UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERIA DE SOFTWARE

Aplicación de los principios del *Chaos Engineering* en entornos de computación *Edge*

Application of *Chaos Engineering* Principles to *Edge Computing* Environments

Realizado por
Alejandro Cortijo López

Tutorizado por
Rodrigo Román Castro

Departamento
Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio de 2021

Resumen

En este proyecto, se ha investigado la aplicabilidad del *Chaos Engineering* sobre *el Edge Computing* y se ha realizado un caso práctico como demostración.

En primer lugar, se ha realizado una investigación sobre las estrategias *Chaos Engineering* utilizadas hasta la fecha y las aplicaciones de este. De esta forma, hemos obtenido una visión bastante amplia y profunda de la disciplina.

En segundo lugar, se han analizado las *estructuras Edge*, siguiente paso a las *tecnologías Cloud*. Se han investigado desde los fundamentos de esta hasta su estructura y la arquitectura interna de los componentes que la forman. En consecuencia, apreciamos el potencial de estas estructuras, y comenzamos a investigar acerca de los desafíos de seguridad que genera este tipo de plataformas, y la aplicabilidad del *Chaos Engineering* sobre ellas. Para ello, exploramos el uso del *Chaos Engineering* sobre el *Cloud Computing* y comenzamos a abstraer los conceptos sobre el *Edge Computing*.

Finalmente, hemos generado un caso práctico en el que desarrollamos y ejecutamos una estrategia *Chaos Engineering* para encontrar los problemas de seguridad, tanto sobre la *plataforma Edge Computing* como sobre la aplicación desplegada, y en consecuencia solucionar los fallos obtenidos a través de las conclusiones aportadas por la experimentación.

Palabras clave:

- Chaos Engineering.

- Edge Computing.

- Seguridad informática.

Abstract

In this project, we have investigated the applicability of *Chaos Engineering* in *Edge Computing* and we have done a practical case as a demonstration.

Firstly, we have done an investigation about the strategies of *Chaos Engineering* used up to now and the applications it has. In this way, we have obtained an extended and deep vision of this discipline.

Secondly, we have analyzed the *Edge structures*, the next step of *Cloud technologies*. We have investigated from their bases to the structure and the internal architecture of the components they are made up of. Consequently, we appreciate the potential of these structures, and we start to investigate the security challenges that generate these types of platforms, and the applicability of *Chaos Engineering* to them. To do these, we explore the use of *Chaos Engineering* in *Cloud Computing* and we start to abstract the concepts of *Edge Computing*.

Finally, we have generated a practical case in which we develop and execute a *Chaos Engineering* strategy to find the security problems, in the *Edge Computing platform* and also in the deployed application, and consequently we solve the errors obtained from the conclusions given by the experimentation.

Keywords:

- Chaos Engineering.

- Edge Computing.

- Computer security.

Glosario y acrónimos

- **5G:** quinta generación de las redes móviles, con una mayor velocidad y procesamiento de la información.

- **Almacenamiento SATA:** tipo de disco duro caracterizado por una interfaz de transferencia de información que permite una mayor velocidad entre el disco duro y la placa base respecto a los modelos anteriores de discos duros.

- **Aplicaciones Cloud:** aplicación lanzada sobre una plataforma Cloud.

- **Arquitectura cliente - *servidor*:** modelo de arquitectura basado en dos entidades: los demandantes o clientes y los proveedores de recursos o *servidores*. Los clientes realizan peticiones sobre el *servidor* y este se encarga de proporcionar respuestas en función a la petición realizada.

- **Arquitectura orientada a eventos:** patrón de arquitectura software basado en eventos o reacciones, las cuales generan un cambio significativo en el estado del sistema.

- **Ataque de denegación de servicio (DDoS):** ofensa caracterizada por la realización masiva de peticiones desde diferentes equipos, con la intención de perjudicar la disponibilidad del equipo atacado.

- **Bare-metal:** software instalado directamente sobre el propio disco duro, sin necesidad de *sistema operativo*.

- ***BD*: *base de datos*.**

- **Cachés:** memoria intermedia entre el hardware y el software (de menor tamaño y en consecuencia más rápida) utilizada para realizar las búsquedas en memoria de forma mucho más rápida. Sigue ciertas políticas a la hora de guardar los archivos más utilizados para conseguir una mayor eficiencia.

- ***Clúster*:** conjunto de ordenadores trabajando de forma ordenada, como si fuesen un único equipo.

- **Compilación en tiempo de ejecución:** técnica de compilación basada en la traducción a código máquina nativo en tiempo de ejecución.

- **Descentralización:** organización de funciones caracterizada por la distribución en varios equipos.

- **Endpoint:** punto de comunicación de los clientes con el *servidor*. Normalmente, suele ser una URL.

- **Feedback:** retroalimentación obtenida a través de experimentos, pruebas o ciertos recursos.

- **Hypercalls:** llamadas internas del *sistema operativo* al hipervisor.

- **I / O de alta velocidad:** procesamiento de la información de entrada / salida cuyo atributo principal es la necesidad del procesamiento rápido.

- **I / O de baja velocidad:** procesamiento de la información de entrada / salida cuyo atributo principal es la necesidad del procesamiento normal o lento.

- **IA:** inteligencia artificial. Capacidad de programar procesos de inteligencia humana en máquinas.

- **Imagen *Docker*:** conjunto de programas, librerías y archivos necesarios para construir un *contenedor Docker* y poder ejecutarlo correctamente.

- **Integridad de la información:** capacidad de mantener la información completa y sin modificaciones con respecto a la información original.

- **IoT:** Internet of Things. Conexión de los objetos cotidianos a Internet.

- **JSON:** JavaScript Object Notation. Formato ligero de intercambio de información con fácil legibilidad, tanto para las máquinas como para los humanos. Sigue varias reglas de notación, como las llaves {}, los corchetes [] o los dos puntos : .

- **Kubernetes:** sistema de código libre para el despliegue y uso de *contenedor*es.

- **Memoria DRAM:** componente similar a la caché. Su funcionamiento principal consiste en ayudar al procesador a realizar las tareas de forma más rápida, debido a que puede utilizar la información almacenada en ella en vez de ir al disco rígido. El proceso de búsqueda en las memorias DRAM es directo a la zona de almacenamiento donde se encuentra la información.

- **Métodos HTTP (*GET*, *POST*, *PUT* y DELETE):** instrucciones utilizadas por el protocolo HTTP para comunicar las acciones a realizar sobre el *servidor*: *GET* (obtener información del *servidor*), *POST* (guardar información en el *servidor*), *PUT* (modificar información del *servidor*) y DELETE (eliminar la información del *servidor*).

- ***Microservicios*:** enfoque arquitectónico para desarrollar software basado en pequeños servicios independientes conectados a través de una interfaz definida.

- **Módulos de servicio:** módulos útiles para el despliegue de memoria RAM.

- **Módulos NPM:** extensión de código propia de *Node.js* similar a las librerías.

- **No repudio:** capacidad de demostrar la participación de los usuarios en las acciones realizadas por estos.

- **NVMe:** protocolo de almacenamiento caracterizado por su alto rendimiento.

- ***Open Source*:** desarrollo de software gratuito de forma colaborativa y abierta.

- ***Peticiones HTTP*:** solicitud realizada mediante el uso del protocolo HTTP, protocolo de transporte basado en la estructura cliente - *servidor*.

- **Plataformas Cloud / nube:** plataforma lanzada en el propio Internet con la intención de aportar un servicio disponible a toda persona con conexión a este.

- **Privacidad:** capacidad de un usuario de limitar las personas con conocimiento de su información personal.

- **Procesador:** componente encargado de interpretar las acciones hardware mediante el uso de operaciones aritméticas y lógicas, además de encargarse de generar la información de salida.

- ***REST*:** conjunto de principios arquitectónicos cuya finalidad principal es el desarrollo de servicios web. Entre estos principios encontramos la transferencia mediante el uso del protocolo HTTP y uso de URIs.

- ***Rollback*:** conjunto de acciones a realizar para volver a un estado pasado.

- ***Servidor*:** equipo capaz de recoger las peticiones realizadas por los clientes y devolver una respuesta acorde a la petición procesada.

- **Simian Army:** conjunto de herramientas diseñadas para comprobar la salud y resiliencia de un servicio desplegado en Internet.

- ***Sistema operativo* (SO):** capa de abstracción entre el hardware y el software. Este se encarga de utilizar y gestionar los recursos del equipo en función a las instrucciones aportadas por el usuario.

- **USB:** dispositivo de almacenamiento con uso de memoria flash, cualidad basada en poder escribir y leer en varias posiciones de memoria durante una misma operación.

- **Virtualización:** representación software de algún tipo de recurso tecnológico.

- ***XML*:** Extensible Markup Language. Lenguaje de etiquetado para documentos utilizado para describir datos de manera estructurada e independiente a la plataforma.

Índice

1

Introducción

1.1 Motivación

El proyecto desarrollado persigue el análisis de varios aspectos relacionados con la seguridad informática. El primero de ellos es el *Chaos Engineering*: esta disciplina consiste en **buscar y forzar errores dentro de un sistema y mejorarlo de forma iterativa**, conforme vamos realizando los experimentos y obteniendo las conclusiones de estos errores **[2]**. El fundamento principal de esta disciplina es siempre dudar del funcionamiento correcto del sistema, y pensar formas en las que el sistema dejaría de aportar su funcionamiento esperado. De esta forma, conseguimos encontrar errores de manera gradual y solucionar problemas no detectables en un proceso normal de desarrollo de software.

Esta novedosa disciplina está siendo utilizada por las grandes multinacionales, debido a su filosofía y a su potencial. El uso principal que se le está dando al *Chaos Engineering* consiste en comprobar la disponibilidad de las plataformas Cloud de estas grandes multinacionales, mediante la eliminación de forma aleatoria de sus recursos y *contenedor*es **[3]**. Además, Amazon ha comenzado un proyecto para ofrecer este testeo a pequeñas plataformas, es decir, *Chaos Engineering* como servicio **[1]**.

Sin embargo, esta técnica está dirigida principalmente a un solo punto de la seguridad informática: *la disponibilidad de los sistemas*. Otros atributos igual de importantes en la

seguridad informática se dejan de lado, como por ejemplo, *la integridad de la información, la privacidad o el repudio* **[4]** (podemos apreciar las áreas que cubren los diferentes *frameworks* en la figura 1). Todos estos atributos se pueden comprobar igualmente a partir del *Chaos Engineering*, mediante experimentos y estados en los que el sistema viole alguno de los atributos anteriores. Es por esto por lo que el primer punto a desarrollar en este proyecto es analizar la capacidad de aportar seguridad a un sistema usando el *Chaos Engineering*.

Framework	Target Stack	Resiliency Attributes	Application Layer
Chaos Kong [6]	AWS Regions	availability (non-security)	cloud network
Chaos Gorilla [7]	AWS Availability Zones	availability (non-security)	cloud network
Chaos Monkey [8]	AWS Availability Zones	availability (non-security)	cloud instances (VMs)
Chaos Monkey for Spring Boot [9]	Spring Boot Applications	availability (non-security e.g. latency, terminations)	Java applications (e.g. REST, inter-service calls)
Royal Chaos [10]	Java applications	availability (non-security)	JVM
Chaos Toolkit [11]	AWS, Azure, Google & Kubernetes	availability (non-security)	cloud & kubernetes network
PowerfulSeal [12]	Kubernetes	availability (non-security)	kubernetes network (e.g. pods, microservices)
ChaosMesh [13]	Kubernetes	availability (non-security)	kubernetes network
ChaoSlingr [14]	AWS	security (confidentiality, integrity & availability)	cloud services (e.g. S3, IAM)
CloudStrike [15]	AWS & GCP	security (confidentiality, integrity & availability)	cloud services (e.g. S3, IAM)

FIGURA 1: *Frameworks* de *Chaos Engineering* con sus respectivas áreas de resiliencia **[4].**

El segundo punto en el que se sustenta este proyecto es el *Edge Computing*, siguiente paso a las *tecnologías Cloud*. Su característica principal es **la capacidad de trasladar el procesamiento y la computación de la nube al límite más cercano a los usuarios**, por ejemplo a los routers o a las antenas. De esta forma, el rendimiento y eficiencia crecen exponencialmente debido a la velocidad de procesamiento de la información, abriendo muchísimas oportunidades a la computación **[5]**.

El problema principal que presentan las plataformas *Edge Computing* es su novedad: como toda nueva tecnología, presenta errores (como errores de seguridad) debido a su arquitectura y su funcionamiento **[6]**. Es en este punto donde podemos comenzar a utilizar el *Chaos Engineering*, ya que podemos aplicarlo al *Edge Computing* al igual que

lo aplicamos al *Cloud*. De esta forma, podremos testear los sistemas *Edge* y encontrar los problemas de seguridad que estos presenten.

1.2 Objetivos

Como se ha mencionado anteriormente, los objetivos principales del proyecto son los siguientes:

1. **OBJETIVO 1**: Realizar un análisis que muestre de forma teórica la aplicabilidad de los principios de *Chaos Engineering* en entornos *Edge Computing*.
2. **OBJETIVO 2**: Desarrollar un caso práctico como prueba de concepto en donde se implemente un servicio de seguridad basado en *Chaos Engineering* para comprobar la seguridad de una plataforma *Edge Computing Open Source*.

Estos objetivos se pueden detallar en las siguientes tareas realizadas a lo largo del proyecto:

- **Hacer un análisis en profundidad** sobre las diferentes estrategias de *Chaos Engineering* y los protocolos utilizados a la hora de realizar la experimentación. De esta forma, tendremos una visión estructurada de las estrategias utilizadas en el *Chaos Engineering* actualmente.

- **Estudiar y realizar un análisis** de las aplicaciones del *Chaos Engineering*. De esta manera, conseguimos una visión lo más amplia posible sobre el área; y también veremos las aplicaciones prácticas utilizadas por las grandes empresas recientemente.

- **Estudio de la *infraestructura Edge***, su funcionamiento y su arquitectura. De este modo, obtenemos el conocimiento necesario para saber cómo funcionan las *plataformas Edge*, concepto vital en el desarrollo del proyecto.

- **Investigación del *Chaos Engineering* sobre *aplicaciones Edge***. En este punto, se desarrollará toda la información obtenida relacionando ambos conceptos y se comenzará a definir de forma teórica el experimento práctico que se va a realizar.

- **Análisis de las tecnologías a utilizar** para el desarrollo del caso práctico. De esta forma, seleccionaremos las tecnologías que más se ajusten al tipo de aplicación a desarrollar.

- **Análisis de las *plataformas Edge***, comprobando las características principales de cada una de ellas, sus atributos más destacados y la plataforma que más se ajuste a las necesidades del caso práctico.

- **Definir la estrategia *Chaos Engineering*** a seguir en el caso práctico; desde los puntos de comienzo, como por ejemplo los esquemas y el equipo de observación, hasta la estructura necesaria para definir los experimentos a realizar.

- **Implementación de la funcionalidad básica**. Tras seleccionar la plataforma con la que realizaremos el caso práctico y las tecnologías para realizar la aplicación a desplegar, debemos crear la aplicación sobre la que vamos a ejercer el *Chaos Engineering*. Una vez creado el *servidor*, analizaremos su diseño y estructura. Finalmente, ejecutaremos la estrategia definida anteriormente.

- **Diseño y ejecución de la experimentación**. Dentro de la estrategia del *Chaos Engineering*, debemos definir los experimentos a realizar sobre el sistema, que ejecutados mejorarán el *servidor* de forma iterativa y aportarán la seguridad necesaria.

- **Implementación de la funcionalidad completa**. Una vez mejorada la aplicación en la funcionalidad básica, desplegaremos la aplicación sobre

la *plataforma Edge*. Primero, analizaremos la estructura de la propia plataforma. posteriormente, diseñaremos y ejecutaremos los experimentos según la estrategia definida y comprobaremos la seguridad de la plataforma y de la aplicación en producción.

1.3 Estructura de la memoria

Esta memoria está organizada en cinco capítulos divididos de la siguiente manera:

- **Capítulo 1 - Introducción:** En este capítulo define las bases de la memoria, especialmente las motivaciones y los objetivos del proyecto a realizar.

- **Capítulo 2 - *Chaos Engineering*:** En este capítulo se explican principalmente los conceptos base del *Chaos Engineering*; se definen los pasos a seguir, es decir, la estrategia, los *"Game Days"* (o experimentación) y las conclusiones. Finalmente, se centra en las aplicaciones del *Chaos Engineering*.

- **Capítulo 3 - Infraestructuras *Edge*:** En este capítulo expone los fundamentos del *Edge Computing*, centrándose principalmente en las definiciones base y en la arquitectura de los *nodos Edge*. Por último, se presentan los desafíos generados por el *Edge Computing* y la aplicación del *Chaos Engineering* sobre este.

- **Capítulo 4 - Tecnologías utilizadas:** En este capítulo se realizará la comparación y elección, tanto de las tecnologías utilizadas para realizar el *servidor* como de las *plataformas Edge* para realizar el caso práctico.

- **Capítulo 5 - Implementación del caso práctico:** En este capítulo se explicará todo el proceso de desarrollo de la experimentación, tanto de la funcionalidad simple (*servidor*) como de la funcionalidad completa (*servidor* desplegado en la *plataforma Edge*).

1.4 Metodología

El proyecto ha sido realizado con el proceso de gestión SCRUM **[7]**. Para ello, se ha dividido en varias secciones y cada una de ellas contiene varios puntos, los cuales se han ido desarrollando de forma explícita. De esta forma, podemos tener un conocimiento detallado del desarrollo del proyecto, así como tener obtener una crítica constructiva con cada entrega parcial o total de cada una de las secciones.

2

Chaos Engineering

Una vez explicada la estructura y los objetivos del proyecto, nos centraremos en el primer punto principal del proyecto, el *Chaos Engineering*. En este capítulo, se realizará primero una introducción, definiendo los conceptos más importantes del *Chaos Engineering*. Seguido a esto, se investigará la estrategia a seguir y cómo generar la experimentación. Finalmente, se explicarán las conclusiones obtenidas y las aplicaciones del *Chaos Engineering*.

2.1 Introducción al *Chaos Engineering*

Actualmente, la necesidad de que los sistemas funcionen correctamente durante todo su tiempo de funcionamiento es el desafío pendiente de toda empresa, desde la empresa más pequeña a la multinacional más grande. Sin embargo, por mucho control que haya sobre el sistema, siempre aparecen nuevos problemas. Estos problemas pueden ser causados de forma intencionada o de forma casual, pero el daño que pueden generar a las empresas es considerable. Esta problemática ha dado mucho que reflexionar hasta el punto de comenzar a generar nuevas disciplinas, como por ejemplo, el *Chaos Engineering* **[2]**.

El *Chaos Engineering* es una **novedosa disciplina basada en forzar los errores, dentro de un sistema de forma controlada**, mediante una serie de pruebas o también llamadas experimentos.

Las inconsistencias, también denominadas *deuda oscura*, son la respuesta a los errores controlados que se han ejercido sobre el sistema. Estas inconsistencias deberán ser analizadas y arregladas de forma iterativa cada vez que sean registradas, a través de las conclusiones que se vayan obteniendo **[2]**.

De esta forma, podremos controlar la *consistencia, seguridad, disponibilidad y rendimiento* de un sistema de forma continua. Aunque un sistema parezca que funciona correctamente, debe ser analizado detenidamente y buscar los posibles fallos para continuar con su mejora. El *Chaos Engineering*, además de abarcar los aspectos técnicos de un sistema, también abarca más niveles como son el tráfico del sistema, la infraestructura o la monitorización **[2]**.

Como podemos apreciar, el *Chaos Engineering* es una disciplina muy completa y novedosa, la cual está creciendo exponencialmente en los últimos años. Es por esto por lo que grandes empresas como Amazon, Google o Netflix han comenzado a utilizarla para comprobar la resiliencia y disponibilidad de sus servicios **[3]**.

No obstante, el *Chaos Engineering* debe seguir una estrategia y un control bastante exhaustivo, ya que cualquier cambio realizado sin posibilidad de *rollback* puede crear errores sin capacidad de recuperación. En el siguiente apartado, se explicarán los pasos a seguir para poder aplicar el *Chaos Engineering* de forma satisfactoria.

2.1.1 Pasos a seguir en el *Chaos Engineering*

Los puntos a seguir en el *Chaos Engineering* son un concepto vital, ya que cualquier paso en falso puede generar errores considerables como la eliminación o desconexión de recursos críticos. A continuación, se irán definiendo y explicando cada uno de ellos.

El primer paso, y la base del *Chaos Engineering*, es la siguiente pregunta: ¿Sabemos qué hará el sistema en el siguiente caso? Esta pregunta será realizada por dos posibles motivos **[2]**:

- El sistema *ha tenido un accidente anteriormente* debido a este caso.

- *Hay incertidumbre por parte de los responsables* sobre el comportamiento del sistema.

Una vez realizada la pregunta, obtenemos las hipótesis sobre el comportamiento del sistema, estas son las bases de los *"Game Day"* o experimentos **[2]**. Estas preguntas se realizan sobre un esquema principal del funcionamiento del sistema, a partir del cual se verán los puntos críticos y de experimentación de forma más clara.

Un ***"Game Day"*** o experimento es un **evento en el cual se pone en ejecución alguna de las turbulencias sobre las cuales tenemos incertidumbre o creemos que ha producido un fallo en el sistema [2]**. Una vez que se ha realizado el *"Game Day"*, recopilamos los datos y obtenemos las evidencias y observaciones al respecto para poder mejorar el sistema.

Un dato a destacar sobre este proceso es su uso en entornos de prueba, estos serán más seguros y no son incorrectos. No obstante, la precisión sobre la realidad no es la misma que realizarlos sobre el propio entorno de producción, por lo que se tendrían que contemplar las ventajas y las desventajas. Una buena práctica para evitar grandes problemas en el entorno de producción consiste en ir aumentando el radio del *"Game Day"* poco a poco, de esta forma obtenemos fiabilidad de forma gradual y evitaremos errores graves **[2]**.

De forma más esquemática, serían los siguientes pasos **[2]**:

- **1º Paso: acumulación de hipótesis**:
  - Para obtener todas las hipótesis posibles, debemos reunirnos con nuestro equipo y estudiar todos los análisis de los accidentes pasados, además de estudiar todos los casos que generan incertidumbre en el equipo. Esta búsqueda debe ser lo más exhaustiva posible. Con ayuda de un *esquema de arquitectura de la aplicación*, podremos ver los puntos más críticos.

- **2º Paso: introducción del impacto y la probabilidad**:
  - Una vez que hemos obtenido la lista de hipótesis, debemos realizar el *análisis de efecto*. Para ello, debemos aportar a cada hipótesis la posibilidad de que ocurra y el grado de impacto que tendría si esta ocurriese. De esta forma, obtenemos un *gráfico de probabilidad / impacto.*

- **3º Paso: categorizar cada una de las hipótesis**:
  - Es necesario saber cada una de las áreas a las que ataca cada hipótesis. De esta forma, acotamos el área de confianza que aportaría la resolución de las hipótesis.

- **4º Paso: comenzar a realizar los *"Game Days"*:**
  - Una vez que sabemos cuáles son los fallos que pueden causar mayores problemas, debemos generar sus respectivos *"Game Days"*. A partir de estos, se podrá explorar cómo funcionan los sistemas de alerta, qué miembros del equipo reaccionarán al accidente, que indicaciones se obtienen sobre la salud del sistema o como responderá este a las condiciones turbulentas.

A continuación, se definirán los pasos a seguir para generar la experimentación del sistema, los *"Game Days"*.

2.1.2 *"Game Days"*, estructura y creación

El *"Game Day"* es el encargado de convertir una hipótesis en un experimento de fácil ejecución para la comprobación de las debilidades del sistema. La creación de los *"Game Days"* debe seguir un protocolo lo más exhaustivo y detallado posible, debido a que cualquier vacío puede generar margen de error. Los pasos de definición de los *"Game Days"* son los siguientes **[2]**:

- **1º Seleccionamos la hipótesis que deseamos estudiar**.

- **2º Seleccionamos el estilo del *"Game Day"*.**
  - Los *"Game Days"* se pueden dividir en *aviso previo*, si los integrantes conocen de la experimentación, y *sin aviso previo*, si necesitamos del desconocimiento del resto de integrantes. De esta forma, podemos ver la capacidad de reacción o el impacto.

- **3º Decisión de quienes participan y quienes observan**:
  - Se deberá realizar una lista con cada uno de los integrantes que participarán en el experimento, aportando su nombre, su rol y la razón por la que deben participar en el experimento.

- **4º Decidimos donde ocurrirá el *"Game Day"*:**
  - Lo ideal es comenzar en el entorno más seguro posible y con un radio de acción muy cercano. A partir de ahí, se deberá ir avanzando gradualmente hasta llegar a producción.

- **5º Decidimos cuando comenzará y cuánto tiempo durará:**
  - El tiempo para conseguir las evidencias y el tiempo máximo de estrés que el equipo puede soportar son las medidas que debemos tener en cuenta. A partir de estas dos, obtenemos el *tiempo óptimo* para realizar el experimento.

- **6º Describimos el experimento:**
  - La descripción del experimento debe ser lo más detallada posible y no dar margen a suposiciones:

    - **El estado estable:** conjunto de medidas que indican que el sistema está funcionando correctamente.

    - **El método:** conjunto de actividades que se realizarán para generar las condiciones turbulentas en el sistema.

    - **Procesos de retroceso:** conjunto de acciones a través de las que volvemos al estado estable remediando todas las turbulencias generadas.

- **7º Obtenemos la aprobación:**
  - Debemos realizar una lista, para comprobar si cada uno de los participantes del experimento ha sido notificado y si aprueba que se realice el experimento.

