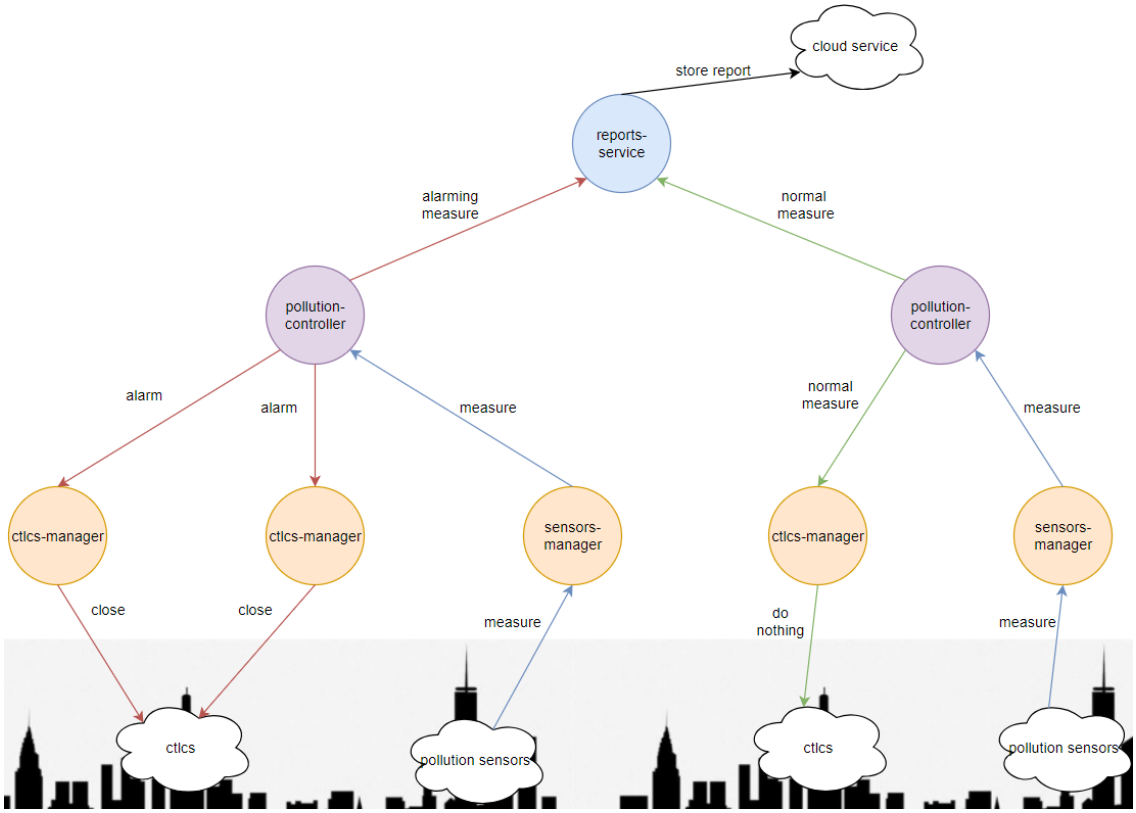

Figura 19 Diagrama con ejemplo de despliegue de la solución propuesta

En los nodos de la capa inferior se desplegará el servicio de gestión de controladores de tráfico conectados (ctlcs-manager). Este servicio se encarga de controlar el estado de los controladores de tráfico que estén registrados, contaré con que se registrarán los controladores de tráfico más cercanos en un perímetro alrededor de cada nodo donde se encuentre el ctlcs-manager. Con este servicio, que estará replicado por todos los nodos de la primera capa, se consigue repartir el control el control sobre todos los controladores de tráfico. Con este sistema distribuido se consigue una estructura con un reparto de responsabilidades más eficiente, partiendo de que el registro de los controladores de tráfico debe hacerse al Fog Node de primera capa más cercano.

En la segunda capa de la red Fog se desplegará el servicio de control de contaminación, este es un servicio que responde a eventos recibidos con medidas de contaminación, tomadas por sensores repartidos por la ciudad, generando alarmas en el caso de que la medida supere ciertos niveles parametrizados. Este servicio genera alarmas que se envían a otros servicios para que el sistema haga sus funciones. Cuando se produce una lectura superior al nivel máximo permitido se envía una alerta para cerrar el tráfico en la zona circundante a donde ese ha producido la lectura a los ctlcs-manager que están en los nodos hijos del nodo en el que se haya recibido la lectura. Del mismo modo cuando se produce una lectura con niveles bajos se comunica a los ctlcs-manager que comprueben si deben abrir de nuevo el tráfico. Por otro lado cuando se produce una lectura dentro de unos niveles intermedios se envía el dato a un tercer servicio que genera un histórico y un informe de la evolución de los niveles de contaminación. No se restringe la forma en que este servicio pueda recibir estos eventos, sería posible hacerlo con colas de mensajería, por REST,... en el caso de esta implementación se hará con un api REST por simplicidad.

Por último en la tercera capa se tiene el servicio de reportes. Este servicio se ha añadido para dar un ejemplo de aplicación del concepto de gestión de los datos asociado a los entornos Fog. Al pretender tener las capacidades de un entorno Cloud centralizado en una plataforma Fog distribuida hay que cambiar la manera con la que se almacenan los datos de las aplicaciones. En escenarios en los que no es viable almacenar todos los datos en el Cloud, por el coste del envío a través de la red, sale a relucir la filosofía del Fog para la gestión de los datos. En entornos Fog los datos deben "vivir" cerca del lugar donde se generan, en el Fog Node que los procesa o en un Fog Node cercano que acumula datos de otros Fog Node. Cuando la cantidad de datos almacenada en un Fog Node es lo suficientemente grande, esta es resumida o comprimida para ser enviada a un sistema de almacenamiento tradicional en el Cloud. Lo que hace el servicio de reportes es justamente eso, recibe todas las alarmas que genera el servicio de control de contaminación y crea un resumen con los datos de las alarmas recibidas, cada día este servicio vuelva toda la información que tiene almacenada a una base de datos en el Cloud. La ventaja de este sistema de persistencia a largo plazo usado en entornos distribuidos es que el movimiento de los datos se hace con procesos batch y que los datos finales almacenados en el sistema Cloud ocupan menos espacio que los datos originales, porque son un resumen de ellos.

6.2.1 Detalles de implementación

A continuación describiré las partes destacables de la implementación de los tres servicios de aplicación implementados para el proyecto de esta demo.

Los tres servicios implementados son aplicaciones SpringBoot.

6.2.1.1 Servicio de control de la contaminación

El servicio de control de la contaminación implementa una función que se expone a través de un api REST. La lógica que se implementa en esta llamada es la descrita sobre la generación de alarmas que se envían a todos los ctlcs-manager que hay en los Fog Node descendientes al Fog Node en el que se haya recibido la medida. Todas estas alarmas también se envían al servicio de reportes para que este almacene y sintetice todos los datos de medidas de contaminación recibidos en el sistema.

Es importante mencionar que también tendría sentido invocar la lógica implementada en este servicio a través de colas de mensajería. Esto sería de utilidad para entornos reales que tengan que soportar grandes cargas de trabajo.

6.2.1.2 Servicio de gestión de controladores de tráfico

Lo más destacable del servicio de gestión de controladores de tráfico es el algoritmo que implementa para controlar el estado de los controladores de tráfico que están bajo su control. Cuando se recibe un mensaje con una medida de contaminación en este servicio se indica si la medida es normal o es una alarma, y en ambos casos se hacen cambios en el estado de los controladores de tráfico más cercanos a la ubicación indicada en el mensaje para que el tráfico se controle con normalidad, mensaje con medida normal, o para que se corte el tráfico en el caso de recibir una alarma.

La coherencia en la gestión del tráfico respecto al sentido de marcha en que se corta el tráfico es una tarea que no corresponde a este servicio. Se va a usar la misma red mesh de controladores de tráfico que ya se viene usando en el proyecto ecoMobility, y dicha red ya se encarga automáticamente de la coherencia del tráfico, ya que los controladores de tráfico están interconectados y entre ellos mantienen el estado de las luces semafóricas de forma coherente. He de puntualizar que este aspecto de la coherencia en el estado de los controladores de tráfico es algo que queda fuera del alcance de la demo desarrollada y por lo tanto no ha sido probado con la demo desarrollada.

6.2.1.3 Servicio de reportes

Este servicio se ha diseñado simplemente para ejemplificar los conceptos de funcionamiento de un sistema Fog en el que se trabaja con grandes volúmenes de datos. La implementación hecha es muy simple y no hace realmente lo que se ha

indicado en el diseño, simplemente almacena los datos con las medidas recibidas y ofrece un api REST para consultar dichos datos.

Como he dicho este servicio se ha diseñado para mostrar cómo sería el reparto de responsabilidades para gestionar grandes volúmenes de datos en sistemas para entornos Fog. En un entorno Fog hay que centrarse en minimizar las conexiones de red que sean costosas o ineficientes y transportar grandes cantidades de datos constantemente hasta un servicio Cloud, que sería el lugar más normal donde realizar tareas de Big Data, es ineficiente. Una mejor solución podría ser utilizar el Fog como un paso intermedio de esos datos hacia un sistema final en el Cloud. En el Fog se podrían ejecutar procesos de análisis de datos, en los nodos que ofrezcan capacidad de cómputo suficiente, o se podrían almacenar dichos datos temporalmente para luego ser enviados en un formato reducido, comprimidos o resumidos, a un sistema final en el Cloud. Estas son simplemente algunas de las opciones para tratar grandes volúmenes de datos que posibilita y fomenta el novedoso modelo de computación en Fog.

6.2.2 Puesta en marcha

Para la puesta en marcha de esta demostración se han usado cinco máquinas virtuales de los servicios de virtualización del DSIC con las que se ha simulado el funcionamiento de la plataforma creada en una determinada estructura Fog. Con estas máquinas se ha podido comprobar, entre otras cosas, el trabajo necesario para implantar la plataforma Fog creada en una sencilla red de tres nodos Fog Node.