Un punto muy importante es el conocimiento que se tiene sobre el sistema. Si este conocimiento es escaso, es necesario una entidad que tenga total conocimiento sobre el sistema y sepa si el experimento no se está encaminando en una dirección peligrosa.

2.1.3 Conclusiones

Como resumen, el esquema de funcionamiento a seguir es el siguiente:

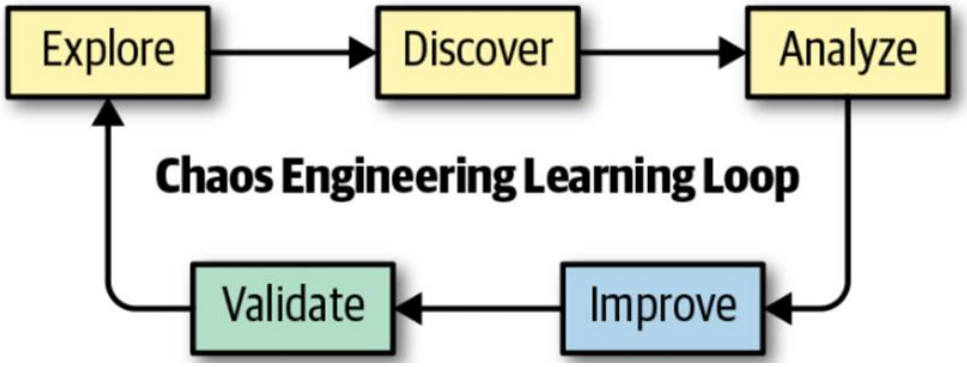

FIGURA 2: Esquema de funcionamiento de las estrategias *Chaos Engineering*.

Como podemos apreciar, esta metodología se puede aplicar a todo tipo de sistemas, desde las propias aplicaciones hasta a una *plataforma Edge*, debido a su abstracción. La metodología tiene unas pautas estrictas a seguir en cualquier sistema, desde sistemas más simples a sistemas con arquitecturas más complejas.

2.2 Aplicaciones del *Chaos Engineering*

Cada día son más las empresas que usan el *Chaos Engineering*, las grandes multinacionales comenzaron a implementarlo hasta el punto de comenzar a ofrecerlo como servicio **[3]**.

Primero, analizaremos el caso de Amazon. Esta empresa utiliza el *Chaos Engineering* para situar a su sistema en los peores casos y de esta forma encontrar errores potenciales antes de que estos ocurran de forma inesperada. No obstante, la relación de Amazon con el *Chaos Engineering* no queda aquí. La compañía comenzó a ofrecer "*Chaos Engineering as a Service*", llamado *AWS Fault Injection Simulator* **[8]**. La idea principal de este servicio es dar la capacidad a las empresas de probar sus *aplicaciones Cloud* a pequeña escala, no solo al nivel de Amazon. El servicio ofrece completa administración y asesoramiento para probar los experimentos diseñados sobre el hardware de *AWS*, facilitando la ejecución de experimentos seguros y obteniendo un informe con los resultados obtenidos.

El proceso de creación es el mismo que seguimos en la estrategia del anterior punto. Primero, fijamos el *estado estable, la hipótesis y las inyecciones* a realizar sobre el sistema. Mientras se realiza el experimento y una vez que finaliza este, se proporcionarán los datos sobre el experimento realizado (*CloudStack* e informes). Tras analizar correctamente estos datos, se realizarán las mejoras oportunas.

Otro caso similar al de Amazon es el de Netflix. La empresa creó *"Chaos Monkey"* como resultado sobre la mejora de la recuperación de la nueva arquitectura de la compañía, basada en la nube. La idea principal de *"Chaos Monkey"* se sustenta en la eliminación de *clústeres de AWS* para ver cómo se comporta la plataforma ante la falta de estos, y de esta forma comprobar su robustez. Sin embargo, esta herramienta no fue la única, Netflix creó la *"Simian Army"* **[9]** cubriendo ciertas áreas como por ejemplo:

- ***"Latency Monkey"***, para reducir la latencia de la comunicación.

- ***"Confirmity Monkey"***, para comprobar que los *servidores* siguen buenas prácticas.

- ***"Security Monkey"***, basada en apagar instancias mediante el uso de violaciones de seguridad.

- ***"Chaos Gorilla"***, parecido al uso de *"Chaos Monkey"* pero mediante zonas de disponibilidad en vez de por equipos.

- ***"Chaos Kong"***, herramienta más alta de la jerarquía. Su funcionamiento se centra en hacer caer una región completa de *AWS*.

- ***"Janitor Monkey"***, se centra en los equipos y recursos no utilizados para evitar desperdicio y desorden.

Con todas estas herramientas, Netflix comprueba constantemente la resistencia de sus *servidores*. Además, hay muchas otras herramientas, como pueden ser *"Tornasol"* o *"ChaosSinglr".*

"ChaosSinglr" **[11]** ha sido la primera herramienta *Open Source* en el ámbito del *Chaos Engineering* centrada en la ciberseguridad. La idea principal de esta herramienta es realizar experimentos de seguridad sobre las *infraestructuras AWS*, y descubrir las debilidades que estas aplicaciones presentan. Normalmente, este instrumento es utilizado sobre sistemas distribuidos complejos.

En cambio, *"Tornasol"* **[10]** es utilizado para encontrar debilidades en los *sistemas kubernetes*, y las aplicaciones en ejecución dentro de los *contenedor*es. Las características principales de esta herramienta son:

- **Proporcionar ejemplos genéricos** listos para su uso.
- **Proporcionar una API** para la gestión del flujo de trabajo.
- **Admite Python, Ansible y Go** para la creación de los experimentos.

Finalmente, esta disciplina no está siendo solo utilizada por Amanzon y Netflix. Fue difundiéndose a otras grandes empresas como Google, IBM o Yahoo.

3

Infraestructuras *Edge*

Una vez analizado el *Chaos Engineering*, sus definiciones principales, y el desarrollo de estrategias y aplicaciones, nos centraremos en el segundo punto principal del proyecto: las infraestructuras *Edge*. En este capítulo, comenzaremos definiendo los conceptos clave y los usos principales de las estructuras *Edge*. Una vez definidos, nos centraremos en la arquitectura del *Edge Computing*: sus elementos principales, los sistemas operativos y las plataformas *Edge*. Por último, se desarrollarán los desafíos generados por el *Edge* y las aplicaciones que unen el *Chaos Engineering* con el *Edge*.

3.1 Introducción al *Edge Computing*

Los *sistemas Edge* son esencialmente sistemas remotos cuyos fundamentos se basan en los *sistemas Cloud*, los *sistemas integrados, la seguridad y las telecomunicaciones* **[5]**.

Por definición, podemos dividir a los componentes entre el *far-Edge* y el *near-Edge* **[5]**:

- **Componentes *near-Edge*:** parte de la infraestructura entre el *far-Edge* y la *nube*. Esta capa hospeda servicios computacionales complejos.

- **Componentes *far-Edge*:** dispositivos de procesamiento capaces de comunicarse, administrar e intercambiar datos con dispositivos en la *nube* y / o *near-Edge*. Esta

capa está más alejada de la *capa de nube general*, pero aún mantiene una relación con la *nube*, así como con sus componentes primarios cercanos al *borde*. Esta capa es la más cercana a los usuarios finales.

Generalmente, cuanto más lejos se encuentre el componente de la *nube*, mayor limitación en los recursos tendrá. El siguiente esquema muestra el funcionamiento de estas dos áreas:

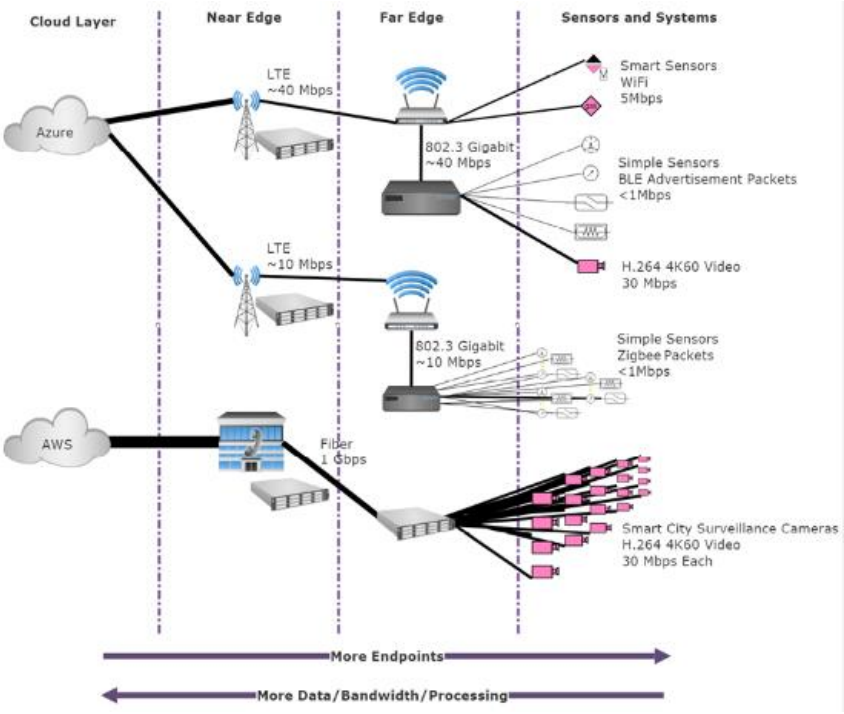

FIGURA 3: estructura de los *nodos Edge*.

Esta estructura tiene ciertas ventajas a la hora de desarrollar aplicaciones, entre ellas encontramos **[5]**:

- **Latencia reducida:** el *Edge Computing* se puede acercar más a los usuarios y a los servicios, reduciendo los saltos de red y la velocidad de envío de los datos.

- **Conservación del *ancho de banda*:** los *entornos Cloud* suelen tener ciertas restricciones acerca del *ancho de banda* o el almacenamiento. En cambio, el

Edge Computing puede ayudar a solventar esta problemática mediante el uso de *caché, compresión de datos o técnicas de filtrado*.

- **Computación resiliente:** hay casos en los que la información es crítica y la pérdida de esta es inaceptable. Mediante el uso de *cachés* en el *Edge*, pueden evitarse estas pérdidas.

- **Seguridad y privacidad:** el *Edge* tiene la capacidad de filtrar toda la información y datos sensibles antes de que estos sean enviados a la *nube*.

Como podemos apreciar, la propia estructura de las *plataformas Edge* ayudan a solventar problemas críticos en la computación. Sin embargo, la estructura no es el único punto necesario para poder solucionar estos problemas. La arquitectura de los nodos que componen la *estructura Edge* es un punto vital, la capacidad de almacenamiento o procesamiento son puntos críticos para solventar estos problemas de forma eficiente. A continuación, en el siguiente apartado, desarrollaremos la arquitectura de los *nodos Edge*.

3.2 Arquitectura del *Edge Computing*

Dependiendo de las circunstancias y del entorno, existen múltiples diseños en el hardware de los *nodos Edge*. No obstante, independientemente del diseño, el hardware final debe atender la carga de trabajo y los casos de uso que se pretenden colocar en el *sistema de borde*. A continuación, abordaremos los puntos más importantes de las *arquitecturas Edge* y las posibles *estructuras Edge*.

3.2.1 Elementos principales

Los elementos principales de los *nodos Edge* son lo esencial para que estos sirvan como *estructura Edge*. Entre los elementos principales de los *nodos* encontramos **[5]**:

- **Procesador:** elemento central del componente.

- ***Cachés y memoria jerárquica*:** todo procesador moderno tiene asociado memorias de alta velocidad actuando como *cachés*, normalmente divididas en dos niveles (dependiendo del uso de la información).

- ***Memoria DRAM* y *memoria volátil*.**

- ***Clases de almacenamient*o:** dependiendo de varios atributos, como la capacidad o la velocidad lectura / escritura. Hay variedad de formas, como por ejemplo *SATA, NVMe* o *USB*.

Además, hay muchos otros atributos como son *I / O de alta velocidad*, *I / O de baja velocidad* o los *módulos de servicio*.

3.2.2 Sistemas Operativos

Las opciones de *sistemas operativos* en el borde para un *sistema Edge* son bastante amplias y significativas.

El ***sistema operativo*** funciona como una **capa de protección y abstracción entre el hardware y las aplicaciones**. No obstante, también proporciona una interfaz para que pueda alojarse la propia aplicación (mediante la creación de procesos o administración de memoria) **[5]**.

La elección del *sistema operativo* depende de varios puntos **[5]**:

- **Coste de distribución del *SO*.**

- **Soporte del *SO*.**

- **Arquitectura de procesadores que soporta el *SO*.**

- Si **soporta todas las características necesarias**: *memoria virtual* o *cachés multinivel*.

- **Extensiones disponibles**.

- **Servicios de seguridad disponibles**.

- **Cómo funcionan los sistemas de ficheros**.

Como se puede apreciar, son muchos los factores a tener en cuenta a la hora de escoger un *SO*.

3.2.3 Plataformas *Edge*

Actualmente, existen múltiples plataformas donde poder alojar nuestro software en un estado seguro y controlado dentro de un *componente Edge*. En cualquiera de los casos, queremos que el software y el sistema sean **[5]**:

- **Robustos:** capaces de recibir, crear *imágenes* y volver a ejecutar software a medida que se van implementando.

- **Controlados:** utilización de un servicio central para supervisión.

- **Responsive:** *feedback* sobre las acciones realizadas con *imágenes*.

Para ello, se aportan múltiples soluciones basadas en la virtualización de recursos computacionales. Algunas son **[5]**:

- **Virtualización hardware**: abstracción a nivel hardware aplicable a partir de cualquier software que se pueda ejecutar en el *bare-metal*. Utiliza un *hipervisor* para administrar las *máquinas virtuales*, además admite la *replicación virtual del hardware*.

- **Paravirtualización**: proporciona una capa de abstracción hardware (HAL) y requiere *controlador*es especiales. Accede al software a partir de *hipercalls*, para ello se necesita modificar el *SO*. De esta forma, se obtiene un mayor rendimiento y capacidad de comunicación con el *hipervisor*.

- ***Contenedor*es:** gestiona la abstracción a nivel de aplicación. Los *contenedor*es solo requieren el *SO* de alojamiento para proporcionar los servicios básicos.

En algunas *aplicaciones Edge*, los *contenedores* son una solución muy atractiva debido a su versatilidad, su ligereza y portabilidad a la hora de implementar aplicaciones en equipos periféricos. No obstante, se debe tener en cuenta los recursos a nivel de sistema, como son el *rendimiento, la capacidad de almacenamiento y las características del procesador*. Debido a la gran versatilidad que nos aportan los *contenedores*, seguiremos nuestra investigación sobre estos.

Un *componente Edge*, así como cualquier otro componente que utilice *Docker*, recoge la *imagen* y la ejecuta de manera similar. Esto facilita exponencialmente la tarea de implementación y desarrollo. Además, permite que los modelos de desarrollo para *componentes Edge* sean como procesos en soluciones *SaaS* (*software como servicio*).

3.3 Aplicaciones del *Edge Computing*

El *Cloud Computing* ha generado grandes avances tecnológicos a lo largo de los últimos años. Sin embargo, la capacidad de procesamiento y almacenamiento de la información en un punto mucho más cercano al usuario aumenta las oportunidades considerablemente. Entre algunas de estas oportunidades encontramos las siguientes **[12]**:

- **Vehículos autónomos:** la capacidad de poder procesar toda la información que va recopilando el coche mientras va circulando es crucial para que el coche pueda tomar decisiones en tiempo real sobre su conducción; como las señales o los otros coches y peatones que van circulando por las vías alrededor.

- **Smart Cities:** la capacidad de que toda la información de una ciudad esté siendo procesada en todo momento necesita un tiempo de latencia y una capacidad de procesamiento que el *Cloud*, a día de hoy, no puede aportar. Sistemas de iluminación, transportes públicos o tráfico controlado por redes son algunos de los casos los cuales pueden generar cantidades masivas de información.

- **Redes eléctricas inteligentes:** las redes inteligentes permiten aportar la demanda de electricidad en tiempo real. De esta forma se evitan incidentes, y en caso de que estos ocurran se puede reducir su duración, además de mejorar la eficiencia y la calidad del sistema.

- **Ámbito gaming:** la velocidad de procesamiento hace que las respuestas sean mucho más rápidas. Además, esta capacidad de procesamiento impulsará la realidad virtual y la realidad aumentada.

- **Industria 4.0:** la maquinaria de la fábrica es capaz de generar información y recopilarla. De esta forma, podremos aumentar la *eficiencia, calidad y sostenibilidad*, además de prevenir accidentes y minimizar sus daños.

- **Asistencia sanitaria:** la telemedicina de calidad, la cirugía robótica y los registros sanitarios digitales son puntos de gran importancia en el ámbito de la sanidad; los cuales pueden ser mejorados con el *Edge Computing*.

Además de estos casos, hay muchos otros que pueden ser mejorados considerablemente con el *Edge Computing*, como pueden ser el sector de las finanzas o el de la educación.

Sin embargo, la gran mayoría de las aplicaciones mencionadas anteriormente son críticas y cualquier fallo en ellas podría suponer la vida de una persona o la caída de una entidad de suma importancia. Es por esto por lo que las plataformas deben ser lo más seguras posibles. En el siguiente capítulo, se investigará sobre la unión del *Edge*

Computing con el *Chaos Engineering* para detectar errores de difícil visión y soportar condiciones extremas.

3.4 Desafíos generados por el *Edge*

Como podemos apreciar, el *Edge Computing* tiene ciertas áreas las cuales generan grandes objetivos de seguridad: la *descentralización de los nodos Edge*, las conexiones entre los diferentes nodos que controlan la información (*agentes* o *controladores*) o la *privacidad de los flujos de datos* que circulan a través de los nodos. Todos estos elementos forman un conjunto de áreas con grandes desafíos de seguridad.

Uno de los desafíos más importantes es el *ataque de denegación de servicio* (*DDoS*). La cantidad de elementos susceptibles de ataque aumenta considerablemente debido a que la computación comienza a realizarse en estos nodos, al contrario que en el *Cloud*, ya que solo es un *servidor* central.

Otro de los desafíos más importantes que presenta el *Edge Computing* es la comunicación y la orquestación de los diferentes nodos que conforman los *sistemas Edge*. Ciertas plataformas, como por ejemplo *IoFog* **[13]**, tienen un nodo central (el *controlador*), ciertos nodos intermedios (*agentes*) y los nodos finales (*contenedores* con la aplicación desplegada).

El sistema debe tener la capacidad de actuar consecuentemente ante ciertos errores que pueden presentar los diferentes nodos:

- **Desconexión del *contenedor*** que hospeda la aplicación.
- **Eliminación de nodos *agente***.
- **Desconexiones ilegítimas de los servicios**.
- **Eliminación del *controlador***.

- **Desconexión del *controlador* y los *agentes***.

Además de todos estos factores, hay que tener en cuenta otras acciones, como por ejemplo, el *repudio del sistema*, la *identificación* de cada uno de las actos realizados y los *rollbacks* para poder solucionar los posibles problemas. En el caso práctico, analizaremos todos estos puntos más detenidamente e intentaremos aportar ciertas soluciones ante este tipo de fallos.

Sin embargo, hay muchos más objetivos por resolver a niveles más bajos. La modularidad que aporta el *Edge Computing* en los diferentes nodos hace que estos tengan que estar conectados continuamente. Es por esto por lo que se deben controlar ciertos factores, entre ellos:

- **Información enviada entre nodos correctamente cifrada**. De esta forma, evitamos ataques de interceptación de la información.

- **Evitar la modificación de la información**.

- **Evitar que la información sea enviada desde un punto externo a la plataforma**. Este punto evita consecuentemente la suplantación de identidad entre nodos y los ataque de denegación de servicio, ya que eliminaría las peticiones ilegítimas de forma inmediata.

- **Protocolo de comunicación** para saber que la información ha sido recibida y enviada correctamente.

- **Capacidad de almacenamiento en caché** si la información no ha sido enviada correctamente.

Como podemos apreciar, son muchos los puntos a analizar sobre el *Edge Computing*. Todos estos tipos de factores hacen que el *Chaos Engineering* sea una buena fórmula

para encontrar estas situaciones comprometidas, ya que estos casos son mucho más probables de encontrar forzando al sistema a condiciones extremas y comprobando su comportamiento.

En los siguientes apartados, analizaremos toda la información posible acerca de la unión del *Chaos Engineering*, tanto con el *Edge Computing* como con el *Cloud Computing*. De esta forma, podremos contemplar el enfoque que se le ha dado a día de hoy y tenerlo en cuenta para el desarrollo del caso práctico.

3.5 *Chaos Engineering* sobre las aplicaciones *Edge*

Actualmente, la literatura relacionada con la aplicación del *Chaos Engineering* en entornos *Edge Computing* es muy limitada, principalmente debido a la novedad de ambas tecnologías. Sin embargo, la unión de ambas áreas genera un campo de investigación bastante amplio.

Los problemas de seguridad generados por el *Edge Computing* son un gran desafío debido a su complejidad a la hora de realizar su búsqueda. Los nodos empleados, las conexiones realizadas entre estos o el formato del transporte de la información son muchos de los factores que hay que tener en cuenta a la hora de realizar la búsqueda.

Sin embargo, el cambio de pensamiento que aporta el *Chaos Engineering* genera grandes posibilidades para encontrar estos posibles fallos sobre el *Edge Computing*. La capacidad de pensar que un sistema no funciona correctamente de forma independiente a su desarrollo, rompe los esquemas de pensamiento sobre cualquier obra de ingeniería. Esta forma de pensamiento ayuda a encontrar los errores menos superficiales y más escondidos, debido al pensamiento escéptico y empírico que esta disciplina genera.

Además, hay que tener en cuenta que existen investigaciones que estudian la aplicación del *Chaos Engineering* al *Cloud Computing*, las cuales se detallarán a continuación. Si el

Chaos Engineering se puede aplicar al *Cloud*, se podría aplicar al *Edge Computing*, debido a que i) el *Edge Computing* puede considerarse tanto una evolución como un complemento al *Cloud Computing*, y ii) el *Edge Computing* utiliza muchas de las tecnologías subyacentes del *Cloud Computing*.

Ejemplos de la aplicación del *Chaos Engineering* al *Cloud* son los estudios realizados por el *Hasso-Plattner-Institute for Digital Engineering* **[4]**, o por el *KTH Royal Institute of Technology* **[14]**. La base de estas investigaciones es las brechas de seguridad debidas a errores humanos y configuraciones erróneas de las plataformas. No obstante, todos los modelos de seguridad se centran en el cliente y dejan de lado los paradigmas de seguridad tradicionales.

Primero, nos centraremos en los puntos más importantes de la investigación *del Hasso-Plattner-Institute for Digital Engineering* sobre la implementación de la seguridad en el *Cloud* mediante el uso del *Chaos Engineering* **[4]**. Para la investigación, fue utilizada la herramienta *CloudStrike*, sistema de seguridad en la nube basado en los principios del *Chaos Engineering*. El funcionamiento principal de *CloudStrike* consiste en la inyección de fallos de seguridad sobre el sistema.

El fundamento principal de esta investigación es la falta de aplicabilidad del *Chaos Engineering* sobre la seguridad de un sistema. El 49% de los problemas de seguridad son generados por los humanos (por ejemplo, fallos de auditoría o fallos en las listas de control de accesos). Estos motivos hacen que sea necesario evolucionar los modelos de seguridad, y de esta forma consolidar el factor humano en la *seguridad del Cloud*.

Para poder identificar estos fallos, adoptaron el *Fault Injection Testing (FIT)* dentro de los principios del *Chaos Engineering*. Esta técnica consiste en la introducción de fallos en producción de forma intencionada para tener un mayor conocimiento del escenario de fallo y ayudar al diseño e implementación de sistemas tolerantes a fallos. Esta extensión del *Chaos Engineering* se denomina *"Security Chaos Engineering"*, y su característica adicional, además de exponer la resiliencia de un sistema, es comprometer los

fundamentos de la seguridad: *integridad, confidencialidad y disponibilidad de un sistema determinado*.

El **"*Security Chaos Engineering*"** consiste en la **identificación de fallos de seguridad mediante la experimentación proactiva** para aportar confianza sobre la capacidad del sistema de defenderse de condiciones maliciosas en producción.

Para ello, se utilizará el *"Risk Driven Fault Injection"*. Este proceso se divide en varias tareas iterativas, teniendo el conocimiento base de la seguridad del sistema como punto principal. Las tareas son las siguientes:

- **Ejecución**: el primer paso consiste en ejecutar las campañas de inyección de fallos en el sistema objetivo y encontrar todos los fallos posibles dentro del alcance definido.

- **Monitorización:** visión en tiempo real para ver cómo reacciona el sistema.

- **Análisis:** observación de la información e identificación tanto de las vulnerabilidades como de las partes implicadas.

- **Plan:** procesamiento de la información, ya sea para mejorar el sistema o para mejorar las inyecciones de fallo.

Finalmente, para el desarrollo de las campañas de inyección de fallos se realizan los siguientes pasos:

- **1º Generar una hipótesis sobre un estado estable**, dando lugar a un estado anormal justificado con parámetros medibles.

- **2º Cambios en los eventos de la vida real**, modificando la magnitud de las acciones normales para conseguir el estado anormal justificado en la hipótesis anterior.

- **3º Realización de los experimentos**, además de un proceso de aumento del radio de acción.

- **4º Automatización de los experimentos** para la continua realización de estos.

Como podemos apreciar, las bases de la investigación son los pilares de las estrategias *Chaos Engineering*, realizando pequeñas modificaciones sobre estas (por ejemplo, énfasis principal a problemas de configuración u obtención de los estados anómalos mediante acciones cotidianas).