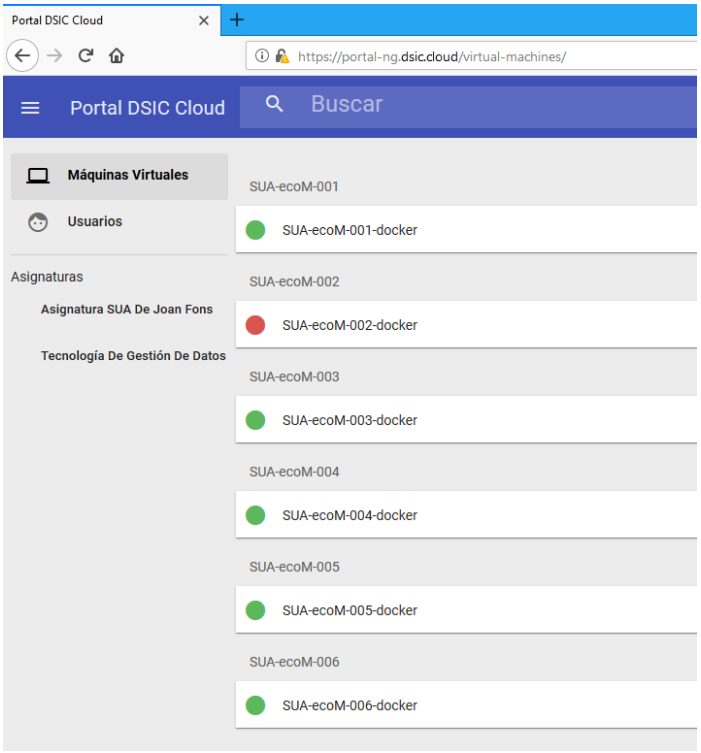

Figura 20 Muestra máquinas virtuales en la web de gestión de las máquinas virtuales del DSIC

De las cinco máquinas usadas una fue destinada para alojar el Fog Manager. Como ya se ha indicado el Fog Manager requiere de una base de datos MongoDB para funcionar. Para hacer más rápida la instalación y despliegue se desplegó la base de datos y el software del Fog Manager en dos contenedores Docker en la máquina virtual indicada.

Por otro lado se han utilizado tres máquinas virtuales para ejecutar los tres Fog Node que componen la red de esta prueba. Un Fog Node se compone del software FogNodeManager que se ejecuta localmente como un archivo jar en la máquina donde se quiere tener un Fog Node y de un Docker instalado también en la máquina local. En cada una de estas tres máquinas se ha parametrizado el FogNodeManager para que apunte al Fog Manager desplegado previamente en la primera máquina virtual y también se han configurado con coordenadas diferentes y ubicado en capas diferentes de la red. Un nodo se ha ubicado en la capa 1, otro en la capa 2 y el restante en la capa 3, con esta configuración se forma una red como la de la siguiente imagen.

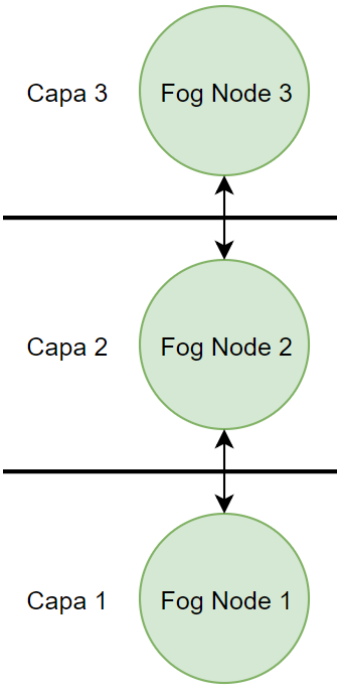

Figura 21 Representación red Fog utilizada en la prueba de la solución propuesta

Por último la quinta máquina virtual utilizada se ha usado para ejecutar dentro de contenedores Docker el software de varios controladores de tráfico con los que finalmente se ha podido probar el funcionamiento general de la plataforma y de la demo desarrollada.

Para comenzar la prueba lo primero necesario fue desplegar los diferentes componentes de la plataforma.

En las siguientes imágenes se documenta el proceso de despliegue de los Fog Node.

En la siguiente imagen se puede ver el comando que laza la aplicación SpringBoot FogNodeManager, el software que implementa la capa de gestión y controla la capa de servicio basada en Docker.

```
[sua@SUA-ecoM-006-docker fognodemanagerrelease]$ sudo java -Dnode-properties-file="/var/famar/fognodemanagerrelease/layer2/node.properties" -jar ./FogNodeManager-0.0.2-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.1.RELEASE)

2018-07-02 22:59:14.662  INFO 4430 --- [           main] c.f.iot.node.FogNodeManagerApplication   : Starting FogNodeManagerApplication v0.0.2-SNAPSHOT on SUA-ecoM-006-docker with PI
0.0.2-SNAPSHOT.jar started by root in /var/famar/fognodemanagerrelease)
2018-07-02 22:59:14.666  INFO 4430 --- [           main] c.f.iot.node.FogNodeManagerApplication   : No active profile set, falling back to default profiles: default
2018-07-02 22:59:14.923  INFO 4430 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebA
59:14 CEST 2018]; root of context hierarchy
2018-07-02 22:59:18.518  INFO 4430 --- [           main] f.a.AutowiredAnnotationBeanPostProcessor : JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
2018-07-02 22:59:18.641  INFO 4430 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.boot.autoconfigure.validation.ValidationAutoConfigurati
e.validation.ValidationAutoConfiguration] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2018-07-02 22:59:18.937  INFO 4430 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'validator' of type [class org.springframework.validation.beanvalidation.Loc
essed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2018-07-02 22:59:19.962  INFO 4430 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 8080 (http)
2018-07-02 22:59:19.998  INFO 4430 --- [           main] o.apache.catalina.core.StandardService   : Starting service Tomcat
2018-07-02 22:59:20.010  INFO 4430 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.11
2018-07-02 22:59:20.275  INFO 4430 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-07-02 22:59:20.275  INFO 4430 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 5373 ms
2018-07-02 22:59:20.681  INFO 4430 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Mapping servlet: 'dispatcherServlet' to [/]
2018-07-02 22:59:20.698  INFO 4430 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'errorPageFilter' to: [/*]
2018-07-02 22:59:20.699  INFO 4430 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-07-02 22:59:20.699  INFO 4430 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-07-02 22:59:20.699  INFO 4430 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-07-02 22:59:20.699  INFO 4430 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-07-02 22:59:23.315  INFO 4430 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.context.embedded.Annotati
date [Mon Jul 02 22:59:14 CEST 2018]; root of context hierarchy
2018-07-02 22:59:23.534  INFO 4430 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/management/service/v2/{projectId}/{serviceName}],methods=[POST]}" onto
```