La segunda investigación que vamos a analizar es la realizada por el *KTH Royal Institute of Technology* **[14]**. Esta investigación se fundamenta en la experimentación sobre tres aplicaciones desplegadas en *contenedor*es (tamaño medio y *arquitectura cliente - servidor*).

El fundamento principal de esta investigación es la capacidad de testear la seguridad de los sistemas a través de peticiones. Para ello, el primer punto es saber que llamadas son las que más interés causan sobre los sistemas. Es por esto por lo que la primera acción a realizar es monitorizar el flujo normal de peticiones sobre las diferentes aplicaciones. Una vez que tenemos las llamadas más críticas, realizamos las evaluaciones de las aplicaciones inyectando fallos a partir de estas llamadas .

Conforme se va realizando la experimentación, se comienza a realizar una *monitorización multicapa* para poder observar el efecto de los experimentos desde diferentes puntos de vista .

Finalmente, se realizan unas conclusiones a partir de los datos obtenidos, teniendo en cuenta el funcionamiento de la propia aplicación, el consumo de recursos y las respuestas de las propias *peticiones HTTP*. De esta forma, se verá si los sistemas son eficientes y tienen buenas respuestas ante todo tipo de peticiones.

Ante esta investigación, se obtiene como conclusión que la *observación multicapa* es muy beneficiosa a la hora de realizar la monitorización. La capacidad de analizar las *peticiones HTTP* y los recursos utilizados muestra de forma más explícita cómo funciona el sistema y comprobar los límites de este.

Una vez analizados los estudios sobre el *Cloud*, nos centraremos en la aplicación de estos conceptos sobre la *plataforma Edge* y sobre la propia aplicación desplegada en el *Edge*. Para ello, utilizaremos la misma metodología iterativa y formas similares de inyección de fallo y observación, tanto sobre la *plataforma Edge* como sobre la aplicación.

4

Tecnologías utilizadas

Para demostrar que el *Chaos Engineering* es utilizable sobre aplicaciones desplegadas en el *Edge Computing* para aportar la seguridad, se realizará un caso práctico general en el que desarrollaremos la experimentación sobre la aplicación y sobre la *plataforma Edge*. En este capítulo, se desglosarán las tecnologías utilizadas en dicho caso práctico y el razonamiento de uso de cada una de ellas.

4.1 Tecnologías de desarrollo

En este punto, se recogerá toda la información referente a las tecnologías de desarrollo utilizadas en el caso práctico. Además, se explicará por qué se ha seleccionado cada una de ellas.

Como detalle a destacar, la aplicación que va a ser desplegada en la plataforma *Edge Computing* es un *servidor web*, por lo que toda la selección de tecnologías va a estar condicionada a este detalle.

En primer lugar, para el desarrollo de la *plataforma Edge*, se ha utilizado *IoFog* **[13]**, tecnología *Open Source* creada por *Eclipse Foundation* **[15]** para el desarrollo de *plataformas Edge* basadas en *microservicios*. Su ámbito principal, además de su documentación y soporte, hicieron que fuese la plataforma seleccionada para el caso

práctico del proyecto. En el siguiente punto, se analiza más detenidamente por qué se seleccionó esta plataforma en vez de otras.

En el desarrollo del *servidor* se utilizó *Node.js* **[16]** , entorno *JavaScript Open Source* con una *arquitectura orientada a eventos*, fácil expansión del código a través de los módulos (*Node Package Manager* con diferentes utilidades) y *compilación en tiempo de ejecución*. De esta forma, la optimización de las funciones es mayor. Además de todas las características anteriores, tenemos muchas otras, como por ejemplo: buen rendimiento ante las solicitudes al *servidor* o fácil conexión con otros servicios. Todas estas características hicieron que me decantara por *Node.js* como entorno de programación para el desarrollo del *servidor*.

Para la creación de la *base de datos*, se ha utilizado una *base de datos NoSQL* mediante el uso de *MongoDB Atlas* **[17]**. Las características que nos han hecho elegir a *MongoDB Atlas* frente a otras bases de datos son las siguientes:

- **Fácil conexión con el *servidor Node.js*** debido a los módulos de *mongoose*. Mediante el uso de dos líneas código, se puede conectar el *servidor* con la *base de datos*.

- ***Clúster* online gratuito:** la capacidad de tener la *base de datos* de forma online aporta muchas facilidades a la hora de poder interactuar con la *base de datos* y realizar las comprobaciones desde varios equipos.

- **Utilización de *JSON* para guardar los datos**. *Node.js* tiene diversos módulos para utilizar los *JSON* de forma sencilla y eficaz.

- **Modularidad de los datos**. La capacidad que aporta *NoSQL* a la hora de captar la información de forma rápida y sin necesidad de simetría en la información hace que sea una gran opción.

Como tecnologías auxiliares, han sido utilizadas *Visual Studio Code* **[18]** para el desarrollo del código, debido a ciertas ventajas como la fácil memorización de funciones, la capacidad de crear archivos fácilmente y la fácil estructuración del código, además de muchas otras funcionalidades. Otra tecnología utilizada ha sido *Docker* **[19]**, para la orquestación de *contenedores* en la plataforma *IoFog* **[13]** debido a su versatilidad y facilidad a la hora de crear las imágenes y desplegar los *contenedores*. Por último, tenemos la consola de comandos para poder realizar todas las acciones sobre la plataforma de *IoFog*.

Finalmente, para el testeo del *servidor* hemos utilizado *Postman* **[20]** , herramienta de testeo de *servidores* web basada en peticiones. Estas son algunas de las características principales que han decantado la balanza a la hora de utilizar *Postman* como herramienta de orquestación de experimentos:

- **Gran cantidad de recursos para realizar las pruebas**. A través de la aplicación, se pueden generar múltiples peticiones realizando cambios en todos los campos de la cabecera o el *path*.

- **Gran cantidad de datos aportados** cuando se realizan las pruebas.

- **Capacidad de guardar los experimentos** para realizarlos en cualquier otro momento.

Una vez explicadas cada una de las herramientas a utilizar para el desarrollo del *servidor*, nos centraremos en la comparación y selección de la *plataforma Edge* para la prueba de concepto.

4.2 Análisis de plataformas *Edge*

El número de *plataformas Edge* ha aumentado recientemente, ya que existen plataformas *Open Source* con diferentes características dependiendo del proyecto a realizar. Por lo tanto, en esta sección analizaremos las *plataformas Edge* más

importantes y seleccionaremos aquella plataforma que sea más adecuada para la realización del caso práctico del proyecto.

4.2.1 Plataformas *Edge*

Entre las plataformas *Open Source*, podemos encontrar la iniciativa *LF Edge*, perteneciente a *Linux Foundation* **[29]**, cuya intención es ofrecer un marco abierto e interoperable para el *Edge Computing*. Existen también otras plataformas como *IoFog* **[13]**, la cual pertenece a la iniciativa de *Eclipse Foundation* **[15]**, y aporta una *plataforma Edge* estructurada en *microservicios*.

Las plataformas *Edge Computing* ofertadas son las siguientes:

- ***Akraino Edge Stack* [21]:** plataforma caracterizada por ofrecer gran nivel de flexibilidad a la hora de escalar *servicios Edge* rápidamente. Tiene como fundamento principal la capacidad de proporcionar una latencia aproximada de 20 milisegundos. Entre las aplicaciones proporcionadas encontramos: aplicaciones *5G, IoT o AI / AR*.

- ***Baetyl* [22]:** plataforma basada principalmente en *contenedores* de uso general. Compatibilidad con *k8s*.

- ***FLEdge* [23]:** plataforma basada en el *Edge Computing* industrial. La intención principal de esta plataforma es acelerar la adopción por parte de la industria. Las operaciones críticas, el mantenimiento predictivo y la seguridad son puntos vitales que se intentan mejorar con esta plataforma.

- ***Edge X Foundry* [24]**: plataforma cuyo objetivo principal es la escalabilidad y la flexibilidad, facilitando la interoperabilidad entre equipos y aplicaciones en el *Edge*.

- ***KubeEdge* [25]:** sistema basado en la orquestación de aplicaciones en *contenedores* cuyas capacidades se pueden desplegar en el *Edge*. No es una *plataforma Edge* pura, ya que ciertas bases se sitúan en el *Cloud.*

- ***Open Horizon* [26]:** plataforma que permite la gestión autónoma de las aplicaciones implementadas en flotas distribuidas a escala web. También permite la gestión autónoma de más de 10.000 dispositivos de borde simultáneamente, es decir, 20 veces más puntos finales que en las soluciones tradicionales.

- ***Home Edge Project* [27]:** plataforma *Edge Computing* dedicada al ecosistema del hogar, principalmente al uso del *IoT* con el uso de APIs que también corren librerías. Las instancias se fundamentan en *contenedores Docker* debido a su fácil uso e independencia.

- ***Proyect EVE* [28]:** *sistema operativo* basado en *Linux* para el *Edge Computing* distribuido. La intención principal es crear una base abierta que simplifique el desarrollo, la seguridad y la orquestación de recursos.

- ***IoFog* [13]:** plataforma para crear un *sistema Edge* distribuido basado en *microservicios* de forma segura, rápida y escalable. Proyecto de *Eclipse Foundation*.

4.2.2 Elección de la *plataforma Edge* para el caso práctico

Una vez analizadas las plataformas, debemos escoger una para realizar nuestro caso práctico. El primer punto que debemos tener en cuenta a la hora de escoger una plataforma es el tipo de aplicación que queremos lanzar en el *Edge*. Hay plataformas que están optimizadas para el *IoT*, otras para el *5G* u otras para *microservicios*. En este caso, tendríamos que buscar alguna plataforma óptima para el lanzamiento de un *servidor Node.js* con *base de datos MongoDB Atlas*. Además, debemos tener en cuenta muchas otras características, como pueden ser la documentación y el soporte aportado.

A continuación, se realizará una tabla para poder comparar todas las plataformas de la forma más precisa posible.

El primer punto a analizar es el área que cubre cada una de las plataformas:

Plataformas Edge	Área
Akraino Edge Stack	Aplicaciones *IoT, 5G* o *IA*
Baetyl	Aplicaciones capaces de desplegar en *contenedor*
FLEdge	Industria *IoT*
Edge X Foundry	*Middleware IoT*
KubeEdge	Aplicaciones capaces de desplegar en *contenedor*
Open Horizon	Aplicaciones de gestión autónoma
Home Edge Project	*IoT* en el hogar
Project EVE	*Sistema operativo* para simplificar el desarrollo
IoFog	Desarrollo basado en *microservicios*

FIGURA 4: áreas de las *plataformas Edge*.

Entre todas estas plataformas, las que más se adaptan a nuestras necesidades son principalmente *Baetyl* **[22]**, *KubeEdge* **[25]** e *IoFog* **[13]**, ya que son las plataformas que más facilidades aportan a la hora de desplegar un *servidor* web y las que están más enfocadas para el despliegue de este tipo de servicios.

Partiendo de esa base principal, se compararán otros atributos como son el soporte y la documentación aportada, ya que estos atributos son de suma consideración debido a la novedad de las *plataformas Edge* y el poco soporte disponible:

Plataformas Edge	Documentación	Soporte
Baetyl	Media / baja	Bajo
KubeEdge	Media / alta	Bajo
IoFog	Alta	Medio

FIGURA 5: comparación de las diferentes plataformas mediante documentación y soporte.

El soporte obtenido por *KubeEdge* **[25]** se basa en discusión a través de *GitHub* con muy poca participación por parte de los usuarios. En cambio, el soporte de *IoFog* **[13]** es similar al de *KubeEdge* **[25]**, pero con bastante más participación por parte de los usuarios. El método de soporte consiste en un foro de discusión con múltiples temas y usuarios activos. En caso de tener alguna duda acerca de la plataforma, tienes la posibilidad de abrir tema, y los usuarios de forma medianamente rápida contestan las dudas. Finalmente, el soporte de *Baetyl* **[22]** se centra en un correo de soporte, pero funciona con tiempos de espera no viables en el desarrollo del proyecto.

En cuanto a la documentación de *Baetyl* **[22]** , tiene puntos con una explicación insuficiente como para realizar la implementación. Ciertos pasos no se ajustan a las necesidades de la versión. La documentación de *KubeEdge* **[25]** en cambio, es más explícita que la de *Baetyl* **[22]**. Sin embargo, la documentación de *IoFog* **[13]** es clara, específica y muy intuitiva. Para la comparación de las tres plataformas, me fui guiando por sus correspondientes guías y tutoriales; y a partir del proceso de implementación con estas plataformas, pude realizar la comparación de la forma más exacta posible.

Finalmente, todos estos aspectos hicieron que me decantara por la plataforma *IoFog* **[13]** para el desarrollo del caso práctico del proyecto. Como síntesis de las ventajas que aporta *IoFog* **[13]** tenemos:

- **Área adecuada para el desarrollo de un *servidor* web**.

- **Soporte adecuado ante dudas**.

- **Documentación clara y específica para el despliegue de la plataforma**.

- **Documentación clara y específica para el despliegue del *microservicio*** sobre la plataforma.

- **Soporte basado en *Docker*** para el despliegue del *microservicio*.

Una vez tenemos seleccionada la plataforma, en el siguiente capítulo, comenzaremos con la parte práctica del proyecto: la prueba de conceptos. En este caso práctico, desarrollaremos un ejemplo en el que utilizaremos todo lo estudiado en los capítulos anteriores, para aportar la seguridad tanto en la aplicación, que va a ser desplegada en el *Edge*, como sobre la propia *plataforma Edge*.

5

Implementación del caso práctico

Una vez estudiadas los dos áreas principales sobre las que se sustenta el proyecto, el *Chaos Engineering* y el *Edge Computing*, procedemos al desarrollo de la prueba de conceptos. Para ello, el objetivo principal consistirá en utilizar el *Chaos Engineering* sobre la aplicación desplegada en el *Edge* y sobre la propia *plataforma Edge*. De esta forma, conseguiremos encontrar las vulnerabilidades y aportar soluciones para mejorar la seguridad del sistema.

5.1 Tecnologías y estructura del caso práctico

La aplicación seleccionada para realizar la prueba de conceptos será un *servidor REST* escrito en *Node.js* y conectado a una *base de datos NoSQL* en *MongoDB Atlas*. La aplicación base consistirá en cuatro *métodos (GET, POST, PUT y DELETE)* para poder acceder y utilizar la *base de datos* de forma completa.

Una vez desarrollado el *servidor*, se comenzará a aplicar las estrategia del *Chaos Engineering* sobre este: *acumulación de hipótesis, cálculo de impacto y probabilidad de las hipótesis y categorización de estas*.

Tras realizar todos estos pasos, se tienen que definir las características de la experimentación: selección de participantes, decisión del entorno de ejecución de la experimentación, decisión de la fecha y duración de las pruebas, selección del estilo de los "Game Days" (con previo aviso o sin este) y obtención de la aprobación por parte de los integrantes de la experimentación.

Finalmente, se generará una batería de experimentos con los siguientes valores por cada una de las pruebas:

- **Definición del experimento**: redacción de hipótesis, estado estable, pasos a seguir para generar la anomalía y proceso de retroceso al estado estable.

- **Personal de observación**.

- **Zonas afectadas**.

- **Conclusiones y mejoras a realizar** (este apartado se redactará una vez realizado el experimento).

Como buenas prácticas del *Chaos Engineering*, la experimentación debe comenzar con un radio de acción reducido e ir aumentando este conforme la aplicación vaya adquiriendo confiabilidad. Es por esto por lo que toda la experimentación sobre el *servidor* será primero realizada en local. Se mejorará el *servidor* de forma iterativa con las conclusiones obtenidas, y toda la experimentación volverá a ser realizada cuando la plataforma este desplegada en producción para poder comprobar que todos los cambios son aceptados correctamente.

Como consecuencia, la implementación será dividida en dos partes: la implementación básica, la cual consistirá en la aplicación del *Chaos Engineering* de forma local sobre el *servidor*, y la funcionalidad completa, basada en el despliegue del *servidor* sobre la *plataforma Edge* para realizar los experimentos en producción y validarlos de forma

definitiva (aumento del radio de acción). Por último, para finalizar con la funcionalidad completa, se realizará la experimentación sobre la propia *plataforma Edge* para encontrar las posibles vulnerabilidades de seguridad que la plataforma tenga.

5.2 Desarrollo del caso práctico (funcionalidad básica)

Siguiendo la estrategia del *Chaos Engineering*, lo primero que debemos realizar es la esquematización del sistema. A través del esquema, se puede ver de forma más visual e intuitiva las zonas de conflicto, a partir de las cuales podemos generar las hipótesis y sus respectivos experimentos. El esquema de funcionamiento del *servidor* es el siguiente:

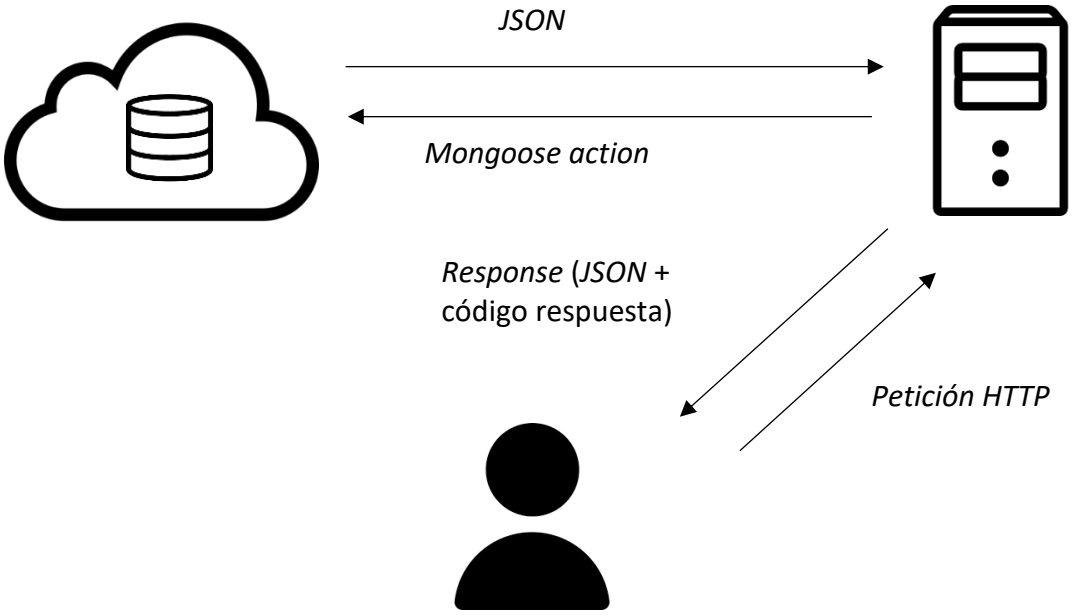

FIGURA 6: esquema de funcionamiento del *servidor*.

5.2.1 Análisis del esquema de funcionamiento del *servidor*

Como podemos apreciar, en el esquema hay dos zonas principales de ataque: el *servidor* y el *clúster*, donde se aloja la *base de datos*. Además de las comunicaciones *BD - servidor* y *servidor* - usuario.

Comenzando por la conexión dentro del *clúster* de *MongoDB Atlas*, encontramos dos tipos de vulnerabilidades: la información recopilada dentro de la *base de datos* y los

accesos a la *base de datos*; es decir, los privilegios que tienen los usuarios sobre la *base de datos* y la capacidad de conexión a esta. Es por esto, por lo que es necesario realizar experimentos sobre estos dos ámbitos que nos generan incertidumbre.

Por otro lado, el *servidor* como entidad principal solo tiene una forma de conexión, a través de *peticiones HTTP*, por lo que toda la experimentación desarrollada debe ser a través de este tipo de peticiones. Un punto de experimentación bastante importante es la capacidad que tiene el *servidor* de tratar las peticiones, peticiones con datos erróneos, cabeceras erróneas, tratamiento de excepciones o *"endpoints"* no recogidos son algunas de las causas principales de fallo en la mayoría de los *servidores*. Nos centraremos especialmente en la experimentación de las peticiones, al igual que se realizó en la investigación del *KTH Royal Institute of Technology* **[14]** .

De forma iterativa, iremos comprobando el comportamiento del *servidor*, la disponibilidad de este y las respuestas a las *peticiones HTTP*. De esta forma, conseguiremos tener información desde diferentes puntos de vista y niveles y obtener las conclusiones más exactas posibles, al igual que en la investigación mencionada en el párrafo anterior.

Una vez desarrollados los fundamentos del sistema, nos centraremos en otros apartados de seguridad. Un punto principal es la falta de *autenticación* e *identificación*, no hay ninguna forma de *identificación*, por lo que los usuarios no autorizados pueden tener acceso a información privada y modificarla. Se deberán realizar experimentos y pruebas que cubran ese ámbito.

Además, otro de los ámbitos que genera bastante incertidumbre es el repudio del *servidor*. No hay constancia de las acciones que realiza cualquier usuario dentro del *servidor*, por lo que debemos realizar experimentos que cubran el área de la *confidencialidad* e *integridad* de los datos, además del repudio del *servidor*.

5.2.2 Formalización de hipótesis

Una vez se ha analizado el esquema de funcionamiento del sistema y hemos obtenido las áreas que generan mayor incertidumbre, comenzamos a formalizarlas mediante las hipótesis:

- ¿Qué ocurrirá si se **desconecta la *base de datos***?
- ¿Qué ocurrirá si **cambiamos los permisos de la *base de datos***?
- ¿Qué ocurrirá si realizamos una **petición a un *"endpoint"* no recogido**?
- ¿Qué ocurrirá si realizamos una **petición a un *"endpoint"* con atributos erróneos**?
- ¿Qué ocurrirá si realizamos una petición a un ***"endpoint"* con más atributos de los necesarios**?
- ¿Qué ocurrirá si se realiza una **petición sin los atributos correspondientes**?
- ¿Qué ocurrirá **si no hay necesidad de credenciales para realizar acciones en el *servidor***?
- ¿Qué ocurrirá **si no hay repudio en el *servidor***?
- ¿Qué ocurrirá **si hacemos un *GET* al *servidor* pidiendo los datos en *XML***?
- ¿Qué ocurrirá **si realizamos un *POST* con JSON poniendo en la cabecera *XML***?
- ¿Qué ocurrirá **si entran en la *base de datos* y roban las credenciales**?

5.2.3 Clasificación de las hipótesis

Una vez redactadas todas las hipótesis de experimentación, debemos clasificar cada una de ellas según el impacto que tendrían sobre el funcionamiento del sistema (de forma aproximada) y el área de la seguridad que abarca, además de la probabilidad de que ocurra cada una de las hipótesis o la necesidad de saber que ocurre ante ciertas acciones.

Clasificación del impacto sobre el sistema y probabilidad de que ocurra:

- **Alto impacto**:
  - ¿Qué ocurrirá si se desconecta la *base de datos*?
    - **PROBABILIDAD BAJA**
  - ¿Qué ocurrirá si cambiamos los permisos de la *base de datos*?
    - **PROBABILIDAD MEDIA / ALTA**
  - ¿Qué ocurrirá si no hay necesidad de credenciales para realizar acciones en el *servidor*?
    - **PROBABILIDAD ALTA**
  - ¿Qué ocurrirá si no hay repudio en el *servidor*?
    - **PROBABILIDAD ALTA**
  - ¿Qué ocurrirá si entran en la *base de datos* y roban las credenciales?
    - **PROBABILIDAD BAJA**

- **Medio impacto:**
  - ¿Qué ocurrirá si realizamos una petición a un *"endpoint"* con atributos erróneos?
    - **PROBABILIDAD ALTA**
  - ¿Qué ocurrirá si hacemos un *GET* al *servidor* pidiendo los datos en *XML*?
    - **PROBABILIDAD MEDIA / BAJA**
  - ¿Qué ocurrirá si realizamos un *POST* con JSON poniendo en la cabecera *XML*?
    - **PROBABILIDAD MEDIA / BAJA**

- **Bajo impacto:**
    - ¿Qué ocurrirá si realizamos una petición a un *"endpoint"* no recogido?
        - **PROBABILIDAD ALTA**
    - ¿Qué ocurrirá si realizamos una petición a un *"endpoint"* con más atributos de los necesarios?
        - **PROBABILIDAD MEDIA**
    - ¿Qué ocurrirá si se realiza una petición sin los atributos correspondientes?
        - **PROBABLIDAD MEDIA**

El criterio a seguir para la clasificación es el daño que puede realizar esa acción sobre el *servidor*; siendo alto impacto un daño que anula al 100% la disponibilidad y funcionamiento del sistema; medio impacto, cierta incertidumbre sobre la forma de procesar los fallos por parte del *servidor*; y bajo impacto, una comprobación de que no hay fallos humanos a la hora de desarrollar el *servidor*, por ejemplo, fallo en control de excepciones.

5.2.4 Categorización de hipótesis

Una vez clasificados según el impacto, debemos categorizar las hipótesis en función de los atributos de seguridad que abarcan:

- ¿Qué ocurrirá si se desconecta la *base de datos*?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si cambiamos los permisos de la *base de datos*?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si realizamos una petición a un *"endpoint"* no recogido?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si realizamos una petición a un *"endpoint"* con atributos erróneos?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si realizamos una petición a un *"endpoint"* con más atributos de los necesarios?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si se realiza una petición sin los atributos correspondientes?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si no hay necesidad de credenciales para realizar acciones en el *servidor*?
    - **CONFIDENCIALIDAD E INTEGRIDAD**

- ¿Qué ocurrirá si no hay control sobre las acciones en el *servidor*?
    - **REPUDIO**

- ¿Qué ocurrirá si hacemos un *GET* al *servidor* pidiendo los datos en *XML*?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si realizamos un *POST* con JSON poniendo en la cabecera *XML*?
    - **DISPONIBILIDAD**

- ¿Qué ocurrirá si entran en la *base de datos* y roban las credenciales?
    - **CONFIDENCIALIDAD E INTEGRIDAD**

5.2.5 Definición y desarrollo de la experimentación

Una vez realizados los primeros pasos, debemos seleccionar a los participantes. En este caso, al ser un proyecto individual, seré el único participante. En cuanto a la fecha y duración, todos los experimentos serán realizados en dos días, y la duración de estos es indefinida, duración necesaria para obtener las máximas conclusiones sobre los experimentos (los niveles de estrés a soportar por el sistema son máximos, ya que no está siendo utilizado por ningún otro usuario) . Finalmente, al ser el único participante, el desarrollo de la experimentación será con previo aviso y doy el visto bueno a la

aprobación del desarrollo de los experimentos, ya que tanto el *servidor* como el *clúster* donde se aloja la *base de datos* se encuentran correctamente y en un *estado estable*.

Finalmente, debemos redactar los experimentos y comenzar a realizarlos. Para el desarrollo de la experimentación, dividimos en tres áreas los experimentos. La primera de ellas es la *base de datos*. El objetivo principal de estas pruebas es testear como actúa el *servidor* ante los fallos que puede tener la *base de datos*. Para ello, se han realizado dos experimentos:

- El primero de ellos consiste en la *desconexión de la base de datos*, eliminando las IPs que se pueden conectar a la *base de datos* y los usuarios con privilegios dentro de esta desde el interior del *clúster*. Una vez realizadas estas acciones, procedemos a realizar una petición al *servidor* y obtenemos un error 500 (problema del *servidor*), accedemos a la consola de comandos del *servidor* y obtenemos como conclusión que este ha caído debido a un fallo en el control de excepciones. Procedemos al arreglo del fallo y volvemos a realizar el experimento, obteniendo un resultado satisfactorio.

- El segundo experimento sobre la *base de datos* consiste en *el cambio de privilegios de los usuarios*. Para conseguir esto, hemos cambiado el rol de los usuarios desde dentro del *clúster* dejando solo a los usuarios leer la *base de datos*. Tras realizar esta acción, procedemos a ejecutar un petición tipo *POST* al *servidor*, para modificar la *base de datos*, y obtenemos un error 500 pero, esta vez el fallo se ha tratado correctamente y el *servidor* sigue funcionando.

La segunda área que se ha experimentado detenidamente ha sido las *peticiones HTTP*. Para comprobar esta área, *se han realizado peticiones en Postman* con todo tipo de atributos en cabecera, como por ejemplo: pedir *XML*, enviar atributos erróneos o no enviar atributos necesarios para el desarrollo de la petición. De esta manera, podemos comprobar si el *servidor* tiene un control adecuado de las peticiones y las excepciones, desde peticiones correctas a peticiones con error. Ante estos experimentos, las

conclusiones han sido bastante satisfactorias, ya que el *servidor* no se ha visto perjudicado por estos, por lo que no se realiza ningún cambio al respecto.

El último área a comprobar es la privacidad, el no repudio de los usuarios y la identificación. Para comprobar estos atributos, se han realizado experimentos en base a unos escenarios:

- El primer escenario, *la identificación de usuarios*, se ha testeado desde el punto de vista de una entidad maliciosa que realiza acciones sobre el *servidor*. Realizamos las acciones sin necesidad de ningún tipo de credencial por lo que obtenemos como conclusión que el *servidor* es muy vulnerable ante usuarios maliciosos. Para ello, hemos programado un sistema de autenticación basado en usuario y contraseña.

- El segundo escenario contemplado ha sido *la entrada de una entidad maliciosa en la base de datos*. Accedemos a la *base de datos* haciéndonos pasar por un usuario ilegítimo y obtenemos la conclusión de que la información sensible (las contraseñas) no se encuentra cifrada y es fácilmente accesible para la entidad maliciosa. En consecuencia, se ha programado un sistema de cifrado de la información sensible. De esta forma, toda esta información se encuentra cifrada en la *base de datos*. Para realizar el cifrado, se ha usado *BCrypt* con 10 rondas de sal.

- El último escenario contemplado es *un usuario legítimo realizando acciones perjudiciales o una entidad maliciosa comprometiendo una cuenta de un usuario legítimo*. Tras realizar las acciones, contemplamos que no hay ninguna forma de saber que acciones y cuando se han realizado, por lo que se ha programado un sistema de logs en el que quede reflejado las acciones realizadas por los usuarios y la fecha en la que han sido realizadas.

A continuación, se mostrará una tabla con la experimentación de forma resumida.

	Área	Experimento	Resultado
1º	*Base de datos*	Desconexión de la *base de datos*	Error 500 de *servidor*, arreglo del control de excepciones
2º	*Base de datos*	Cambio de permisos en la *base de datos*	Error 500 sin caída del *servidor*
3º	*Peticiones HTTP*	Petición a un *"endpoint"* no recogido	Error 404 y *servidor* funciona correctamente
4º	*Peticiones HTTP*	Petición a un *"endpoint"* recogido con atributos erróneos	Error 500 y *servidor* funciona correctamente
5º	*Peticiones HTTP*	Petición a un *"endpoint"* recogido con más atributos de los necesarios	Error 404 y *servidor* funciona correctamente
6º	*Peticiones HTTP*	Petición a un *"endpoint"* recogido sin atributos	Error 404 y *servidor* funciona correctamente
7º	*Identificación*	Falta de credenciales	Entidad no autorizada realiza peticiones, introducimos parámetro de autenticación
8º	*No repudio*	Repudio del *servidor*	No hay sistema de logs, introducimos sistema de historial de acciones
9º	*Peticiones HTTP*	Petición con cabeceras modificadas (accept / *XML*)	Resultado 200 pero devolución en JSON, el *servidor* funciona correctamente
10º	*Peticiones HTTP*	Petición con cabeceras modificadas (content type / *XML*)	Resultado 200 pero la petición no se procesa correctamente, *servidor* funciona correctamente

11º	*Privacidad*	Entrada y robo de credenciales	Es posible el robo de información sensible, esta se comienza a cifrar

FIGURA 7: resumen de la experimentación básica.

Una vez que hemos realizado la prueba de conceptos sobre la aplicación que va a ser desplegada en el *Edge Computing*, nos centraremos en la aplicación del *Chaos Engineering* sobre la propia *plataforma Edge*.

5.3 Desarrollo del caso práctico (funcionalidad completa)

La arquitectura de la plataforma *IoFog* consiste en un conjunto de nodos relacionados entre ellos formando una jerarquía. De esta forma, generan una red *Edge Computing* de *microservicio*s. Los nodos que forman esta jerarquía son:

- ***Controlador*:** núcleo de la red *Edge Computing*, se encarga de orquestar todos los *agentes*. Este se encuentra desplegado en un hardware cuyo acceso es inmediato por todos los nodos.

- ***Agente*:** componente encargado de correr los *microservicio*s y manejar los recursos. El *agente* se encarga de informar al *controlador* de todo su funcionamiento. La administración del *microservicio* se fundamenta en *contenedore*s *Docker* y es responsable de la vida útil del *contenedor* y las imágenes de este.

Un esquema de funcionamiento básico es el siguiente:

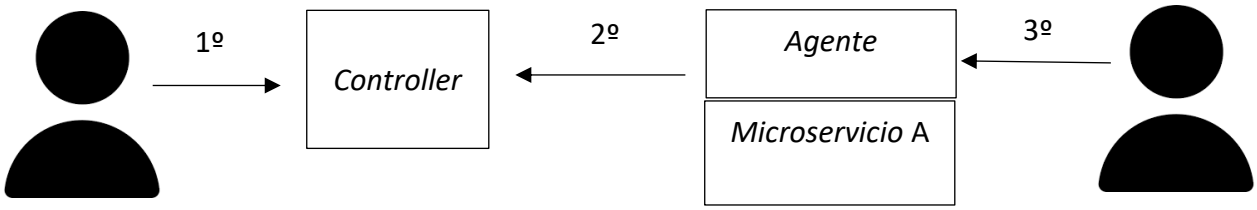

FIGURA 8: esquema de funcionamiento de la plataforma *IoFog*.

- 1º El *controlador* se maneja mediante el uso del CLI o de la API *REST*.
- 2º El *agente* está en continua comunicación con el *"controller".*
- 3º El usuario utiliza el *microservicio* a través del *agente*.

5.3.1 Análisis del esquema de funcionamiento

Por lo que podemos apreciar, de forma teórica hay tres zonas principales de acceso y ataque: el *controlador* y el *agente*, mediante el uso del CLI, y el propio *contenedor* con la aplicación. En este caso, como el *microservicio* es un *servidor REST* (el *servidor* analizado anteriormente), realizaremos la experimentación de las pruebas anteriores para saber cómo reacciona la plataforma ante fallos forzados sobre las peticiones al *servidor* y cómo reaccionará el *servidor* en producción.

5.3.2 Formalización de hipótesis

Las hipótesis obtenidas sobre este esquema son las siguientes, dando como condición que comenzamos los ataques desde el interior del sistema que contiene el control de la *plataforma Edge*:

- ¿Qué ocurrirá si se **desconecta el *agente* del sistema *Edge***?
- ¿Qué ocurrirá si se **desconecta el *controlador* del sistema *Edge***?

- ¿Qué ocurrirá si se **desconecta el *contenedor* que contiene la aplicación**?

- ¿Qué ocurrirá si se **para la aplicación *Edge***?

- ¿Qué ocurrirá si se **elimina el *controlador* del sistema *Edge***?

- ¿Qué ocurrirá si se **elimina el *agente* del sistema *Edge***?

5.3.3 Clasificación y categorización de hipótesis

Como primera apreciación, el impacto de todas las hipótesis es alto y todas afectan a la *disponibilidad* del sistema. En cuanto a la probabilidad de que ocurran las hipótesis, todas las acciones son realizadas desde la consola de comandos del equipo que sustenta la plataforma. Estas acciones pueden ser realizadas de forma casual o de forma malintencionada, por lo que le daremos a todas las hipótesis una probabilidad media.

5.3.4 Definición y desarrollo de la experimentación

En cuanto a la selección de participantes, seré el único, al igual que en el desarrollo de la experimentación en la funcionalidad simple. Los experimentos se realizarán durante dos días y la duración será indefinida. De esta forma, obtendremos las conclusiones más exactas posibles. Finalmente, la experimentación será con previo aviso desarrollada en producción y doy el visto bueno, ya que la plataforma está desplegada correctamente en un estado estable y el *servidor* y el *clúster* funcionan de forma adecuada en la *plataforma Edge*.

Como podemos apreciar, en el desarrollo de la experimentación, tanto de la funcionalidad básica como de la funcionalidad completa, se están teniendo en cuenta las estrategias utilizadas en las dos investigaciones analizadas anteriormente.

En la experimentación del *servidor*, se están utilizando *peticiones HTTP* para poder realizar el testeo, además de las comprobaciones en varios niveles (nivel interno del *servidor*, *peticiones HTTP* y *disponibilidad* del sistema), al igual que en la investigación

del *KTH Royal Institute of Technology* **[14]** . Además, se están teniendo en cuenta los pasos a seguir en el *"Risk Driven Fault Injection"* **[4]** de la investigación del *Hasso-Plattner-Institute for Digital Engineering* **[4]** (tanto en la funcionalidad simple como en la completa):

- **Ejecutamos la experimentación en base a acciones normales** modificadas con la intención de obtener estados anómalos.

- **Realizamos la monitorización en tiempo real** de los experimentos para ver **cómo se comporta el sistema en todos los niveles**.

- **Analizamos la información para identificar vulnerabilidades** y las partes implicadas.

- **Procesamos la información** para la mejora del sistema.

- **Analizamos el sistema de forma completa**, desde el funcionamiento hasta la arquitectura de este, mediante el uso del esquema (conocimiento base del sistema).

Todos los experimentos parten del escenario de que una entidad maliciosa se encuentra dentro de la consola de comando de la *plataforma Edge*. Los experimentos siguen una jerarquía de funcionalidades. Primero, nos centramos en los experimentos sobre la aplicación y su respectivo *contenedor*. Para ello, realizamos la prueba de desconectar el *contenedor* con la aplicación alojada y comprobaremos el funcionamiento de la plataforma. Una vez realizado el experimento, apreciamos que el *contenedor* se vuelve a reconectar de forma automática, buena solución para recuperarse del error.

El siguiente punto de la jerarquía es los experimentos sobre los *agente*s. Las acciones posibles a realizar sobre el *agente* son la eliminación y la desconexión, por lo que procedemos a realizar ambas acciones para comprobar los sistemas de seguridad que presentan la plataforma a la hora de realizarlas. Sobre la función de eliminación, no es posible realizarla bajo ningún procedimiento. En cambio, la función de desconexión se

puede realizar sin ninguna medida de protección, eliminando la coordinación y el control de la *plataforma Edge* sobre el *microservicio*. Como soluciones oportunas ante estas acciones encontramos el uso de credenciales, evitar la desconexión de *agente*s si quedan *contenedor*es huérfanos o establecer una jerarquía de desconexiones adecuada, en caso de que se quiera desconectar un *microservicio*, así evitamos el goteo de memoria que generan los *contenedor*es huérfanos.

El último punto de la jerarquía, y el más importante, es la experimentación sobre el *controlador*, nodo principal de la *plataforma Edge*. Al igual que en los *agente*s, se puede realizar tanto la desconexión como la eliminación. A diferencia de los *agente*s, en el *controlador*, se pueden realizar ambas acciones y su consecuencia es la misma, se pierde la conexión directamente con el *agente* y con el *contenedor* que tiene al *servidor*. El *servidor* sigue funcionando sin ningún tipo de monitorización ni control por parte de la plataforma. Como solución ante este tipo de problemas, se pueden sacar unas conclusiones muy similares a las obtenidas con los *agente*s: uso de credenciales, evitar *contenedor*es huérfanos o el sistema de recuperación si se pierde la conexión del *controlador*.

Finalmente, queda por comprobar dos acciones. La primera es la desconexión de la aplicación completa. Esta acción se puede realizar sin ningún tipo de medida de seguridad, por lo que sería necesario al menos un sistema de credenciales para poder confirmar que la persona que realiza la acción es legítima. La segunda acción es el conjunto de experimentos realizados sobre el *servidor* en local. Debemos aumentar el radio de confianza a producción, como buena práctica del *Chaos Engineering*, y comprobar si los resultados son los mismos a los obtenidos anteriormente. En este caso, es así y el sistema se comporta igual que en local, por lo que las conclusiones son bastante satisfactorias.

A continuación, se resumirá la experimentación en la siguiente tabla.

	Jerarquía	Acción	Resultado
1º	*Agente*	Desconexión del *agente* en la red *Edge*	Desaparecen todos los componentes de la red *Edge*, *contenedor* de aplicación sin control
2º	*Controlador*	Desconexión del *controlador* en la red *Edge*	Se pierde toda la conexión a la red *Edge*, *contenedor* de aplicación sin control
3º	*Contenedor* de la aplicación	Desconexión del *contenedor* de aplicación	*Servidor* invalidado por segundos, se recupera automáticamente
4º	Aplicación *Edge*	Parar la aplicación *Edge Computing*	La aplicación se detiene y no se pueden realizar peticiones
5º	*Controlador*	Eliminación del *controlador* en la red *Edge*	Se pierde la conexión con el *agente* y el *contenedor*, *contenedor* de aplicación sin control
6º	*Agente*	Eliminación del *agente* en la red *Edge*	Nos pide el uso de la acción "force", aún así no se permite la eliminación del *agente*

FIGURA 9: resumen experimentación completa.

5.4 Conclusiones del caso práctico

Tras la realización del caso práctico, he obtenido como conclusión los siguientes puntos:

- **Todo experimento**, por muy básico que parezca, **puede encontrar errores bastante perjudiciales para el sistema**. Por muy simple que parezcan los experimentos sobre el tratamiento de excepciones, si este no está bien implementado sobre el sistema, puede generar errores considerables (el 50% de los problemas de seguridad son por fallos humanos en el desarrollo del sistema).

- **La importancia de una estrategia bien definida es vital**. Todos los puntos deben estar bien explicados y no puede faltar ningún punto por realizar. En caso de que la estrategia no esté bien definida, aumenta considerablemente el tiempo de espera y dificulta la experimentación.

- **Los *rollbacks* deben ser lo suficientemente explícitos**. En consecuencia al punto anterior, los *rollbacks* deben estar perfectamente explicados y ser entendibles, ya que un proceso de *rollback* mal explicado puede prolongar la experimentación o incluso no poder volver a partir desde el mismo punto.

- **La observación multinivel agiliza mucho los procesos de mejora** y ayuda a obtener conclusiones más exactas. La capacidad de ver el resultado de las *peticiones HTTP*, unidas a la visión de la consola de comandos interna al *servidor*, hace que se vean de forma mucho más acentuada donde se encuentran los errores y si las peticiones están siguiendo su curso normal de funcionamiento.

- **La experimentación realizada** sobre la *plataforma Edge Computing* no tiene **prácticamente ninguna medida de seguridad que controle las acciones realizadas sobre esta**. Una vez accedes a la consola de comandos, la gran mayoría de las acciones se pueden realizar sin ningún tipo de medida de seguridad.

- **El *Chaos Engineering* es una herramienta con muchísimo potencial**, pero a su vez muy peligrosa en caso de que esta no se utilice correctamente. Esta disciplina encuentra errores poco comunes y ayuda a los sistemas a aguantar condiciones extremas. Sin embargo, la forma de obtención de estos errores es bastante peligrosa si no se aborda de forma adecuada, ya que sitúa al sistema en sus límites. Es por tanto aconsejable realizar aquellos experimentos que puedan modificar el estado del sistema en entornos controlados que reflejen el estado del sistema en producción.

6

Conclusiones y líneas futuras

En esta última sección, se recogerán las conclusiones que se han alcanzado a lo largo de la investigación y el caso práctico, además de algunas líneas futuras de investigación.

Como se ha ido comentando a lo largo de toda la memoria, ambos conceptos, tanto el *Chaos Engineering* como el *Edge Computing*, son dos áreas muy novedosas y con un potencial enorme. Sin embargo, este potencial está comenzando a ser algo tangible actualmente, por lo que tiene una gran capacidad de mejora y utilidad. Las grandes empresas están siendo las primeras en usarlas y en conseguir beneficios de estas. No obstante, estas herramientas están a mano de todo el mundo y pueden ser escalables a tamaños menores, como hemos podido apreciar en el caso práctico. Las plataformas *Open Source* dan la posibilidad a toda persona de usar las *tecnologías Edge* para diferentes tipos de utilidades.

En este proyecto, hemos querido demostrar el potencial que tienen ambos conceptos. El primero de ellos es el *Chaos Engineering*, con sus estrategias y la capacidad de

abstracción para todo tipo de plataformas y áreas que queramos evaluar. Esta disciplina actualmente está siendo utilizada principalmente para la evaluación de la *resiliencia* y *disponibilidad* de los sistemas. Sin embargo, se le puede dar un enfoque distinto, dependiendo del área a evaluar, en nuestro caso es la seguridad de los sistemas.

Utilizo este último punto para indicar una posible área de investigación futura, la capacidad de generar estrategias *Chaos Engineering* más dedicadas a las diferentes áreas, en este caso la seguridad informática. Las estrategias *Chaos Engineering* actuales son bastante precisas y detalladas. Sin embargo, se podrían formalizar ciertos cambios sobre esta, por ejemplo: clasificación de los ataques a partir de la *Matriz de Mitre* **[30]** o la formalización de los experimentos mediante el uso de escenarios de ataque (un mismo error se puede forzar desde diferentes escenarios), entre otros.

Otra posible área de investigación, aunque de mayor complejidad, sería la generación de una herramienta de orquestación de experimentos de seguridad, desde *ataques DDoS* hasta comprobación de las *peticiones HTTP*. De esta forma, podremos comprobar de forma automática y frecuente el funcionamiento del sistema. Actualmente, las herramientas de seguridad disponibles no se encuentran al 100% de madurez y hay muchas áreas mejorables sobre estas.

El segundo punto es el *Edge Computing*. La intención principal de la investigación es demostrar el potencial de estas plataformas, desde los conceptos principales hasta los desafíos que estas generan. Como se ha podido apreciar, el *Edge Computing* tiene grandes aptitudes en desarrollo. Sin embargo, como hemos analizado en este proyecto, los desafíos que generan las plataformas son de gran complejidad. Es en este punto donde se ha querido demostrar que el *Chaos Engineering* es una herramienta muy adecuada para la búsqueda de estos desafíos, como se ha podido apreciar en el caso práctico.

En cuanto a líneas futuras de investigación sobre el *Edge Computing* hay varias posibles:

- **Realizar búsquedas exhaustivas de desafíos y problemas de seguridad en las diferentes plataformas *Edge*.** Las plataformas *Edge* tienen muchas áreas de búsqueda; desde altos niveles, como ha sido el caso práctico, a niveles menores, como pueden ser los protocolos de comunicación entre nodos o el cifrado de la información que circula entre nodos. Esta área de investigación es muy general y ha sido abordada en este proyecto, sin embargo, son muchos los desafíos aún por descubrir en el *Edge Computing*, por lo que he decidido que tenga un lugar en la memoria.

- **Búsquedas de estructuras óptimas para las plataformas *Edge*.** El número de nodos a proteger ha aumentado considerablemente con el uso del *Edge Computing*. Es por esto por lo que se deberán buscar estructuras optimizadas, tanto para el aumento de la eficiencia de la plataforma como para la seguridad de estas.

- **Búsquedas de arquitecturas óptimas de los nodos *Edge*.** Al igual que es importante la estructura de la plataforma, también es importante una arquitectura individual de cada uno de los nodos que conforman la plataforma. La seguridad debe escalarse desde los puntos menores, los nodos, a los puntos más altos de la jerarquía, la estructura general de la plataforma.

Finalmente, como conclusión de lo aprendido a lo largo de este proyecto, hay varios puntos importantes a tener en cuenta en relación a lo estudiado en el grado. El primer punto es el aprendizaje de una nueva estrategia de desarrollo de software, el *Chaos Engineering*. A lo largo de la carrera, se aprenden múltiples estrategias de desarrollo; sin embargo, el *Chaos Engineering* rompe con todas las ideologías de desarrollo anteriores. El mero hecho de intentar forzar y romper lo desarrollado para seguir mejorando el software es lo contrario a lo aprendido en estos cuatro años de carrera. Tras realizar un análisis en profundidad de la disciplina y realizar un caso práctico, apreciamos el

potencial y la gran mejora que aporta un cambio en la ideología normal de desarrollo del software.

Como segundo punto, tenemos el aprendizaje de las tecnologías *Edge Computing*. Estas tecnologías tienen un potencial enorme para el futuro; no obstante, en el grado, el *Edge Computing* no se desarrolla en ninguna asignatura, el *Cloud Computing* sí. La capacidad de haber analizado y estudiado esta nueva tecnología, me ha aportado un conocimiento bastante significativo para el desarrollo profesional como ingeniero de software, debido a sus futuras utilidades en el ámbito empresarial.

Por último, tenemos el desarrollo de un *servidor* web, además de todos los puntos académicos que se utilizan en la experimentación. Ejemplos de estos son: el estudio de las *peticiones HTTP*, el análisis de bases de datos *NoSQL* o la utilización de la consola de comandos para la ejecución de determinadas instrucciones. Todos estos conceptos han sido estudiados de forma académica en cuarto curso y han sido utilizados satisfactoriamente en el desarrollo del proyecto.

Referencias

[1] *Chaos Engineering* as a Service. https://techcrunch.com/2020/12/15/aws-introduces-new-chaos-engineering-as-a-service-offering/

[2] Miles, R. (2019). *Learning Chaos Engineering*: Discovering and Overcoming System Weaknesses Through Experimentation (1). Sebastopol: O'Reilly Media, Incorporated

[3] *Chaos Engineering* en empresas https://www.bbvaapimarket.com/es/mundo-api/chaos-monkey-la-herramienta-que-provoca-pequenos-fallos-para-evitar-otros-mayores/

[4] CloudStrike: *Chaos Engineering* for Security and Resiliency in Cloud Infrastructure Hasso-Plattner-Institute for Digital Engineering.

[5] Perry, L. (2020). *IoT and Edge Computing for architects*: implementing *Edge* and IoT systems from sensors to clouds with communication systems, analytics and security (Second Edition) Birmingham, England ; Mumbai : Packt

[6] VII Jornada de Ciberseguridad en Andalucía. WORKSHOP TÉCNICO: Ponencia Rodrigo Román. https://www.youtube.com/watch?v=9NxH2yBtvl8&t=89s

[7] SCRUM https://www.scrum.org/resources/blog/que-es-scrum

[8] AWS Fault Injection https://www.infoq.com/news/2020/12/aws-fault-injection/

[9] The Simian Army https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116

[10] Herramientas *Chaos Engineering*: Tornasol https://geekflare.com/es/chaos-engineering-tools/#anchor-litmus

[11] ChaosSinglr https://github.com/Optum/ChaoSlingr

[12] Aplicaciones del *Edge Computing* https://www.iberdrola.com/innovacion/que-es-Edge-computing

[13] *IoFog* https://*IoFog*.org/

[14] Observability and *Chaos Engineering* on system calls for containerized applications in *Docker* KTH Royal Institute of Technology, https://www.sciencedirect.com/science/article/pii/S0167739X21001163

[15] Eclipse Foundation https://www.eclipse.org/org/foundation/

[16] *Node.js* https://nodejs.org/es/docs/

[17] *MongoDB Atlas* https://www.mongodb.com/cloud/atlas/lp/try2?utm_source=google&utm_campaign=gs_emea_spain_search_core_brand_atlas_desktop&utm_term=mongodb&utm_medium=cpc_paid_search&utm_ad=e&utm_ad_campaign_id=12212624563&gclid=CjwKCAjw2ZaGBhBoEiwA8pfP_kqJ9FjKLCBBZPBOZydGYg6JuhxjdevkuDUHCz75opMCYWv-7ucQKhoCoqYQAvD_BwE

[18] Visual Studio Code https://code.visualstudio.com/

[19] *Docker* https://www.*Docker*.com/

[20] *Postm*an https://www.*Postman*.com/

[21] Akraino *Edge* Stack https://www.lf*Edge*.org/projects/akraino/

[22] *Baetyl* https://www.lf*Edge*.org/projects/*Baetyl*/

[23] Fl*Edge* https://www.lf*Edge*.org/projects/fl*Edge*/

[24] *Edge* X Foundry https://www.*Edge*xfoundry.org/

[25] *KubeEdge* https://*KubeEdge*.io/en/

[26] Open Horizon https://www.lf*Edge*.org/projects/openhorizon/

[27] Home *Edge* Project https://www.lf*Edge*.org/projects/home*Edge*/

[28] Project EVE https://www.lf*Edge*.org/projects/eve/

[29] Linux Foundation https://www.linuxfoundation.org/

[30] Matriz de Mitre https://attack.mitre.org/

Apéndice A

Manual de instalación

Para la instalación de la funcionalidad básica, necesitamos solo el programa *Node.js*. Para ello, debemos ir a la página oficial (https://nodejs.org/es/) y realizar la descarga del archivo. Una vez realizada, debemos ejecutar el archivo e instalar el entorno de programación. Teniendo instalado el entorno, debemos introducirlo en las variables de entorno de Windows. De esta forma, podremos llamar al comando "node" desde cualquier directorio en la consola de comandos de Windows.