Figura 22 Muestra FogNodeManager arrancando

Cuando termina el arranque del nodo podemos comprobar en el Fog Manager que el nodo se ha registrado correctamente y que el Fog Manager le ha asignado un identificador.

Para ello hacemos una petición al api REST del Fog Manager para ver los nodos registrados. En la siguiente imagen se puede ver dicha petición y su respuesta.

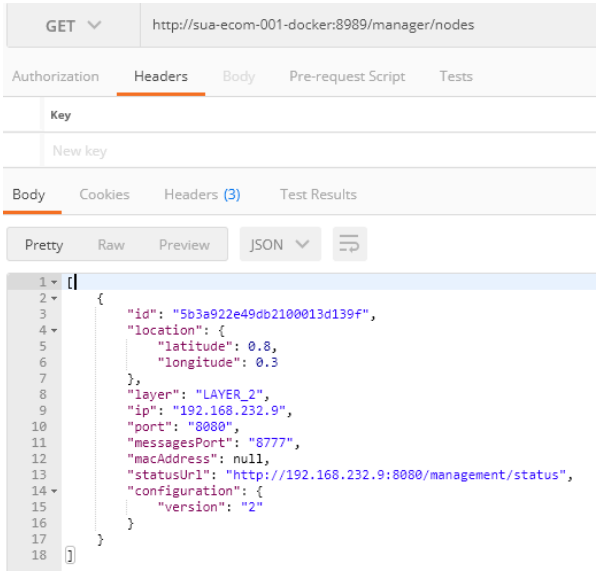

Figura 23 Ejemplo consulta de nodos registrados

Se continúa el despliegue y se hace lo mismo con los otros dos Fog Node. Una vez desplegados todos se comprueba que están todos registrados en el Fog Manager. La petición con el estado final de los nodos registrados se puede ver en la siguiente imagen.

GET http://sua-ecom-001-docker:8989/manager/nodes

Pretty Raw Preview JSON


```
[
    {
        "id": "5b3a922e49db2100013d139f",
        "location": {
            "latitude": 0.8,
            "longitude": 0.3
        },
        "layer": "LAYER_2",
        "ip": "192.168.232.9",
        "port": "8080",
        "messagesPort": "8777",
        "macAddress": null,
        "statusUrl": "http://192.168.232.9:8080/management/status",
        "configuration": {
            "version": "2"
        }
    },
    {
        "id": "5b3a931549db2100013d13a0",
        "location": {
            "latitude": 0.8,
            "longitude": 0.3
        },
        "layer": "LAYER_1",
        "ip": "192.168.232.8",
        "port": "8080",
        "messagesPort": "8777",
        "macAddress": null,
        "statusUrl": "http://192.168.232.8:8080/management/status",
        "configuration": {
            "version": "2"
        }
    },
    {
        "id": "5b3a935d49db2100013d13a1",
        "location": {
            "latitude": 0.8,
            "longitude": 0.3
        },
        "layer": "LAYER_3",
        "ip": "192.168.232.6",
        "port": "8080",
        "messagesPort": "8777",
        "macAddress": null,
        "statusUrl": "http://192.168.232.6:8080/management/status",
        "configuration": {
            "version": "2"
        }
    }
]
```


Figura 24 Consulta de nodos registrados en la prueba de la solución propuesta

Respecto a las propiedades que se pueden ver en la imagen de los nodos quiero destacar el “statusUrl” que es usado por el Fog Manager para comprobar periódicamente si los Fog Node siguen funcionando correctamente.

Una vez desplegados el Fog Manager y todos los nodos de la red ya se puede decir que la plataforma está en funcionamiento. Para continuar con las pruebas hay que configurar en la plataforma el software que se quiere desplegar, concretamente hay que configurar el proyecto con los tres servicios descritos en apartados anteriores.

En la siguiente imagen se muestra la petición al Fog Manager que se realiza para guardar la configuración de un proyecto. En el cuerpo de la petición se pasa el objeto en formato JSON que define el proyecto completo, con los servicios, parámetros y configuraciones necesarios para su correcto despliegue.