Finalmente, cuando tenemos realizados todos estos pasos, debemos introducirnos dentro del directorio del proyecto en la consola de comandos y ejecutar el comando "node index.js". De esta forma, conseguimos desplegar el *servidor* de forma local para su uso, este se conecta de forma inmediata a la *base de datos* online siempre que el ordenador tenga conexión a internet.

INSTALACION EN LINUX DE LA *PLATAFORMA EDGE*

Para la instalación del *servidor Node.js* sobre la *plataforma Edge*, necesitaremos los archivos de la propia plataforma. Para ello, se irá a la página (https://*Iofog*.org/) y seguiremos los comandos aportados. Primero descargaremos los archivos:

```
Curl
https://packagecloud.io/install/repositories/iofog/iofogctl/script.deb
.sh | sudo bash

sudo apt-get install iofogctl=2.0.4
```

Y seguido a esto, generamos los componentes fundamentales para el desarrollo mediante el quick - start que aporta la propia plataforma:

```
echo "---

apiVersion: iofog.org/v2

kind: LocalControlPlane

metadata:

  name: ecn

spec:

  iofogUser:

    name: Quick
```

```
    surname: Start

    email: user@domain.com

    password: q1u45ic9kst563art

  controller:

    container:

      image: iofog/controller:2.0.1

---

apiVersion: iofog.org/v2

kind: LocalAgent

metadata:

  name: local-agent

spec:

  container:

    image: iofog/agent:2.0.4

" > /tmp/quick-start.yaml

iofogctl deploy -f /tmp/quick-start.yaml
```

Una vez ejecutado, podemos apreciar cómo se han generado las bases de una plataforma *Edge Computing*, obtenemos algo similar a esto:

```
alejandrocl99@acl-ubuntu:/tmp$ cd ..
alejandrocl99@acl-ubuntu:/$ sudo iofogctl deploy -f /tmp/quick-start.yaml
✓ Successfully deployed resources
alejandrocl99@acl-ubuntu:/$ sudo iofogctl get all
NAMESPACE
default

CONTROLLER      STATUS          AGE             UPTIME          VERSION         ADDR            PORT
local           online          52s             55s             2.0.1           0.0.0.0         51121

AGENT           STATUS          AGE             UPTIME          VERSION         ADDR
local-agent     RUNNING         51s             33s             2.0.4           localhost

APPLICATION     RUNNING         MICROSERVICES

MICROSERVICE    STATUS          AGENT           VOLUMES         PORTS

VOLUME          SOURCE          DESTINATION     PERMISSIONS     AGENTS

ROUTE           SOURCE MSVC     DEST MSVC
```

FIGURA 10: bases de la *plataforma Edge Computing*.

Y estos *contenedor*es:

```
alejandrocl99@acl-ubuntu:/$ sudo docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS              PORTS
                NAMES
bc2d39b2ea2c   iofog/router:latest     "/qpid-dispatch/rout…"   29 seconds ago       Up 28 seconds       0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 0.0.0
56721-56722/tcp   iofog_CPYf6BzWDXCN68FX3CtpwWn9rHdnKcLb
e8b590f50d7c   iofog/agent:2.0.4      "sh /start.sh"           54 seconds ago       Up 54 seconds
                iofog-agent
8f7479b69c4e   iofog/controller:2.0.1   "node /usr/local/lib…"   About a minute ago   Up About a minute   0.0.0.0:51121->51121/tcp, 0.0.0.0:8008->80/tcp
                iofog-controller
```

FIGURA 11: *contenedor*es desplegados.

Una vez tenemos las bases, nos centramos en el desarrollo del *microservicio* en la plataforma. Para ello, debemos modificar el config.yaml e introducir nuestro nuevo *microservicio* (introducción de puertos, la imagen, etc.) y se ha tenido que crear antes el *Docker*file de nuestro *servidor Node.js* y generar su respectiva imagen para que el config.yaml la tenga disponible.

Las imágenes disponibles son las siguientes:

```
alejandrocl99@acl-ubuntu:/$ sudo docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
iofog/tfg           v1        d7b056e41a7a   47 hours ago     917MB
iofog/agent         2.0.4     095c6588135f   2 months ago     969MB
iofog/controller    2.0.1     f65d228e2ab6   6 months ago     337MB
iofog/router        latest    14b5a9f3d6b0   7 months ago     349MB
node                8         8eeadf3757f4   15 months ago    901MB
```

FIGURA 12: imágenes generadas para el desarrollo de la *plataforma Edge*.

Y el config.yaml modificado es el siguiente:

```yaml
apiVersion: iofog.org/v2
kind: Application
metadata:
  name: tfg
spec:
  microservices:
    - name: nodejs-server
      agent:
        name: local-agent
      config: {}
      images:
        x86: iofog/tfg:v1
        registry: remote
      container:
        volumes: []
        ports:
          - internal: 8000
            external: 10101
        env: []
  routes: []

```

FIGURA 13: config.yaml modificado para el despliegue del servicio.

En él, modificamos el nombre que le damos al *microservicio*, el *agente* responsable del *microservicio*, la imagen necesaria para establecer el *microservicio*, si es remoto, y la unión de puertos (el puerto 8000 es el que se usa de forma interna en el *servidor* y el 10101 es el que usa el *microservicio*).

Finalmente, ejecutamos el siguiente comando desde la carpeta donde se encuentra el config.yaml:

```
Sudo iofogctl deploy -f config.yaml
```

Y se generará el *microservicio* en la *plataforma Edge*. Lo podremos comprobar con el comando:

```
sudo iofogctl GET microservices
```

```
sudo iofogctl GET all
```

```
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl deploy -f config.yaml
✔ Successfully deployed resources
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl get microservices
NAMESPACE
default

MICROSERVICE    STATUS          AGENT           VOLUMES         PORTS
nodejs-server   RUNNING         local-agent                     10101:8000

alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl get all
NAMESPACE
default

CONTROLLER      STATUS          AGE             UPTIME          VERSION         ADDR            PORT
local           online          5m43s           5m45s           2.0.1           0.0.0.0         51121

AGENT           STATUS          AGE             UPTIME          VERSION         ADDR
local-agent     RUNNING         5m41s           5m28s           2.0.4           localhost

APPLICATION     RUNNING         MICROSERVICES
tfg             1/1             nodejs-server

MICROSERVICE    STATUS          AGENT           VOLUMES         PORTS
nodejs-server   RUNNING         local-agent                     10101:8000

VOLUME          SOURCE          DESTINATION     PERMISSIONS     AGENTS

ROUTE           SOURCE MSVC     DEST MSVC
```

FIGURA 14: despliegue completo de la *plataforma Edge* y el *microservicio*.

Apéndice B

Desarrollo de experimentos en la funcionalidad básica

EXPERIMENTO 1

Definición:

Desconexión de la *base de datos*.

Hipótesis:

El *servidor* intentará la petición y tras el *"timeout"* enviará una respuesta de error.

Zonas afectadas:

BBDD y el *servidor*.

Estado estable:

El *servidor* está operativo a las peticiones en todo momento.

Esquema de funcionamiento:

- 1.- Se entrará en el *clúster* de *MongoDB Atlas* y se desconectará temporalmente la *base de datos*. Eliminaremos las conexiones a la *BD*.

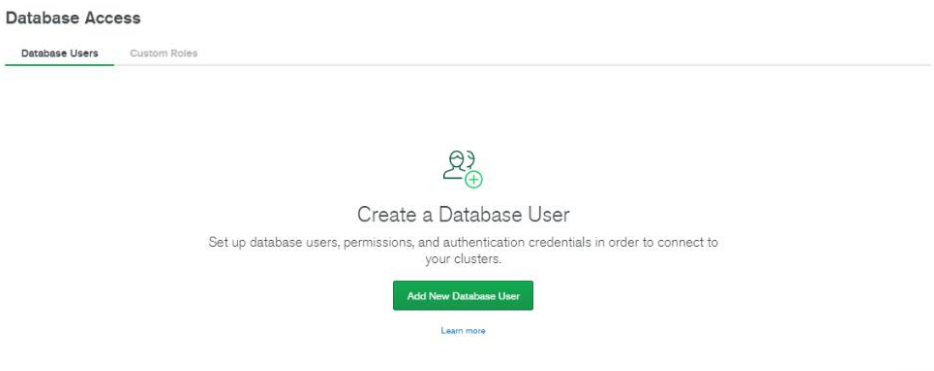

FIGURA 15: interfaz de accesos a la *base de datos MongoDB Atlas*.

❖ 2.-Desconectamos todas las IPs que se puedan conectar.

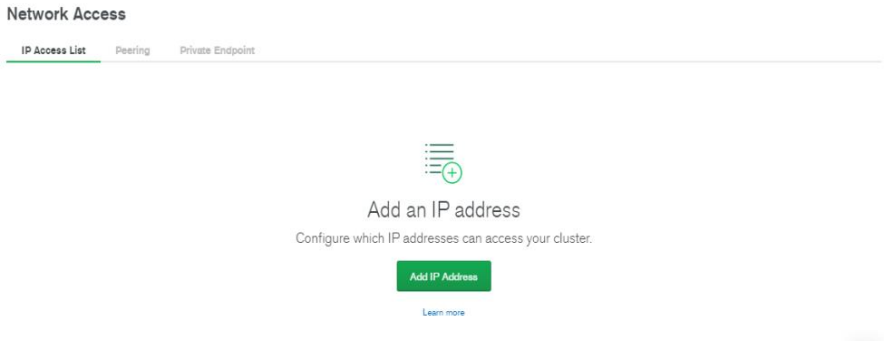

FIGURA 16: interfaz de conexiones a la *base de datos MongoDB Atlas*.

❖ 3.-Realizamos una petición *GET* a la url http://localhost:8000/api/usuarios/.

***Rollback*:**

Conexión nuevamente con la *base de datos* y reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Obtenemos un error 500 (internal server error) del *servidor*:

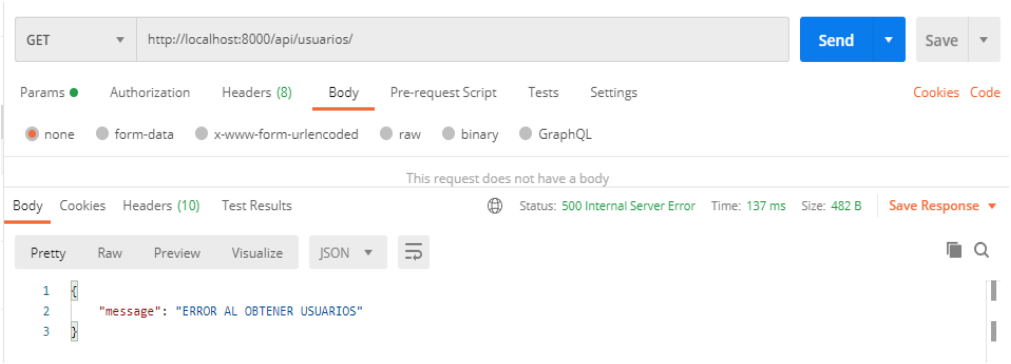

FIGURA 17: respuesta del *servidor* a nivel de *petición HTTP* (*Postman*).

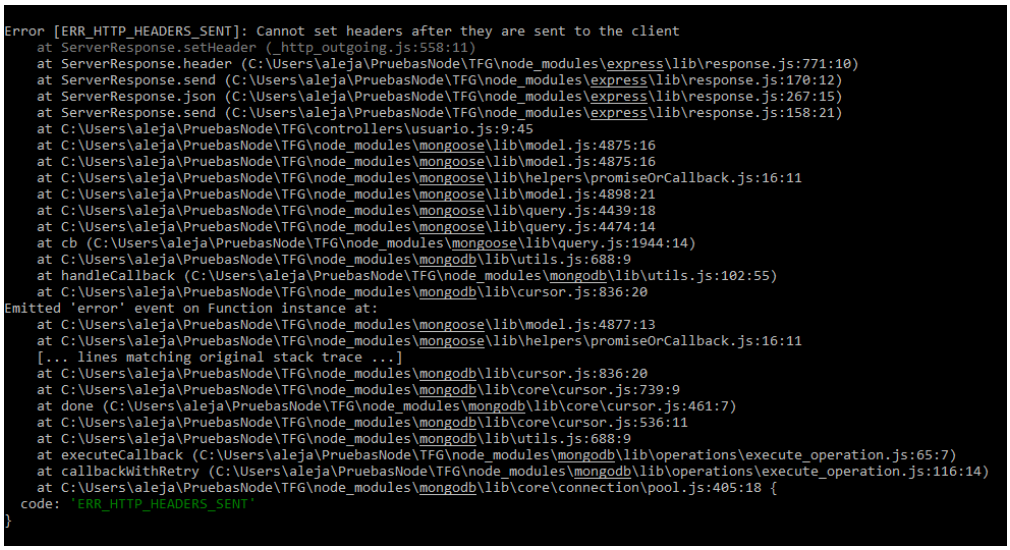

FIGURA 18: respuesta del *servidor* de forma interna.

El fallo se ha solucionado modificando las excepciones de error. En caso de que el *servidor* reciba el fallo, debe realizar una devolución del error y no seguir leyendo el código de esa función. A continuación, tras solucionar el fallo, no cae el *servidor*:

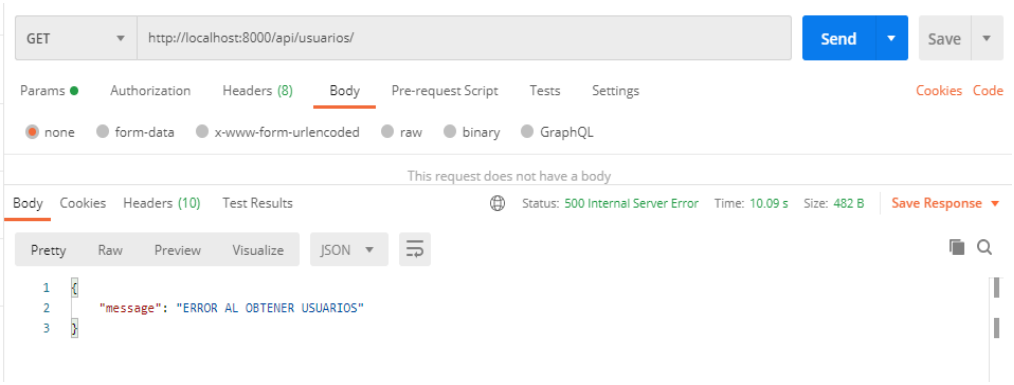

FIGURA 19: respuesta del *servidor* a la *petición HTTP* (*Postman*).

El *servidor* sigue a la espera de peticiones:

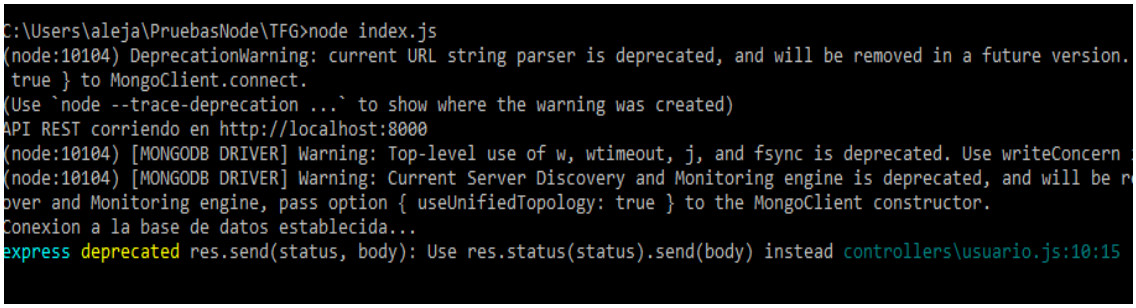

FIGURA 20: respuesta de la petición de forma interna al *servidor*.

EXPERIMENTO 2

Definición:

Cambio de permisos en el acceso a la *base de datos*.

Hipótesis:

El *servidor* intentará la petición y tras el *"timeout"* enviará una respuesta de error.

Zonas afectadas:

BBDD y el *servidor*.

Estado estable:

El *servidor* está operativo a las peticiones en todo momento y se pueden realizar todo tipo de operaciones, incluidas las de escritura.

Esquema de funcionamiento:

- 1.- Se entrará en el *clúster* de *MongoDB Atlas* y se cambiarán los permisos de la *base de datos* (pondremos que el administrador solo pueda leer las *BD*).
- 2.- Realizaremos un *POST* a la url http://localhost:8000/api/usuarios/ con el *JSON*:

```
{
        "username": "admin",
        "pwd": "admin",
        "email": "admin@gmail.com"
}
```

***Rollback*:**

Conexión nuevamente con la *base de datos* y reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

El *servidor* nos devuelve un error 500:

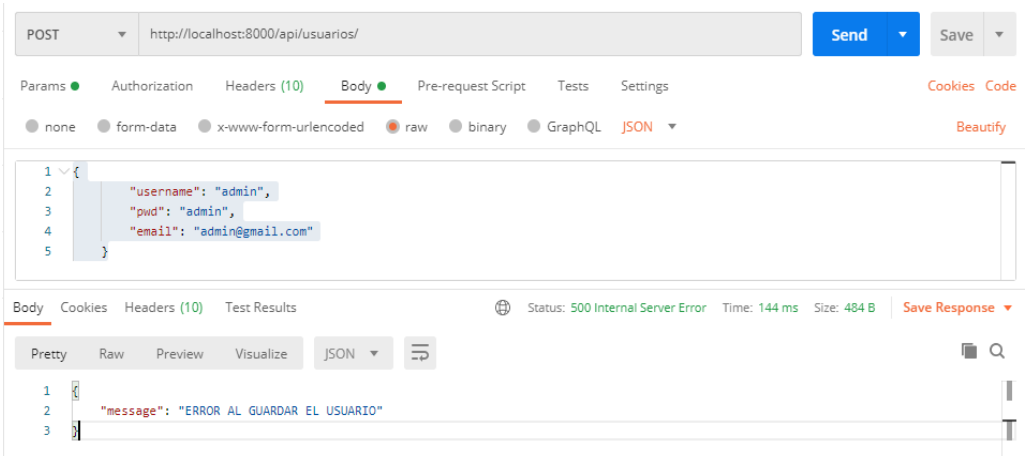

FIGURA 21: respuesta del *servidor* a la *petición HTTP* (*Postman*).

Pero el *servidor* no cae y sigue a la espera de nuevas peticiones:

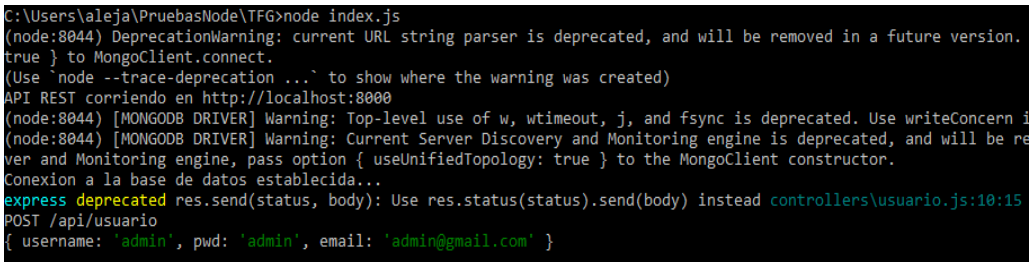

FIGURA 22: respuesta del *servidor* de forma interna.

Tras devolverle los permisos al administrador, todo funciona correctamente:

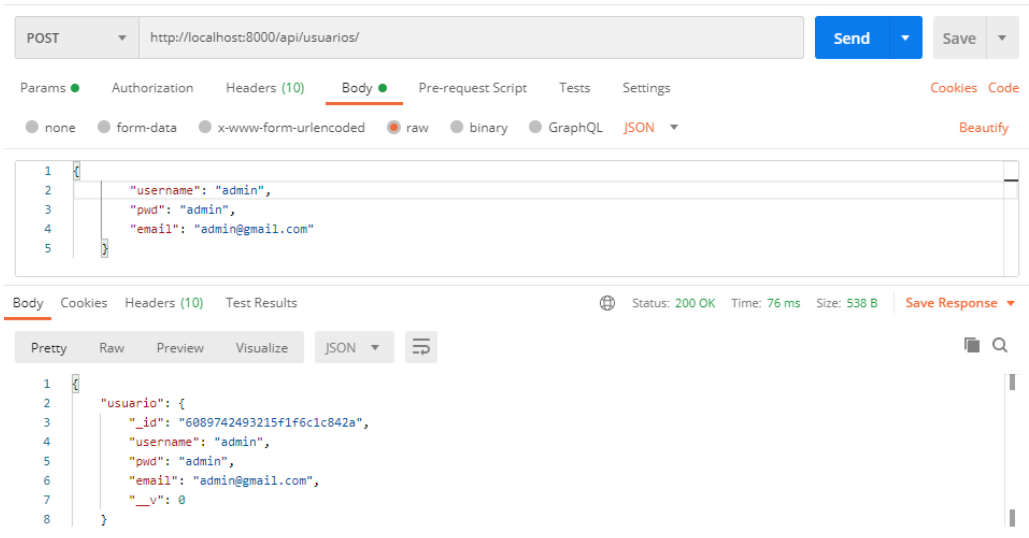

FIGURA 23: respuesta correcta del *servidor* a la *petición HTTP* (*Postman*).

EXPERIMENTO 3

Definición:

Petición a un *"endpoint"* no recogido.

Hipótesis:

El *servidor* intentará la petición y obtendremos un error 404 de *"endpoint"* no contemplado.

Zonas afectadas:

Servidor.

Estado estable:

El *servidor* debe estar operativo en todo momento para la resolución de las peticiones.

Esquema de funcionamiento:

- 1.- En *Postman* realizaremos una petición *GET* a la dirección http://localhost:8000/usuarios.

***Rollback*:**

Reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Se cumple la hipótesis y el *servidor* sigue en pie:

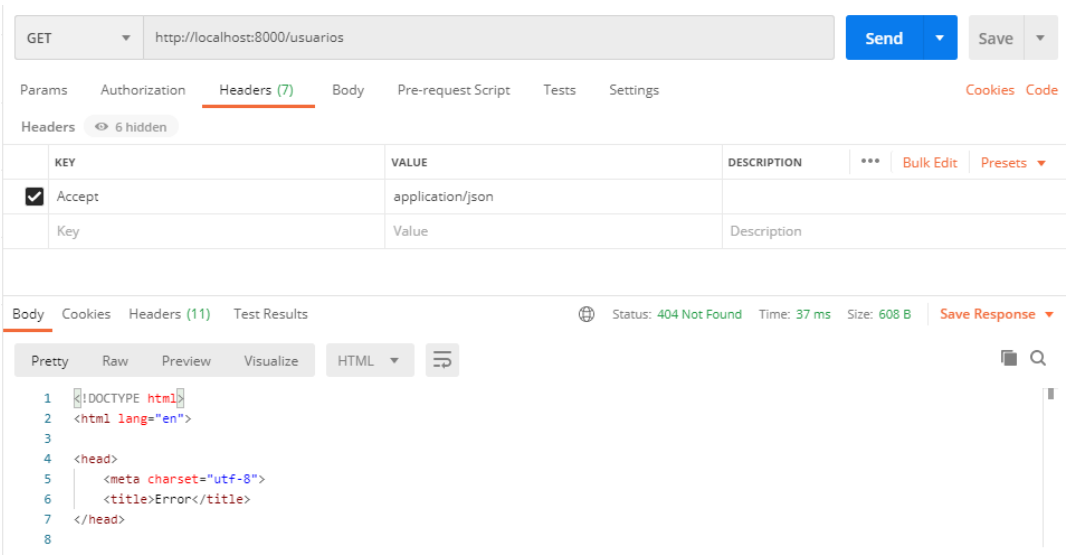

FIGURA 24: repuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 4

Definición:

Petición a un *"endpoint"* recogido pero con atributos erróneos.

Hipótesis:

El *servidor* dará un error 500.

Zonas afectadas:

B*BD*D y el *servidor*.

Estado estable:

El *servidor* debe estar operativo en todo momento para la resolución de las peticiones.

Esquema de funcionamiento:

- 1.- En *Postman*, realizaremos una petición DELETE a la dirección http://localhost:8000/api/usuarios/juan

***Rollback*:**

Conexión nuevamente con la *base de datos* y reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Se cumple la hipótesis y el *servidor* devuelve el error 500:

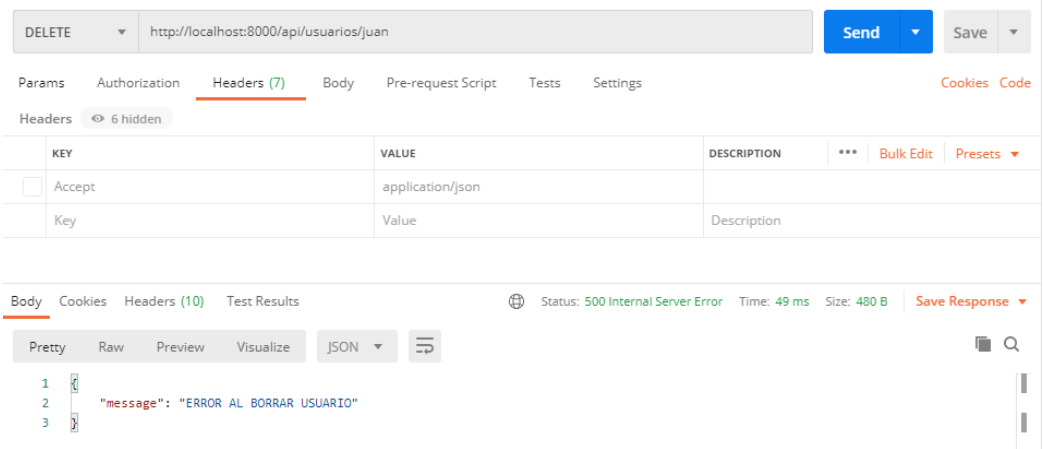

FIGURA 25: respuesta a la *petición HTTP* (*Postman*).

El *servidor* sigue funcionando correctamente.

EXPERIMENTO 5

Definición:

Petición a un *"endpoint"* recogido pero con más atributos de los necesarios.

Hipótesis:

El *servidor* dará un error 404.

Zonas afectadas:

Servidor.

Estado estable:

El *servidor* debe estar operativo en todo momento para la resolución de las peticiones.

Esquema de funcionamiento:

- 1.- En *Postman*, realizaremos una petición *GET* a la dirección http://localhost:8000/api/usuarios/juan/lopez .

***Rollback*:**

Conexión nuevamente con la *base de datos* y reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

La hipótesis se cumple y el *servidor* sigue en funcionamiento:

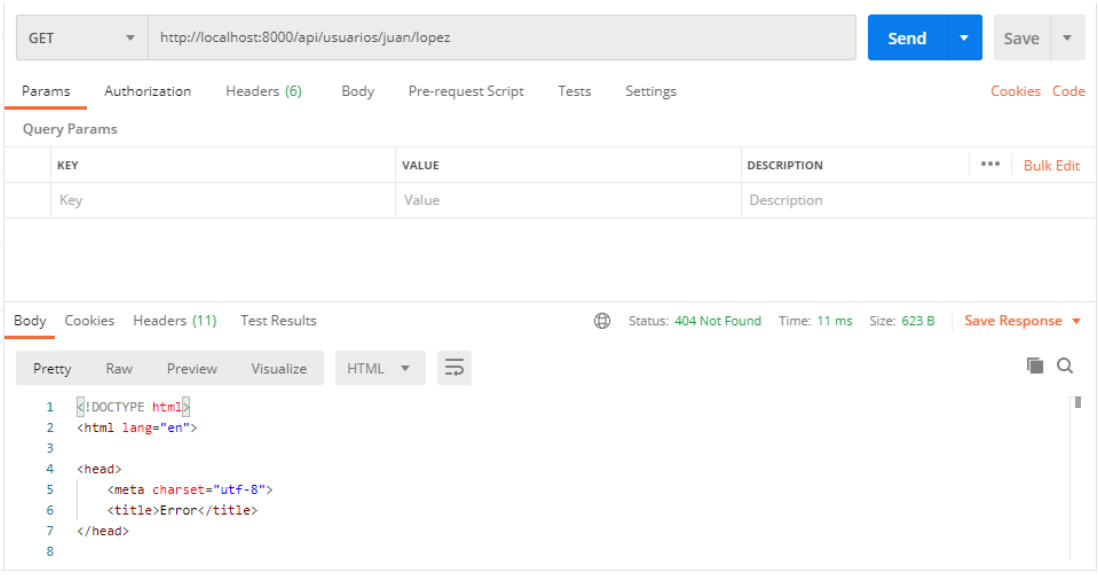

FIGURA 26: respuesta a la *petición HTTP* (*Postman*).

EXPERIMENTO 6

Definición:

Petición a un *"endpoint"* recogido pero sin atributos.

Hipótesis:

El *servidor* dará un error 404.

Zonas afectadas:

Servidor.

Estado estable:

El *servidor* debe estar disponible para responder a todas las peticiones.

Esquema de funcionamiento:

- 1.- En *Postman*, realizaremos una petición DELETE a la dirección http://localhost:8000/api/usuarios .

***Rollback*:**

Conexión nuevamente con la *base de datos* y reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

La hipótesis se cumple y el *servidor* sigue en funcionamiento:

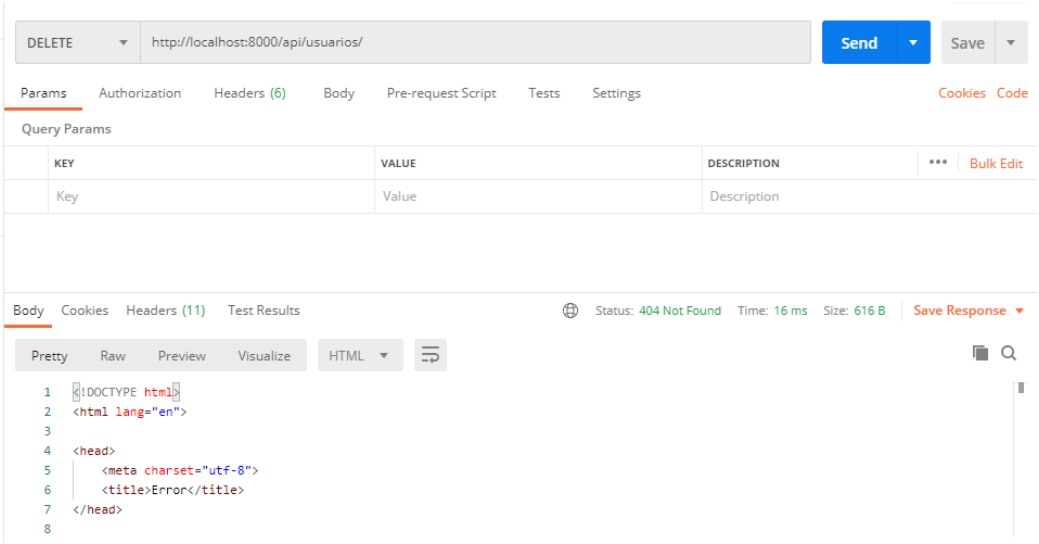

FIGURA 27: respuesta a la *petición HTTP* (*Postman*).

EXPERIMENTO 7

Definición:

Falta de credenciales.

Hipótesis:

El sistema no tiene recogido un sistema de credenciales, por lo que podrá hacer todos los cambios que quiera.

Zonas afectadas:

Servidor y *BBDD*.

Estado estable:

El sistema deberá permitir realizar acciones sobre la *base de datos*, en caso de que el usuario sea legítimo y esté identificado.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada realiza cambios en la *BD*.
- 2.- En *Postman*, realizamos un *PUT* a la dirección http://localhost:8000/api/usuario/x y modificaremos la *BD*.

***Rollback*:**

Reseteo de los datos modificados.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

La entidad no autorizada puede realizar peticiones sin problema:

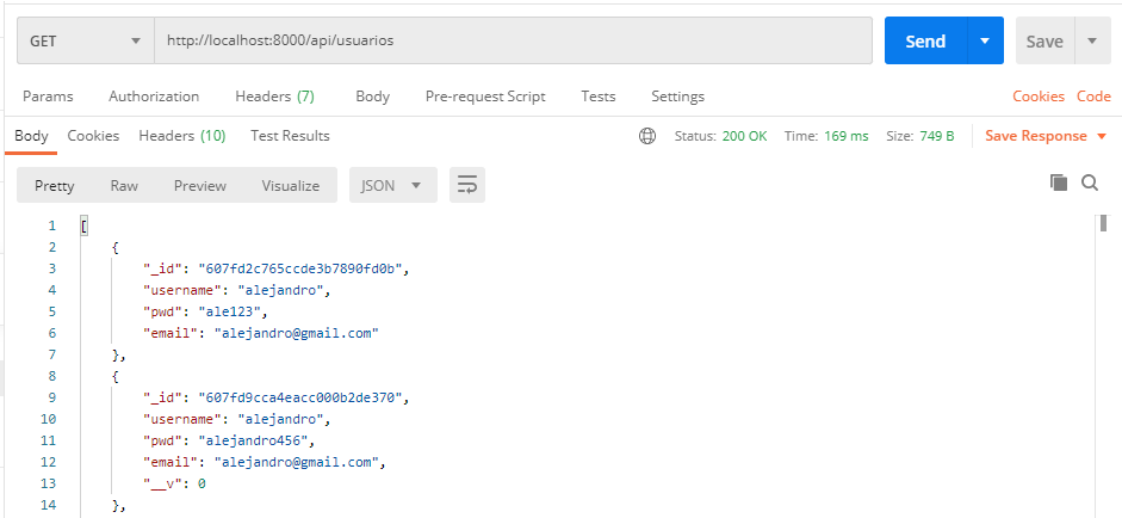

FIGURA 28: respuesta a la *petición HTTP* (*Postman*).

Como solución a la falta de credenciales, hemos introducido un parámetro de identificación. Se debe introducir la contraseña del usuario para poder realizar las peticiones:

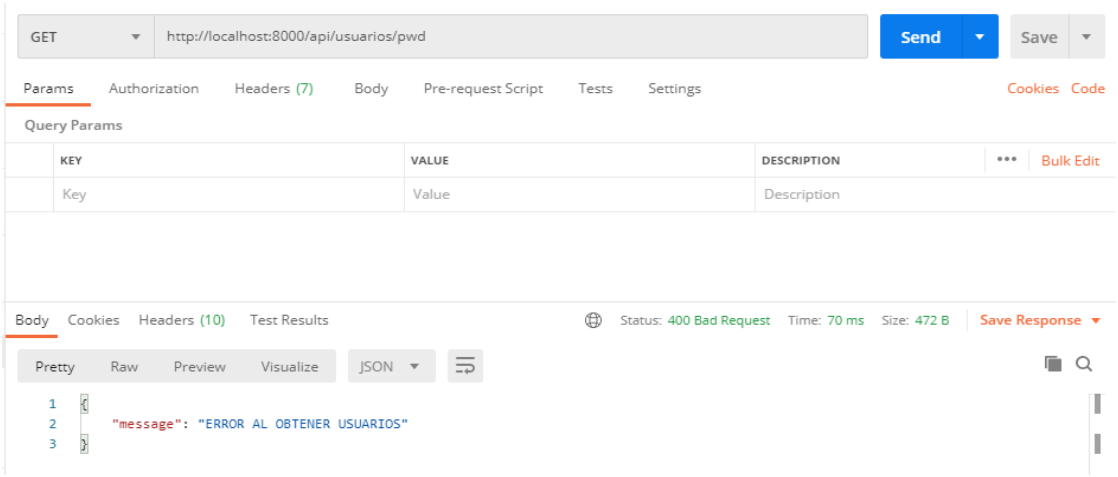

FIGURA 29: respuesta a la *petición HTTP* (*Postman*).

Ahora con las credenciales:

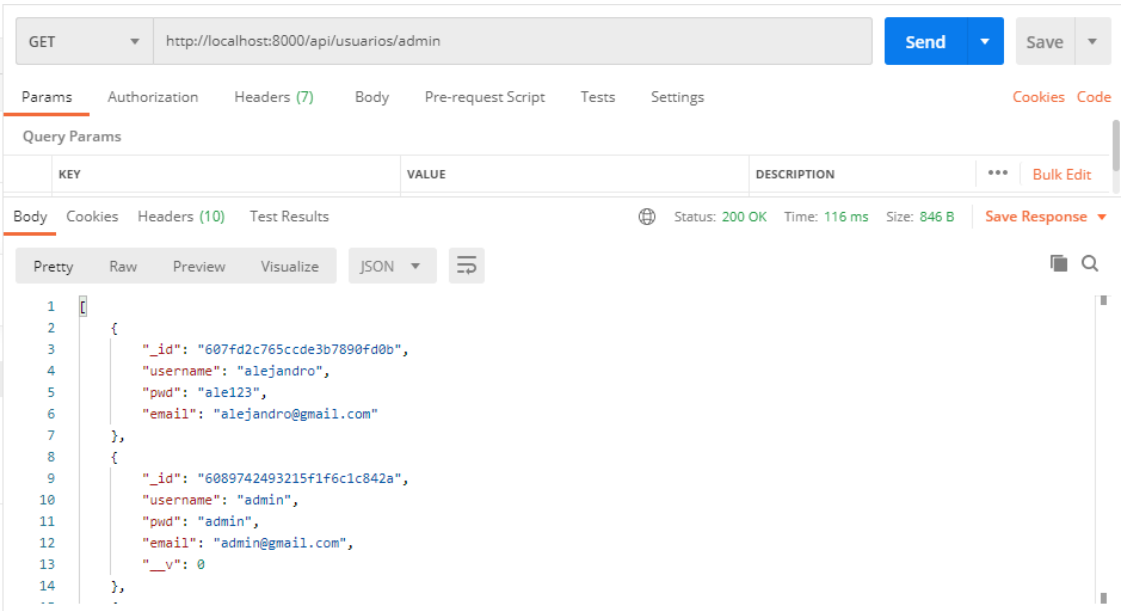

FIGURA 30: respuesta a la *petición HTTP* (*Postman*).

EXPERIMENTO 8

Definición:

Repudio del *servidor* (una entidad autorizada realiza cambios ilegítimos).

Hipótesis:

El sistema no tiene recogido un sistema de logs, por lo que no se puede cuestionar el repudio a una entidad determinada.

Zonas afectadas:

Servidor y *BBDD*.

Estado estable:

El *servidor* debe tener constancia de todas las acciones realizadas sobre él y el usuario que las ha realizado.

Esquema de funcionamiento:

- 1.- Una entidad autorizada realiza cambios en la *BD*.
- 2.- En *Postman*, realizamos un *DELETE* a la dirección http://localhost :8000/api/usuario/x y modificaremos la *BD*.

***Rollback*:**

Reseteo de los datos modificados.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Se cumple la hipótesis y no quedan recogidos los cambios, por lo que he introducido una colección de logs donde se guardan en la *BD* con el usuario que los ha realizado, la acción y la hora a la que los ha realizado:

FIGURA 31: logs del sistema en la *BD MongoDB Atlas*.

EXPERIMENTO 9

Definición:

Petición con cabeceras modificadas, uso del accept type / *XML*.

Hipótesis:

El *servidor* lanzará un error 400.

Zonas afectadas:

Servidor.

Estado estable:

El sistema deberá continuar estando operativo para seguir tratando las peticiones.

Esquema de funcionamiento:

- 1.- En *Postman*, realizamos un *GET* a la dirección http://localhost :8000/api/usuario/x y añadiremos en la cabecera accept type / *XML*.

***Rollback*:**

Reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

El *servidor* lanza un 200 pero con el resultado en JSON en vez de en *XML*.

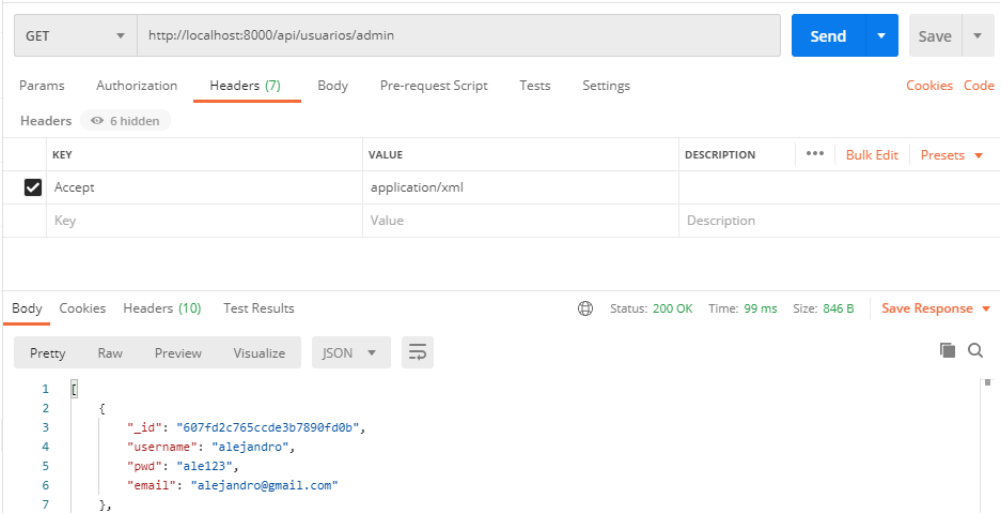

FIGURA 32: respuesta a la *petición HTTP* (*Postman*).

El *servidor* sigue funcionando correctamente.

EXPERIMENTO 10

Definición:

Petición con cabeceras modificadas, uso del content type / *XML*.

Hipótesis:

El *servidor* lanzará un error 400.

Zonas afectadas:

Servidor.

Estado estable:

El sistema deberá continuar estando operativo para seguir tratando las peticiones.

Esquema de funcionamiento:

- 1.- En *Postman*, realizamos un *POST* a la dirección http://localhost :8000/api/usuario/x y añadiremos en la cabecera content type / *XML*.

***Rollback*:**

Reseteo del *servidor*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

El *servidor* lanza un 200 pero no se guarda correctamente el usuario.

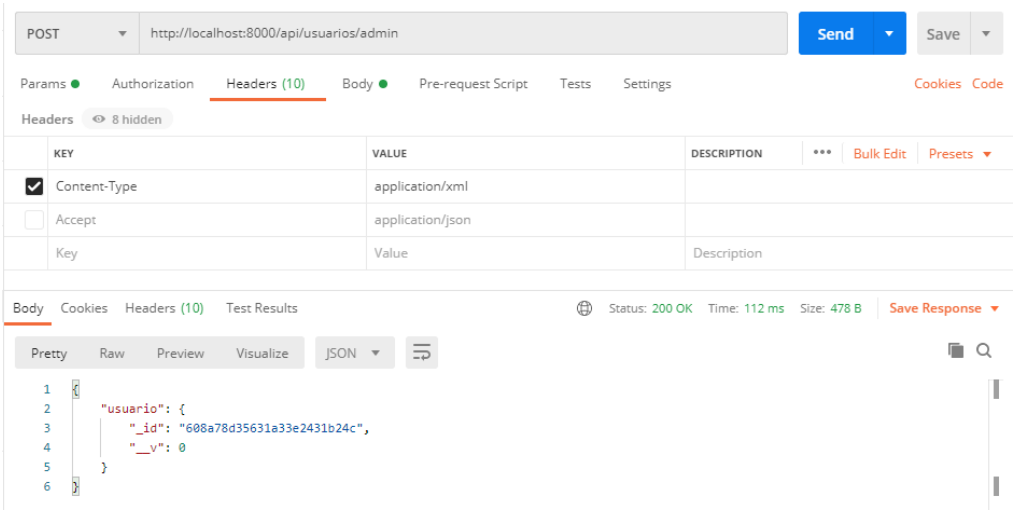

FIGURA 33: respuesta a la *petición HTTP* (*Postman*).

El *servidor* sigue funcionando correctamente.

EXPERIMENTO 11

Definición:

Entrada y robo de credenciales en la *BD*.

Hipótesis:

El *servidor* no tiene encriptadas las contraseñas, por lo que podrá utilizar todas las credenciales robadas.

Zonas afectadas:

Servidor y *BBDD*.

Estado estable:

Las credenciales deben estar protegidas y cada usuario debe saber, solo y exclusivamente, sus credenciales de acceso.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro del *clúster* y roba las credenciales.

***Rollback*:**

Reseteo de los datos modificados.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Es cierta la hipótesis, y las contraseñas se encontraban en la *BD* en texto plano. Para solucionar el problema, se ha utilizado BCrypt para guardar el hash de las contraseñas y de esta forma están cifradas con 10 rondas de sal.

FIGURA 34: registro de los usuarios en la *base de datos*.

Apéndice C

Desarrollo de experimentos en la funcionalidad completa

EXPERIMENTO 1

Definición:

Desconexión del *agente* en la red *Edge Computing*.

Hipótesis:

El *contenedor* se detendrá o funcionará sin ningún control por parte del *agente*.

Zonas afectadas:

Agente.

Estado estable:

El *microservicio* funciona correctamente con el control tanto del *agente* correspondiente como del *controlador*.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro de la máquina virtual y desconecta al *agente*.

Rollback:

Reseteo de la red *Edge Computing*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Se desconecta el *agente* y desaparecen todos los componentes de la *red Edge*:

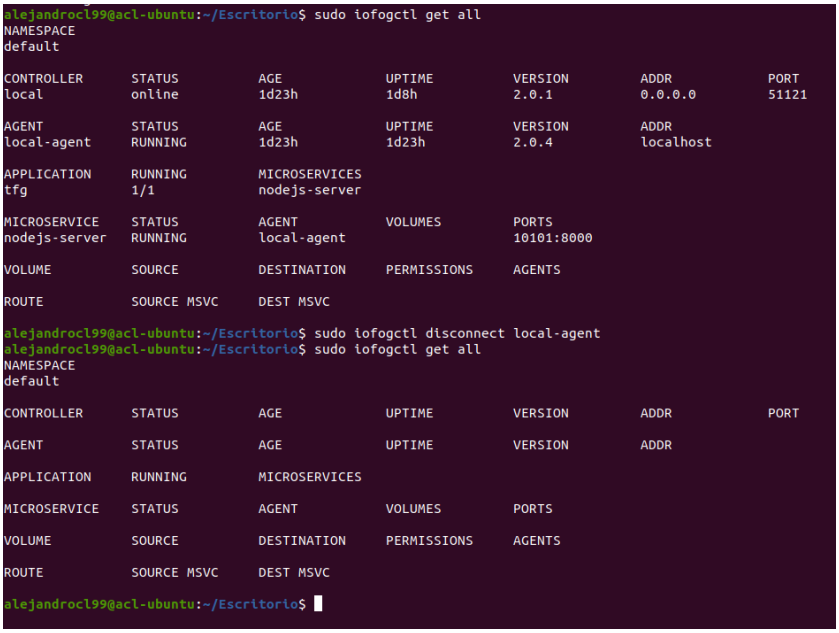

FIGURA 35: consola con los comandos realizados sobre la plataforma *IoFog*.

En cambio, el *contenedor* con el *servidor* continua trabajando, pero sin control ninguno del *"controller"* y de su respectivo *agente*:

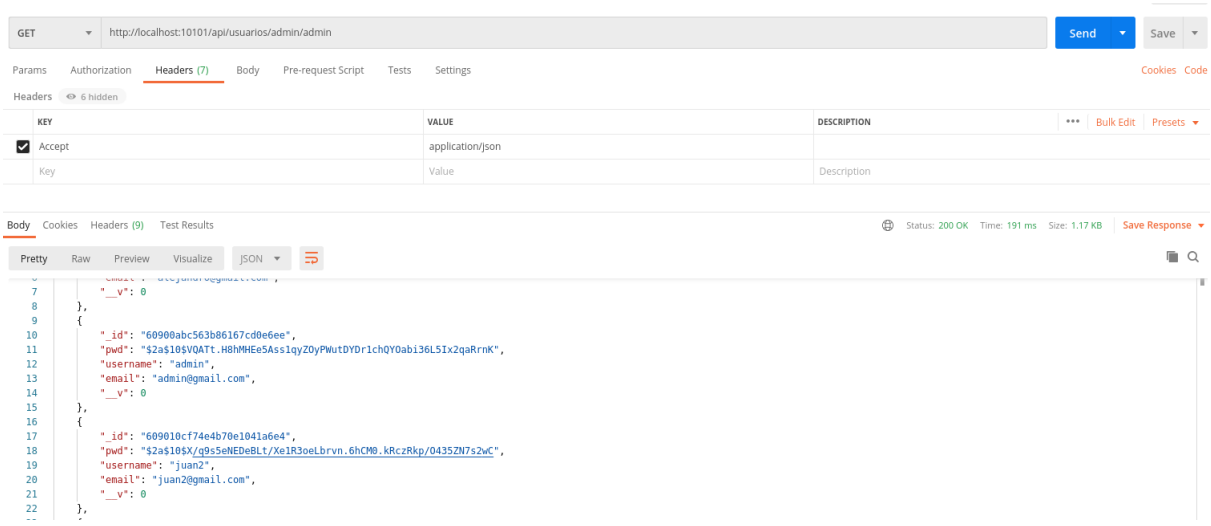

FIGURA 36: comprobación *petición HTTP* (*Postman*).

Los *contenedor*es siguen disponibles pero sin ningún tipo de coordinación:

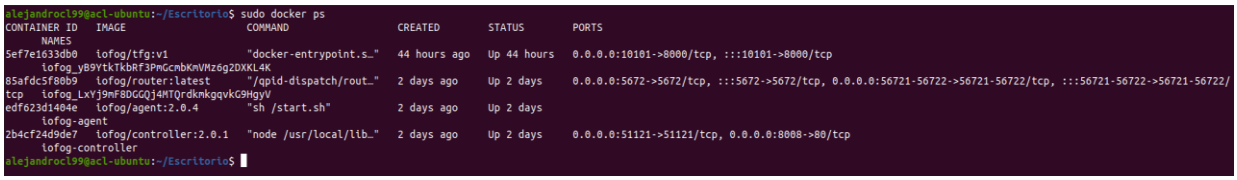

FIGURA 37: *contenedor*es desplegados en la experimentación.