POST http://sua-ecom-001-docker:8989/manager/projects/

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "projectName":"ecomobility",
    "services":[
        {
            "serviceName":"pollution-controller",
            "serviceLayer":"LAYER_2",
            "servicePort":8383,
            "properties":{
                "FOG_ENV":"PROD",
                "MAX_POLLUTION_LEVEL":"14",
                "WARNING_THRESHOLD":"5",
                "MQTT_URI":"tcp://sua-ecom-004-docker:1883"
            },
            "portBindings":{
                "8383":8383
            },
            "repositoryReference":{
                "repository":"https://famargon@bitbucket.org/famarsystems/pollution-controller.git"
            }
        },
        {
            "serviceName":"ctlcs-manager",
            "serviceLayer":"LAYER_1",
            "servicePort":8282,
            "portBindings":{
                "8282":8282
            },
            "repositoryReference":{
                "repository":"https://famargon@bitbucket.org/famarsystems/ctlcs-manager.git"
            }
        },
        {
            "serviceName":"reports-service",
            "serviceLayer":"LAYER_3",
            "servicePort":8484,
            "portBindings":{
                "8484":8484
            },
            "repositoryReference":{
                "repository":"https://famargon@bitbucket.org/famarsystems/reports-service.git"
            }
        }
    ]
```

Figura 25 Muestra petición para guardar configuración del proyecto en la prueba de la solución propuesta

A destacar de esta petición. Con el objeto “properties” se pueden indicar variables de entorno que se pasarán al servicio cuando se despliegue. Con la propiedad “serviceLayer” se indica en que capa de la red se debe desplegar el servicio.

Esta llamada responde con el identificador de la entidad guardada con la configuración, para poder hacer después la petición de despliegue de un proyecto en la red e indicar el proyecto que se quiere desplegar. En la siguiente imagen se puede ver un ejemplo de respuesta donde se puede ver el projectId del proyecto recién guardado.

```
{
    "projectId": "5b3a90f949db2100013d139e",
    "projectName": "ecomobility",
    "services": [
        {
            "serviceName": "pollution-controller",
            "serviceLayer": "LAYER_2",
            "servicePort": 8383,
            "properties": {
                "FOG_ENV": "PROD",
                "MAX_POLLUTION_LEVEL": "14",
                "WARNING_THRESHOLD": "5",
                "MQTT_URI": "tcp://sua-ecom-004-docker:1883"
            },
            "portBindings": {
                "8383": 8383
            },
            "repositoryReference": {
                "repository": "https://famargon@bitbucket.org/famarsystems/pollution-controller.git",
                "username": null,
                "password": null
            }
        },
        {
            "serviceName": "ctlcs-manager",
            "serviceLayer": "LAYER_1",
            "servicePort": 8282,
            "properties": null,
            "portBindings": {
                "8282": 8282
            },
            "repositoryReference": {
                "repository": "https://famargon@bitbucket.org/famarsystems/ctlcs-manager.git",
                "username": null,
                "password": null
            }
        },
        {
            "serviceName": "reports-service",
            "serviceLayer": "LAYER_3",
            "servicePort": 8484,
            "properties": null,
            "portBindings": {
                "8484": 8484
            },
            "repositoryReference": {
                "repository": "https://famargon@bitbucket.org/famarsystems/reports-service.git",
                "username": null,
                "password": null
            }
        }
    ]
}
```


Figura 26 Respuesta a la petición de guardado de la configuración del proyecto

En la imagen siguiente se puede ver un ejemplo de una petición al Fog Manager para desplegar un proyecto en la red Fog. La respuesta que se obtiene da información sobre los servicios que se han desplegado y el nodo en el que se han desplegado cada uno de ellos.

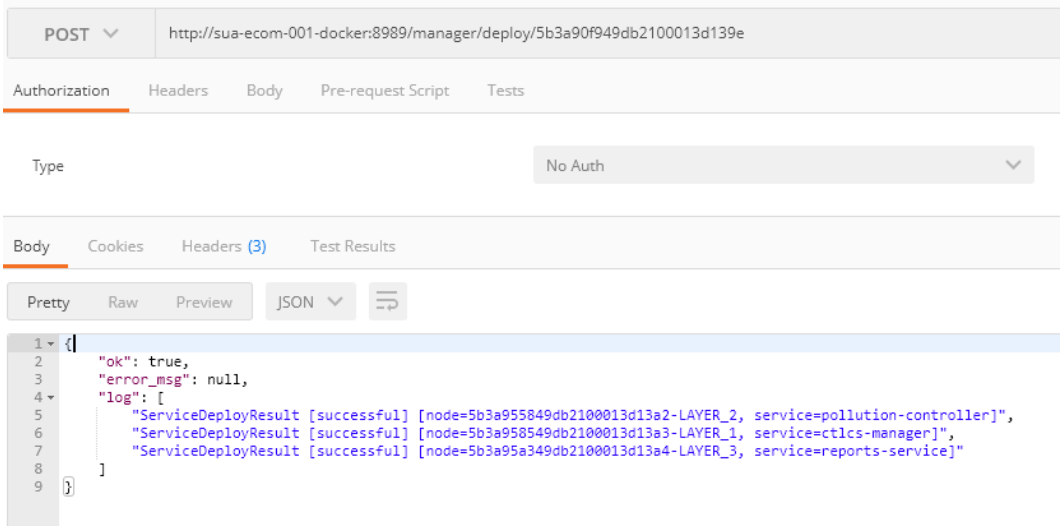

Figura 27 Muestra petición y respuesta para el despliegue del proyecto de la solución propuesta

Una vez desplegado el sistema faltaría configurar el ctlcs-manager registrando un controlador de tráfico para poder probar el flujo completo del sistema. Con la petición que se muestra en la siguiente imagen se registra un controlador de tráfico en el ctlcs-manager.

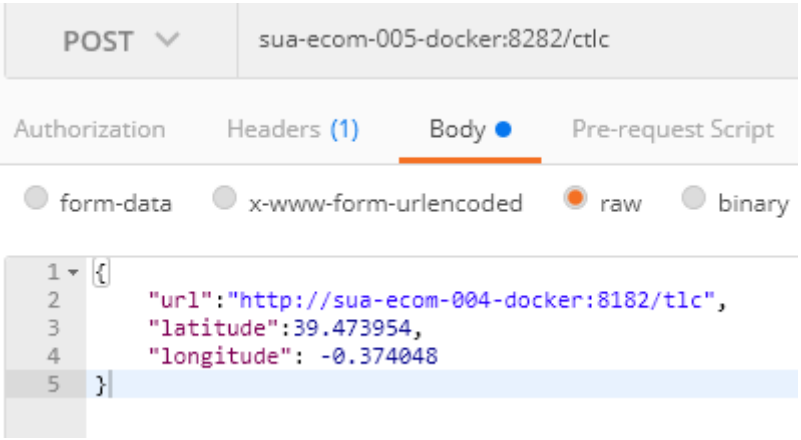

Figura 28 Muestra petición registro controlador de tráfico en un ctlcs-manager

Se obtiene un estado de respuesta http 200 ok como confirmación de que el controlador de tráfico se ha registrado correctamente.

Solo a modo de comprobación, observamos que el controlador de tráfico está funcionando forma normal.

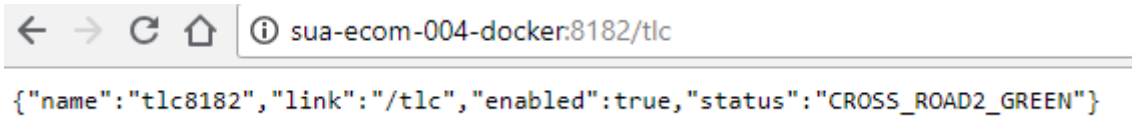

Figura 29 Prueba consulta estado controlador de tráfico, semáforo verde

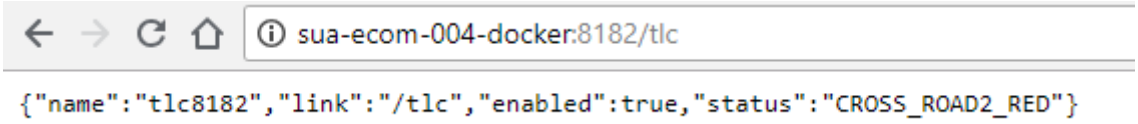

Figura 30 Prueba consulta estado controlador de tráfico, semáforo rojo

Para hacer funcionar el sistema hay que simular una lectura de contaminación y enviarla al pollution-controller para que la procese y se desencadenen las acciones oportunas según la cantidad de contaminación indicada en la medida.

Por lo tanto, la primera prueba consiste en enviar una petición al pollution-controller con una medida de contaminación elevada y en unas coordenadas cercanas a las del controlador de tráfico registrado. Con esto se conseguirá que el pollution-controller genere una alarma que se envía al ctlc-manager que a su vez comprueba que debe dar la orden de cortar el tráfico a su controlador de tráfico registrado.

En la siguiente imagen tenemos un ejemplo de la petición al pollution-controller con una medida de contaminación superior al límite máximo permitido, el cual es configurado con una variable de entorno en el servicio del pollution-controller.

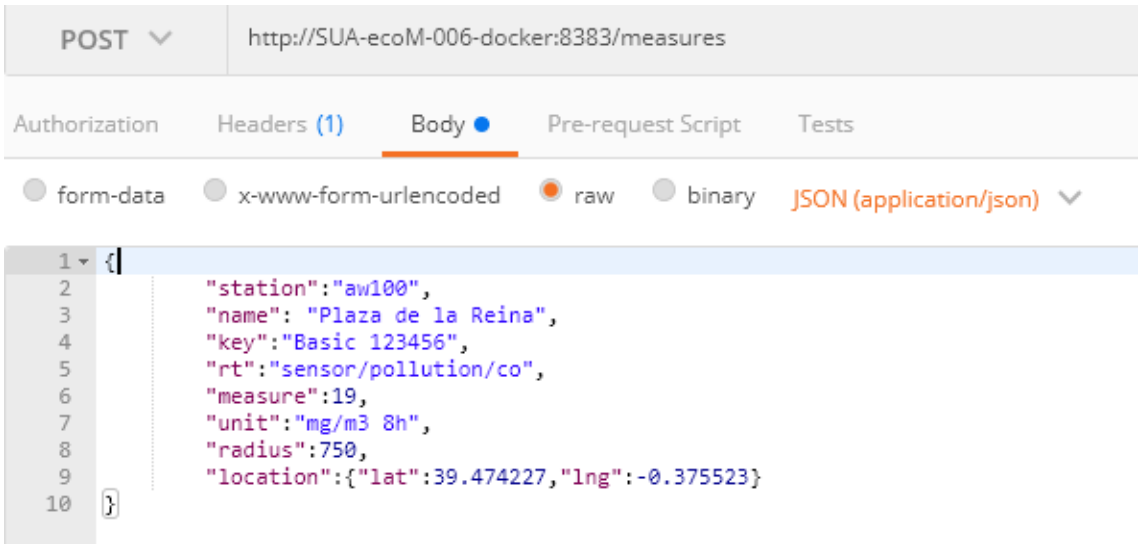

Figura 31 Muestra petición con lectura contaminación al pollution-controller

El servicio responde con un estado http 200 ok tras procesar la medida correctamente. Si se revisan los logs del pollution-controller podemos asegurarnos que se ha procesado la medida como esperábamos. En la siguiente imagen se puede ver el log que ha generado el pollution-controller, indicativo de que ha procesado la medida enviada. Se puede ver que envía una alarma a su nodo descendiente, el nodo de la capa 1, y a su nodo superior, donde está el servicio de reportes.

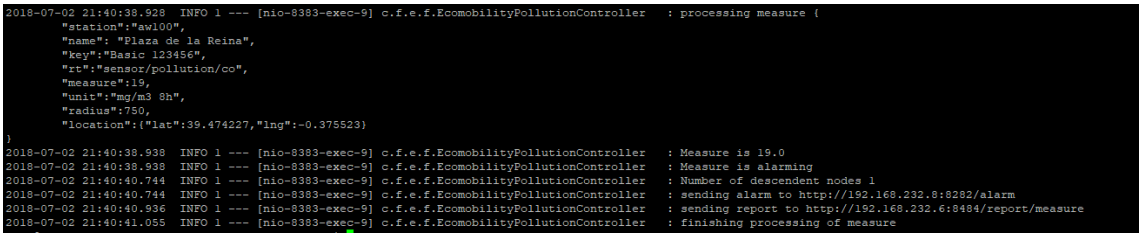

Figura 32 Muestra logs pollution-controller tras procesar lectura de contaminación

La última comprobación necesaria es comprobar el estado del controlador de tráfico, que debería haberse cerrado al tráfico. En la siguiente imagen se puede ver que se ha modificado el estado del controlador de tráfico, el cual está registrado en el ctlc-manager que es quien le ha enviado una petición para cortar el tráfico.

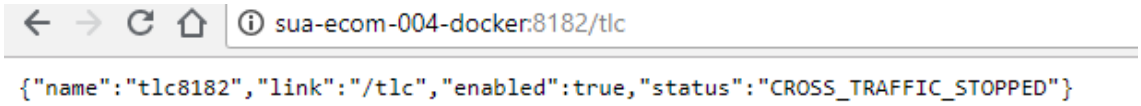

Figura 33 Muestra del estado final del controlador de tráfico tras la prueba realizada, semáforo corta el tráfico

La otra prueba que se puede hacer es probar el caso inverso, es decir, enviar una medida de contaminación con valores normales para que el controlador de tráfico vuelva a su estado normal de funcionamiento.

En la siguiente imagen se puede ver que se envía una medida de contaminación dentro de los niveles permitidos porque en la configuración del servicio pollution-controller se había indicado el MAX_POLLUTION_LEVEL a 14.

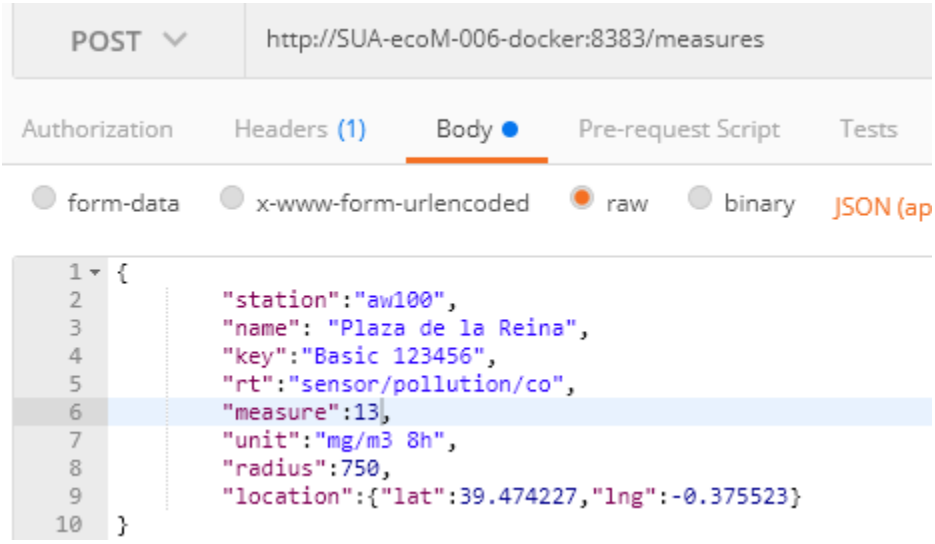

Figura 34 Muestra petición al pollution-controller con lectura de contaminación dentro de los niveles permitidos

El pollution-controller procesa esta medida, se puede ver en la siguiente imagen.

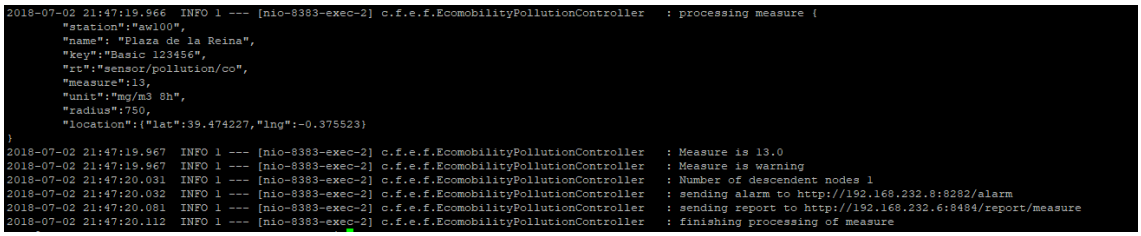

Figura 35 Muestra logs pollution-controller tras procesar lectura de contaminación dentro de los niveles permitidos

Y finalmente en la siguiente imagen se puede ver que el controlador de tráfico vuelve a su estado normal de funcionamiento.

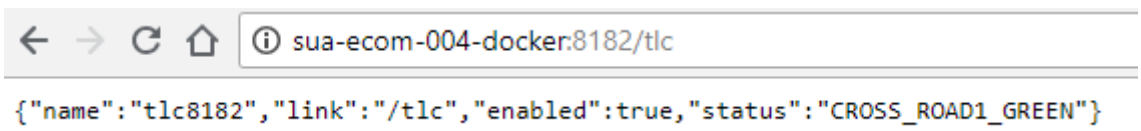

Figura 36 Estado final del controlador de tráfico tras procesado de lectura dentro de los niveles permitidos, semáforo verde funcionando de forma normal

7 Conclusiones

Concluyo este trabajo final de máster muy satisfecho con el proyecto realizado. Sobre los resultados obtenidos puedo decir que han sido muy buenos respecto a los objetivos planteados. Considero que los objetivos planteados no eran fáciles de cumplir pero se han alcanzado con creces.

El hecho de hacer un trabajo que me planteaba muchos retos me ha motivado mucho. He disfrutado mucho durante todo el proceso que ha requerido este trabajo y creo que esto se ha plasmado en los resultados obtenidos.

Se ha conseguido implementar un prototipo de plataforma que permite ofrecer el acceso a una determinada estructura Fog como un servicio.

He de remarcar que ha habido detalles que se han dejado para trabajos futuros o simplemente aspectos que se ha tomado la decisión de diseñarlos y posteriormente implementarlos de un modo más simple que otras alternativas más efectivas pero mucho más complejas.

Respecto a trabajos futuros puedo enumerar los siguientes puntos:

- Posibilitar el registro de nodos en la red Fog cuya ubicación es dinámica. Con esto se posibilitaría el uso de la plataforma en el sector automovilístico para el desarrollo de vehículos inteligentes conectados.
- Desarrollar un mecanismo para hacer de forma automática la instalación de servicios de aplicación en los Fog Node cuando estos se conecten a la red.
- Implementar más servicios de sistema según las necesidades que se requieran en usos futuros de la plataforma.
- Hacer un rediseño e implementación generalizado de la plataforma revisando los aspectos de seguridad, es decir, añadir autenticación, gestión de usuarios,…
- Hacer un rediseño e implementación de ciertas responsabilidades del Fog Manager para ejecutarlas con un modelo distribuido repartiendo dichas responsabilidades entre los Fog Node.

8 Referencias

[1] «Statista,» julio 2018. [En línea]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[2] Cisco Systems Inc, «Fog Computing and Its Role in the Internet of Things,» [En línea]. Available: http://www.ce.uniroma2.it/courses/sdcc1415/progetti/fog_bonomi2012.pdf.

[3] «OpenFog Consortium,» [En línea]. Available: https://www.openfogconsortium.org/. [Último acceso: Mayo 2018].

[4] O. Consortium, «OpenFog Reference Architecture,» [En línea]. Available: https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf. [Último acceso: Mayo 2018].

[5] «SONM,» [En línea]. Available: https://sonm.com/. [Último acceso: Junio 2018].

[6] E. Foundation, «Eclipse Oxygen,» [En línea]. Available: https://www.eclipse.org/oxygen/. [Último acceso: Junio 2018].

[7] Oracle, «Java,» [En línea]. Available: https://www.java.com/es/about/whatis_java.jsp?bucket_value=desktop-chrome67-windows10-64bit&in_query=no. [Último acceso: Junio 2018].

[8] S. Framework, «Spring Boot,» [En línea]. Available: https://spring.io/projects/spring-boot. [Último acceso: Junio 2018].

[9] Apache, «Apache Maven Project,» [En línea]. Available: https://maven.apache.org/. [Último acceso: Junio 2018].

[10] MongoDB, «MongoDB,» [En línea]. Available: https://www.mongodb.com/. [Último acceso: Junio 2018].

[11] Docker, «Docker,» [En línea]. Available: https://www.docker.com/. [Último acceso: Junio 2018].

[12] Docker, «Docker Swarm,» [En línea]. Available: https://docs.docker.com/engine/swarm/. [Último acceso: Junio 2018].

[13] Spotify, «Docker Client,» [En línea]. Available: https://github.com/spotify/docker-client. [Último acceso: Mayo 2018].

[14] eclipse.org, «JGit,» [En línea]. Available: https://www.eclipse.org/jgit/. [Último

acceso: Junio 2018].

[15] OpenFog Consortium, «definition-of-fog-computing,» [En línea]. Available: https://www.openfogconsortium.org/what-we-do/#definition-of-fog-computing. [Último acceso: junio 2018].

[16] IFTT, «IFTT,» [En línea]. Available: https://ifttt.com. [Último acceso: Mayo 2018].

[17] Oracle, «Nashorn Documentation,» [En línea]. Available: https://docs.oracle.com/javase/10/nashorn/introduction.htm#JSNUG136. [Último acceso: mayo 2018].