Las posibles soluciones que se pueden aportar a este problema son:

- **Utilizar algún tipo de credencial** necesario para poder realizar funciones de desconexión o eliminación.

- **Evitar la desconexión de *agentes*** si se queda algún *contenedor* huérfano.

- **Eliminación de *contenedor*es de *agente*s y aplicación en cascada** para evitar el goteo de memoria en caso de eliminación o desconexión de *agente*s.

EXPERIMENTO 2

Definición:

Desconexión del *controlador* en la red *Edge Computing*.

Hipótesis:

La red se desactivará completamente o el *contenedor* funcionará íntegramente por su cuenta sin control ninguno.

Zonas afectadas:

Controlador.

Estado estable:

El *microservicio* funciona correctamente con el control tanto del *agente* correspondiente como del *controlador*.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro de la máquina virtual y desconecta el *controlador*.

Rollback:

Reseteo de la red *Edge Computing*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Se desconecta el *controlador* y se pierde la conexión a toda la red *Edge*:

```
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl disconnect local
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl get all
NAMESPACE
default

CONTROLLER      STATUS          AGE             UPTIME          VERSION         ADDR            PORT

AGENT           STATUS          AGE             UPTIME          VERSION         ADDR

APPLICATION     RUNNING         MICROSERVICES

MICROSERVICE    STATUS          AGENT           VOLUMES         PORTS

VOLUME          SOURCE          DESTINATION     PERMISSIONS     AGENTS

ROUTE           SOURCE MSVC     DEST MSVC
```

FIGURA 38: consola con los comandos realizados sobre la *plataforma Edge*.

El *contenedor* con el *servidor* sigue funcionando sin ningún tipo de monitorización:

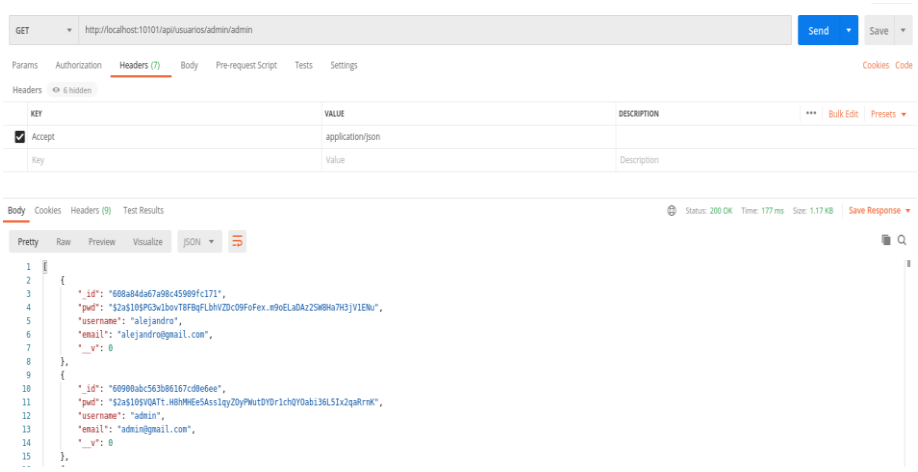

FIGURA 39: respuesta de la *petición HTTP* (*Postman*).

```
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED          STATUS          PORTS
          NAMES
9a1cc4b460b9   iofog/tfg:v1             "docker-entrypoint.s…"   36 minutes ago   Up 35 minutes   0.0.0.0:10101->8000/tcp, :::10101->8000/tcp
          iofog_ft47xpwy3GMvDJVk9f96qG7rZzyfxKWZ
e84ac9115360   iofog/router:latest      "/qpid-dispatch/rout…"   47 minutes ago   Up 47 minutes   0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 0.0.0.0:5
722/tcp   iofog_gmjKJzjpzkhwn43jGy2yZCFb62Nby4VB
9985d601dec7   iofog/agent:2.0.4        "sh /start.sh"           48 minutes ago   Up 48 minutes
          iofog-agent
6185e44f72af   iofog/controller:2.0.1   "node /usr/local/lib…"   51 minutes ago   Up 50 minutes   0.0.0.0:51121->51121/tcp, 0.0.0.0:8008->80/tcp
          iofog-controller
```

FIGURA 40: *contenedor*es desplegados en *Docker*.

Las posibles soluciones que se puede aportar a este problema son:

- **Utilizar algún tipo de credencial** necesario para poder realizar funciones de desconexión o eliminación.

- **Mecanismo de recuperación del sistema** si se pierde la conexión con el *controlador*.

- **Reinicio automático del *controlador*** si hay desconexión de este.

- **Eliminación de *contenedor*es de *controlador*, *agente*s y aplicación en cascada** para evitar el goteo de memoria en caso de eliminación o desconexión del *controlador*.

EXPERIMENTO 3

Definición:

Desconexión del *contenedor* que tiene la aplicación.

Hipótesis:

La aplicación dejará de funcionar.

Zonas afectadas:

Contenedor del *microservicio*.

Estado estable:

El *microservicio* funciona correctamente con el control tanto del *agente* correspondiente como del *controlador*.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro de la máquina virtual y elimina el *contenedor Docker* que tiene la aplicación.

Rollback:

Reseteo de la red *Edge Computing*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Eliminamos el *contenedor*:

```
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl get all
NAMESPACE
default

CONTROLLER      STATUS          AGE             UPTIME          VERSION         ADDR            PORT
local           online          7m51s           7m54s           2.0.1           0.0.0.0         51121

AGENT           STATUS          AGE             UPTIME          VERSION         ADDR
local-agent     RUNNING         7m50s           6m24s           2.0.4           localhost

APPLICATION     RUNNING         MICROSERVICES
tfg             1/1             nodejs-server

MICROSERVICE    STATUS          AGENT           VOLUMES         PORTS
nodejs-server   RUNNING         local-agent                     10101:8000

VOLUME          SOURCE          DESTINATION     PERMISSIONS     AGENTS

ROUTE           SOURCE MSVC     DEST MSVC

alejandrocl99@acl-ubuntu:~/Escritorio$ sudo docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED             STATUS             PORTS
                NAMES
cb7b23e70d71   iofog/tfg:v1           "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:10101->8000/tcp, :::10101->8000/tcp
                iofog_Ffb34vmQG76jg2thyyZGMFW8H3JDHYNn
b203e7ab860b   iofog/router:latest    "/qpid-dispatch/rout…"   5 minutes ago       Up 5 minutes       0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 0.0.0.0:56721-56722->56721-56722/tcp, :::56721-56722->
56721-56722/tcp   iofog_zdzj4PghnbwtjBkPBnx7JwMLg2NJFjdV
720f2d4a6f07   iofog/agent:2.0.4      "sh /start.sh"           6 minutes ago       Up 6 minutes
                iofog-agent
f45b2b8eff14   iofog/controller:2.0.1   "node /usr/local/lib…"   8 minutes ago       Up 7 minutes       0.0.0.0:51121->51121/tcp, 0.0.0.0:8008->80/tcp
                iofog-controller
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo docker rm -f cb7b23e70d71
cb7b23e70d71
alejandrocl99@acl-ubuntu:~/Escritorio$
```

FIGURA 41: consola con los comandos realizados sobre la plataforma *IoFog*.

Y el *servidor* queda invalidado:

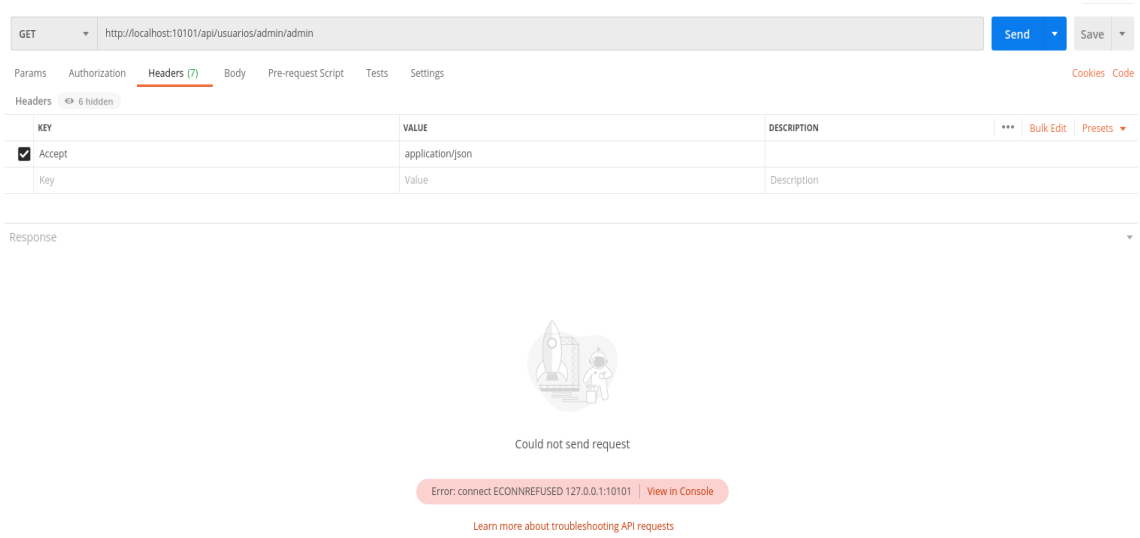

FIGURA 42: respuesta de la *petición HTTP* (*Postman*).

Sin embargo, el *servidor* es lanzado de forma automática a los segundos de eliminarlo, teniendo el servicio disponible de forma casi inmediata:

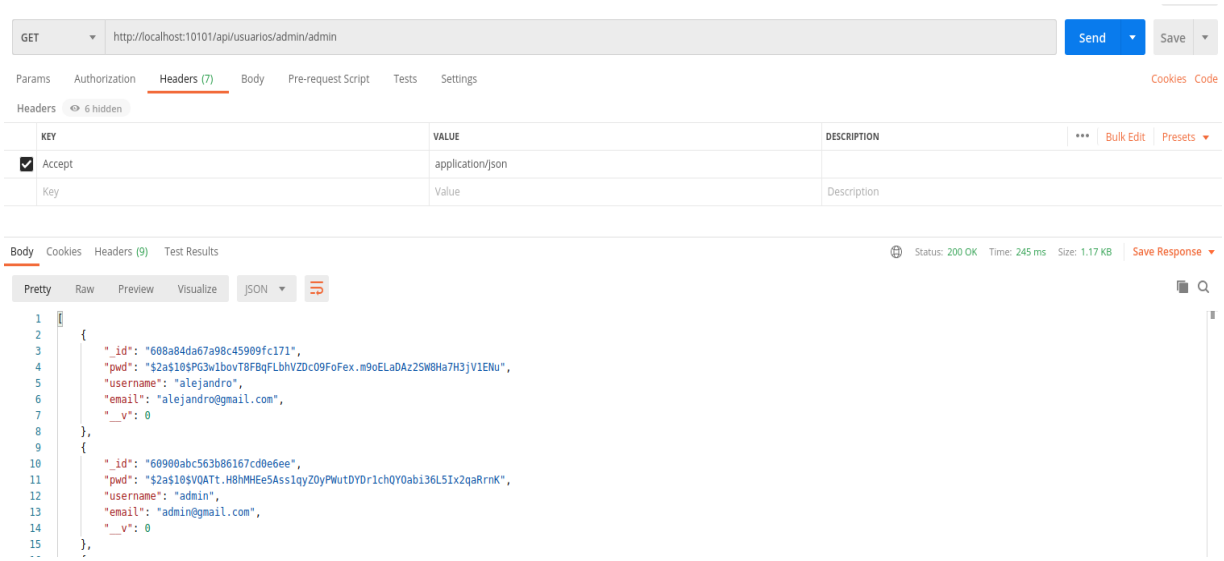

FIGURA 43: respuesta de la *petición HTTP* segundos después (*Postman*).

No tiene apenas repercusión tirar el *contenedor*, a menos que se pare el autolanzamiento.

En este caso, la solución está bien implementada, ya que tiene un sistema de recuperación automático ante este tipo de problemas.

EXPERIMENTO 4

Definición:

Parar la aplicación *Edge Computing*.

Hipótesis:

Se parará el *agente* y el *contenedor* correspondiente.

Zonas afectadas:

Aplicación (el *agente* y el *contenedor*).

Estado estable:

El *microservicio* funciona correctamente con el control tanto del *agente* correspondiente como del *controlador*.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro de la máquina virtual y para la aplicación.

Rollback:

Volver a activar la aplicación.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

La aplicación se detiene y no se pueden realizar peticiones al *servidor*:

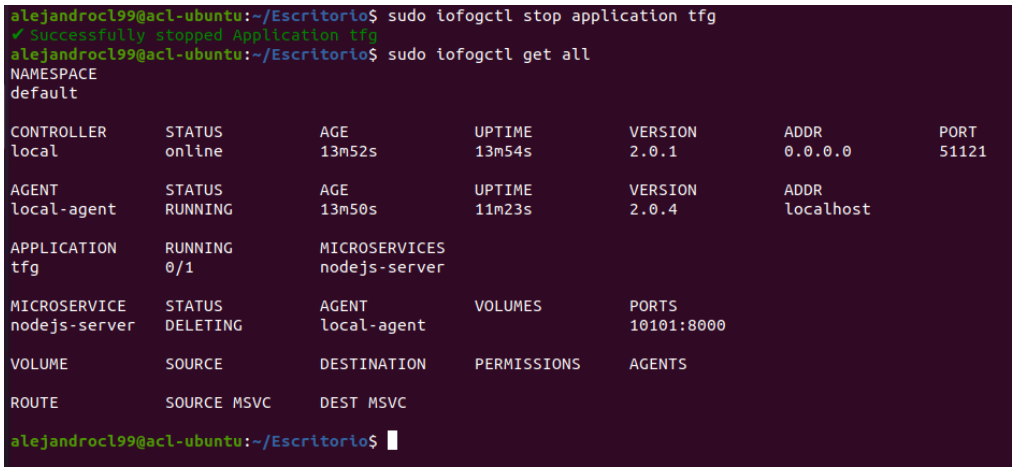

FIGURA 44: consola con los comandos realizados sobre la plataforma *IoFog*.

Cuando realizamos una petición al *servidor*:

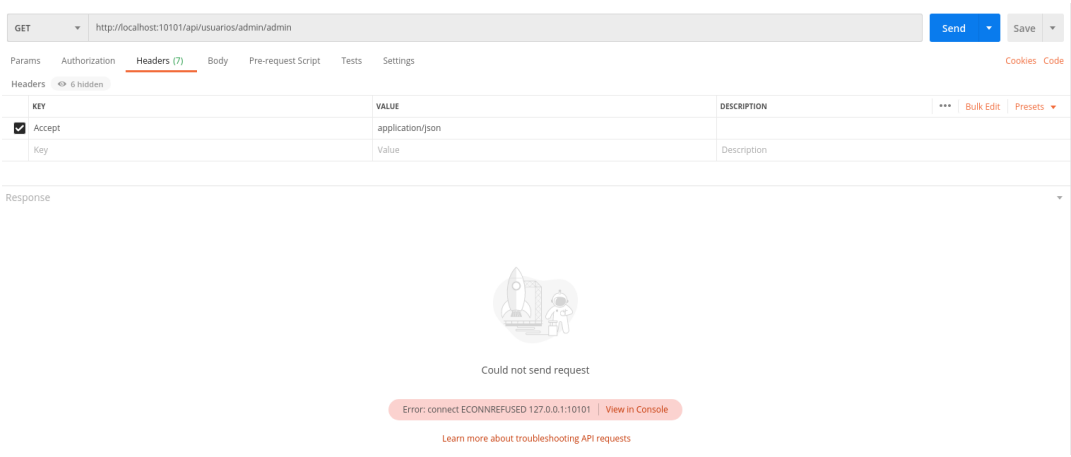

FIGURA 45: respuesta *petición HTTP* (*Postman*).

Si conectamos de nuevo la aplicación:

```
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl start application tfg
✔ Successfully started Application tfg
alejandrocl99@acl-ubuntu:~/Escritorio$ sudo iofogctl get all
NAMESPACE
default

CONTROLLER      STATUS          AGE             UPTIME          VERSION         ADDR            PORT
local           online          15m17s          15m20s          2.0.1           0.0.0.0         51121

AGENT           STATUS          AGE             UPTIME          VERSION         ADDR
local-agent     RUNNING         15m16s          12m53s          2.0.4           localhost

APPLICATION     RUNNING         MICROSERVICES
tfg             1/1             nodejs-server

MICROSERVICE    STATUS          AGENT           VOLUMES         PORTS
nodejs-server   RUNNING         local-agent                     10101:8000

VOLUME          SOURCE          DESTINATION     PERMISSIONS     AGENTS

ROUTE           SOURCE MSVC     DEST MSVC

alejandrocl99@acl-ubuntu:~/Escritorio$
```

FIGURA 46: consola con los comandos realizados sobre la plataforma *IoFog*.

Obtenemos respuesta a las peticiones:

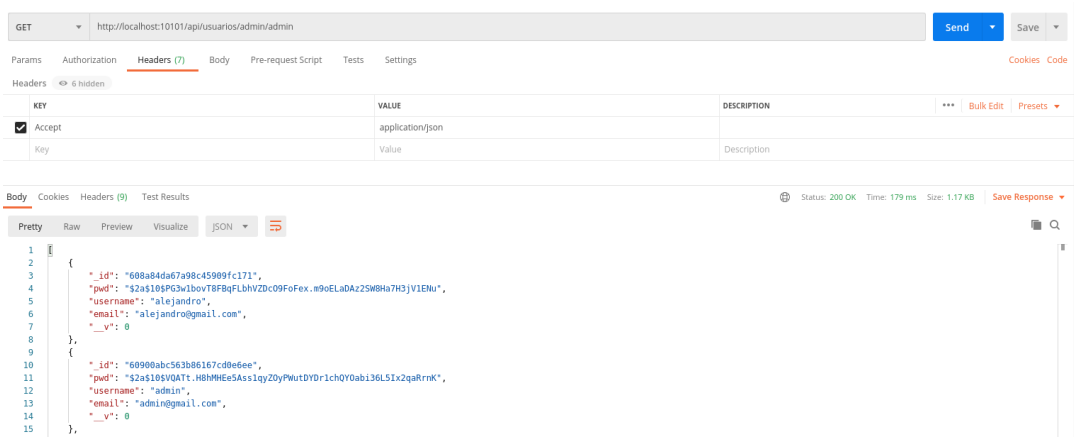

FIGURA 47: repuesta *petición HTTP* (*Postman*).

Solución a este tipo de problemas:

- **Pedir algún tipo de credencial** que permita a una persona legítima parar la aplicación.

- **Sistema de reconexión** de la aplicación en caso de que esta esté inactiva.

EXPERIMENTO 5

Definición:

Eliminación del *controlador*.

Hipótesis:

La red se desactivará completamente o el *contenedor* funcionará íntegramente por su cuenta sin control ninguno.

Zonas afectadas:

Controlador.

Estado estable:

El *microservicio* funciona correctamente con el control tanto del *agente* correspondiente como del *controlador*.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro de la máquina virtual y elimina el *controlador*.

Rollback:

Reseteo de la red *Edge Computing*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Se puede eliminar el *controlador* y se pierde la conexión con el *agente* y con el *contenedor* que tiene al *servidor*. El *servidor* sigue funcionando sin ningún tipo de monitorización:

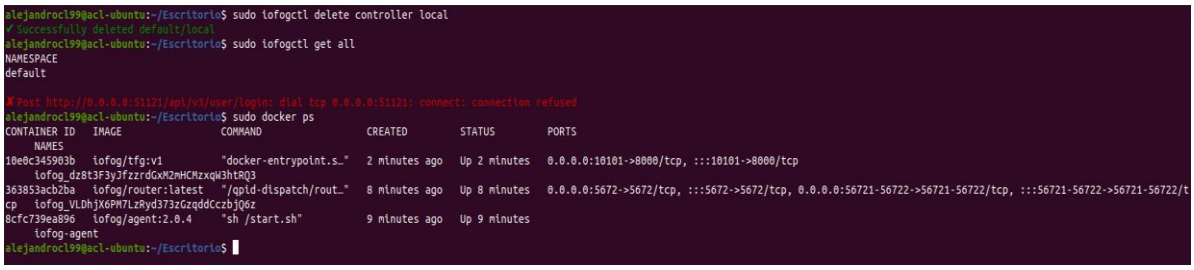

FIGURA 48: consola con los comandos realizados sobre la plataforma *IoFog*.

Las posibles soluciones que se le pueden aportar a este problema son:

- **Utilizar algún tipo de credencial** necesario para poder realizar funciones de desconexión o eliminación.

- **Evitar la eliminación del *controlador*** si se queda algún *contenedor* huérfano.

- **Eliminación de *contenedor*es de *agente*s y aplicación en cascada**, para evitar el goteo de memoria en caso de eliminación o desconexión del *controlador*.

- **Mecanismo de recuperación del sistema** si se pierde la conexión con el *controlador*.

- **Reinicio automático del *controlador*** si hay desconexión de este.

EXPERIMENTO 6

Definición:

Eliminación del *agente*.

Hipótesis:

La red se desactivará completamente o el *contenedor* funcionará íntegramente por su cuenta sin control ninguno.

Zonas afectadas:

Agente.

Estado estable:

El *microservicio* funciona correctamente con el control tanto del *agente* correspondiente como del *controlador*.

Esquema de funcionamiento:

- 1.- Una entidad no autorizada entra dentro de la máquina virtual y elimina el *agente*.

Rollback:

Reseteo de la red *Edge Computing*.

Personal de observación:

Alejandro Cortijo López.

Conclusión:

Nos pide utilizar la función force:

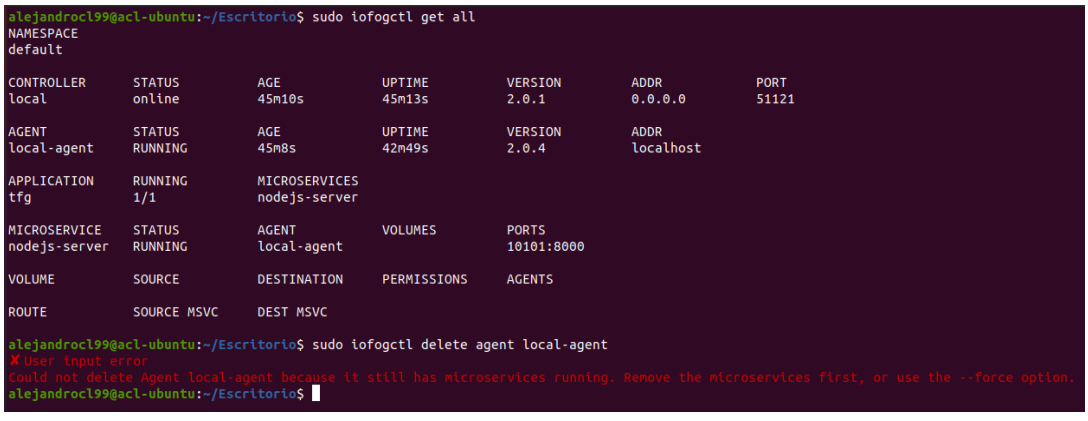

FIGURA 49: consola con los comandos realizados sobre la plataforma *IoFog*.

A continuación, utilizamos la función force:

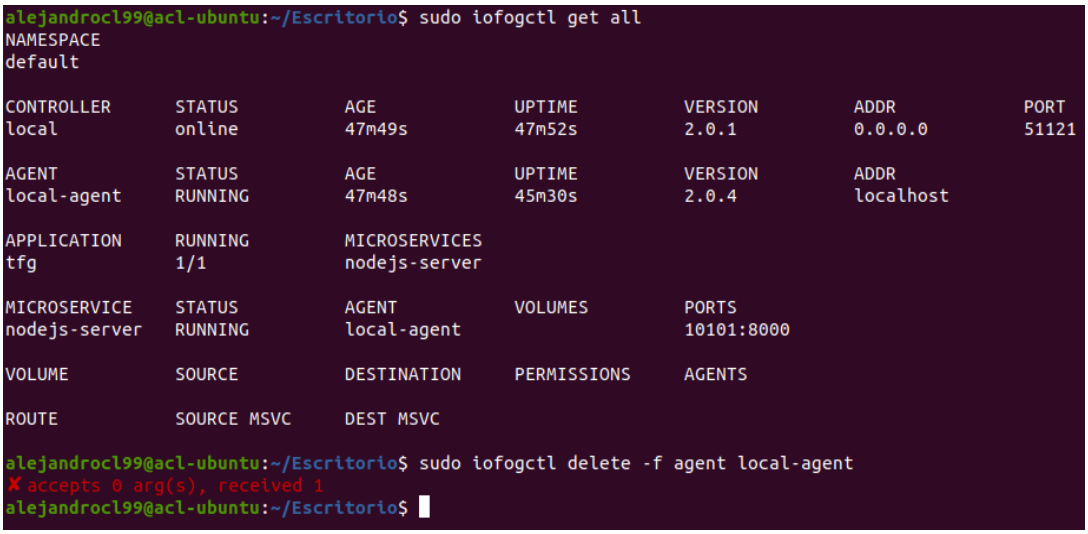

FIGURA 50: consola con los comandos realizados sobre la plataforma *IoFog*.

No se nos permite eliminar el *agente*.

Las posibles soluciones que se le pueden aportar a este problema son:

- **Utilizar algún tipo de credencial** necesaria para poder realizar funciones de desconexión o eliminación.

- **Evitar la eliminación de *agentes*** si se queda algún *contenedor* huérfano.

- **Eliminación de *contenedores* de *agentes* y aplicación en cascada** para evitar el goteo de memoria en caso de eliminación o desconexión de *agentes*.

Además, para finalizar la funcionalidad completa realizaremos todos los experimentos de la funcionalidad básica sobre el *servidor*, pero esta vez en producción.

EXPERIMENTO 1

Definición:

Desconexión de la *base de datos*.

Conclusión en producción:

En producción y tras la realización de todas las mejoras, cuando realizamos la desconexión de la *BD*, obtenemos un error 400:

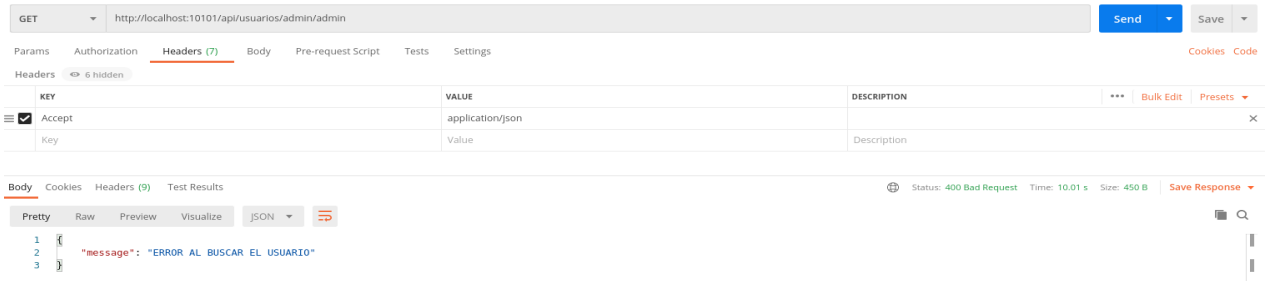

FIGURA 51: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 2

Definición:

Cambio de permisos en el acceso a la *base de datos*.

Conclusión en producción:

Solo le damos el permiso de leer la *BD*:

FIGURA 52: interfaz de accesos *MongoDB Atlas*.

Y el resultado es un error 500:

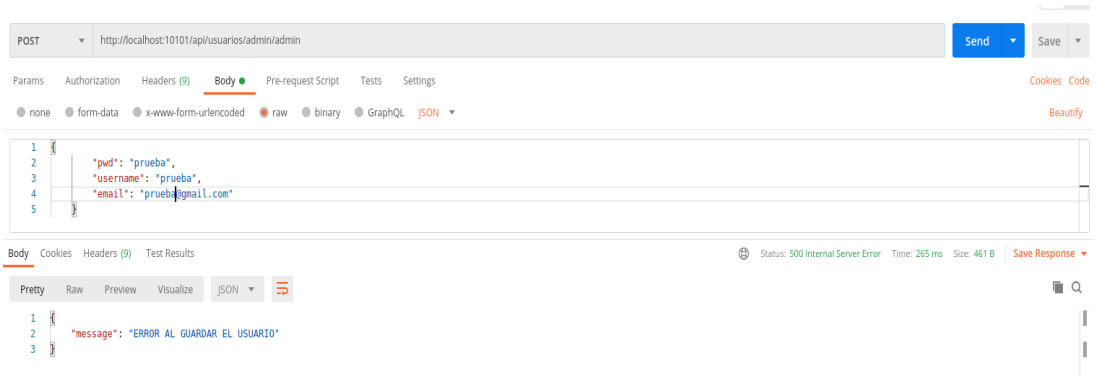

FIGURA 53: respuesta de la *petición HTTP* (*Postman*).

El *servidor* sigue a la espera de peticiones.

EXPERIMENTO 3

Definición:

Petición a un *"endpoint"* no recogido.

Conclusión en producción:

El *servidor* continua dando un error 404, pero sigue funcionando a la espera de peticiones.

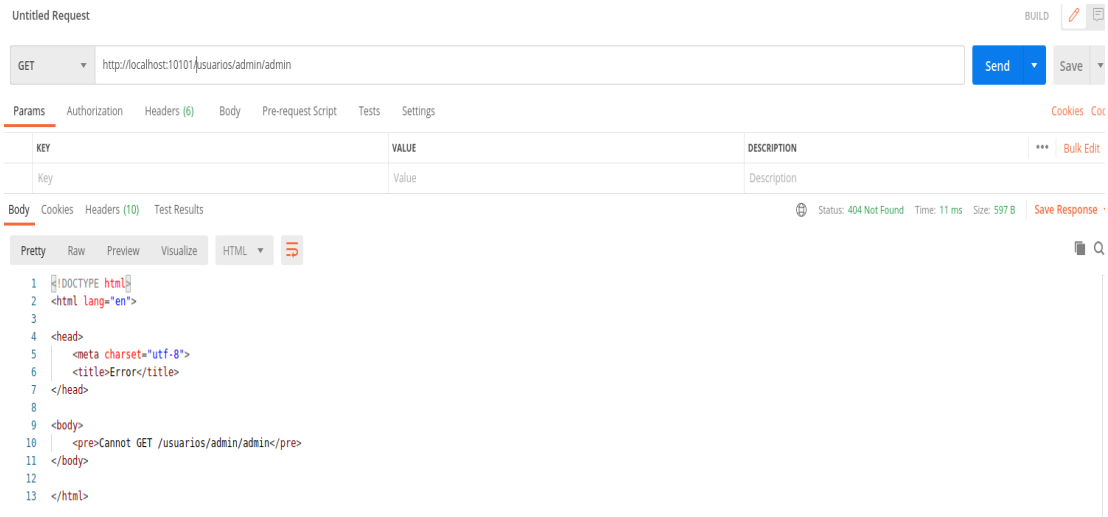

FIGURA 54: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 4

Definición:

Petición a un *"endpoint"* recogido pero con atributos erróneos.

Conclusión en producción:

El *servidor* continua dando un error 500, pero sigue funcionando a la espera de peticiones:

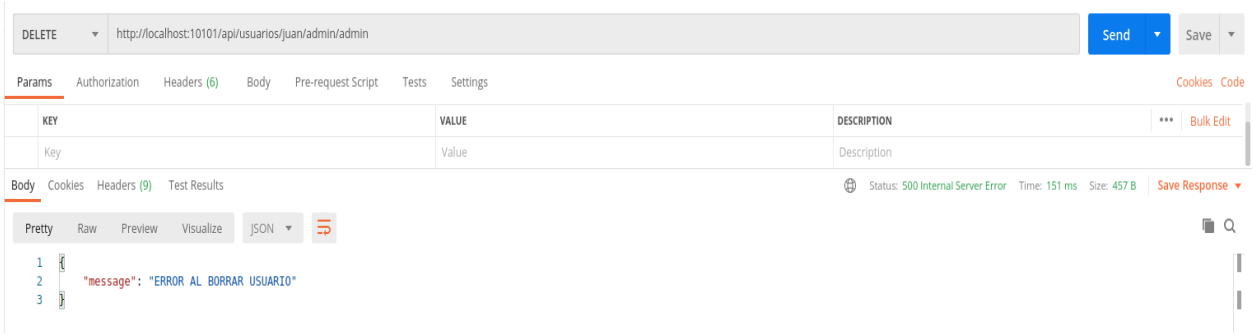

FIGURA 55: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 5

Definición:

Petición a un *"endpoint"* recogido pero con más atributos de los necesarios.

Conclusión en producción:

El *servidor* continua dando un error 404, pero sigue funcionando a la espera de peticiones.

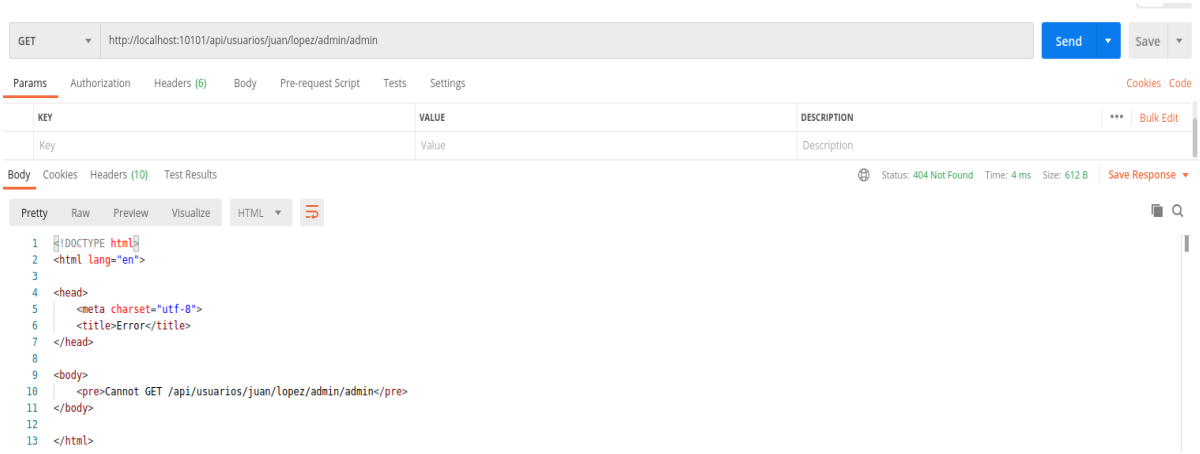

FIGURA 56: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 6

Definición:

Petición a un *"endpoint"* recogido pero sin atributos.

Conclusión en producción:

El *servidor* continúa dando un error 404, pero sigue funcionando a la espera de peticiones.

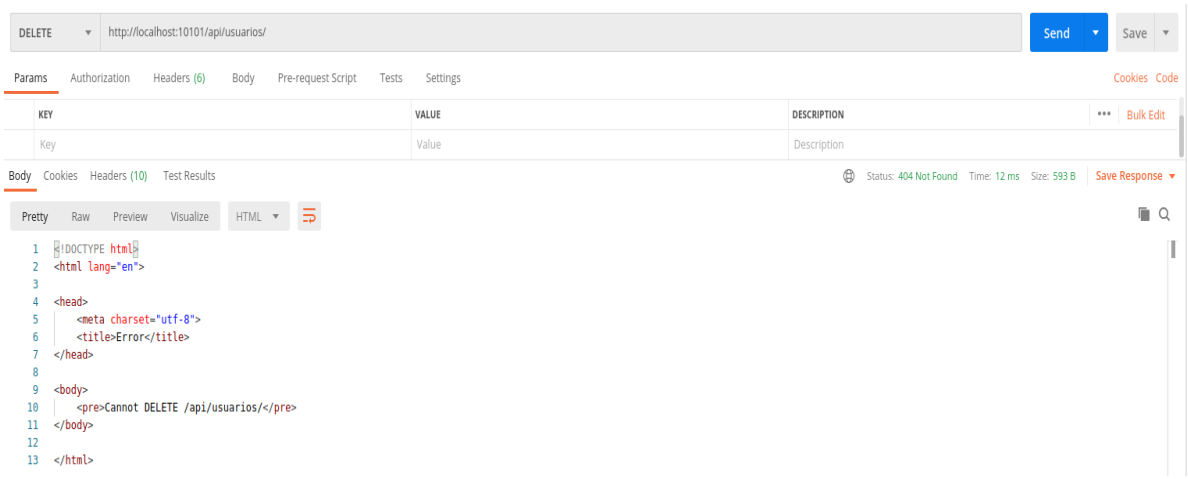

FIGURA 57: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 7

Definición:

Falta de credenciales.

Conclusión en producción:

Se ha implementado un sistema de credenciales para poder realizar las peticiones, basado en username y password:

- Si introduces ambas de forma correcta:

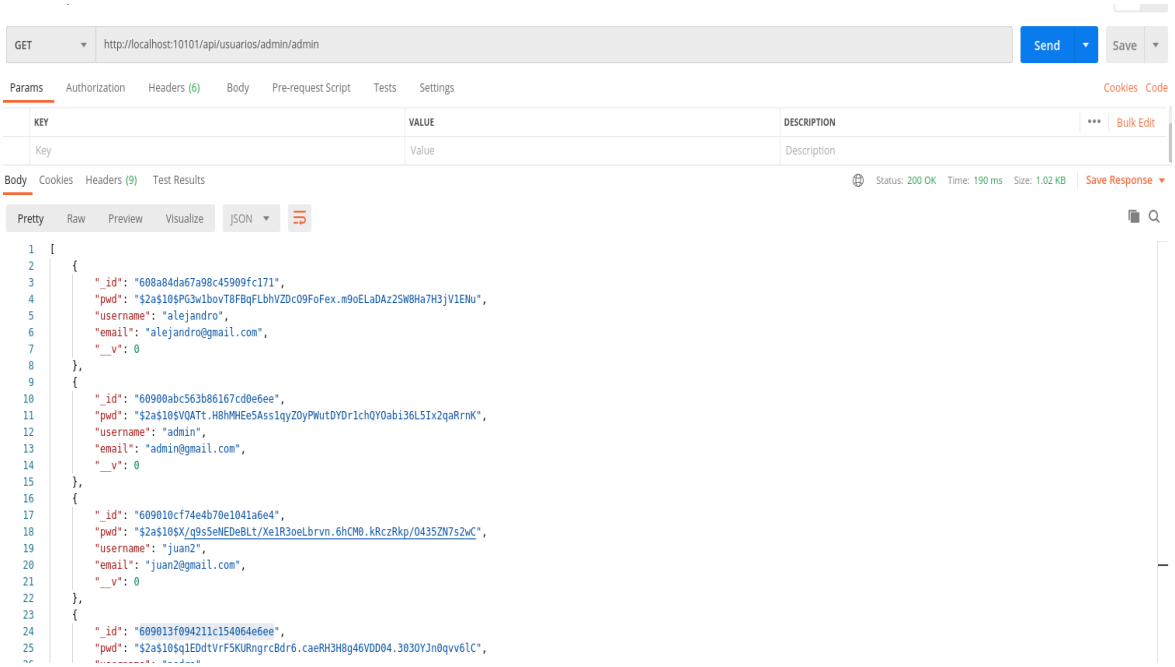

FIGURA 58: respuesta de la *petición HTTP* (*Postman*).

- Si introduces una de ellas mal:

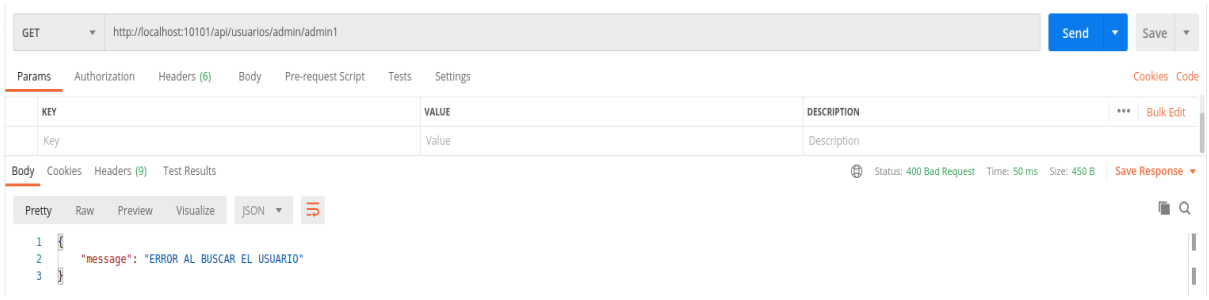

FIGURA 59: respuesta de la *petición HTTP* (*Postman*).

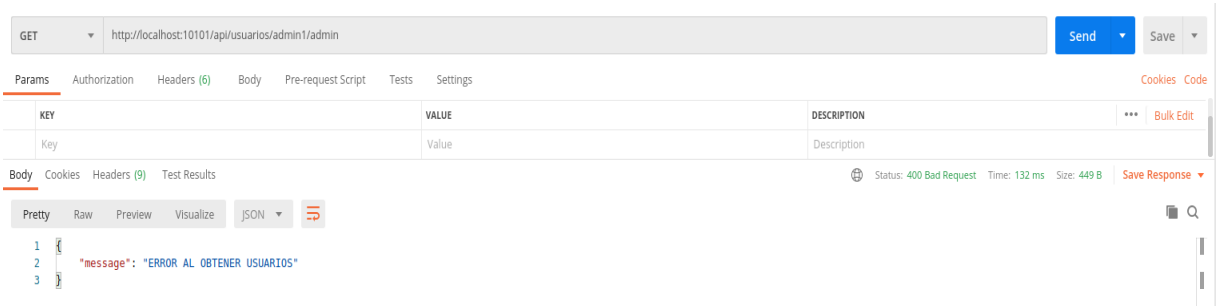

FIGURA 60: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 8

Definición:

Repudio del *servidor* (una entidad autorizada realiza cambios ilegítimos).

Conclusión en producción:

Se ha implementado un historial de logs, con el usuario que ha realizado la petición, la petición realizada y la hora a la que se ha realizado, para que, de la siguiente forma, quede constancia de todas las acciones realizadas sobre el *servidor*.

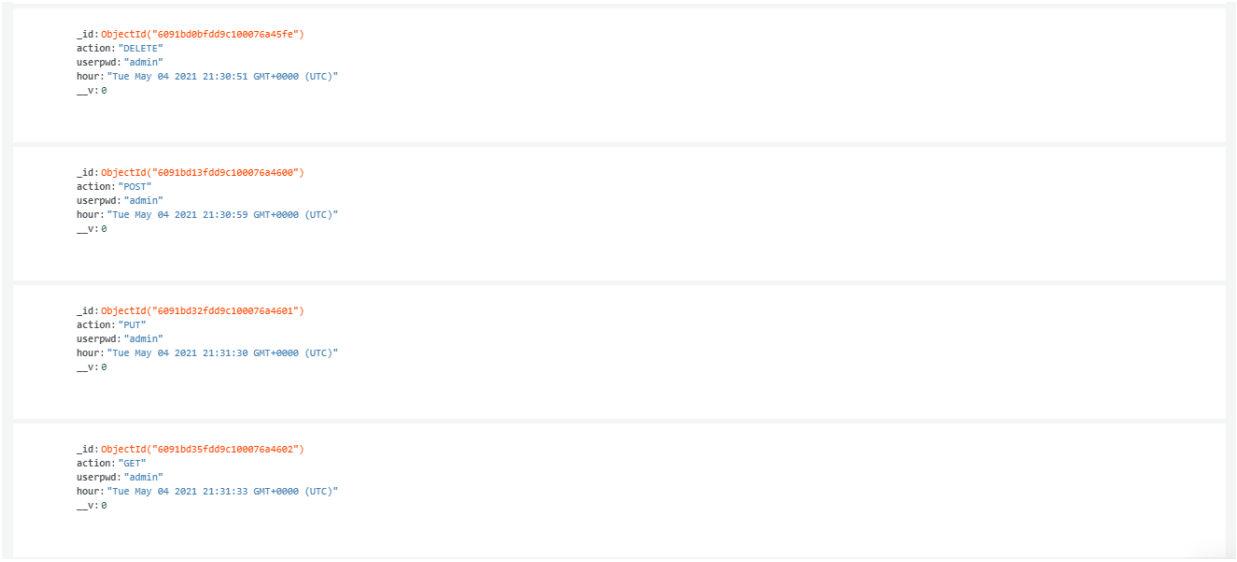

FIGURA 61: logs guardados en la *base de datos*.

EXPERIMENTO 9

Definición:

Petición con cabeceras modificadas (accept type / *XML*).

Conclusión en producción:

El *servidor* lanza un 200 pero con el resultado en JSON, en vez de en *XML*.

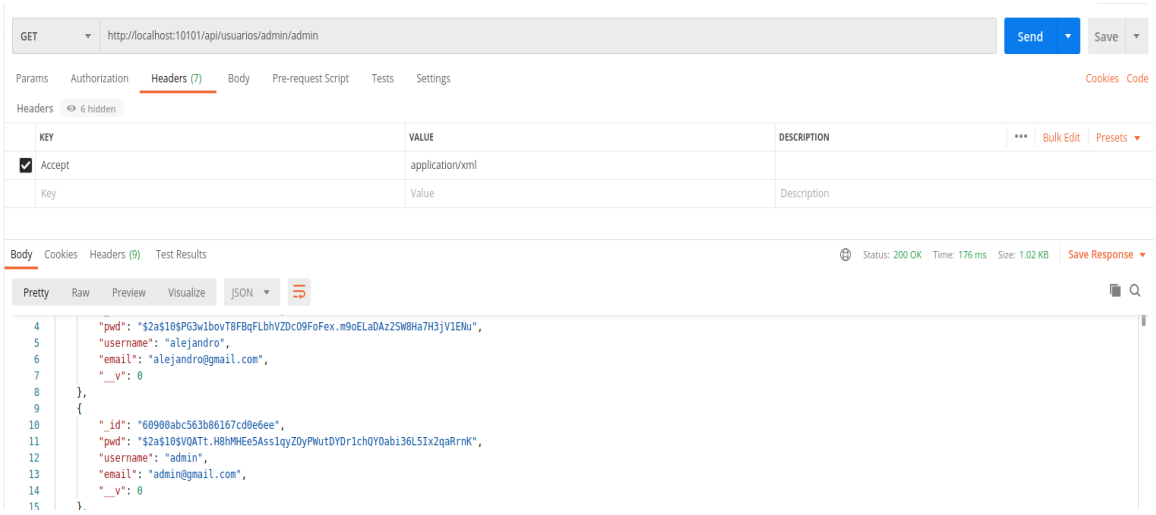

FIGURA 62: respuesta de la *petición HTTP* (*Postman*).

EXPERIMENTO 10

Definición:

Petición con cabeceras modificadas (content type / *XML*).

Conclusión en producción:

El *servidor* no capta bien la petición y salta error de identificación:

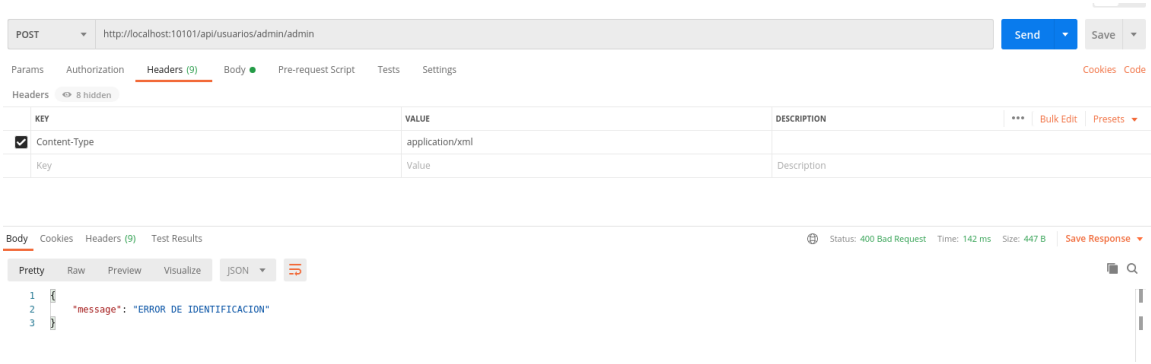

FIGURA 63: respuesta de la *petición HTTP* (*Postman*).

El *servidor* continua funcionando correctamente.

EXPERIMENTO 11

Definición:

Entrada y robo de credenciales en la *BD*.

Conclusión en producción:

Al realizar la implementación del BCrypt, las contraseñas están hasheadas:

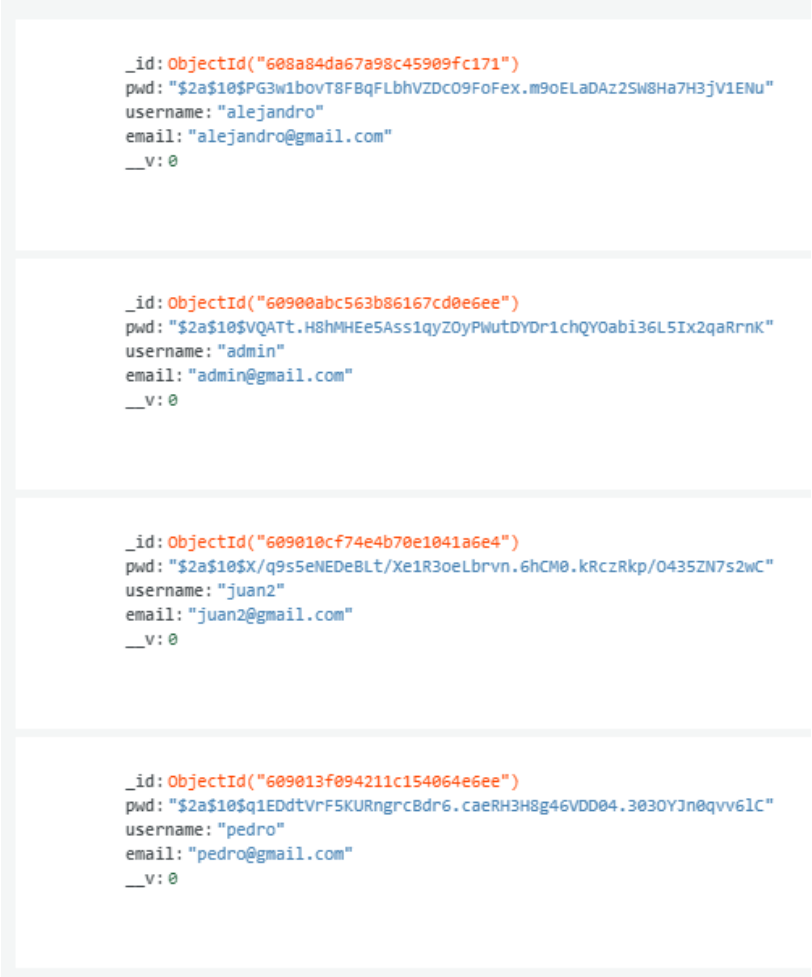

FIGURA 64: registro de los usuarios en la *base de datos*.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA