Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Estudio de rendimiento de una plataforma IoT en escenarios sanitarios

Autor: Ignacio Gómez Gómez
Tutor: Jorge Calvillo Arbizu

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Estudio de rendimiento de una plataforma IoT en escenarios sanitarios

Autor:
Ignacio Gómez Gómez

Tutor:
Jorge Calvillo Arbizu
Departamento de Ingeniería Telemática

Dpto. Ingeniería Telématica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo de Fin de Grado: Estudio de rendimiento de una plataforma IoT en escenarios sanitarios

Autor: Ignacio Gómez Gómez

Tutor: Jorge Calvillo Arbizu

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A pesar de que en un comienzo decidí cursar una ingeniería por curiosidad e interés de conocer un mundo nuevo, finalmente estoy orgulloso de la decisión que tomé, ya que el mundo de la ingeniería y las telecomunicaciones me ha resultado realmente apasionante. Por ello estoy plenamente agradecido de tener la oportunidad de conocerlo y haber abierto la puerta a semejante fuente de conocimiento y posibilidades en mi vida.

Agradecer a todas las personas que me han apoyado durante este duro pero bello trayecto, tanto familia, como amigos, anteriores y nuevos, que han surgido a lo largo de esta aventura y me han ayudado cuando el camino se hacía denso.

Gracias a Jorge Calvillo, tutor que me ha guiado en este trabajo de fin de carrera, aportando respuesta a cualquier duda y estando atento en todo momento para buscar una solución o referencia, lo cual se agradece bastante.

Ignacio Gómez Gómez

Sevilla, 2021

Resumen

El estudio del rendimiento en cualquier sistema es una tarea fundamental que aporta una visión de los diferentes puntos críticos del mismo. En una plataforma de "Internet of things" (IoT) en la que existen flujos de datos, es de gran importancia encontrar aquellos módulos del sistema que son considerados como "Cuellos de Botella" ya que es en estos puntos donde se encuentra el límite de rendimiento que condiciona al sistema completo.

Para una plataforma IoT en un escenario sanitario, es importante la consistencia de los datos, los cuales pueden ser adulterados por un mal rendimiento debido a, por ejemplo, un módulo del sistema actuando como cuello de botella, que se atasca y provoca que se pierdan paquetes y peticiones, en este caso perderíamos información valiosa de los pacientes y la fiabilidad del sistema sería mucho menor, por ello es realmente importante el estudio del rendimiento del sistema.

En este proyecto se propone el diseño de pruebas de carga mediante las que se evalúa el rendimiento de un sistema IoT basado en diferentes agentes que envían información a una base de datos, y otros agentes que consultan esta información. Todo ello controlado mediante un sistema de seguridad para la autenticación. Los componentes de FIWARE utilizados para el control de acceso son: Authforze y Keyrock. Para el almacenamiento de la información de los agentes, dedicada al control de acceso de los mismos, se utiliza una base de datos MySQL conectada a Keyrock. Para almacenar los datos, se utiliza una base de datos MongoDB. Las operaciones de extracción e inyección de datos en el sistema, se llevan a cabo a través de un proxy conocido como PEP Proxy Wilma.

En primer lugar, se realizó un estudio previo del sistema para conocer cómo funcionan los diferentes módulos que lo componen. Posteriormente, para lograr el objetivo de encontrar los puntos críticos del sistema, y con ello los límites de funcionamiento a pleno rendimiento del mismo, y tras el estudio de diferentes herramientas de testeo, se ha utilizado la herramienta JMeter para lanzar las pruebas de carga y comprobar mediante gráficas, los límites donde el sistema se vuelve inestable y empeora su rendimiento. Para ello se han cargado los diferentes módulos del sistema de manera individual, y se han realizado pruebas al sistema para comprobar cómo afecta la carga de cada módulo, y así poder establecer las limitaciones del sistema.

Tras el estudio del sistema basado en los resultados obtenidos de las pruebas de carga realizadas a los diferentes módulos del sistema, se obtiene que algunos servicios como MongoDB o Keyrock limitan al sistema, de diferentes maneras, tanto en tiempos de respuesta como en la cantidad de peticiones concurrentes que el sistema es capaz de procesar.

Abstract

The study of the performance of any system is a fundamental task that provides insight into the different critical points of the system. In an "Internet of things" (IoT) platform where there are data flows, it is of great importance to find those modules of the system that are considered as "Bottlenecks" since it is at these points where the performance limit that conditions the entire system is found.

For an IoT platform in a healthcare scenario, it is important the consistency of the data, which can be adulterated by a bad performance due to, for example, a system module acting as a bottleneck, which gets stuck and causes packets and requests to be lost, in this case we would lose valuable patient information and the reliability of the system would be much lower, so it is really important to study the performance of the system.

This project proposes the design of load tests to evaluate the performance of an IoT system based on different agents that send information to a database, and other agents that consult this information. All this is controlled by a security system for authentication. The FIWARE components used for access control are: Authforze and Keyrock. A MySQL database connected to Keyrock is used to store agent information for access control. A MongoDB database is used to store the data. Data extraction and injection operations in the system are carried out through a proxy known as PEP Proxy Wilma.

First of all, a preliminary study of the system was carried out in order to know how the different modules that compose it work. Subsequently, to achieve the objective of finding the critical points of the system, and thus the limits of its full performance, and after the study of different testing tools, the JMeter tool has been used to launch the load tests and check through graphs, the limits where the system becomes unstable and its performance worsens. For this purpose, the different modules of the system have been loaded individually, and the system has been tested to check how the load of each module affects the system, in order to establish the limitations of the system.

After studying the system based on the results obtained from the load tests performed to the different modules of the system, it is obtained that some services such as MongoDB or Keyrock limit the system, in different ways, both in response times and in the amount of concurrent requests that the system is able to process.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Tablas **xvi**

Índice de Figuras **xviii**

1 Introducción **1**
1.1 Motivación *1*
1.2 Objetivos del Proyecto *1*
1.3 Plan de trabajo *2*
1.4 Escenario de prueba *4*

2 Estado del arte **6**
2.1 Conceptos *6*
2.2 Testing *6*
2.3 Internet of Things (IoT). *7*
2.4 FIWARE *7*
2.4.1 Orion Context Broker 8
2.4.2 Identity Manager Keyrock 9
2.4.3 PEP Proxy Wilma 9
2.4.4 Authzforce 10
2.5 Tecnologías y recursos software utilizados. *10*
2.4.5 Máquina Virtual Ubuntu 20.04.0 LTS 10
2.4.6 VMware Workstation Player 16 11
2.4.7 Postman 11
2.4.8 Docker y Docker-Compose 11
2.4.9 MongoDB y MySQL 12
2.4.10 Wireshark 13
2.4.11 Herramientas de Testing 13

3 Estudio del sistema **15**
3.1 Orion Context Broker *16*
3.2 Keyrock *17*
3.3 MySQL *18*
3.4 MongoDB *18*
3.5 PEP Proxy Wilma *19*
3.6 Authzforce *28*

4 Resultados **29**
4.1 Puntos de carga *29*
4.1.1 Petición de token a Keyrock 29
4.1.2 Base de datos MySQL 30
4.1.3 Base de datos MongoDB 31
4.1.4 Políticas de Authzforce 32

4.1.5 Pruebas de tensión 32
4.2 Prueba 1: Petición de token a Keyrock 33
4.3 Prueba 2: Base de datos MySQL 41
4.4 Prueba 3: Base de datos MongoDB 46
4.5 Prueba 4: Políticas de Authzforce 49
4.6 Prueba 5: Pruebas de tensión 52

5 Conclusiones y posibles mejoras 56
5.1 Resultados finales del proyecto 56
5.2 Posibles mejoras para el proyecto. 58
5.2.1 Separación de Peticiones 58
5.2.2 Duplicación base de datos MongoDB 59
5.2.3 Cola de Peticiones 59

Anexo A: Instalación y configuración de Docker y Docker-compose 60

Anexo B: Instalación y Configuración de Los Componentes De La Plataforma 62

Anexo C: Instalación de Postman 67

Anexo D: Instalación y configuración de JMeter: 68

Referencias 74

ÍNDICE DE TABLAS

Tabla 1. Consulta de información. 2
Tabla 2. Trabajo inicial de los componentes. 2
Tabla 3. Conocimiento de Wilma y Authzforce. 3
Tabla 4. Estudio de herramientas de testing. 3
Tabla 5. Implementación de los escenarios. 3
Tabla 6. Pruebas y validación. 3
Tabla 7. Documentación. 4
Tabla 8. Tiempo total. 4
Tabla 9. Resumen Resultados. Peticiones de token a Keyrock 41
Tabla 10. Resumen resultados. Base de datos MySQL 45
Tabla 11. Resumen resultados. Base de datos MongoDB 49
Tabla 12. Política base para pruebas. Políticas de Authzforce 50
Tabla 13. Resultados de pruebas para diferente número de usuarios paralelos. Pruebas de Tensión 53
Tabla 14. Tiempos 200 Usuarios. Pruebas de Tensión 53
Tabla 15. Tiempos 300 Usuarios. Pruebas de Tensión 54
Tabla 16. Resumen de los resultados. Límites del sistema. 56

ÍNDICE DE FIGURAS

Figura 1. Esquema del sistema 5
Figura 2. Logo de Fiware 7
Figura 3. Logo OCB 8
Figura 4. Esquema Context Element 8
Figura 5. Logo de Keyrock 9
Figura 6. Logo Authzforce 10
Figura 7. Logo Ubuntu 20.04 LTS 10
Figura 8. WMware Workstation Player 16 11
Figura 9. Logo de Postman 11
Figura 10. Logo Docker 12
Figura 11. Logo Docker-Compose 12
Figura 12. Logo MongoDB 12
Figura 13. Logo MySQL 12
Figura 14. Logo Wireshark 13
Figura 15. Logo Jmeter 14
Figura 16. Esquema de la plataforma a testear 15
Figura 17. Diagrama funcionamiento plataforma. Ejemplo GET 16
Figura 18. Ejemplo de entidad de tipo "ActividadFisica" 17
Figura 19. Petición de token a Keyrock por parte de agente o usuario 17
Figura 20. Respuesta de Keyrock con token tras validación 18
Figura 21. Petición POST a Wilma por parte de agente IoT 20
Figura 22. Wilma pregunta a Keyrock para que autorice a un usuario con un token 20
Figura 23. Respuesta de Keyrock a Wilma con los datos del usuario o agente 21
Figura 24. Wilma consulta a Authzfoce la validez de la acción demandada 22
Figura 25. Respuesta afirmativa de Authzforce. 25
Figura 26. Wilma redirige la petición a Orion Context Broker 25
Figura 27. Información de la entidad a crear con el POST que viaja a Orion Context Broker 26
Figura 28. Respuesta de Orion Context Broker a Wilma 27
Figura 29. Reenvío del mensaje por parte de Wilma al agente o usuario demandante 27
Figura 30. Ejemplo de política básica de Authzforce 28
Figura 31. Petición de token a Keyrock 29
Figura 32. Conexión MySQL JMeter 30
Figura 33. INSERT de users 31

Figura 34. INSERT de roles de los usuarios 31
Figura 35. Carga MongoDB 31
Figura 36. Ejemplo de añadir Política a Authzforce 32
Figura 37. Configuración JMeter 2.000. Peticiones de token a Keyrock 33
Figura 38. Transacciones JMeter 2000. Peticiones de token a Keyrock 33
Figura 39. Hilos JMeter 2000. Peticiones de token a Keyrock 34
Figura 40. Tiempos JMeter 2000. Peticiones de token a Keyrock 34
Figura 41. Tabla JMeter 2000. Peticiones de token a Keyrock 34
Figura 42. Configuración JMeter 5000. Peticiones de token a KeyrockJMeter 35
Figura 43. Transacciones JMeter 5000. Peticiones de token a KeyrockJMeter 35
Figura 44. Hilos JMeter 5000. Peticiones de token a Keyrock.JMeter 36
Figura 45. Tiempos JMeter 5000. Peticiones de token a Keyrock 36
Figura 46. Tabla JMeter 5000. Peticiones de token a Keyrock 36
Figura 47. Configuración JMeter 8500. Peticiones de token a Keyrock 37
Figura 48. Transacciones JMeter 8500. Peticiones de token a Keyrock 37
Figura 49. Hilos JMeter 8500. Peticiones de token a Keyrock 38
Figura 50. Tiempos JMeter 8500. Peticiones de token a Keyrock 38
Figura 51. Tabla JMeter 8500. Peticiones de token a Keyrock 38
Figura 52. Configuración JMeter 10000. Peticiones de token a Keyrock 39
Figura 53. Transacciones JMeter 10000. Peticiones de token a Keyrock 39
Figura 54. Hilos JMeter 10000. Peticiones de token a Keyrock 39
Figura 55. Tiempo JMeter 10000. Peticiones de token a Keyrock 40
Figura 56. Tiempo JMeter 10000. Peticiones de token a Keyrock 40
Figura 57. Errores JMeter 10000. Peticiones de token a Keyrock 40
Figura 58. Peticiones lineales. Peticiones de token a Keyrock 41
Figura 59. Esquema JMeter 1 Usuario. Base de datos MySQL 42
Figura 60. Hilos JMeter 1 Usuario. Base de datos MySQL 42
Figura 61. Tiempos JMeter 1 Usuario. Base de datos MySQL 43
Figura 62. Tabla JMeter 1 Usuario. Base de datos MySQL 43
Figura 63. Esquema JMeter 10000 Usuarios. Base de datos MySQL 43
Figura 64. Hilos JMeter 10000 Usuarios. Base de datos MySQL 44
Figura 65. Tiempos JMeter 10000 Usuarios. Base de datos MySQL 44
Figura 66. Tabla JMeter 10000 Usuarios. Base de datos MySQL 44
Figura 67. Errores JMeter 10000 Usuarios. Base de datos MySQL 45
Figura 68. Esquema JMeter 10000 Entidades. Base de datos MongoDB 46
Figura 69. Hilos JMeter 10000 Entidades. Base de datos MongoDB 46
Figura 70. Tiempos JMeter 10000 Entidades. Base de datos MongoDB 47
Figura 71. Tabla JMeter 10000 Entidades. Base de datos MongoDB 47
Figura 72. Esquema JMeter 50000 Entidades. Base de datos MongoDB 47

Figura 73. Hilos JMeter 50000 Entidades. Base de datos MongoDB 48
Figura 74. Tiempos JMeter 50000 Entidades. Base de datos MongoDB 48
Figura 75. Tabla JMeter 50000 Entidades. Base de datos MongoDB 48
Figura 76. Tiempos 1 Usuario. Política Básica. Políticas de Authzforce 50
Figura 77. Tiempos 200 Usuarios. Política Básica. Políticas de Authzforce 51
Figura 78. Tiempos 1 Usuario. Política Compleja. Políticas de Authzforce 51
Figura 79. Tiempos 200 Usuarios. Política Compleja. Políticas de Authzforce 52
Figura 80. Tiempos 400 Usuarios. Pruebas de Tensión 54
Figura 81. Tiempos 500 Usuarios. Pruebas de Tensión 55
Figura 82. Error User Token not authorized 55
Figura 83. Evolución tiempos con carga de POST directa a MongoDB 57
Figura 84. Tiempos GET ORION con MongoDB cargado 57
Figura 85. Tiempos petición POST aislada 58
Figura 86. Distribución directorios y ficheros del sistema 62
Figura 87. Opciones script services.sh 63
Figura 88. Docker-compose. Orion Context Broker 63
Figura 89. Docker-compose. Keyrock 64
Figura 90. Docker-compose. Authzforce 65
Figura 91. Docker-compose. MongoDB y MySQL 65
Figura 92. Interfaz de Postman 67
Figura 93. Acceso directo JMeter 68
Figura 94. Interfaz JMeter 69
Figura 95. Botones Interfaz JMeter 69
Figura 96. Grupo de Hilos JMeter 70
Figura 97. Añadir grupo de hilos JMeter 70
Figura 98. Añadir elementos JMeter 71
Figura 99. Pestaña Plugins Manger JMeter 71
Figura 100. Plugins Manager Menu JMeter 72
Figura 101. Conector mysql JMeter. Carpeta de plugins 73
Figura 102. Script conexión base de datos MySQL 73

1 INTRODUCCIÓN

1.1 Motivación

Actualmente, la mayoría de dispositivos electrónicos de nuestro entorno son capaces de conectarse a internet y formar parte de lo que se conoce como "Internet of Things", o más conocido por sus siglas "IoT", conectándose a otros y siendo capaces de interactuar entre ellos. Esta tecnología la cual se encuentra en pleno auge, será probablemente una de las principales que nos acompañarán en el futuro. Términos relacionados con IoT pueden ser las "Smart Cities" o "Industria 4.0" donde se utiliza esta tecnología en dispositivos que permiten el control del tráfico, control de la electricidad, suministros de agua… además de las fábricas donde los diferentes instrumentos empleados en los procesos de producción se conectan entre ellos e interactúan sin necesidad de la acción humana. En nuestro caso, trataremos la aplicación en el ámbito sanitario, en el que las diferentes tecnologías existentes se apoyan en el uso de dispositivos IoT.

El IoT en sanidad supone la integración de forma inteligente de los datos recogidos por los sensores de los dispositivos médicos y tecnologías de comunicación móviles que se utilizan en la asistencia sanitaria. En este caso se estudiará el uso de sensores que recopilan información útil de las constantes vitales de los pacientes y la envían a una base de datos donde estos datos convergen, permitiendo el estudio de la salud de los pacientes.

Como en cualquier sistema, en las plataformas IoT el testeo es un proceso fundamental para garantizar su calidad y funcionamiento óptimo. En un sistema IoT que recibe información constante de dispositivos, existiendo un flujo de datos importante, es necesario comprobar de qué forma se ve afectado el rendimiento cuando se aumenta el número de elementos conectados, y con ello el flujo de información y peticiones.

Por ello, en este trabajo se van a llevar a cabo una serie de pruebas de rendimiento sobre una plataforma IoT para estudiar aquellos módulos del sistema que soportan menor carga, y que, por lo tanto, limitan al sistema en cuestión.

Como caso de uso se utilizará un dispositivo sensor integrado en una camiseta inteligente que obtiene datos de la actividad física de los pacientes que la visten. Este dispositivo ha sido desarrollado por el Grupo de Ingeniería Biomédica de la ETSI. La información obtenida por las camisetas es publicada por agentes IoT y, mediante los componentes FIWARE, se asegura la integridad de la información y la seguridad del sistema. Este sistema almacena una gran cantidad de información (de autenticación de pacientes y médicos, de constantes vitales de los pacientes…) y mediante pruebas de cargas aumentaremos esta información y flujo de datos para comprobar cómo rinde el sistema bajo determinadas circunstancias.

Una motivación añadida para llevar a cabo este trabajo es el interés personal por conocer nuevas herramientas relacionadas con "QA tester", o "quality assurance tester", un puesto de trabajo que abarca el estudio de calidad y rendimiento de sistemas, y que es muy importante y valorado hoy en día. Añadiendo que previamente he recibido una oferta para un empleo en una empresa como Ingeniero de Rendimiento, en cuyo puesto se requiere el uso de JMeter para testear el rendimiento de determinados sistemas.

1.2 Objetivos del Proyecto

El objetivo principal del proyecto es el diseño y ejecución de baterías de pruebas de carga usando JMeter para comprobar el rendimiento del sistema bajo determinadas circunstancias y localizar los puntos críticos del mismo también conocidos como cuellos de botella.

Para poder diseñar y lanzar las diferentes pruebas de carga con JMeter, primero tenemos que hacer un estudio del sistema, para conocer el funcionamiento de los componentes del mismo y cómo se relacionan entre ellos. Esto se debe principalmente a que nuestro sistema (y cualquier otro) estará formado por varios componentes, ya sean bases de datos, proxys, servidores, clientes, sistemas de seguridad… y cada uno de ellos funcionará de una manera diferente, aportando ciertas limitaciones al sistema.

Una vez analizado el sistema componente a componente nos dispondremos a crear diferentes escenarios, en cada uno de los cuales colocaremos bajo carga un componente del sistema para comprobar cómo afecta al funcionamiento global, y encontrar los factores que resultan más limitantes lanzando pruebas con una herramienta de pruebas de carga, en este caso y tras el estudio de varias herramientas, he escogido JMeter.

1.3 Plan de trabajo

En este apartado se van a detallar las tareas que se han realizado para llevar a cabo el proyecto, y una comparación del tiempo estimado y el tiempo real dedicado a cada una de ellas.

Estas tareas son las siguientes:

- Tarea 1: Búsqueda y obtención de la información. En esta actividad inicial se estudiará la documentación de los diferentes componentes de la plataforma IoT, tanto los componentes funcionales como los de seguridad.

Consulta de información	**Tiempo estimado**	**Tiempo real**
Investigación de componentes del sistema.	35 horas	48 horas
Total	35 horas	48 horas

Tabla 1. Consulta de información

- Tarea 2: Instalación de los diferentes componentes. En esta tarea también se ha tenido en cuenta la comprobación de la conexión entre todos los componentes y el tiempo invertido en la solución de los errores que se han obtenido en la instalación de estos.

Trabajo inicial de los componentes	**Tiempo estimado**	**Tiempo real**
Instalación de los componentes y comprobación de la conexión entre ellos	25 horas	45 horas
Total	25 horas	45 horas

Tabla 2. Trabajo inicial de los componentes

- Tarea 3: Esta tarea va a consistir en tratar de entender los diferentes archivos de configuración de Wilma y de conocer un lenguaje nuevo, XACML, que es usado en Authzforce para establecer diferentes políticas con el fin de autorizar a diferentes usuarios.

Conocimiento de Wilma y Authzforce	**Tiempo estimado**	**Tiempo real**
Conocimiento de los archivos de configuración de Wilma	10 horas	10 horas
Búsqueda de información sobre el lenguaje XACML	12 horas	15 horas

Total	22 horas	25 horas

Tabla 3. Conocimiento de Wilma y Authzforce

- Tarea 4: El propósito de esta tarea es investigar herramientas de testing como SoapUI, o JMeter, para decidir cuál usar, en nuestro caso, JMeter.

Estudio de herramientas de Testing	**Tiempo estimado**	**Tiempo real**
Recolección de información sobre herramientas de testing.	10 horas	15 horas
Elección de la mejor para nuestro caso, JMeter, y estudio de la misma para hacer uso de ella	100 horas	120 horas
Total	110 horas	135 horas

Tabla 4. Estudio de herramientas de testing

- Tarea 5: Esta tarea va a tratar sobre el diseño y desarrollo de los posibles escenarios que se pueden implementar y el tiempo dedicado sobre los mismos

Implementación de los escenarios	**Tiempo estimado**	**Tiempo real**
Planteamiento de los posibles escenarios	10 horas	15 horas
Desarrollo de los escenarios	70 horas	90 horas
Total	80 horas	105 horas

Tabla 5. Implementación de los escenarios.

- Tarea 6: Pruebas y validación de los diferentes escenarios planteados

Pruebas y validación	**Tiempo estimado**	**Tiempo real**
Pruebas de cada escenario	5 horas	8 horas
Corrección de errores en la puesta en marcha de los escenarios	5 horas	10 horas
Total	10 horas	18 horas

Tabla 6. Pruebas y validación

- Tarea 7: Realización de la memoria del trabajo fin de Grado

Documentación	**Tiempo estimado**	**Tiempo real**
Memoria del trabajo	80 horas	85 horas
Total	80 horas	85 horas

Tabla 7. Documentación

- Tiempo total estimado y real concluido.

Total	**362 horas**	**461 horas**

Tabla 8. Tiempo total

1.4 Escenario de prueba

La comprobación del rendimiento de la plataforma en cuestión parte del estudio de cada componente del sistema, para entender su funcionamiento y encontrar la manera de someterlo a carga.

En la Figura 1. Esquema del sistema, podemos observar los diferentes componentes del sistema y cómo se relacionan entre ellos. Existen diferentes acciones que un Agente IoT o un usuario registrado, como por ejemplo un médico, pueden realizar. Ya sea publicar o solicitar información, subscribirse a determinada entidad o actualizar sus datos, para todas estas acciones, Wilma PEP Proxy actuará de intermediario, recogiendo dicha solicitud y consultando con Keyrock la autenticidad del agente o usuario que desea acceder al sistema mediante la validación de una contraseña llamada “token” que debe ir en la cabecera de la solicitud de la acción. Este token es aportado previamente por Keyrock al usuario o agente, cuando este está previamente registrado en la base de datos MySQL conectada a Keyrock. Una vez Keyrock valida el token, y con ello el acceso al sistema, Wilma consulta a Authzforce para comprobar si el usuario está autorizado a acceder a determinado recurso, o a realizar determinada acción. En caso afirmativo, Wilma redirige la petición a Orion Context Broker donde, dependiendo de la acción demandada, se obtendrá una respuesta u otra. En caso de que no esté autorizado se mostrará un mensaje indicándolo, lo mismo ocurre cuando el token es inválido.

Con JMeter realizaremos la carga de la plataforma, simulando peticiones o introduciendo datos simulando agentes IoT o usuarios del sistema. Postman será utilizado para cambiar las políticas de Authzforce por la sencillez que aporta, ya que en caso de no utilizarlo, habría que hacerlo por línea de comandos, siendo esto bastante engorroso teniendo en cuanta que las políticas de acceso que maneja Authzforce pueden llegar a ser de miles de líneas.

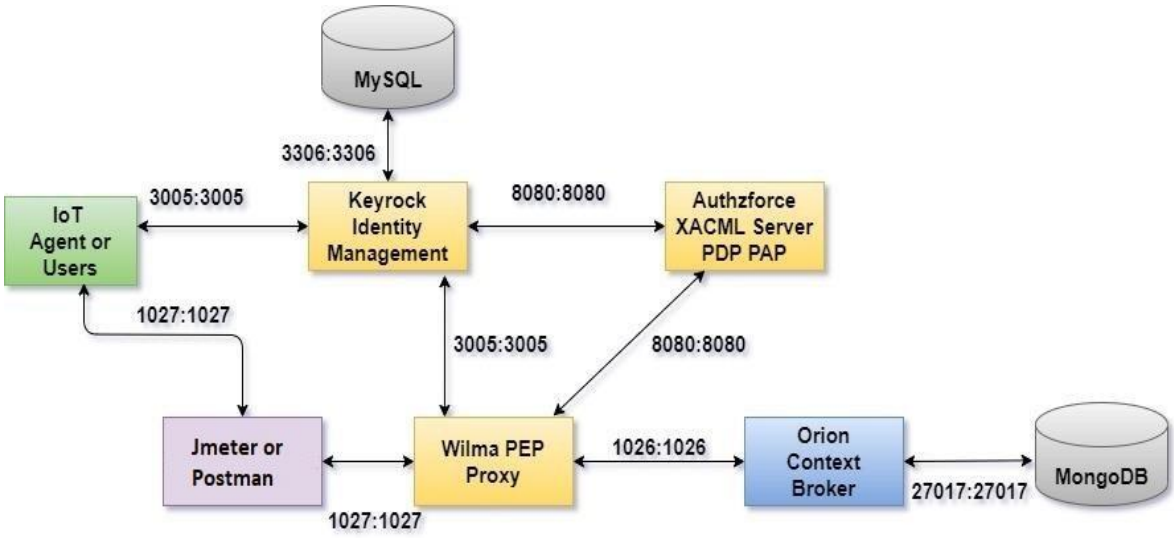


Figura 1. Esquema del sistema

Una vez estudiado los componentes del sistema, identificamos los puntos de interés para someter bajo carga, en este caso, el número de Agentes IoT o Users mandando peticiones, ya sean de publicar información, obtenerla, subscribirse a entidades o actualizarlas. Otros puntos de interés son las bases de datos, tanto MySQL, con la información de autenticación de usuarios, como MongoDB que contiene las entidades (información de los sensores de las camisetas de los agentes IoT), de este modo comprobaremos también el funcionamiento de Keyrock y Orion Context Broker bajo carga. Por último, identificamos Authforce como otro punto de interés a someter bajo presión, cargándolo con políticas de acceso densas para comprobar cómo afecta al sistema. Con todo esto evaluaremos cómo afecta al rendimiento del sistema cada componente y propondremos soluciones para la optimización del sistema.

2 ESTADO DEL ARTE

En este apartado vamos a explicar brevemente cada una de las tecnologías utilizadas en este proyecto, y se detallarán los recursos utilizados para la implementación de este trabajo.

2.1 Conceptos

Se considera de importancia el conocimiento de determinados conceptos antes de proseguir con el estudio del rendimiento de un sistema:

- Testing de software: consiste en poner a prueba el software para conocer su calidad, descubrir determinados errores, límites de carga, e incluso prevenir futuros bugs. Es un proceso fundamental en cualquier desarrollo de un producto software.
- Cuello de botella: se produce cuando la falta de potencia de uno de los componentes lastra el rendimiento del resto, impidiendo que éste desarrolle su potencia completa.
- Prueba de carga: comprueba el comportamiento del sistema, en cuestiones de rendimiento, estando sometido a diferentes tipos de carga (como el número de usuarios, el número de transacciones, etc.), mientras que la configuración permanece igual.
- API: es la abreviatura de "Application Programming Interfaces", que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas
- Pruebas unitarias: Pruebas que se realizan a cada componente por separado para comprobar su correcto funcionamiento antes de integrarlo con el resto.
- Pruebas de integración: Pruebas que se realizan una vez los componentes han sido integrados entre ellos para comprobar el correcto funcionamiento cuando interactúan entre ellos.
- Pruebas End to End: Serán las pruebas que hagamos para comprobar el correcto funcionamiento de la aplicación en un caso real, ya con todo integrado y funcionando.

2.2 Testing

El objetivo de las pruebas es representar la calidad de un producto en cuestiones de rendimiento y eficiencia. Las pruebas de calidad buscan encontrar defectos, prevenirlos o incluso ayudar en la toma de decisiones sobre el producto.

Dependiendo del contexto en que se desarrolle la evaluación de calidad, ya sea una plataforma, una aplicación web, un sistema informático, una red… la información que se desea obtener de las pruebas es muy diferente, y con ello la naturaleza de las mismas, existiendo una gran cantidad de tipos de pruebas.

Podemos clasificar las pruebas según su función, existiendo dos grandes grupos:

- Pruebas funcionales: Pruebas que comprueban el correcto funcionamiento de los componentes del sistema previamente diseñados, desde lo más bajo, componente a componente (pruebas unitarias) hasta el nivel más alto, con todos los componentes del sistema integrados y funcionando en conjunto (integración y End to End)

- Pruebas no funcionales: Pruebas cuyo objetivo es verificar el cumplimiento de determinados requisitos previamente establecidos para el sistema, como por ejemplo la disponibilidad, el retardo, la capacidad, la seguridad, el rendimiento…

En este caso, dado que lo que se quiere evaluar es el rendimiento de una plataforma IoT, los diferentes escenarios de este proyecto se centran en el desarrollo de pruebas funcionales de carga que evalúen el rendimiento de la plataforma y cómo afecta cada módulo que la compone. No obstante, al montar la plataforma inicialmente, se han realizado pruebas unitarias para comprobar que los componentes funcionan de manera individual, y luego en conjunto. Una vez verificado que el sistema funciona como se desea, procederemos a realizar las pruebas de rendimiento.

2.3 Internet of Things (IoT).

Internet of Things se define como el conjunto de tecnologías que permiten interconectar objetos físicos capaces de intercambiar datos entre ellos o con otros dispositivos y sistemas, formando una red. Estos objetos interconectados pueden ser desde simples herramientas domésticas hasta complejas herramientas utilizadas en la industria.

Esta tecnología, la cual se encuentra actualmente en auge, se espera que siga creciendo en los próximos años, formando parte de una de las principales tecnologías del futuro, en el cual no sería extraño que los hospitales la incorporen en la mayoría de los seguimientos a sus pacientes, o que multitud de hogares la incluyan en sus elementos más básicos, como ventanas, puertas, luces… La importancia de IoT radica en la interacción que crea entre personas, procesos y objetos. Todo esto gracias a tecnologías como Big Data, Cloud, Machine Learning y Analítica, Inteligencia Artificial, electrónica, informática de bajo coste o tecnologías móviles.

La recolección de datos por parte de los dispositivos se lleva a cabo gracias a sensores. De esta forma los objetos se relacionan con el medio de una forma u otra y recolectan información, que son capaces de enviar o recibir a otros dispositivos.

Por otro lado, cabe resaltar la importancia del testeo de estos sistemas de IoT, dado que pueden llegar a manejar una cantidad de datos que crece con el número de dispositivos que se incorporan a la red, pudiendo determinado sistema colapsar o presentar un rendimiento deficiente. Es por ello que resulta necesario testear las plataformas IoT que manejan dicha cantidad de datos, como es nuestro caso, para conocer los límites de la misma y no llegar a saturarla, o desarrollar soluciones para aumentar los límites del sistema en caso necesario.

2.4 FIWARE

FIWARE [1] es una plataforma de código abierto que ofrece diferentes estándares para la gestión de información de contexto siguiendo el paradigma IoT. FIWARE proporciona una serie de componentes que facilitan el despliegue de soluciones inteligentes con el fin de obtener y gestionar información del entorno.

Figura 2. Logo de Fiware

El elemento principal que permite a FIWARE recopilar y administrar la información de contexto es el Orion Context Broker, un servidor web que permite peticiones de actualización o suscripción a los datos, utilizando una API de tipo REST sobre el protocolo de transporte HTTP.

FIWARE también proporciona otros elementos, llamados Generic Enabler, que permiten llevar a cabo la tarea de securización del sistema y el procesamiento de la información de contexto producida por los agentes IoT.

2.4.1 Orion Context Broker

Orion Context Broker (OCB) [2] es el componente principal para cualquier solución que se desarrolle bajo el paraguas de FIWARE, ya que mediante él se puede administrar la información de contexto, consultarla y actualizarla. El Context Broker está rodeado a su vez de elementos adicionales que permiten suministrar datos de diversas fuentes y entornos que, trabajando en este elemento se convierten en herramientas para el procesamiento de datos, análisis, control de acceso o la implementación y propia configuración de la plataforma FIWARE.

Figura 3. Logo OCB

Una función principal soportada por el OCB es el de lograr una disociación total entre productores y consumidores de contexto. Los productores de contexto son aquellos que publican datos sin saber qué, dónde y cuándo los consumidores de contexto consumirán los datos publicados; por lo tanto, no necesitan estar conectados a ellos. Por otro lado, los consumidores de contexto consumen información de contexto de su interés, sin que conozcan al productor de contexto que publica un evento en particular. Están interesados en el evento en sí, y no en quien lo generó.

Un elemento de contexto se refiere a la información que es producida u observada y que puede ser relevante para su procesamiento, análisis y extracción de nuevo conocimiento. Tiene asociado un valor definido, que consiste en una secuencia de una o más triplas que se refieren a atributos de un elemento de contexto. Además, proporciona información relevante a una entidad en particular, la cual puede ser un componente físico o parte de una aplicación. Por ejemplo, un elemento de contexto puede contener valores de atributos asociados a una habitación como la última temperatura medida, los metros cuadrados que mide y el color de la pared. Por lo general, elemento de contexto contiene un id (EntityId) y un tipo (EntityType) que identifica exclusivamente a una entidad. Adicionalmente, pueden existir metadatos, vinculados a los atributos a un elemento de contexto. Sin embargo, la existencia de metadatos vinculados a un atributo de elemento de contexto es opcional.

En resumen, un elemento de contexto puede contener los siguientes parámetros:

- Un EntityId y un EntityType único que identifica la entidad a la cual los datos de contexto hacen referencia.
- Una secuencia de uno o más atributos de elementos de datos.
- Metadatos opcionales vinculados a atributos.

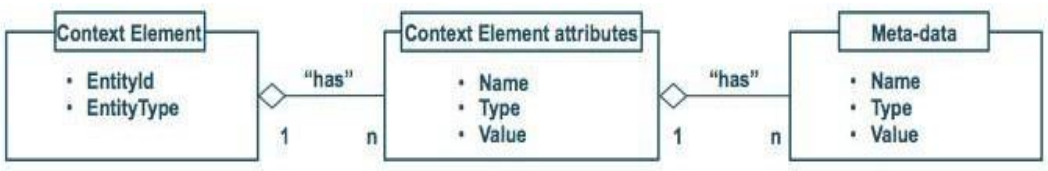


Figura 4. Esquema Context Element

El OCB utiliza la base de datos MongoDB para almacenar el estado actual de las entidades, sin almacenar información histórica de sus cambios. Para este propósito se debe utilizar una base de datos externa al OCB.

Orion no proporciona autenticación "nativa" ni ningún mecanismo de autorización para hacer cumplir el control de acceso, por tanto, para llevar a cabo los diferentes escenarios debemos utilizar una serie de componentes FIWARE para garantizar diferentes niveles de seguridad.

2.4.2 Identity Manager Keyrock

Keyrock [3] es el componente encargado de proporcionar mecanismos para gestionar la identidad de aquellos usuarios que intenten acceder al sistema, con la finalidad de conocer en todo momento quién o qué está accediendo a la información de contexto del sistema.

Keyrock implementa una interfaz gráfica donde el administrador puede registrar aplicaciones, organizaciones y usuarios y asignar diferentes roles a los usuarios creados. Keyrock incorpora una base de datos MySQL para almacenar toda la información relativa a aplicaciones, organizaciones, usuarios, token de acceso y roles.

Para la autenticación, lo habitual, será primero solicitar a Keyrock el token de acceso, indicando el usuario y contraseña en el mensaje de solicitud. En caso de estar registrado dicho par usuario-contraseña en la base de datos, Keyrock dará como respuesta el token de acceso, el cual identificará al usuario.

Por consiguiente, el usuario para solicitar algún recurso en el sistema ha de indicar en la propia solicitud el token de acceso. La finalidad de esto sería que Keyrock decida si se ha autenticado correctamente en función de si el token indicado por el usuario está almacenado en la base de datos

Figura 5. Logo de Keyrock

2.4.3 PEP Proxy Wilma

El FIWARE Generic Enabler PEP Proxy Wilma [4], es el componente que va a permitir la activación del mecanismo de autenticación y autorización de nuestro sistema, siendo el intermediario entre Keyrock, la aplicación, Authzforce y el OCB.

Las solicitudes al proxy deben hacerse con una cabecera especial HTTP Header: X-Auth-Token. Este encabezado contiene el token de acceso obtenido mediante Keyrock.

PEP-Proxy Wilma actuará como Punto de Aplicación de la Política (PEP en sus siglas del inglés), teniendo el siguiente funcionamiento:

1. Intercepta las peticiones de los usuarios a un recurso del OCB.
2. Realiza una petición de autenticación a Keyrock, el cual consultará si dicho usuario está registrado mediante su token y, por consiguiente, emitirá una decisión de acceso.
3. Wilma escuchará la decisión de autenticación, denegando el acceso del usuario o realizando una petición a Authzforce para obtener una decisión de autorización en función del usuario, recurso y acción que pretenda realizar.

Wilma denegará el acceso al usuario o dará como respuesta la petición que ha solicitado consultando de esta manera a Orion.

2.4.4 Authzforce

AuthzForce [5]es un componente FIWARE que proporciona un marco de control de acceso basado en atributos (ABAC), facilitando una API para obtener decisiones de autorización basadas en políticas de autorización y solicitudes de autorización desde un PEP. Authzforce evalúa decisiones de autorización basadas en políticas XACML y atributos relacionados con una solicitud de acceso determinada (por ejemplo, identidad del solicitante, recurso solicitado, acción solicitada), siguiendo la lógica de evaluación de políticas definida en XACML.

Figura 6. Logo Authzforce

Authzforce además de actuar como PDP, puede actuar como punto de administración de políticas (PAP), esto significa que los PolicySets se pueden crear y modificar utilizando llamadas API directamente a Authzforce.

Sin embargo, no existe una GUI para crear o modificar una <PolicySet>. Por tanto, utilizaremos Postman para modificar las políticas ya que aporta cierta simplicidad al proceso, en vez de usar la consola de comandos que resulta más problemático dada la magnitud de algunas políticas.

2.5 Tecnologías y recursos software utilizados.

Para realizar el proyecto, ha sido necesario utilizar diferentes herramientas software, tanto para el despliegue de la plataforma, para el despliegue del entorno en el que se va a instalar y desarrollar las pruebas, para la realización de las propias pruebas y para la propia plataforma.

2.4.5 Máquina Virtual Ubuntu 20.04.0 LTS

Tanto la plataforma como la herramienta de pruebas de carga han sido desplegados en una máquina virtual Ubuntu 20.04.0 LTS [6], sistema operativo Linux. Este sistema operativo ha sido escogido ya que la administración y despliegue de aplicaciones es más sencillo en este entorno que en Windows, y es un sistema operativo con el que hemos trabajado a lo largo de la carrera por lo que el conocimiento sobre este es de gran ayuda para llevar a cabo el trabajo.

Figura 7. Logo Ubuntu 20.04 LTS

2.4.6 VMware Workstation Player 16

VMware Workstation Player 16 [7]es una herramienta que permite ejecutar una máquina virtual con un sistema operativo diferente al que se use en el PC. He escogido esta herramienta ya que es la que hemos estado utilizando a lo largo de la carrera y el conocimiento previo es de ayuda.

Figura 8. WMware Workstation Player 16

2.4.7 Postman

Postman [8] es un servicio que permite simular peticiones REST a una API, cuenta con una interfaz gráfica bastante detallada e intuitiva por lo que su manejo resulta sencillo. Con Postman podemos hacer todo tipo de peticiones como POST, GET, PUT, PATCH …

Ha sido utilizado para comprobar el correcto funcionamiento de la plataforma IoT en las pruebas unitarias, haciendo peticiones a la API de cada módulo por separado, y luego para comprobar el funcionamiento en conjunto, haciendo peticiones de GET para recoger entidades de Orion Context Broker, almacenadas en MongoDB, para almacenar nuevas entidades con POST, para actualizarlas con PATCH u otras de POST para que un usuario se subscriba a una entidad y reciba información sobre el cambio cuando esta sea actualizada.

También ha sido de gran ayuda para el manejo de políticas de Authforce ya que hacerlo por la terminal era bastante farragoso pero con la interfaz gráfica de Postman se ha solucionado ese problema.

Figura 9. Logo de Postman

2.4.8 Docker y Docker-Compose

Docker [9]es un Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contendores software a partir de imágenes previamente creadas, proporcionando una capa adicional de abstracción y permitiendo la virtualización de aplicaciones en múltiples sistemas operativos.

Docker consta de imágenes y contenedores:

- Una imagen es la especificación inerte, inmutable, una foto del estado y de unas piezas de software que incluyen desde la aplicación que queremos ejecutar hasta las librerías y todo lo necesario para que se ejecute encima del sistema operativo que lo contiene.
- Un contenedor es un entorno aislado con la instanciación de una imagen, el cual se puede configurar.

Docker permite la independencia de las aplicaciones que desplegamos en los contenedores, del sistema operativo. Esto se usa cuando por ejemplo queremos desplegar una plataforma en una gran cantidad de equipos, cada uno con un entorno diferente, si tratamos de instalarlos directamente en el sistema, en cada equipo requeriremos ciertos complementos que serán diferentes a los requeridos en otro. Con Docker aislamos las imágenes del sistema pudiendo replicar la plataforma en diferentes entornos sin problemas de compatibilidad.

Figura 10. Logo Docker

También será necesario el uso de la herramienta Docker-Compose [10] para poder desplegar aplicaciones formadas por diferentes contenedores que se relacionan e interactúan entre ellos. Por ejemplo, en este proyecto, Orion Context Broker y MongoDB están relacionados, al igual que Keyrock con MySQL y Authforce.

Figura 11. Logo Docker-Compose

2.4.9 MongoDB y MySQL

MongoDB [11]es una base de datos orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Orion Context Broker va a utilizar MongoDB para almacenar toda la información relativa a las entidades.

Figura 12. Logo MongoDB

Figura 13. Logo MySQL

MySQL [12]es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo

2.4.10 Wireshark

Wireshark [13]ha sido utilizado para capturar los mensajes que viajan entre los diferentes módulos del sistema, para comprobar el funcionamiento de este y analizar el contenido de los mensajes, entendiendo así el comportamiento interno de la plataforma, como veremos en el siguiente apartado.

Figura 14. Logo Wireshark

2.4.11 Herramientas de Testing

Existen diferentes herramientas para el estudio del rendimiento en un sistema o aplicación, y con ellas se puede, desde lanzar diversas peticiones para poner aprueba una API hasta cargar una red de o una base de datos para el posterior estudio del rendimiento, siendo esto lo que más nos interesa.

2.4.11.1 Análisis de herramientas existentes

Existen diferentes herramientas para el estudio del rendimiento en un sistema o aplicación, y con ellas se puede, desde lanzar diversas peticiones para poner aprueba una API hasta cargar una red de o una base de datos para el posterior estudio del rendimiento, siendo esto lo que más nos interesa. Las principales herramientas estudiadas son:

JMeter: Sin duda la herramienta más recomendable para realizar pruebas de carga y estudiar el rendimiento, contando con una gran cantidad de complementos para llevarlas a cabo y dibujar los resultados en gráficas o mostrar tablas con los mismos. Lleva bastante tiempo siendo utilizada en el mundo laboral.

- SoapUI: Potente herramienta de testeo de API bastante utilizada en el mundo laboral. Pese a que es capaz de realizar pruebas de carga, son bastante específicas, no tan flexibles ni diversas como en JMeter.
- LoadRunner: Sin duda una de las mejores herramientas para pruebas de rendimiento en plataformas y sistemas. Es una herramienta de pago, por lo que he priorizado JMeter al ser de código abierto. A pesar de ser bastante potente, JMeter es más fácil de utilizar, siendo bastante sencillo añadir complementos, guardar variables, se puede realizar scripting sin necesidad de conocimientos previos, además de mostrar los resultados en gráficas de manera bastante intuitiva, con una fácil interpretación de los resultados. Todo esto hace que muchas empresas prefieran JMeter en vez de LoadRunner.
- Blazemeter: Orientado principalmente a aplicaciones móviles, aplicaciones web o testeo de API en continuo desarrollo, es decir, agrupando los resultados y comprobando la evolución del proyecto. No sería mala idea utilizarlo para comprobar mejorías en el desarrollo de la plataforma, pero dado que nuestro objetivo es evaluarla en su forma actual y conocer sus límites de rendimiento actuales con el fin de mejorarlos en caso de ser necesario, o de tenerlos en cuenta para no sobrepasarlos (monitorizar el sistema), es más preferible utilizar JMeter.

Finalmente, tras evaluar las ventajas y desventajas de cada una, la elegida es JMeter, la cual es bastante potente, cuenta con una gran cantidad de recursos y oportunidades de pruebas, cumpliendo sobradamente con lo que necesitamos para nuestro caso.

2.4.11.2 JMeter

JMeter [14]es un software de código abierto utilizado en la realización de test de carga para comprobar el comportamiento y medir el rendimiento de aplicaciones y sistemas.

Con JMeter se pueden realizar pruebas de aplicaciones web, mandado peticiones HTTP/HTTPS, también se pueden testear webservices basados en SOAP/REST, establecer conexión a una base de datos y llevar a cabo acciones en la misma como cargarla con un elevado número de entidades o solicitudes y así comprobar el comportamiento, pudiendo mostrar gráficas de cómo el sistema evoluciona en cuanto a velocidad en las respuestas, errores, número de hilos…

Figura 15. Logo Jmeter

3 ESTUDIO DEL SISTEMA

Uno de los pasos más importantes que se debe dar a la hora de resolver un problema es el correcto análisis del mismo. Esta sección refleja el trabajo previo que se realizó para nuestro trabajo y que debería de servir de ejemplo para cualquier proceso de testing de un sistema. Analizar cada módulo y su funcionamiento es vital para diseñar las diferentes pruebas y sus limitaciones de rendimiento. En la siguiente imagen se muestra el sistema y cómo los diferentes módulos se relacionan entre sí:

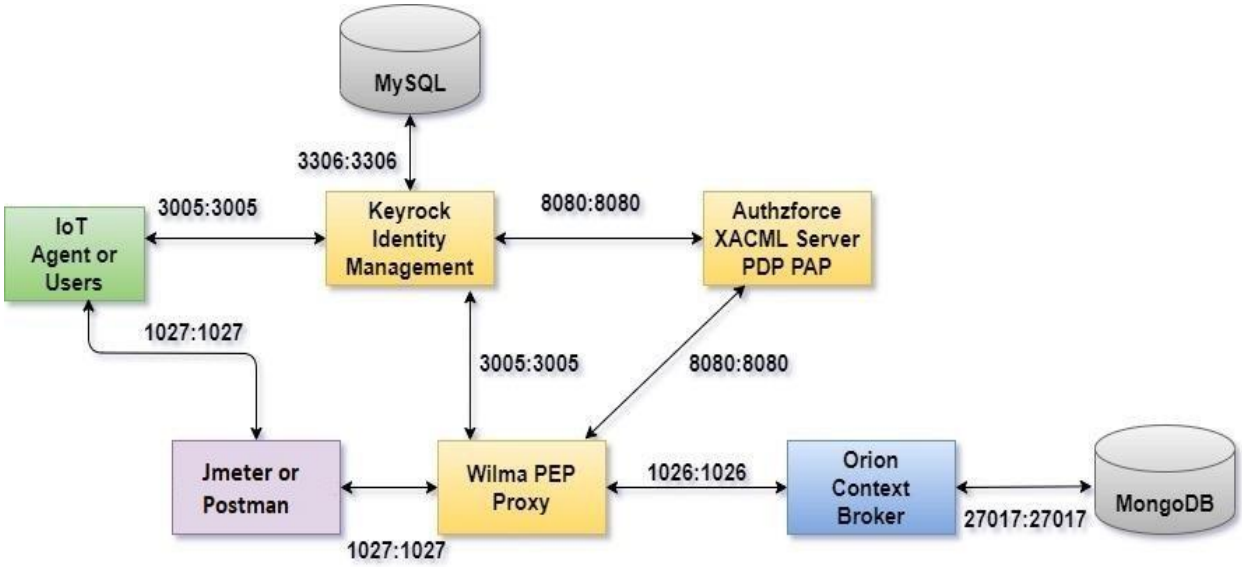


Figura 16. Esquema de la plataforma a testear

A grandes rasgos, los agentes IoT o usuarios, que son emulados por una herramienta de pruebas, en este caso JMeter, dirigirá las peticiones a Wilma PEP Proxy, en estas peticiones irá el token de acceso que previamente Keyrock debe haber otorgado a los usuarios previamente registrados, cuya información se almacena en la base de datos MySQL. Wilma validará este token de acceso preguntando a Keyrock. En caso de ser válido, consultará con Authforce si el usuario o agente está autorizado para realizar determinada acción demandada en la petición y, en caso afirmativo, redirigirá la petición al Orion context Broker, quien procesará la petición y responderá de una forma u otra dependiendo de la acción que se requiera, consultando la base de datos MongoDB donde se encuentran las entidades, ya sea para actualizar una (PATCH), añadir una nueva (POST), obtener una entidad (GET) o establecer una relación de subscripción de un usuario a una entidad (SUBSCRIBE).

A continuación se muestra un diagrama con las interacciones entre los diferentes módulos en el caso de un usuario enviando una petición GET para obtener la información de un sensor.

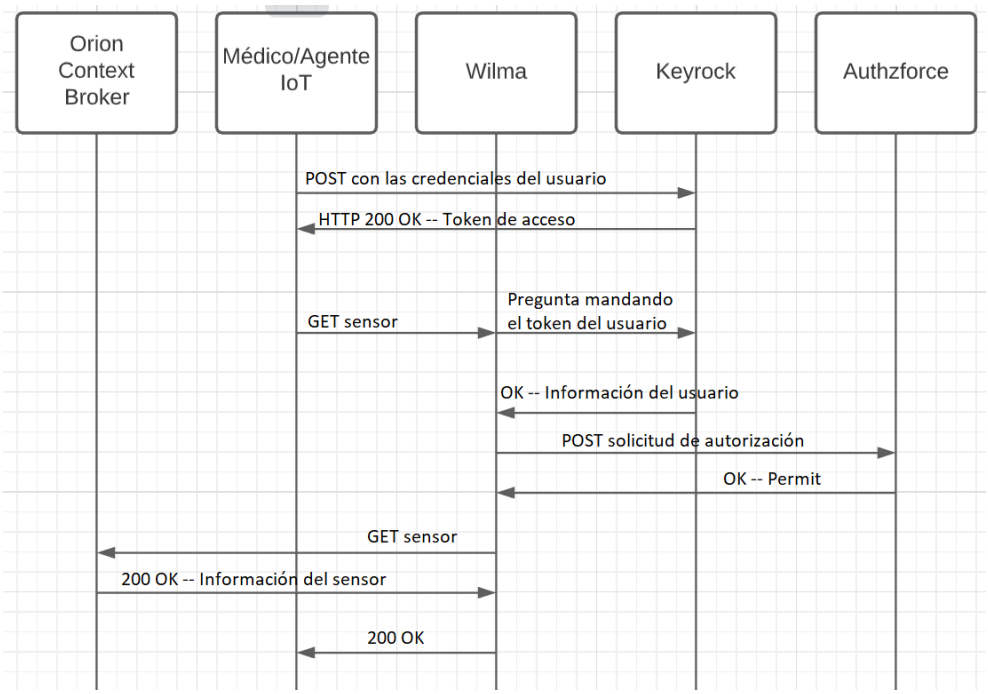


Figura 17. Diagrama funcionamiento plataforma. Ejemplo GET

A continuación, se describirá cada componente de manera individual.

3.1 Orion Context Broker

Orion Context Broker será el manejador de las solicitudes de obtención, publicación, actualización y subscripción que los agentes o usuarios lanzarán. Orion estará protegido frente a vulnerabilidades gracias a Keyrock y Authforce, encargados de autenticar a los usuarios y agentes que desean acceder al sistema, y de validar sus peticiones, ya que las políticas de Authforce se diseñan para limitar las acciones que un usuario puede realizar y proteger el sistema.

Para este proyecto se va a trabajar con entidades que contienen datos de la actividad física de un paciente, a partir de la información recogida y enviada a un agente IoT gracias a los sensores de las camisetas inteligentes. Esta información puede ser consultada, actualizada por un usuario (médico) que puede subscribirse a los cambios que experimente una entidad. Las entidades cuentan con una serie de atributos:

- "id": Representa el identificador de cada entidad
- "type": En este caso serán todos de "ActividadFisica"
- "publisher": Es el agente IoT que publica la entidad.
- "id_patient": Es el usuario que utiliza la camiseta.
- "timestamp": Indica la hora a la que se realizó la prueba.
- "id_device": Identifica la camiseta usada.
- "organization": Organización a la que pertenece el paciente y la camiseta.
- "accumulates_metabolic_expenditure". Se trata del gasto calórico total del paciente.
- "instant_metabolic_expenditure". Es el gasto calórico instantáneo en el momento en el que se envía el mensaje.

```
{
  "id":"urn:ngsi-ld:sensor:001","type":"ActividadFisica",
 "publisher": "Agente2001",
 "id_patient": "222222222A",
 "timestamp": {"type": "DateTime", "value":"2020-07-24T09:30:23.238Z"},
  "id_device" : "11:1E:C0:25:E6:99",
  "organization": "ResidenciaSevilla",
  "accumulated_metabolic_expenditure" :{"type": "float", "value": "5.59"},
  "instant_metabolic_expenditure":{"type": "float", "value": "0.05"}
}
```


Figura 18. Ejemplo de entidad de tipo "ActividadFisica"

Orion Context Broker será un punto que cargaremos, llenando de entidades la base de datos MongoDB donde almacena la información de las entidades y acumulando una serie de usuarios de manera paralela que manden peticiones, de este modo comprobaremos cómo afecta la carga de este módulo al rendimiento del sistema.

3.2 Keyrock

Keyrock se ha configurado para escuchar peticiones en el puerto 3005, y su función consiste en garantizar la identidad de todo aquel usuario o agente IoT que quiera acceder al sistema para lo cual tendrá que estar registrado previamente.

El registro de los usuarios, organizaciones, PEP Proxy, roles, etc., lo puede realizar el administrador del sistema mediante la interfaz gráfica de Keyrock. Pero se ha optado a crear un estado inicial de la base de datos MySQL donde ya se configura todo el registro. El estado inicial se explica en el Anexo B.

Wilma acudirá a Keyrock para autenticar al usuario o agente IoT.

Cuando un agente envía a Keyrock sus credenciales (Figura 19), este verifica al usuario buscando en su base de datos MySQL las credenciales, y responde, en caso afirmativo, con el token que el usuario debe utilizar para acceder al sistema en sus peticiones (Figura 20).


```
▾ Hypertext Transfer Protocol
  ▾ POST /oauth2/token HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): POST /oauth2/token HTTP/1.1\r\n]
      Request Method: POST
      Request URI: /oauth2/token
      Request Version: HTTP/1.1
    Connection: keep-alive\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Accept: application/json\r\n
  ▸ Authorization: Basic ZXNjZW5hcmlvX3Nhbml0YXJpbzplc2NlbmFyaW9fc2FuaXRhcmlvX3NlY3JldA==\r\n
  ▸ Content-Length: 60\r\n
    Host: localhost:3005\r\n
    User-Agent: Apache-HttpClient/4.5.12 (Java/11.0.11)\r\n
    \r\n
    [Full request URI: http://localhost:3005/oauth2/token]
    [HTTP request 1/1]
    [Response in frame: 6]
    File Data: 60 bytes
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▾ Form item: "grant_type" = "password"
      Key: grant_type
      Value: password
  ▾ Form item: "username" = "Plato21@tfg.com"
      Key: username
      Value: Plato21@tfg.com
  ▾ Form item: "password" = "test"
      Key: password
      Value: test
```


Figura 19. Petición de token a Keyrock por parte de agente o usuario

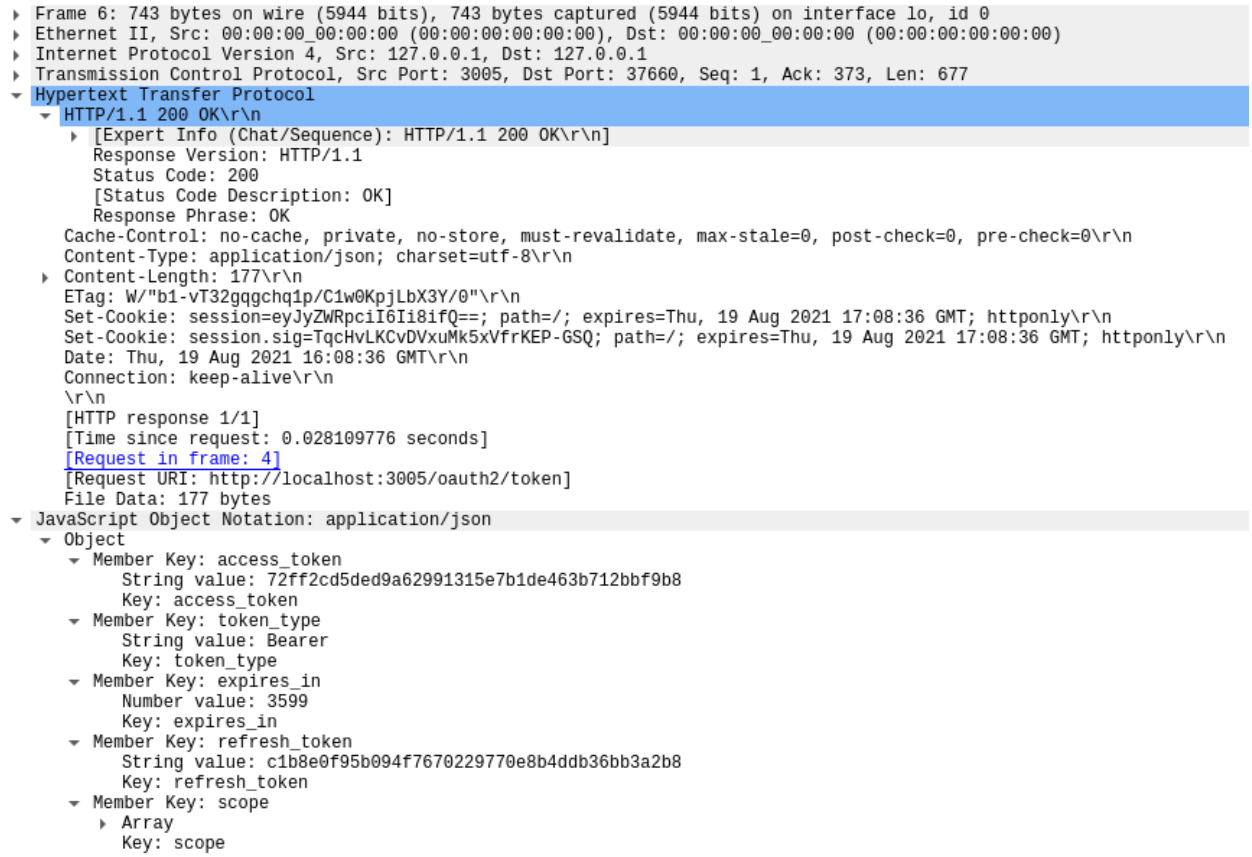


Figura 20. Respuesta de Keyrock con token tras validación

3.3 MySQL

En la base de datos MySQL se almacenará la información relativa a los usuarios y agentes registrados en el sistema, la cual será utilizada por Keyrock para validar la identidad de los usuarios que tratan de acceder al sistema. Contaremos con diferentes tablas dentro de esta base de datos, las cuales aumentarán su tamaño conforme más usuarios o agentes IoT estén registrados, debido a esto, el elemento crítico en este módulo del sistema será la cantidad de usuarios registrados, tanto agentes (camisetas con sensores IoT) como médicos registrados en la plataforma.

3.4 MongoDB

MongoDB será la base de datos que utilizará Orion Context Broker para almacenar la información de las diferentes entidades existentes, y sobre la cual llevará a cabo las operaciones de consulta e inserción de nuevas entidades, por lo que esta base de datos será un punto importante en el estudio del sistema para comprobar si su carga afecta o no al rendimiento del sistema, debido a la posibilidad de que se lleve de una gran cantidad de datos.

MongoDB presenta un esquema libre, es decir, cada entrada o registro almacenado en la base de datos puede tener un esquema de datos diferente. Estos registros pueden cambiar en cualquier momento siendo actualizados. A cada registro se le denomina documento. Los documentos se componen de pares clave-valor, con una composición similar a una estructura JSON (Figura 18).

En MongoDB los documentos son almacenados en formato BSON, una versión binaria de JSON que acelera la búsqueda de datos ya que guarda información como longitudes o índices que facilitan el trabajo posterior de búsqueda de entidades.

En este punto del sistema, para MongoDB el parámetro que afectará al rendimiento será la cantidad de entidades

que existan, es decir, la cantidad de información recolectada por los sensores de las camisetas y almacenadas en esta base de datos. Se presumen que a medida que MongoDB esté más cargado, se ralentizará la búsqueda de entidades.

3.5 PEP Proxy Wilma

Wilma está configurado para escuchar peticiones en el puerto 1027. Cuando intercepte una petición, Wilma comprobará la autenticación del usuario consultando a Keyrock, proporcionándole a este componente el token de acceso indicado por el usuario. Si el usuario está registrado, Wilma enviará a Authzforce una solicitud de decisión de autorización para comprobar si el usuario puede realizar una consulta, publicación o suscripción de una entidad. En dicha solicitud, Wilma indicará una serie de parámetros de gran importancia para Authzforce:

- El rol que tiene el usuario que está intentando acceder. Wilma obtiene este valor a partir de la respuesta que le daKeyrock al ser autenticado correctamente.
- El id de la aplicación donde está registrado el usuario.
- La acción que pretende realizar, por ejemplo, POST, GET, DELETE, PATCH.
- El recurso que pretende acceder el usuario, en nuestro proyecto será /v2/entities o /v2/subscriptions
- Se enviarán atributos que tratan el tiempo y aspectos dinámicos del escenario de control de acceso, como por ejemplo los atributos expuestos en la publicación de una entidad.

Siendo Wilma el punto central del sistema por el que pasa toda la información, se describirá aquí un ejemplo básico del sistema con los mensajes filtrados por Wireshark donde vemos el funcionamiento de Wilma y del sistema completo. En la Figura 21 se muestra un ejemplo de una petición POST, creada con JMeter, de una nueva entidad, formulada por un supuesto agente IoT previamente registrado en Keyrock. Esta petición el agente IoT la dirige al proxy Wilma, [Full request URI: http://localhost:1027/v2/entities?options=keyValues], que con estos parámetros y el X-Auth-Token, validará al usuario preguntando a Keyrock, y posteriormente a Authforce para validar la acción, en función del valor de los parámetros en cuestión.

Figura 21. Petición POST a Wilma por parte de agente IoT

Cuando Wilma recibe esta petición de un agente IoT, preguntará a Keyrock utilizando un mensaje como el de la Figura 21, portando el token que desea validar.

Figura 22. Wilma pregunta a Keyrock para que autorice a un usuario con un token

Keyrock responde a Wilma con un mensaje en el que especifica toda la información del usuario, su organización, role, app_id, email, nombre, id, incluso con el dominio de Authzforce que debe ser utilizado para validar la

acción (Figura 22).

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
       [HTTP/1.1 200 OK\r\n]
       [Severity level: Chat]
       [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Cache-Control: no-cache, private, no-store, must-revalidate, max-stale=0, post-check=0, pre-check=0\r\n
  Content-Type: application/json; charset=utf-8\r\n
  Content-Length: 280\r\n
  ETag: W/"118-yMkGDg611xgPq6NBhQMi6NQctSQ"\r\n
  Set-Cookie: session=eyJyZWRpciI6Ii8ifQ==; path=/; expires=Thu, 19 Aug 2021 17:08:36 GMT; httponly\r\n
  Set-Cookie: session.sig=TqcHvLKCvDVxuMk5xVfrKEP-GSQ; path=/; expires=Thu, 19 Aug 2021 17:08:36 GMT; httponly\r\n
  Date: Thu, 19 Aug 2021 16:08:36 GMT\r\n
  Connection: close\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.024277131 seconds]
  [Request in frame: 16]
  [Request URI: http://localhost:3005/user?access_token=72ff2cd5ded9a62991315e7b1de463b712bbf9b8&authzforce=true]
  File Data: 280 bytes
JavaScript Object Notation: application/json
  Object
    Member Key: organizations
    Member Key: displayName
    Member Key: roles
       Array
       Key: roles
    Member Key: app_id
       String value: escenario_sanitario
       Key: app_id
    Member Key: trusted_apps
    Member Key: isGravatarEnabled
    Member Key: image
       String value:
       Key: image
    Member Key: email
       String value: Plato21@tfg.com
       Key: email
    Member Key: id
       String value: Plato2
       Key: id
    Member Key: authorization_decision
    Member Key: app_azf_domain
       String value: ffsR-txyEeqvSwJCrBIBDA
       Key: app_azf_domain
    Member Key: attributes
       Object
       Key: attributes
    Member Key: username
       String value: Plato2
```

Figura 23. Respuesta de Keyrock a Wilma con los datos del usuario o agente

Una vez Wilma ha recibido el 200 OK, ha validado el usuario y se dispone a consultar con Authzforce para comprobar si el usuario está autorizado a realizar determinada acción. El mensaje que manda Wilma a Authzforce es el tipo que se muestra en la Figura 23.

Figura 24. Wilma consulta a Authzfoce la validez de la acción demandada

Con un campo llamado *eXtensible Markup Language*, donde viaja la información del usuario o agente que va a ser evaluada en Authzforce:

Frame 26: 2641 bytes on wire (21128 bits), 2641 bytes captured (21128 bits) on interface lo, id 0

Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 46492, Dst Port: 8080, Seq: 1, Ack: 1, Len: 2575

Hypertext Transfer Protocol

eXtensible Markup Language

<?xml

version="1.0"

encoding="UTF-8"

standalone="yes"

?>

<Request

xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"

CombinedDecision="false"

ReturnPolicyIdList="false">

<Attributes

Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">

</Attributes>

<Attributes

Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">

<Attribute

AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"

IncludeInResult="false">

<AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">

```
            escenario_sanitario
            </AttributeValue>
        </Attribute>
    <Attribute
        AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id"
        IncludeInResult="false">
        <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            /v2/entities
            </AttributeValue>
        </Attribute>
    </Attributes>
<Attributes
    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        IncludeInResult="false">
        <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            POST
            </AttributeValue>
        </Attribute>
    </Attributes>
<Attributes
    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:id_device"
        IncludeInResult="false">
        <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            00:1E:C0:25:E6:99
            </AttributeValue>
        </Attribute>
    </Attributes>
<Attributes
    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:organization"
```

```
        IncludeInResult="false">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          HospitalCentral
          </AttributeValue>
        </Attribute>
      </Attributes>
    <Attributes
      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
      <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:id_patient"
        IncludeInResult="false">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          954179770F
          </AttributeValue>
        </Attribute>
      </Attributes>
    <Attributes
      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
      <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:publisher"
        IncludeInResult="false">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">
          Agente0
          </AttributeValue>
        </Attribute>
      </Attributes>
    </Request>
```

Podemos observar como este agente, llamado Agente0, con un id para el paciente 954179770F, con id_device: 00:1E:C0:25:E6:99, del HospitalCentral con petición POST dirigida a /v2/entities/ será evaluado en Authzforce con estos valores para saber si puede realizar la acción o no.

En caso afirmativo, Authzforce devuelve un 200 OK aprobando la acción (Figura 25).

```
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 200 \r\n
    ▾ [Expert Info (Chat/Sequence): HTTP/1.1 200 \r\n]
        [HTTP/1.1 200 \r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
    Date: Thu, 19 Aug 2021 16:08:38 GMT\r\n
    Content-Type: application/xml\r\n
  ▸ Content-Length: 446\r\n
    Connection: close\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.179743462 seconds]
    [Request in frame: 26]
    [Request URI: http://localhost:8080/authzforce-ce/domains/ffsR-txyEeqvSwJCrBIBDA/pdp]
    File Data: 446 bytes
▾ eXtensible Markup Language
  ▾ <?xml
      version="1.0"
      encoding="UTF-8"
      standalone="yes"
      ?>
  ▾ <ns5:Response
      xmlns="http://authzforce.github.io/core/xmlns/pdp/6.0"
      xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
      xmlns:ns3="http://www.w3.org/2005/Atom"
      xmlns:ns4="http://authzforce.github.io/pap-dao-flat-file/xmlns/properties/3.6"
      xmlns:ns5="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    ▾ <ns5:Result>
      ▾ <ns5:Decision>
          Permit
          </ns5:Decision>
        </ns5:Result>
      </ns5:Response>
```

Figura 25. Respuesta afirmativa de Authzforce.

A continuación Wilma redirige la petición a Orion Context Broker (Figura 26)

```
▾ Hypertext Transfer Protocol
  ▾ POST /v2/entities?options=keyValues HTTP/1.1\r\n
    ▾ [Expert Info (Chat/Sequence): POST /v2/entities?options=keyValues HTTP/1.1\r\n]
        [POST /v2/entities?options=keyValues HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: POST
    ▾ Request URI: /v2/entities?options=keyValues
        Request URI Path: /v2/entities
      ▾ Request URI Query: options=keyValues
          Request URI Query Parameter: options=keyValues
      Request Version: HTTP/1.1
    user-agent: Apache-HttpClient/4.5.12 (Java/11.0.11)\r\n
    accept: application/json\r\n
    Content-Type: application/json\r\n
    x-auth-token: 72ff2cd5ded9a62991315e7b1de463b712bbf9b8\r\n
    X-Nick-Name: Plato2\r\n
    X-Display-Name: \r\n
    X-Roles: []\r\n
    X-Organizations: []\r\n
    X-Eidas-Profile: [object Object]\r\n
    X-App-Id: escenario_sanitario\r\n
    x-forwarded-for: ::ffff:127.0.0.1\r\n
    Host: localhost:1026\r\n
  ▸ Content-Length: 472\r\n
    Connection: close\r\n
    \r\n
    [Full request URI: http://localhost:1026/v2/entities?options=keyValues]
```

Figura 26. Wilma redirige la petición a Orion Context Broker

Este mismo mensaje, contiene un JSON con la información que va a ser almacenada en MongoDB al crear la nueva entidad (detalle en Figura 27).

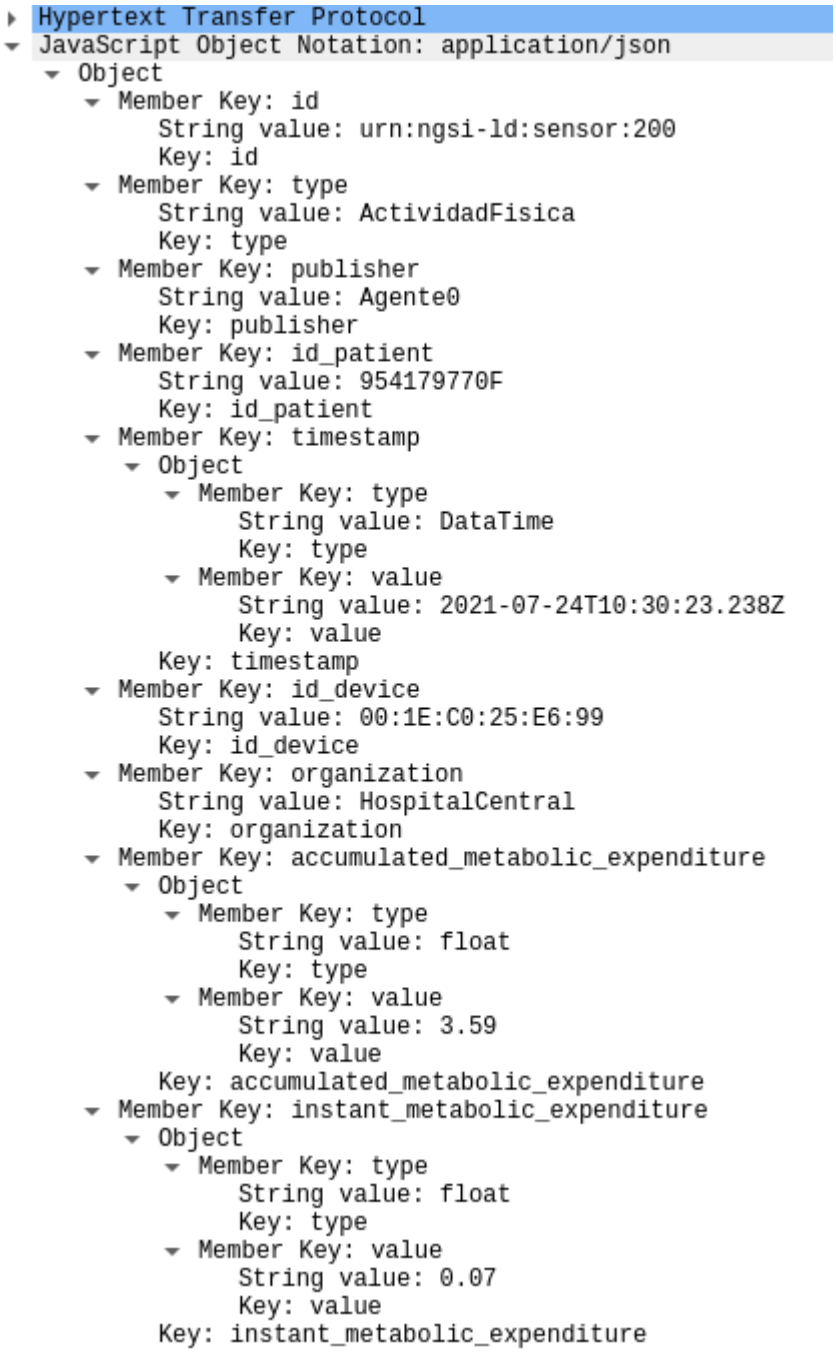


Figura 27. Información de la entidad a crear con el POST que viaja a Orion Context Broker

Finalmente, Orion Context Broker responde con el resultado de la operación, en este caso “created” (Figura 28) y Wilma termina su función reenviando este mensaje al usuario o agente que realizó la petición (Figura 29).

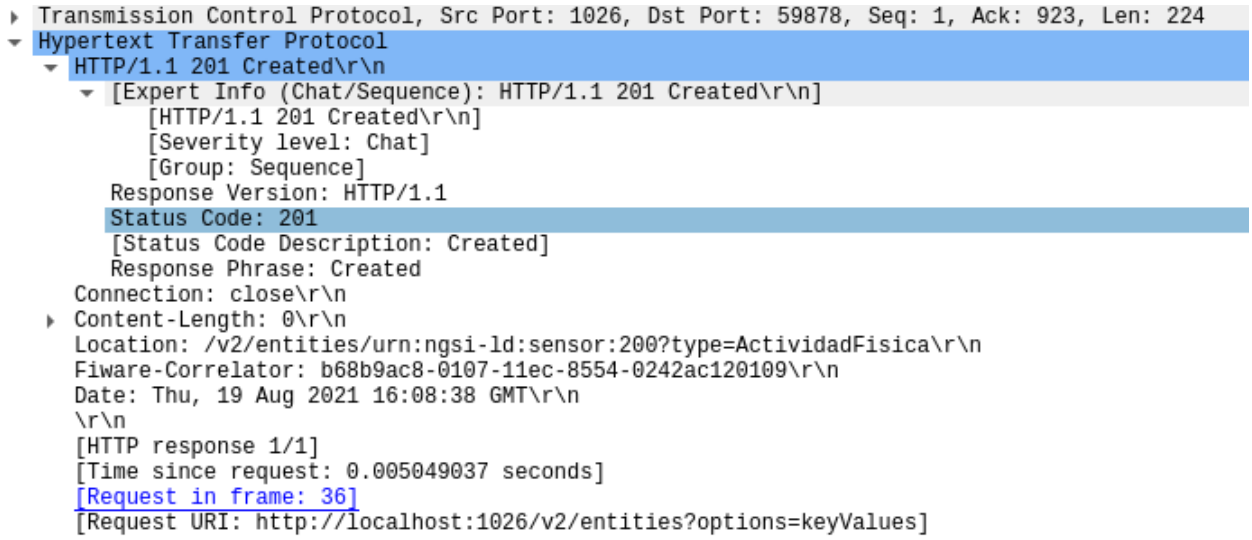


Figura 28. Respuesta de Orion Context Broker a Wilma

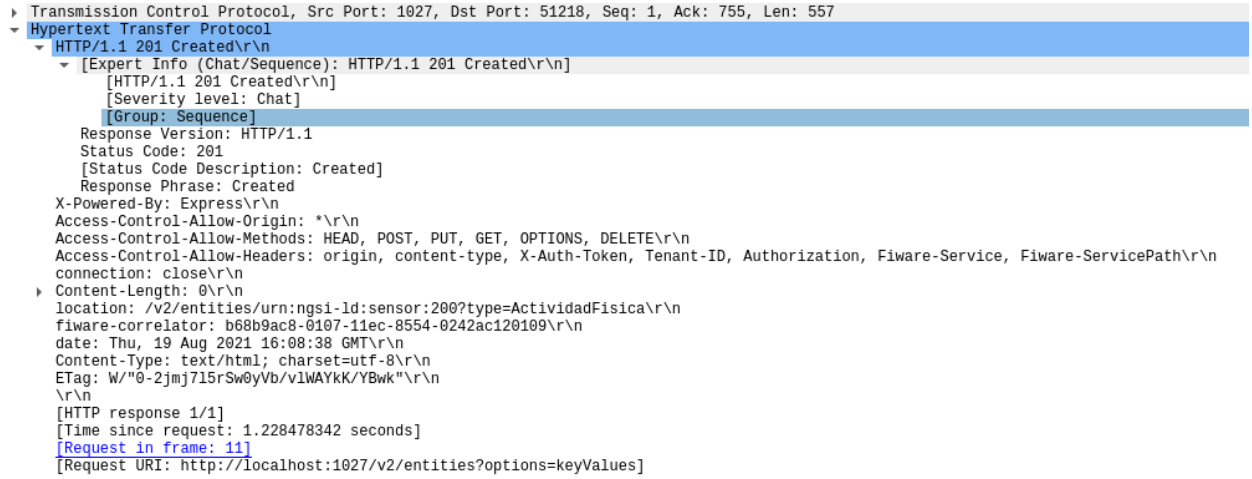


Figura 29. Reenvío del mensaje por parte de Wilma al agente o usuario demandante

Todo este es el trabajo de Wilma, actuar como Proxy para distribuir los mensajes por la aplicación. Dependiendo de la acción este proceso puede variar en algún mensaje, pero ese es el funcionamiento básico.

Este estudio del intercambio de mensajes y de la información que viaja en ellos ha sido útil para el diseño de políticas para poner a prueba Authzforce y así saber qué campos e información estaba siendo tenida en cuenta y así confeccionar políticas más complejas y tediosas, que veremos si afecta o no al rendimiento del sistema.

3.6 Authzforce

Authzforce se encargará de verificar si el usuario puede realizar determinada acción, basándose en la información recibida por Wilma en el mensaje (Figura 24) X Su decisión estará supeditada a una serie de políticas.

Authzforce poseerá un dominio de políticas con una serie de PolicySets y de VariableDefinitions. Estas políticas se escriben en el lenguaje XACML. Un ejemplo sería la política siguiente, en la que se verifica el recurso es escenario_sanitario, que la acción será PATCH y la url a la que va dirigida sea /v2/entities, gracias al método de MatchId string-equal:

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicySetId="f8194af5-8a07-486a-9581-c1f05d05483c"
    Version="93" PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
    <Description>Politicas para escenarios sanitarios</Description>
    <Target />
    <Policy PolicyId="escenario_sanitario_suscripcion" Version="1.0"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
        <Description>Politicas para la suscripcion por parte de los medicos</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">escenario_sanitario
                        </AttributeValue>
                        <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Rule RuleId="Reglas_Medicos_Hospital_Central_suscripcion" Effect="Permit">
            <Description>Reglas medicos del Hospital Central suscripcion</Description>
            <Target>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">PATCH</AttributeValue>
                            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                                AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                                DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
                        </Match>
                    </AllOf>
                </AnyOf>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-starts-with">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">/v2/entities
                            </AttributeValue>
                            <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                                AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id"
                                DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
                        </Match>
                    </AllOf>
                </AnyOf>
            </Target>
        </Rule>
    </Policy>
</PolicySet>
```

Figura 30. Ejemplo de política básica de Authzforce

La política debe ser activada para que funcione, no obstante, en nuestro caso, introduciremos en la base de datos el dominio que se encontrará activo y la política, y lo que haremos será añadir diferentes versiones de la misma con Postman. De este modo actualizaremos las políticas para las diferentes pruebas de manera sencilla y sin tener que añadir diferentes dominios o políticas que deban ser activadas.

4 RESULTADOS

4.1 Puntos de carga

Una vez hemos analizado el funcionamiento del sistema, estamos preparados para preparar nuestras pruebas de carga que van a analizar el rendimiento del mismo. Para someter nuestro sistema bajo carga vamos a utilizar la ya nombrada herramienta: JMeter. Postman será utilizada para cambiar las políticas del sistema y así poner a prueba el módulo Authzforce.

Tanto la carga del sistema de datos para su posterior análisis de rendimiento, como las propias pruebas de rendimiento, han sido lanzadas con JMeter. Por ejemplo, si queremos saber cómo se comporta la base de datos cuando posee 1000 usuarios registrados, con JMeter conectaremos con la base de datos, introduciremos la información de los 1000 usuarios y posteriormente, también con JMeter trataremos de poner a funcionar la base de datos con peticiones para comprobar cómo rinde.

Se han identificado 5 parámetros en nuestro sistema cuya carga puede afectar al rendimiento:

- Petición de token a Keyrock.
- Base de datos MySQL.
- Base de datos MongoDB.
- Políticas de Authzforce.
- Pruebas de tensión, con el fin de encontrar el límite de peticiones simultáneas que se soporta.

Realizaremos diferentes tipos de pruebas para cada parámetro, tanto lineales, con peticiones que llegan una detrás de otra, y paralelas, con usuarios que envían sus peticiones a la vez. Centraremos el estudio principalmente en ver cómo varía el retraso de la respuesta del sistema y el porcentaje de errores.

4.1.1 Petición de token a Keyrock

El paso previo a cualquier petición que un agente o usuario realice al sistema es pedir un token a Keyrock. Así, se estudiará el caso en el que una gran cantidad de usuarios pidan su token a la vez a Keyrock para comprobar cómo afecta al rendimiento. Esta prueba no afecta al sistema completo ya que son peticiones que se realizan previamente al envío de cualquier petición a Wilma, ya que se requiere para ello el token previo que entrega Keyrock como se explicó previamente.

Para realizar estas peticiones hemos utilizado una HTTP Request al puerto 3005 que contiene el usuario y contraseña de los agentes o usuarios registrados en el sistema que desean obtener su token de acceso. Estos usuarios deben haber sido previamente registrado en la base de datos MySQL para que Keyrock les otorgue su token.

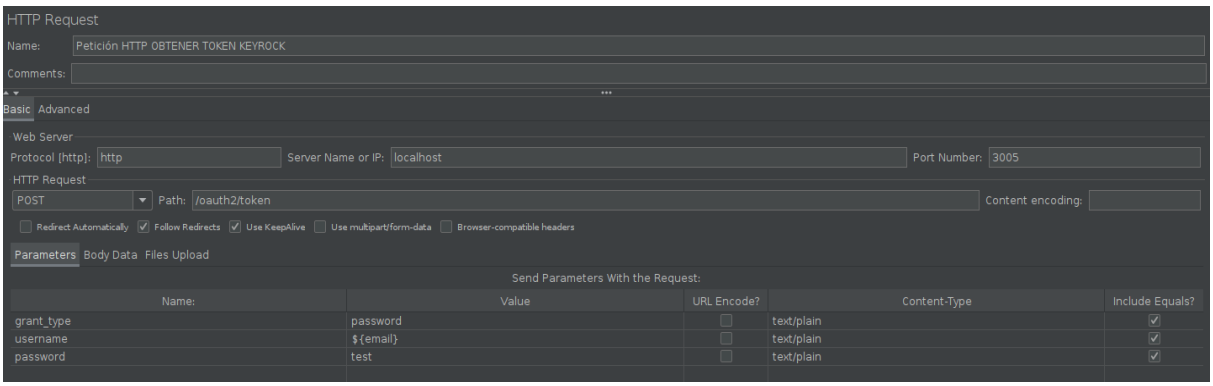


Figura 31. Petición de token a Keyrock

Se realizarán pruebas de carga tanto lineales como en paralelo para comprobar el rendimiento.

4.1.2 Base de datos MySQL

Cada vez que un usuario accede al sistema con su token, Wilma necesita preguntarle a Keyrock si el token es válido y Keyrock consulta la base de datos MySQL para verificarlo. Cargaremos esta base de datos con grandes cantidades de usuarios para ver si esto afecta al rendimiento del sistema.

Para cargar la base de datos de MySQL nos conectamos con JMeter gracias al conector JDBC Connection Configuration. Para usar este conector hacen falta algunos pasos previos que serán explicados en el Anexo D,

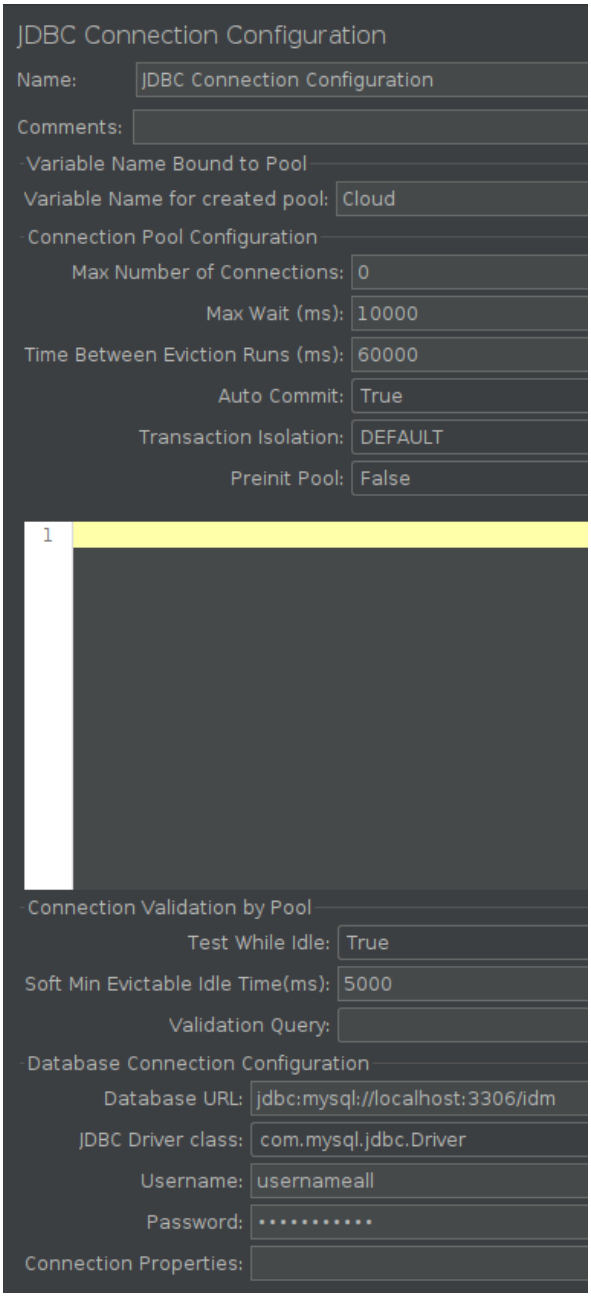


Figura 32. Conexión MySQL JMeter

Con este complemento de JMeter nos conectamos a la base de datos con un Username y Password que previamente hemos introducido con un script y hemos habilitado para que JMeter pueda acceder (el Anexo D).

Una vez establecida la conexión con la base de datos, introducimos las sentencias que deseamos, utilizando el

nombre de la Pool que hemos creado, en este caso “Cloud”:

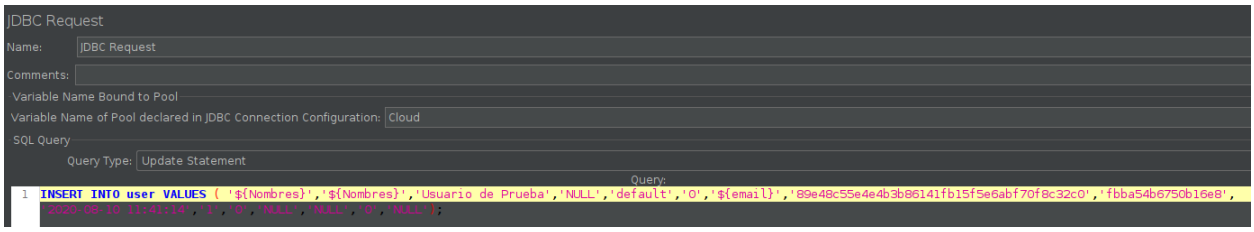


Figura 33. INSERT de users

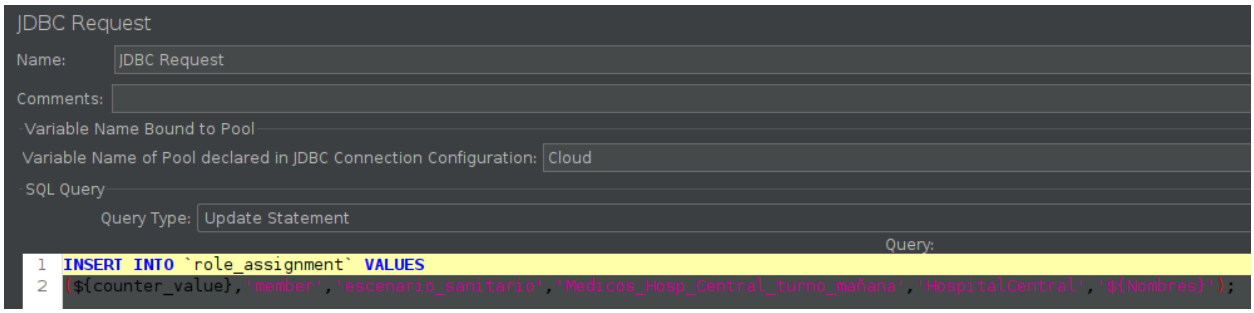


Figura 34. INSERT de roles de los usuarios

Introduciremos diferentes cantidades de usuarios en MySQL para comprobar el rendimiento. Para cargarlo con usuarios válidos que puedan ser utilizados en las pruebas de login posteriormente, se ha creado un fichero de formato CSV que contiene cerca de 10.000 nombres y emails, de este modo podemos utilizar el mismo fichero para el registro y la posterior obtención de token para una gran cantidad de usuarios.

4.1.3 Base de datos MongoDB

Orion Context Broker utiliza esta base de datos para almacenar la información de los sensores de cada agente IoT. Así que querremos también conocer cómo afecta al rendimiento la cantidad de entidades almacenadas en esta base de datos. Para llenar de entidades MongoDB hemos utilizado una petición HTTP con el método POST dirigida directamente al puerto 1026, con el JSON en el cuerpo del mensaje:

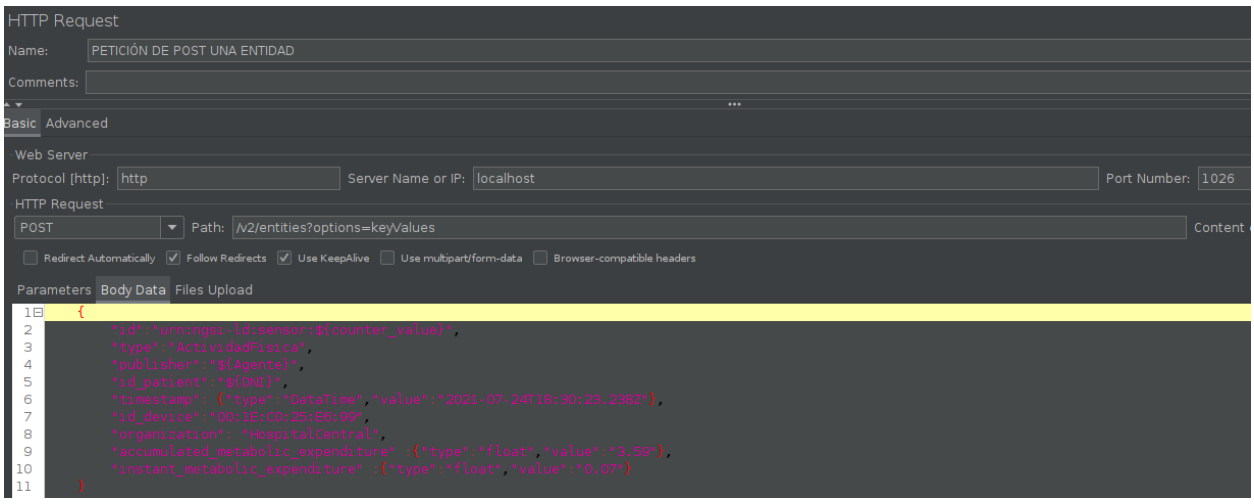


Figura 35. Carga MongoDB

Este JSON se rellena con los datos de un fichero CSV que contiene una gran cantidad de Agentes con diferente DNI. Además de un contador que va creciendo para que cada POST sea de un sensor diferente.

4.1.4 Políticas de Authzforce

Authzforce debe evaluar en función de las políticas previamente declaradas si la realización de determinada actividad es posible o no para determinado agente o usuario. Añadiendo políticas de gran longitud comprobaremos si esto afecta al rendimiento del sistema.

Compararemos el rendimiento del sistema cuando no hay carga de políticas, y cuando tenemos una política de casi 50.000 líneas, para comprobar si la lectura de esta política afecta o no.

Las políticas serán añadidas utilizando Postman ya que su interfaz facilita la tarea además de mostrarlas de manera visualmente sencillo (Figura 35).

Figura 36. Ejemplo de añadir Política a Authzforce

4.1.5 Pruebas de tensión

También analizaremos cómo afecta al sistema la simultaneidad de peticiones, es decir, una gran cantidad de usuarios o agentes emitiendo peticiones a la vez. Así comprobaremos cuánta carga simultánea es capaz de soportar nuestro sistema, y de este modo comprobaremos si el Proxy Wilma es capaz de soportar todo ese manejo de peticiones, o si es otro componente el que se satura antes.

Para esta labor utilizaremos un complemento que añadimos a JMeter, explicado en el Anexo D, con el que se pueden hacer pruebas de carga ascendentes, donde el número de usuarios que envía peticiones al sistema aumenta gradualmente, en este caso hacemos la prueba con todos los módulos descargados, ya que el objetivo es comprobar si Wilma es capaz de manejar altas cantidades de peticiones (o quizá es otro módulo el que satura antes excepto la base de datos de usuarios MySQL, ya que para poder hacer peticiones válidas que sean procesadas por el sistema los usuarios deben estar previamente registrados).

4.2 Prueba 1: Petición de token a Keyrock

El paso previo a toda petición que un usuario o agente realiza es pedir un token de acceso a Keyrock. Vamos a comprobar cómo afecta al rendimiento la cantidad de usuarios que piden un token a Keyrock en un corto intervalo de tiempo. Para ello vamos a evaluar la velocidad de respuesta del sistema en función de la cantidad de usuarios emitiendo peticiones, comprobando en qué punto comienzan a aparecer errores. El número de usuarios en cada escenario será de 2000, 5000, 8500 y 10000.

Con 2K usuarios pidiendo un token:

Configuración JMeter:

Figura 37. Configuración JMeter 2.000. Peticiones de token a Keyrock

Transacciones por segundo:

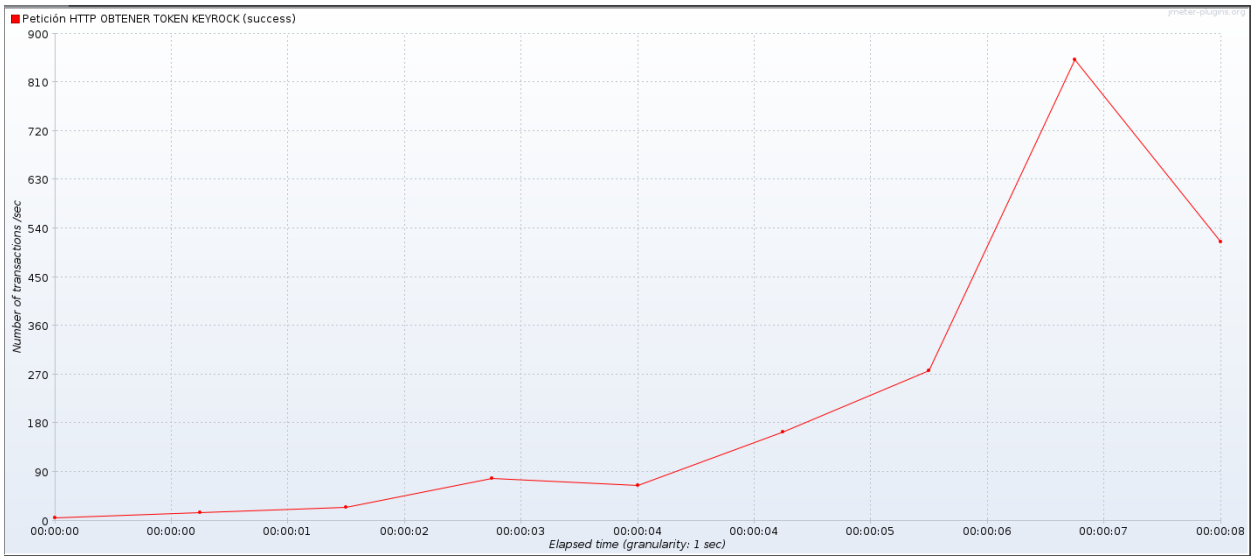


Figura 38. Transacciones JMeter 2000. Peticiones de token a Keyrock

Hilos activos:

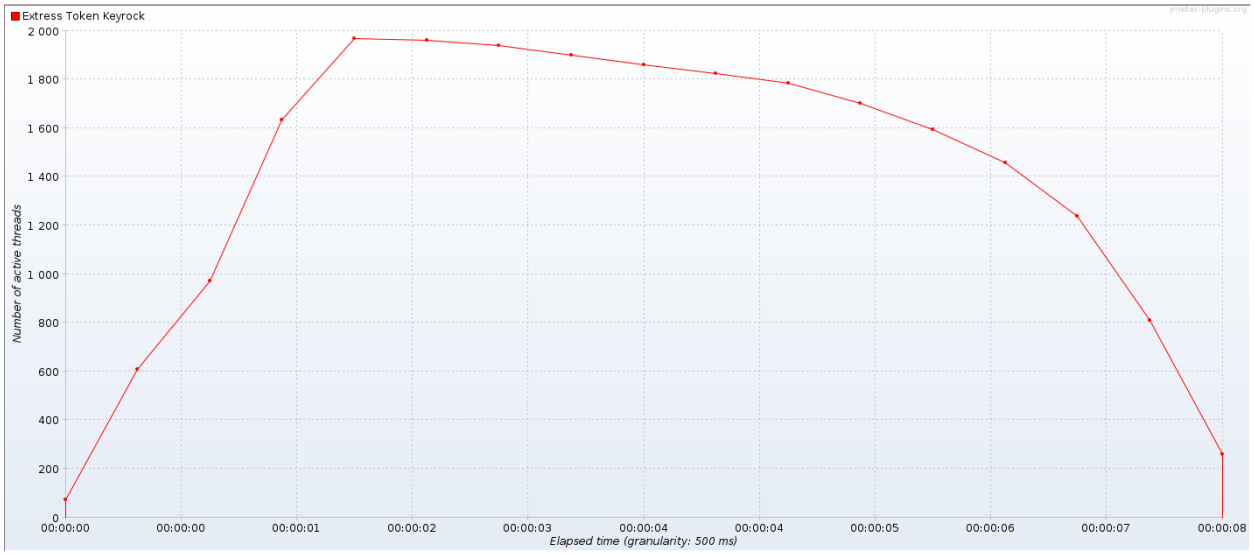


Figura 39. Hilos JMeter 2000. Peticiones de token a Keyrock

Tiempos:

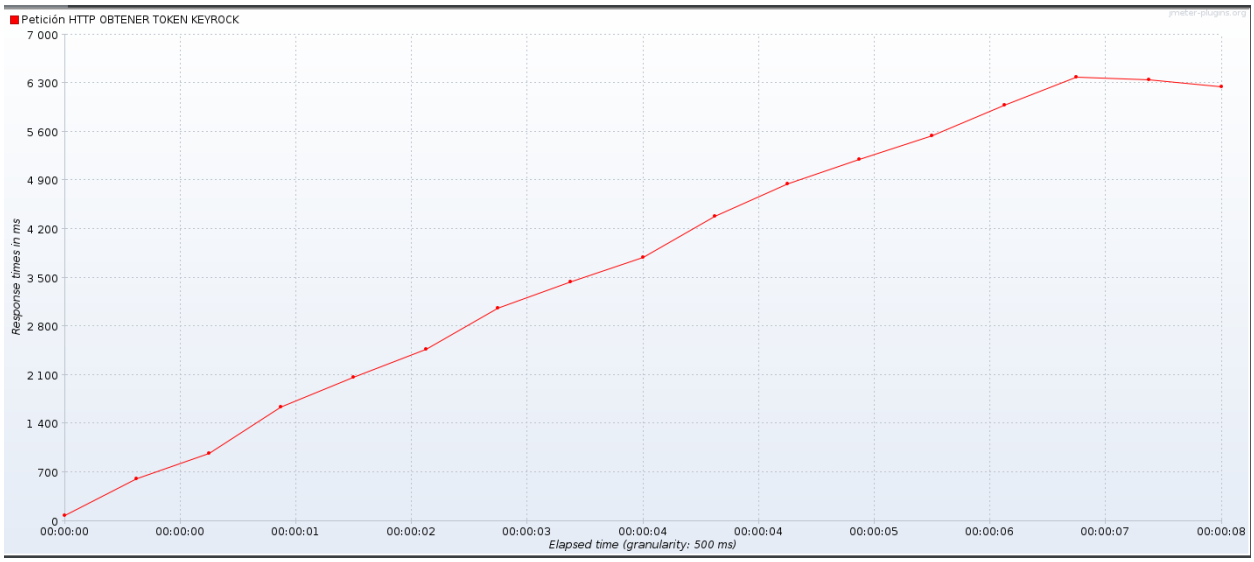


Figura 40. Tiempos JMeter 2000. Peticiones de token a Keyrock

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP...	2000	5853	6265	6385	6400	6422	28	6450	0.00%	243.8/sec	161.17	89.32
TOTAL	2000	5853	6265	6385	6400	6422	28	6450	0.00%	243.8/sec	161.17	89.32

Figura 41. Tabla JMeter 2000. Peticiones de token a Keyrock

Como se puede apreciar en los resultados, conforme se van acumulando las peticiones de los usuarios los tiempos de respuesta aumentan notablemente, despachando las 2000 peticiones concurrentes de los 2000 usuarios sin que ocurra ningún error.

Con 5k usuarios pidiendo un token:

Configuración JMeter:

Figura 42. Configuración JMeter 5000. Peticiones de token a KeyrockJMeter

Transacciones por segundo:

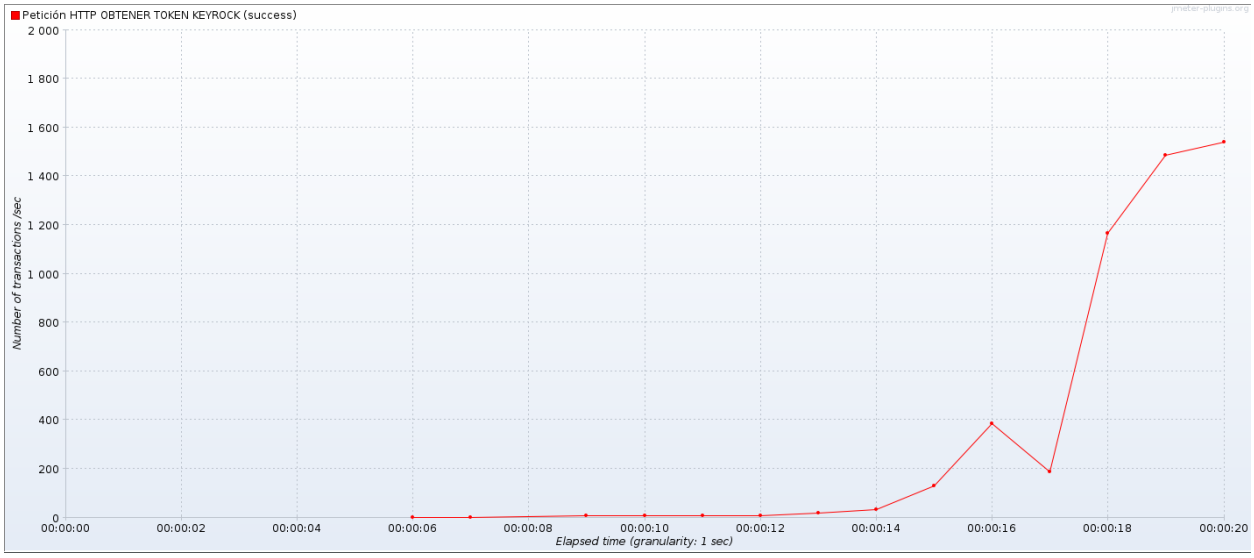


Figura 43. Transacciones JMeter 5000. Peticiones de token a KeyrockJMeter

Hilos activos:

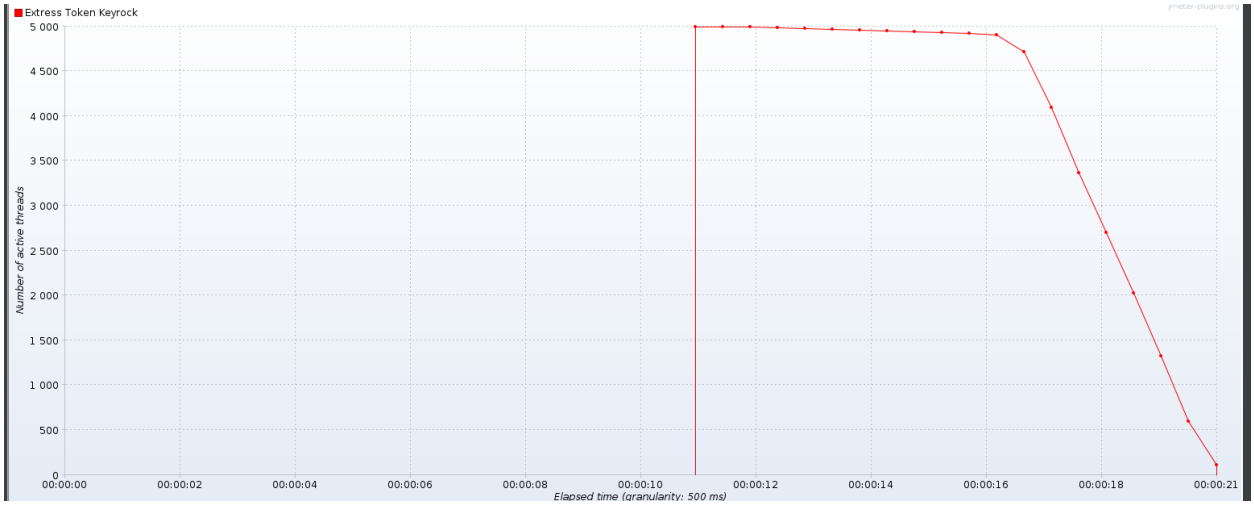


Figura 44. Hilos JMeter 5000. Peticiones de token a Keyrock.JMeter

Tiempos:

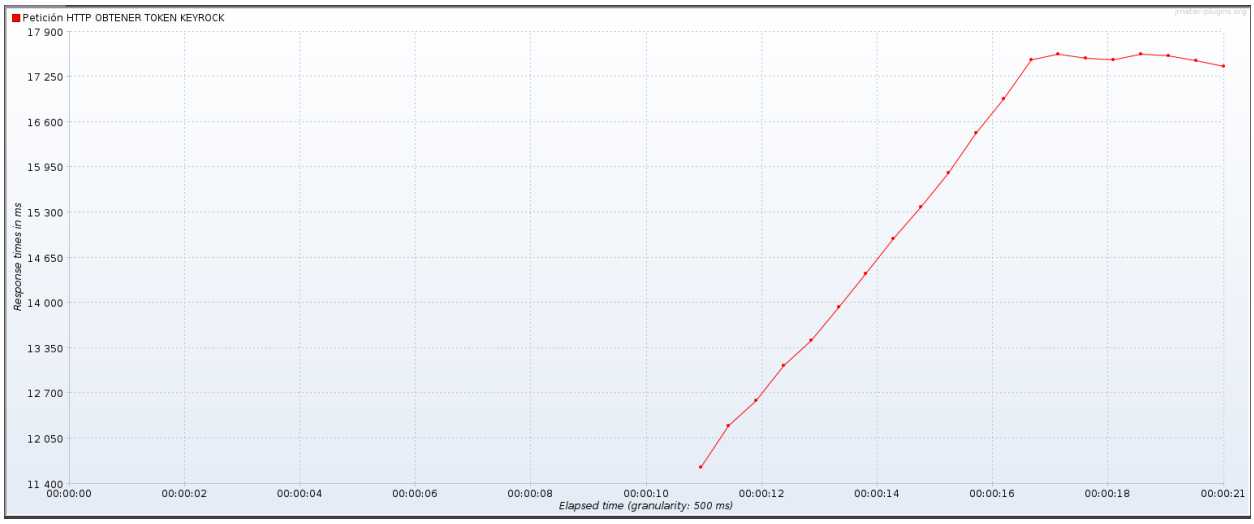


Figura 45. Tiempos JMeter 5000. Peticiones de token a Keyrock

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP ...	5000	17473	17534	17594	17608	17624	11635	17635	0.00%	239.6/sec	158.39	87.86
TOTAL	5000	17473	17534	17594	17608	17624	11635	17635	0.00%	239.6/sec	158.39	87.86

Figura 46. Tabla JMeter 5000. Peticiones de token a Keyrock

En este caso, como en el anterior, pese al aumento de los tiempos de respuesta, Keyrock es capaz de despachar todas las peticiones sin ningún error.

Con 8.5k usuarios pidiendo un token:

Configuración JMeter:

Figura 47. Configuración JMeter 8500. Peticiones de token a Keyrock

Transacciones por segundo:

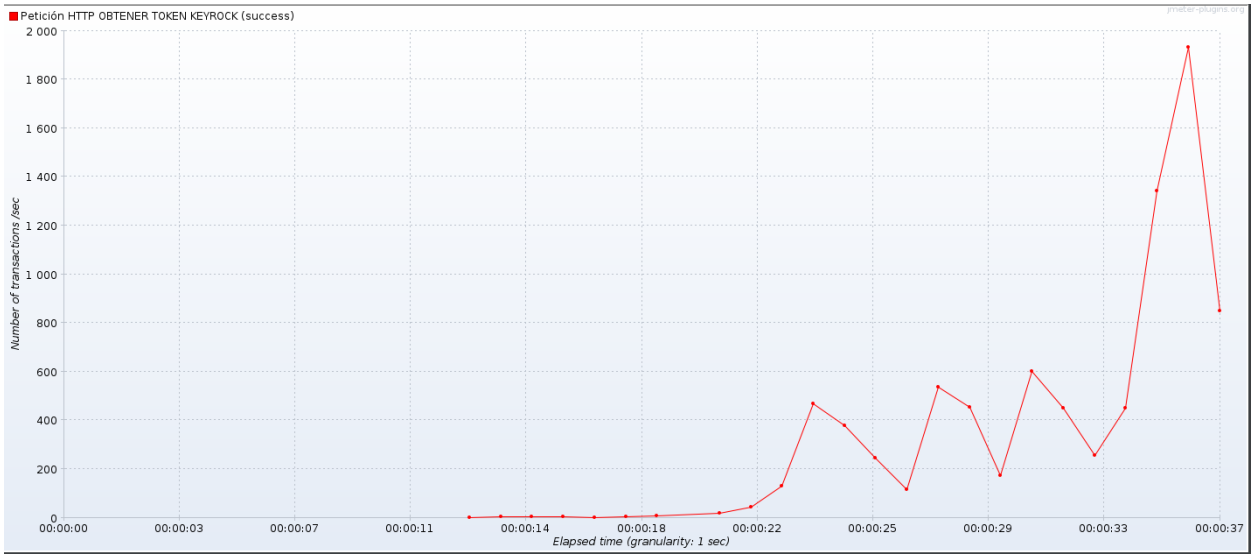


Figura 48. Transacciones JMeter 8500. Peticiones de token a Keyrock

Hilos activos:

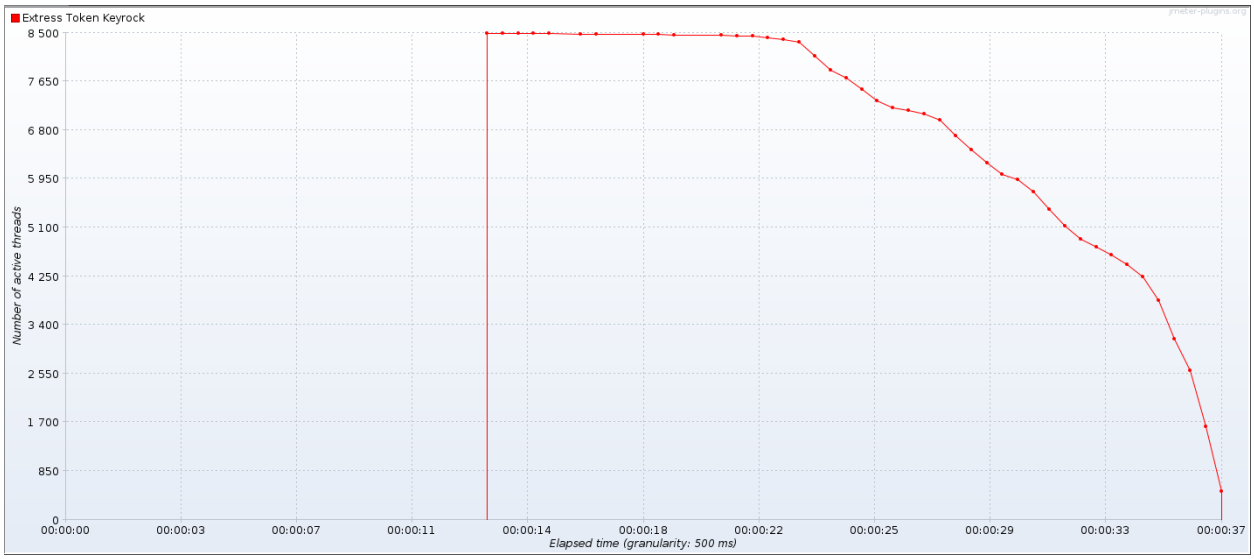


Figura 49. Hilos JMeter 8500. Peticiones de token a Keyrock

Tiempos:

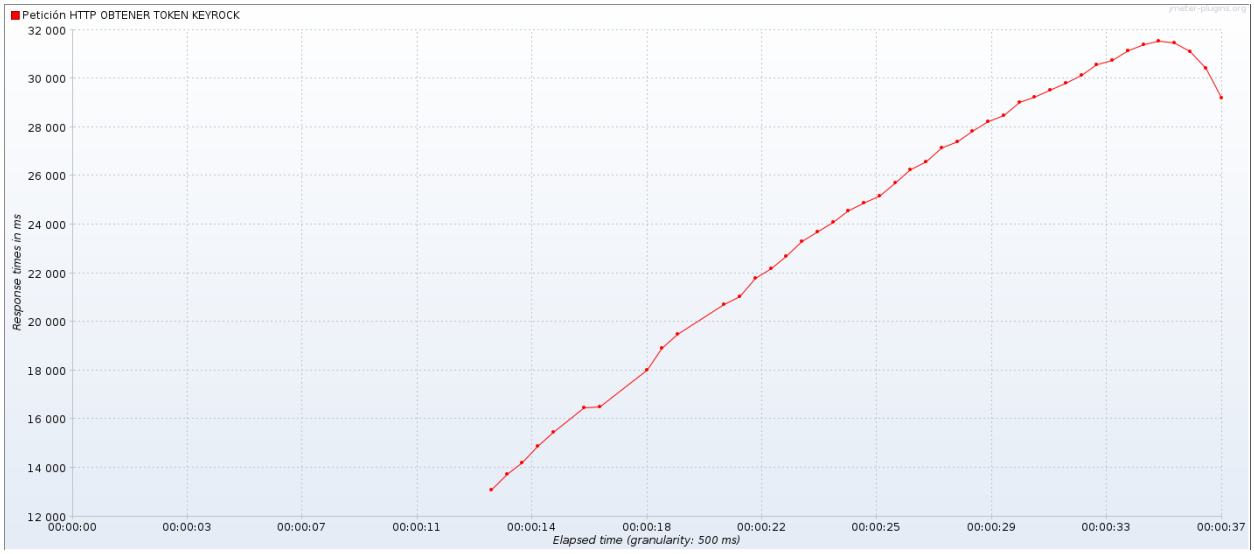


Figura 50. Tiempos JMeter 8500. Peticiones de token a Keyrock

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP...	8500	29109	29944	31515	31537	31588	13097	31612	0.00%	229.7/sec	151.89	84.27
TOTAL	8500	29109	29944	31515	31537	31588	13097	31612	0.00%	229.7/sec	151.89	84.27

Figura 51. Tabla JMeter 8500. Peticiones de token a Keyrock

En este caso sucede como en los anteriores, aunque el tiempo en que Keyrock tarda en procesar todas las solicitudes aumenta, como es lógico, pero sigue sin presentar errores.

Con 10k usuarios pidiendo un token:

Configuración JMeter:

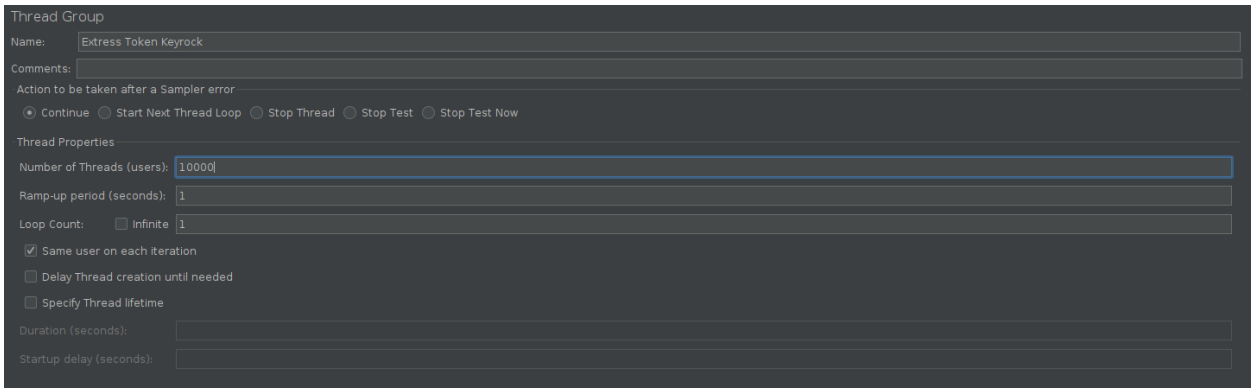


Figura 52. Configuración JMeter 10000. Peticiones de token a Keyrock

Transacciones por segundo:

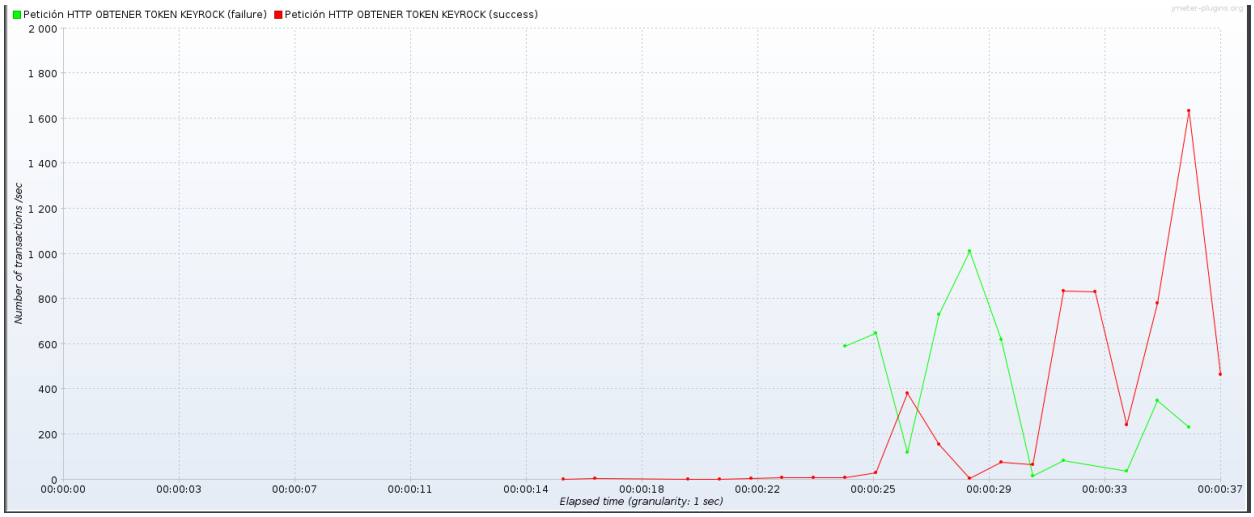


Figura 53. Transacciones JMeter 10000. Peticiones de token a Keyrock

Hilos activos:

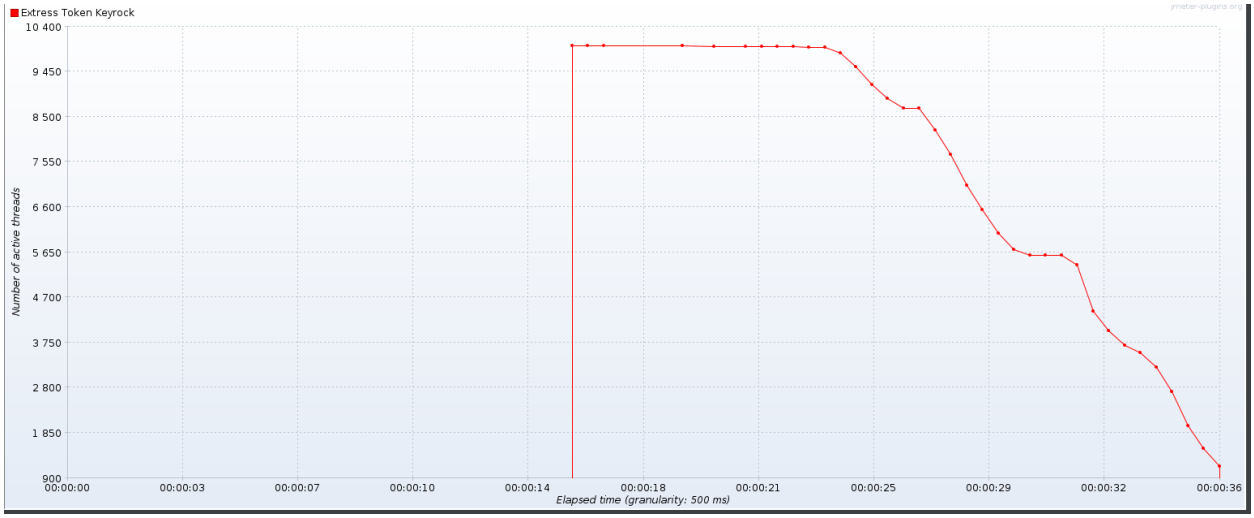


Figura 54. Hilos JMeter 10000. Peticiones de token a Keyrock

Tiempos:

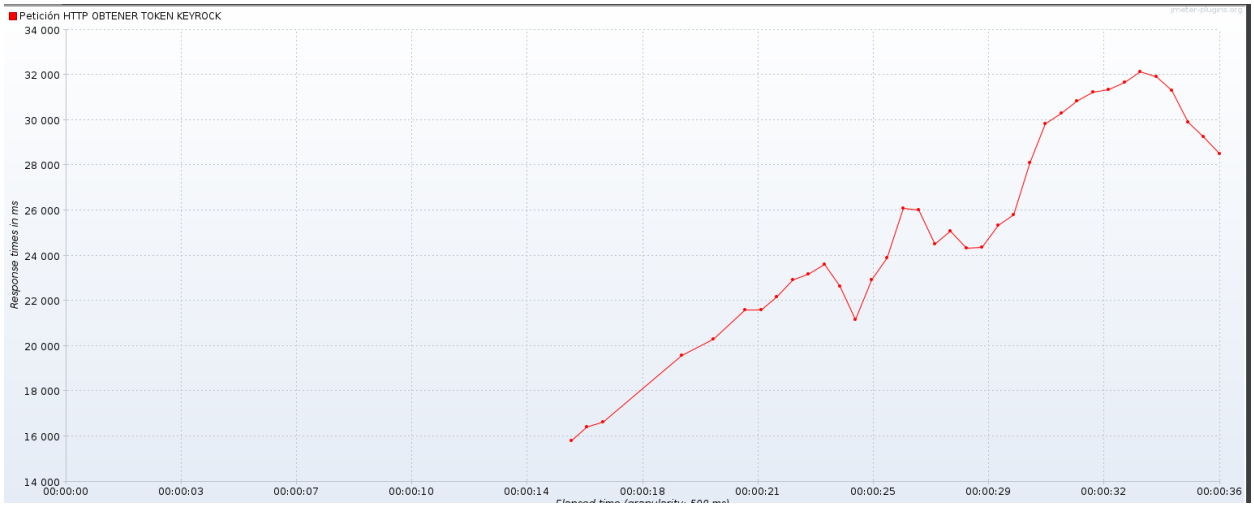


Figura 55. Tiempo JMeter 10000. Peticiones de token a Keyrock

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP ...	10000	27688	28937	31427	31869	32163	15805	33221	44.42%	274.8/sec	166.79	100.81
TOTAL	10000	27688	28937	31427	31869	32163	15805	33221	44.42%	274.8/sec	166.79	100.81

Figura 56. Tiempo JMeter 10000. Peticiones de token a Keyrock

Errores:

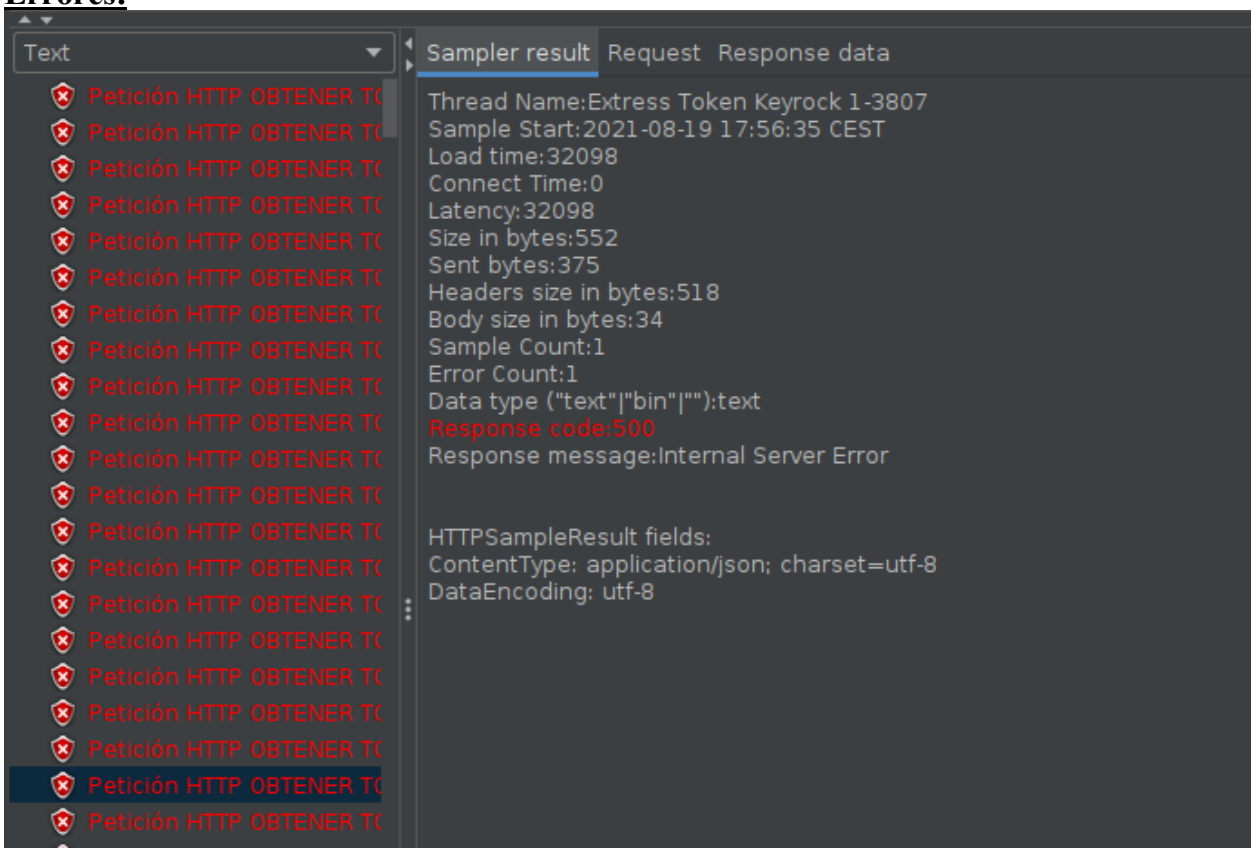


Figura 57. Errores JMeter 10000. Peticiones de token a Keyrock

En este caso, vemos que se produce un 44% de error en las peticiones, por lo que hemos alcanzado el límite en cuanto a capacidad de Keyrock.

RESUMEN RESULTADOS:

Nº Users	Av Delay	Max Delay	Error	Throughput	Received kb/sec	Sent Kb/sec
2.000 Users	5853 ms	6450 ms	0.00 %	243.8/sec	161.18	89.32
5.000 Users	17473 ms	17635 ms	0.00 %	239.6/sec	158.39	87.86
8.500 Users	29109 ms	31612 ms	0.00 %	229.7/sec	151.89	84.27
10.000 Users	27688 ms	33221 ms	44.42 %	274.8/sec	166.79	100.81

Tabla 9. Resumen Resultados. Peticiones de token a Keyrock

En base a estos resultados, encontramos que Keyrock empeora su rendimiento conforme más cargado de peticiones se encuentre, dado que el retardo aumenta drásticamente con el número de peticiones. Esto sólo ocurre si estas peticiones se realizan de forma paralela, ya que, en caso de que fuera una detrás de otra, el rendimiento no empeoraría, como podemos ver en la Figura 57 en la que se ejecutan 10.000 peticiones, una detrás de otra.

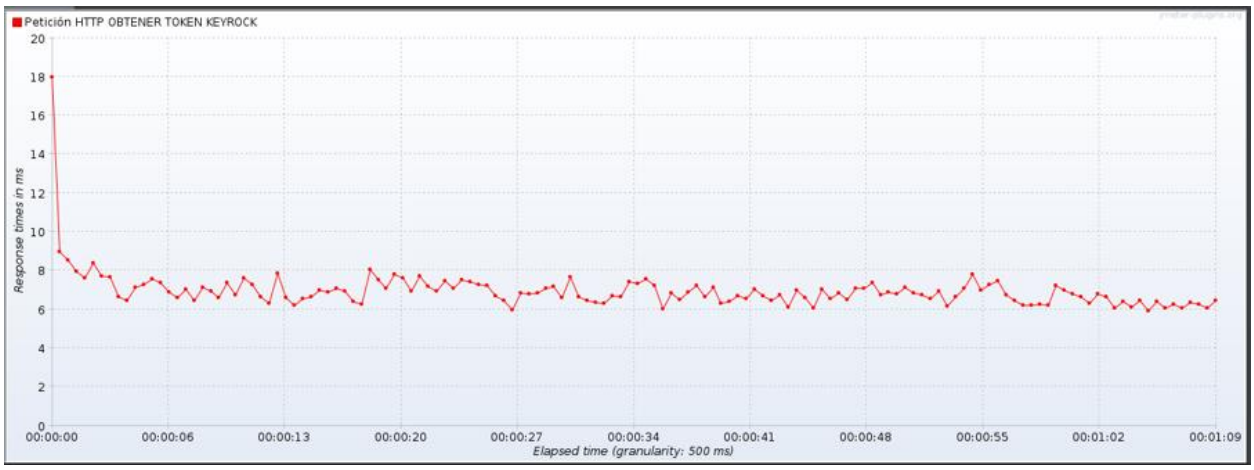


Figura 58. Peticiones lineales. Peticiones de token a Keyrock

Observamos que el problema de retardo proviene de la acumulación de peticiones simultáneas. No obstante, además existe un límite a partir del cual el sistema comienza a rechazar peticiones, obteniendo un error: "500 Internal Server Error". Este límite se alcanza en torno a los 9.000-9500 peticiones paralelas, llegando, como vemos en las gráficas de 10.000 usuarios, a casi un 45% de error por saturación en las peticiones, por lo que este sería el límite para Keyrock en nuestro entorno.

En cuanto a las transacciones por segundo, podemos ver que para 10.000 usuarios las transacciones llegan a alcanzar mínimos de 0/seg, dado que el sistema llega a colapsarse. No obstante, en el resto de casos, las transacciones por segundo son estables, alcanzando mínimos que no llegan en ningún caso a 0.

4.3 Prueba 2: Base de datos MySQL

En la base de datos MySQL donde se guarda la información de autenticación de usuarios, el problema que puede existir es que el número de usuarios registrados sea demasiado grande y el rendimiento del sistema empeore por el tiempo de búsqueda en la base de datos.

Vamos a comparar el funcionamiento del sistema cuando un solo usuario está registrado, frente a cuando tenemos 10.000 usuarios registrados. Se va a utilizar una batería de prueba llamada Ultimate Thread Group.

Este tipo de prueba de carga nos permite definir una escalera ascendente de usuarios que con el tiempo va incrementando la cantidad de usuarios que emiten peticiones al sistema repetidamente. Así, por ejemplo, podemos emular que al principio hay 5 usuarios emitiendo peticiones constantemente, tras 10 segundos asciende a 20 usuarios, luego cuando alcanzamos un minuto, asciende a 200 usuarios, y así sucesivamente. De este modo comprobamos el comportamiento del sistema en diferentes puntos de carga.

Para esta prueba hemos realizado dos acciones, primero los usuarios piden su token a Keyrock, y luego hacen un GET de un sensor concreto al que todos tienen acceso.

Un usuario en MySQL:

Esquema de carga:

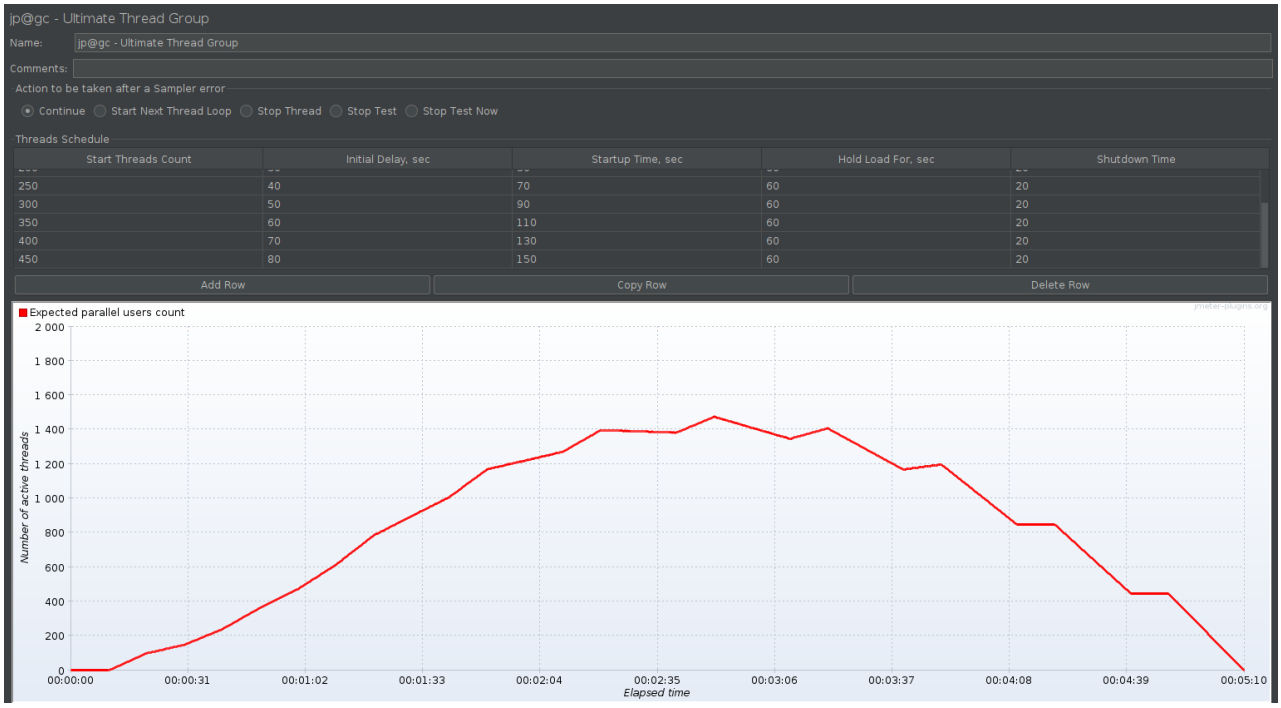


Figura 59. Esquema JMeter 1 Usuario. Base de datos MySQL

Número de hilos en el tiempo:

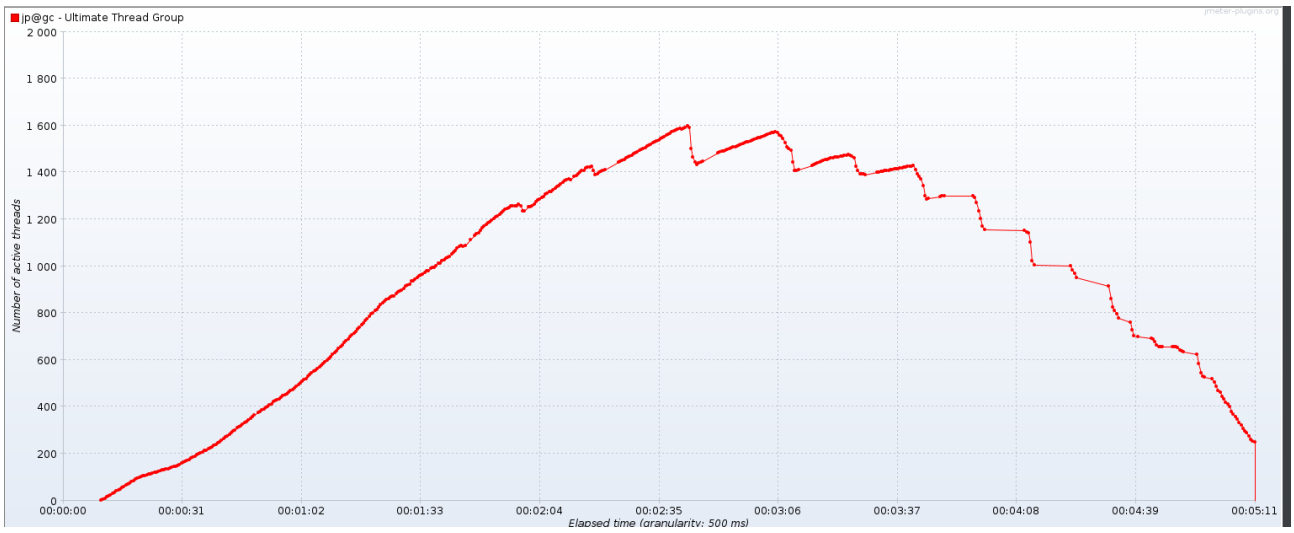


Figura 60. Hilos JMeter 1 Usuario. Base de datos MySQL

Retardo en las peticiones:

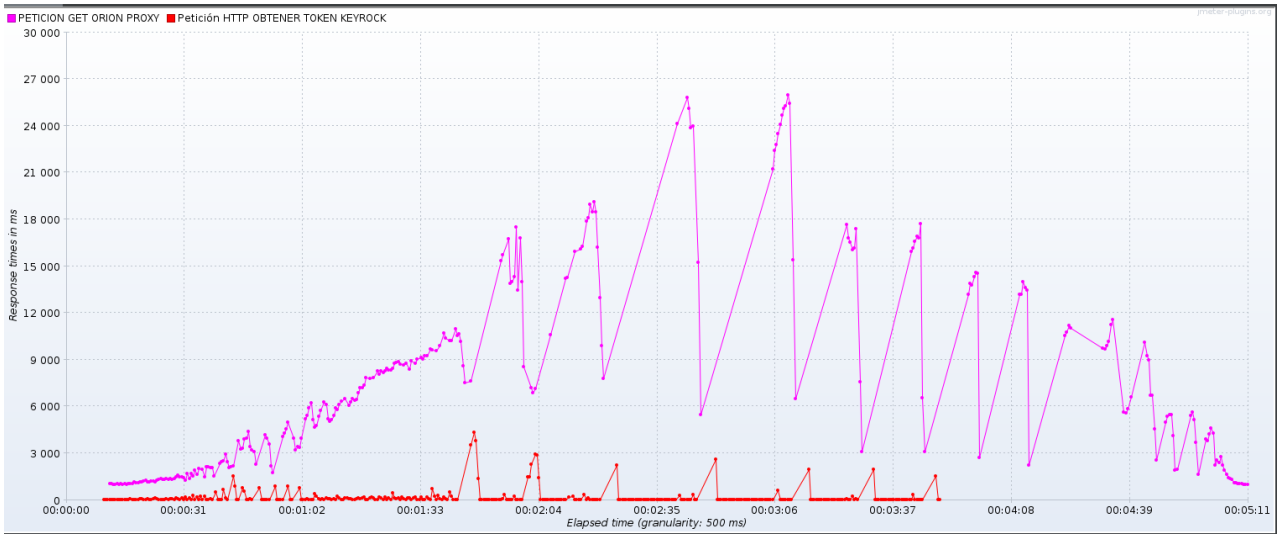


Figura 61. Tiempos JMeter 1 Usuario. Base de datos MySQL

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP ...	2200	404	15	1418	2754	4190	6	4815	0.00%	10.0/sec	6.61	3.74
PETICION GET ...	23486	10879	9759	23295	25263	26433	1016	27192	0.00%	78.1/sec	84.42	18.30
TOTAL	25686	9982	8994	21553	25193	26403	6	27192	0.00%	85.4/sec	89.25	21.04

Figura 62. Tabla JMeter 1 Usuario. Base de datos MySQL

Podemos apreciar cómo el retardo del sistema aumenta proporcionalmente con el número de usuarios que envían peticiones.

10.000 usuarios en MySQL:

Esquema de carga:

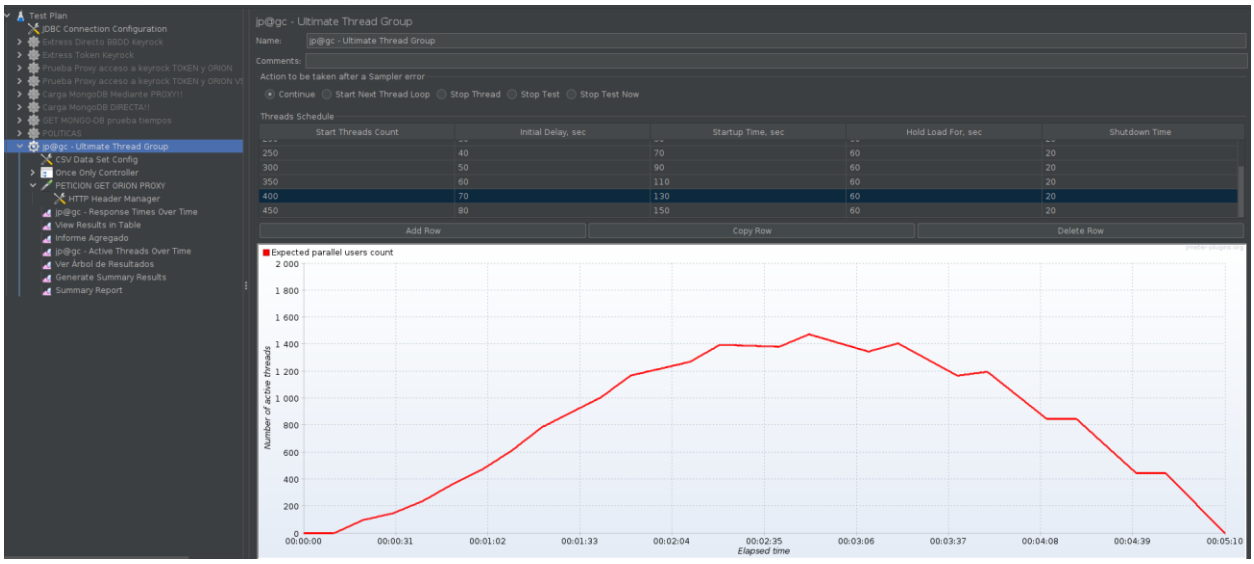


Figura 63. Esquema JMeter 10000 Usuarios. Base de datos MySQL

Número de hilos en el tiempo:

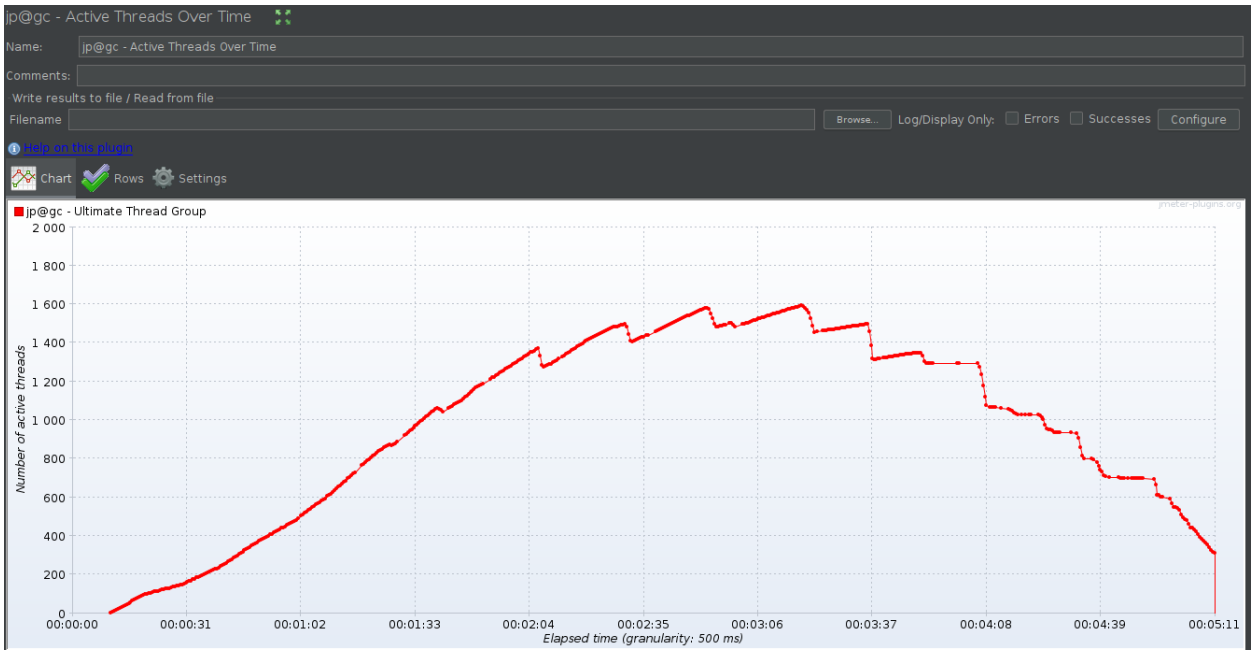


Figura 64. Hilos JMeter 10000 Usuarios. Base de datos MySQL

Retardo en las peticiones:

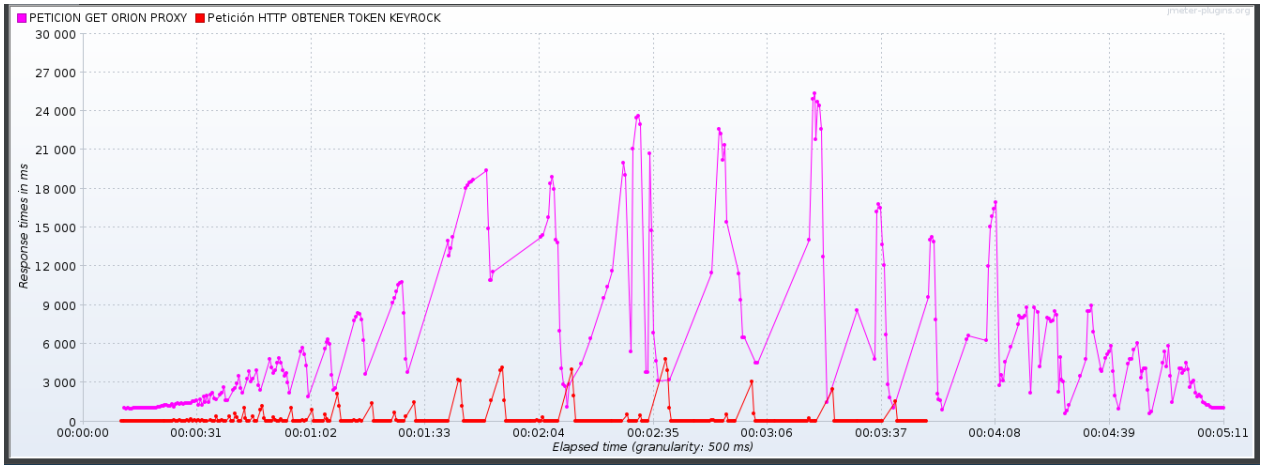


Figura 65. Tiempos JMeter 10000 Usuarios. Base de datos MySQL

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
Petición HTT...	2200	478	12	1917	2899	4290	4	5295	0.82%	10.0/sec	6.60	3.67
PETICION GE...	23096	10625	8567	22620	23982	26222	13	27533	3.67%	76.8/sec	81.29	17.94
TOTAL	25296	9743	8133	22532	23747	26145	4	27533	3.42%	84.1/sec	86.12	20.63

Figura 66. Tabla JMeter 10000 Usuarios. Base de datos MySQL

Errores:

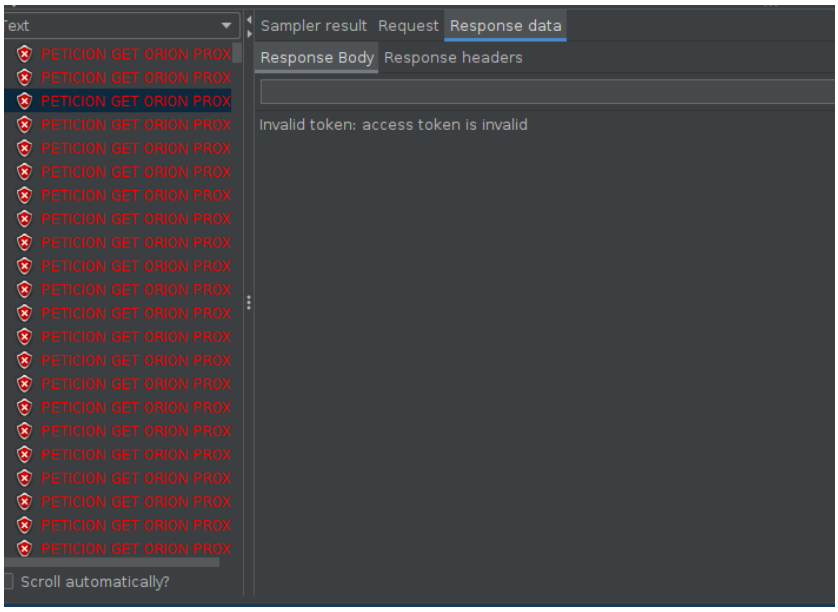


Figura 67. Errores JMeter 10000 Usuarios. Base de datos MySQL

RESUMEN DE RESULTADOS:

Nº Users	Av Delay	Max Delay	Error	Throughput	Received kb/sec	Sent Kb/sec
1	9982 ms	27192 ms	0.00 %	85,4/sec	89.25	21.04
10.000	9743 ms	27533 ms	3.42 %	84.1/sec	86.12	20.63

Tabla 10. Resumen resultados. Base de datos MySQL

Como podemos observar, el hecho de que la base de datos MySQL este más cargada de datos o menos, apenas afecta al rendimiento, obteniendo valores muy similares para ambos casos, de un usuario, y de 10.000 usuarios.

No obstante, las oscilaciones en el caso de 10.000 usuarios en cuanto a tiempos de respuesta, son mayores, y encontramos un 3.42 % de paquetes erróneos, debido a que el token que utilizan es inválido. Esto se debe a que el token expira, ya que cuando un usuario pide un token en el primer instante de la simulación, seguirá utilizando este mismo token durante el resto de tiempo debido al módulo “Once Only Controller” utilizado. Este módulo se utiliza para que la acción de pedir un token solo se realice una vez al principio por cada usuario, y no una vez por cada petición consecutiva, lo cual no tendría sentido, ya que el token es un “login” que dura un intervalo de tiempo durante el cual el usuario realiza determinadas acciones.

En el caso de un solo usuario no sucede, ya que el usuario que accede es siempre el mismo.

Como conclusión, la carga de la base de datos MySQL no afecta al rendimiento del sistema, dentro del rango de valores que estamos manejando y de las limitaciones de nuestro entorno virtual. Cabe detallar que en otro entorno más potente podríamos llegar a cargas mucho mayores de la base de datos que seguramente llegarían a afectar al rendimiento del sistema, pero trabajando en el rango de valores de 0 a 10.000 usuarios, obtendremos, a menor escala, cuáles son los componentes del sistema cuya carga afecta en mayor medida, sin necesidad de utilizar un servidor de pruebas con mayor potencia.

4.4 Prueba 3: Base de datos MongoDB

La base de datos donde se encuentra almacenada la información correspondiente a cada sensor es un punto de interés que conviene ponerlo a prueba debido a que su carga puede afectar al rendimiento en cuestiones de retraso en la respuesta del sistema ante una petición. Se van a realizar pruebas con la base de datos cargada con 10.000 entidades, simulando la existencia de 10.000 camisetas con sensor, y con 50.000 sensores, para comprobar cómo evoluciona el rendimiento del sistema.

10.000 entidades almacenadas:

Esquema:

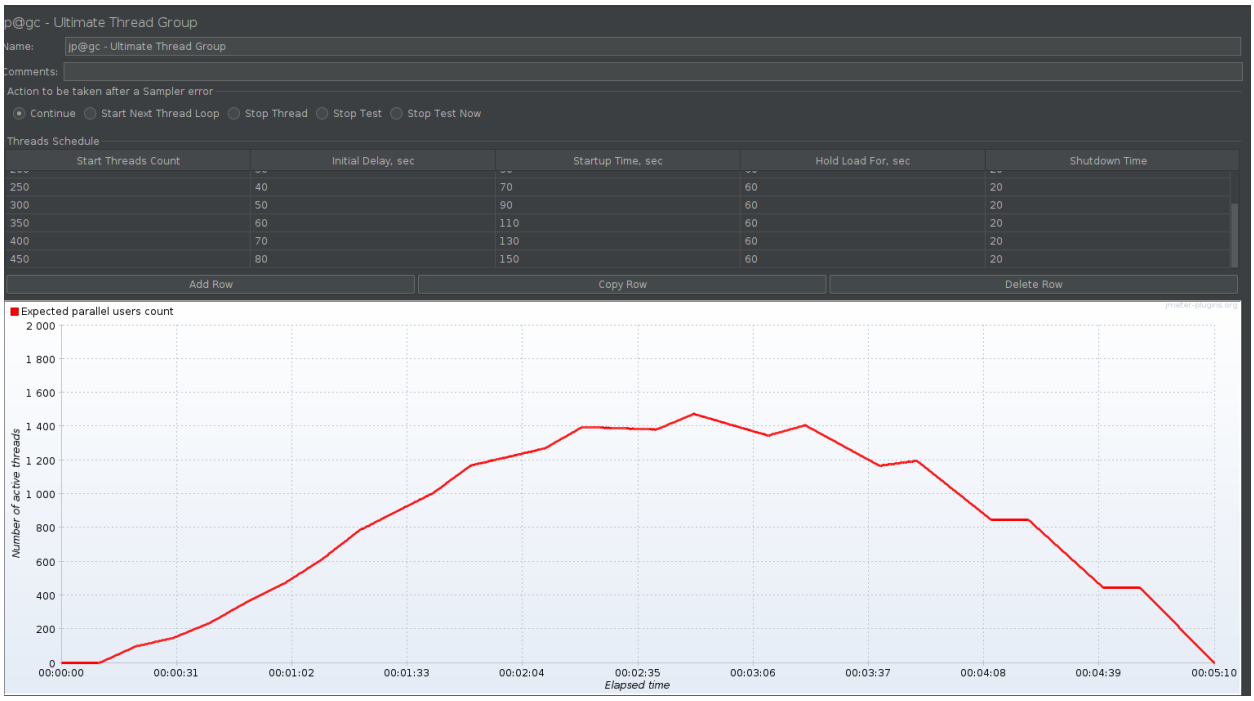


Figura 68. Esquema JMeter 10000 Entidades. Base de datos MongoDB

Hilos:

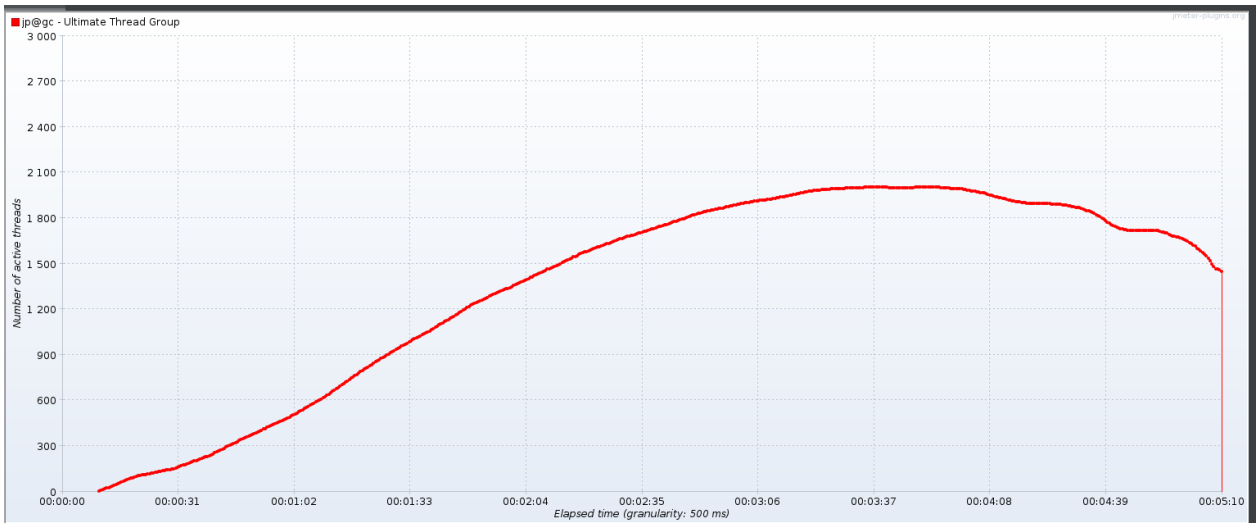


Figura 69. Hilos JMeter 10000 Entidades. Base de datos MongoDB

Tiempos del sistema:

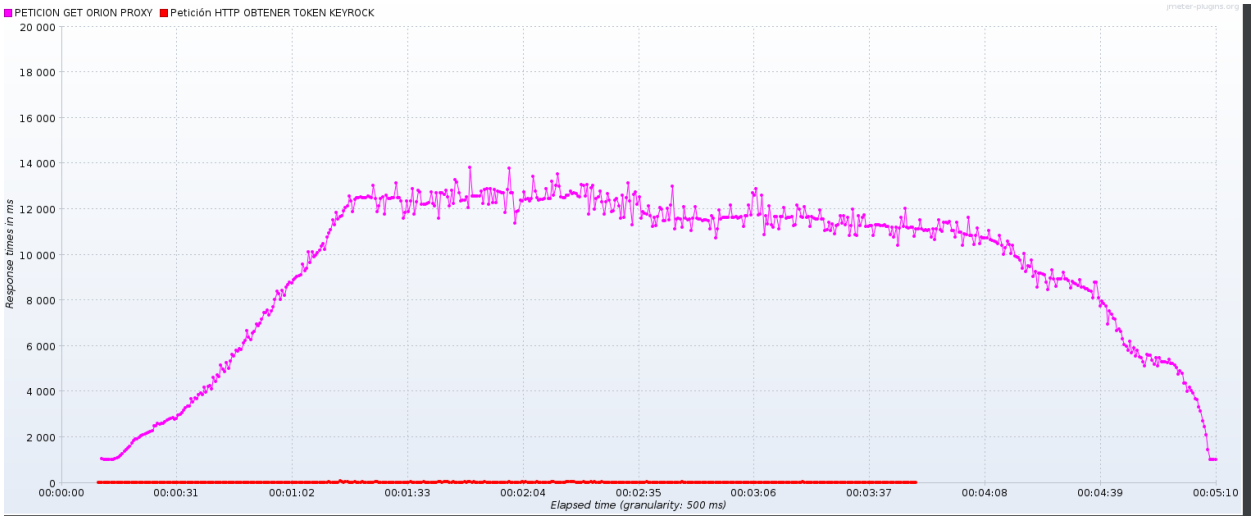


Figura 70. Tiempos JMeter 10000 Entidades. Base de datos MongoDB

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP ...	2200	23	15	50	74	124	6	201	0.00%	10.0/sec	6.61	3.74
PETICION GET ...	14041	9543	11184	12512	12645	14354	1025	34665	0.00%	46.7/sec	50.62	10.95
TOTAL	16241	8253	10729	12485	12618	12984	6	34665	0.00%	54.0/sec	55.45	13.68

Figura 71. Tabla JMeter 10000 Entidades. Base de datos MongoDB

En este caso el retardo no continúa aumentando, permanece en torno a cierto valor, sin ocurrir errores.

50.000 entidades almacenadas:

Esquema:

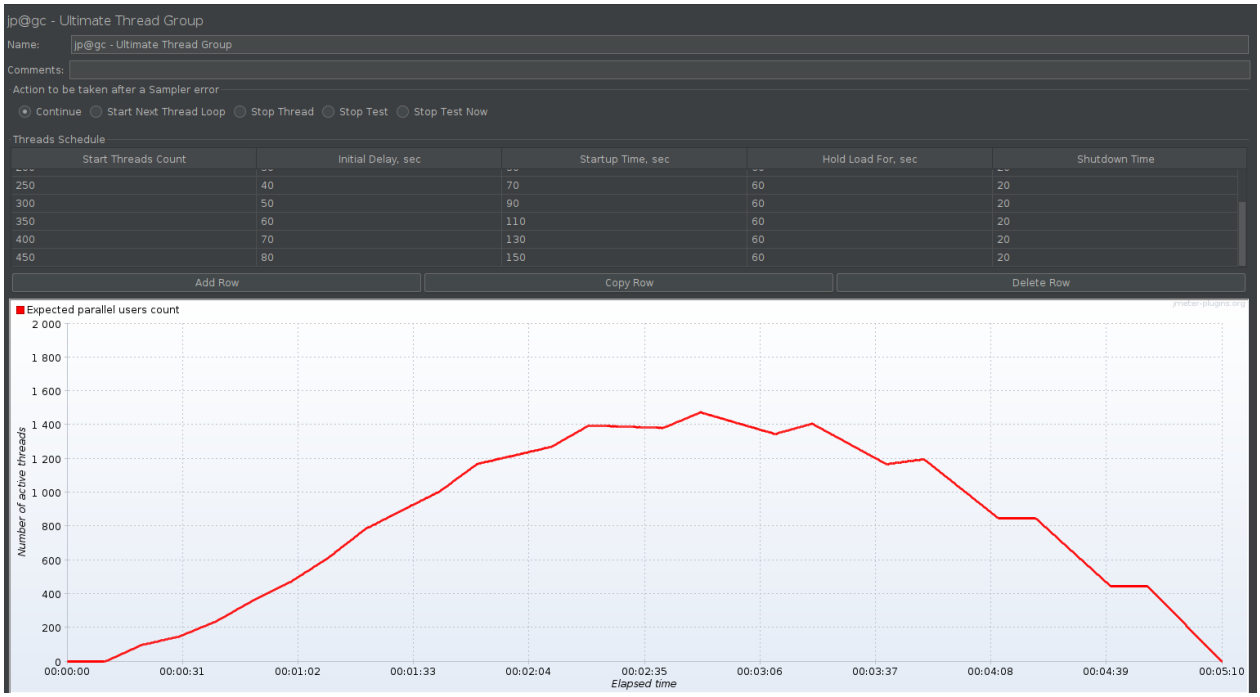


Figura 72. Esquema JMeter 50000 Entidades. Base de datos MongoDB

Hilos:

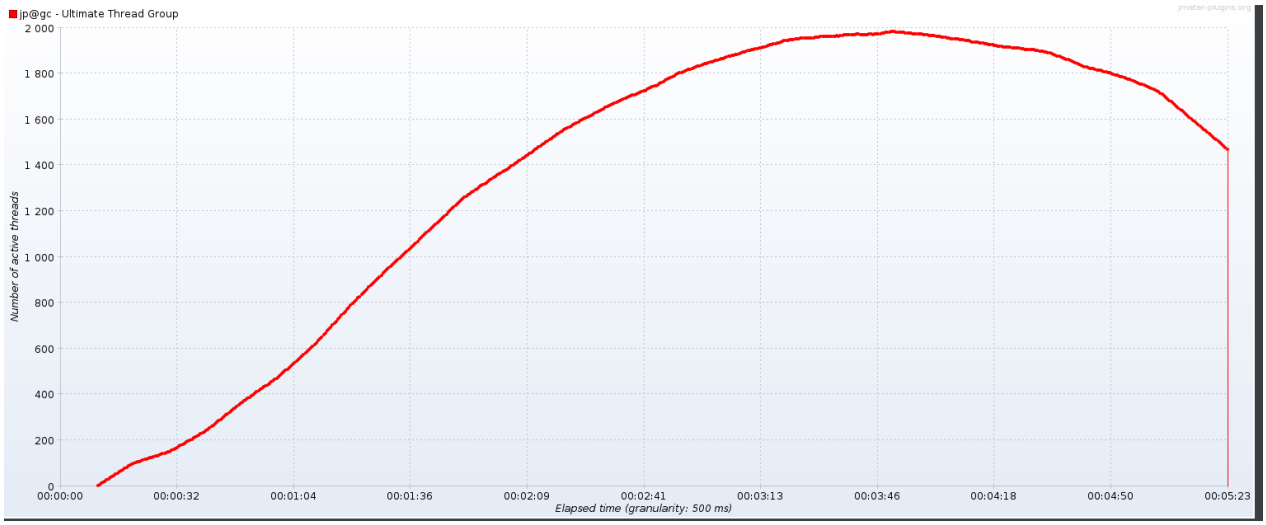


Figura 73. Hilos JMeter 50000 Entidades. Base de datos MongoDB

Tiempos:

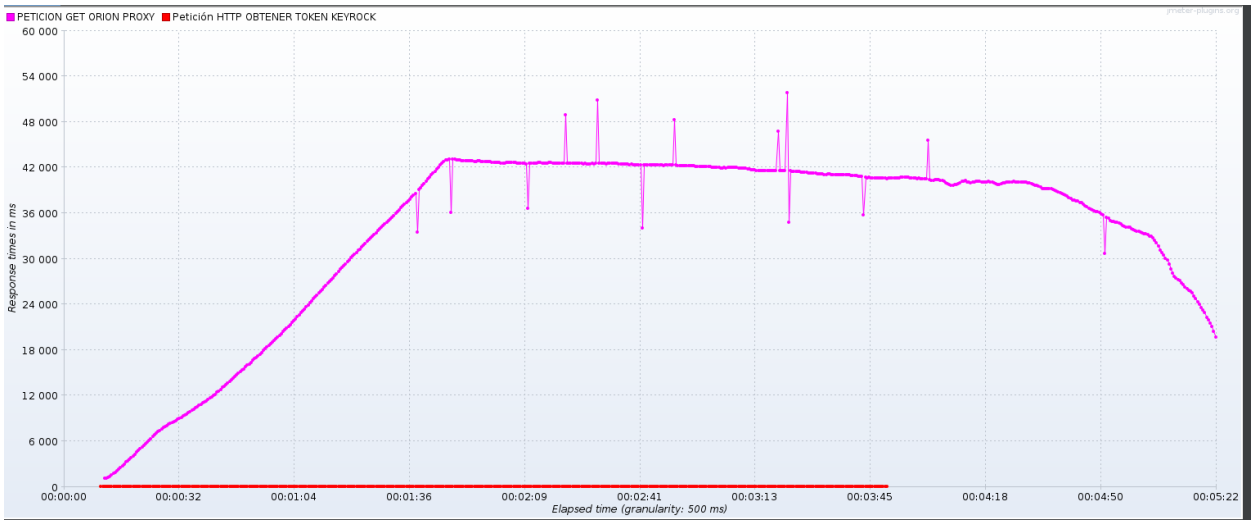


Figura 74. Tiempos JMeter 50000 Entidades. Base de datos MongoDB

Tabla de resultados:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
Petición HTTP ...	2200	12	11	20	24	35	6	141	0.00%	10.0/sec	6.61	3.74
PETICION GET ...	3853	33670	40045	42574	42689	43031	1055	84464	0.00%	12.3/sec	13.36	2.89
TOTAL	6053	21437	25280	42456	42606	42936	6	84464	0.00%	19.4/sec	18.02	5.52

Figura 75. Tabla JMeter 50000 Entidades. Base de datos MongoDB

Ahora el tiempo, como antes, permanece en torno a cierto valor sin continuar aumentando, no obstante, son bastante elevados.

RESUMEN RESULTADOS:

Nº Entities	Av Delay	Max Delay	Error	Throughput	Received kb/sec	Sent Kb/sec
10.000.	8253 ms	34665 ms	0.00 %	54.0/sec	55.45	13.68
50.000	21437 ms	84464 ms	0.00 %	19.4/sec	18.02	5.52

Tabla 11. Resumen resultados. Base de datos MongoDB

En este caso podemos comprobar cómo la carga de la base de datos MongoDB afecta directamente al rendimiento del sistema. Las peticiones GET lanzadas para obtener la información de un sensor se ralentizan drásticamente cuando existen una gran cantidad de entidades almacenadas. Cuando tenemos 5 veces más entidades almacenadas tenemos 2.6 veces mayor tiempo de retardo, empeorando el caudal de datos de la aplicación y haciendo que su rendimiento disminuya, ocasionando un funcionamiento ciertamente lento, ya que una media de 21437 ms de retardo de media en servir una respuesta a una petición es demasiado tiempo para un sistema IoT que maneja una gran cantidad de operaciones por segundo.

4.5 Prueba 4: Políticas de Authzforce

Para comprobar cómo afecta la carga de Authzforce al rendimiento del sistema, se van a utilizar políticas de diferentes longitudes. Así comprobaremos si la existencia de políticas largas, de miles de líneas, afecta al sistema ocasionando un mayor retardo en el proceso de solicitudes, o incluso errores.

Vamos a comparar el funcionamiento del sistema, cuando la política en uso de Authzforce es una política básica de 635 líneas, en la que se evalúan diferentes parámetros en función de la acción a realizar:

Acción	URL	Organization	Current-time	DNI	Agente	Role
GET	/v2/entities	Hospital Central	09:00 a 23:00	10 primeros DNI del fichero CSV	Agente 0-9	No
POST	/v2/entities	Hospital Central	09:00 a 23:00	10 segundos DNI del fichero CSV	Agente10-19	No
PATCH (médicos de mañana)	/v2/entities	Hospital Central	09:00 a 23:00	No	No	Medicos_Hosp_Central_turno_mañana
PATCH (médicos de tarde)	/v2/entities	Hospital Central	15:00 a 15:00	No	No	Medicos_Hosp_Central_turno_tarde

POST de subscripción (mañana)	http://172.18.1.[1-3]:1028/subscriptions	Hospital Central	09:00 a 15:00	No	No	Medicos_Hosp_Central_turno_mañana
POST de subscripción (tarde)	http://172.18.1.[1-3]:1028/subscriptions	Hospital Central	15:00 a 23:00	No	No	Medicos_Hosp_Central_turno_tarde

Tabla 12. Política base para pruebas. Políticas de Authzforce

Esta política básica que consta de 1 Política con 1 Regla para cada acción (GET, PUT, PATCH y Suscripción) separando turno de mañana del turno de tarde, pero siendo la misma regla, será multiplicada por 10 Políticas de 10 Reglas cada una para cada acción. Obteniendo una política como la de la Tabla 12 pero 100 veces más grande, evaluando hasta 2000 usuarios registrados. Con este grosor de políticas se espera comparar el rendimiento y ver si afecta a la respuesta del sistema o es despreciable.

Para realizar estas pruebas se han emitido peticiones de dos formas diferentes, primero de manera linear un solo usuario, y luego con 200 usuarios emitiendo peticiones POST, POST para subscribirse, y PATCH.

RESULTADOS DE LAS PRUEBAS:

Política Básica:

1-User

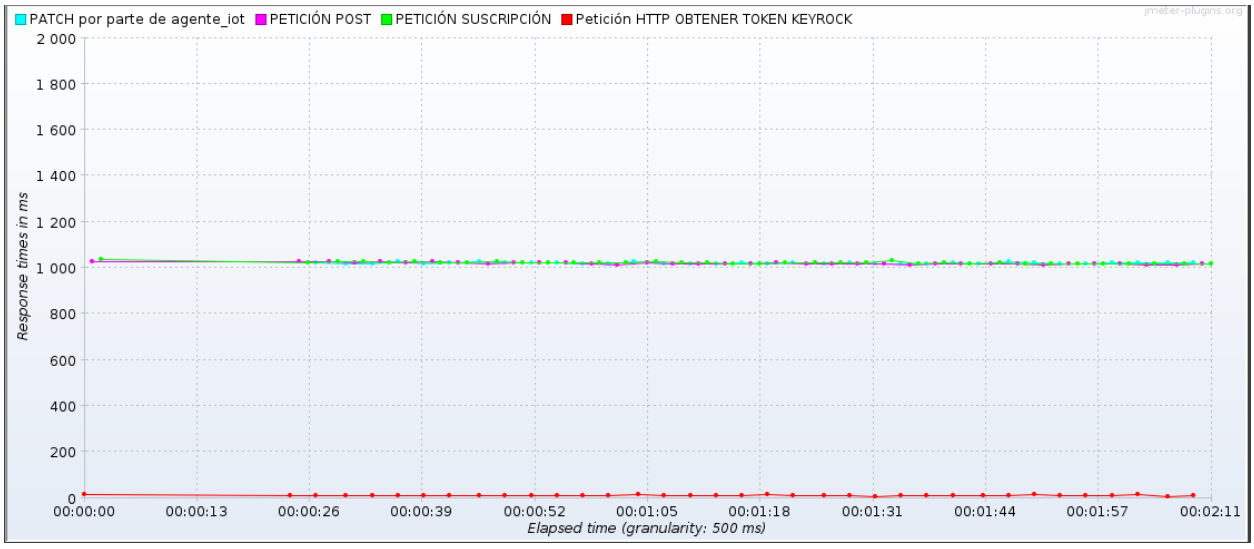


Figura 76. Tiempos 1 Usuario. Política Básica. Políticas de Authzforce

200-Users

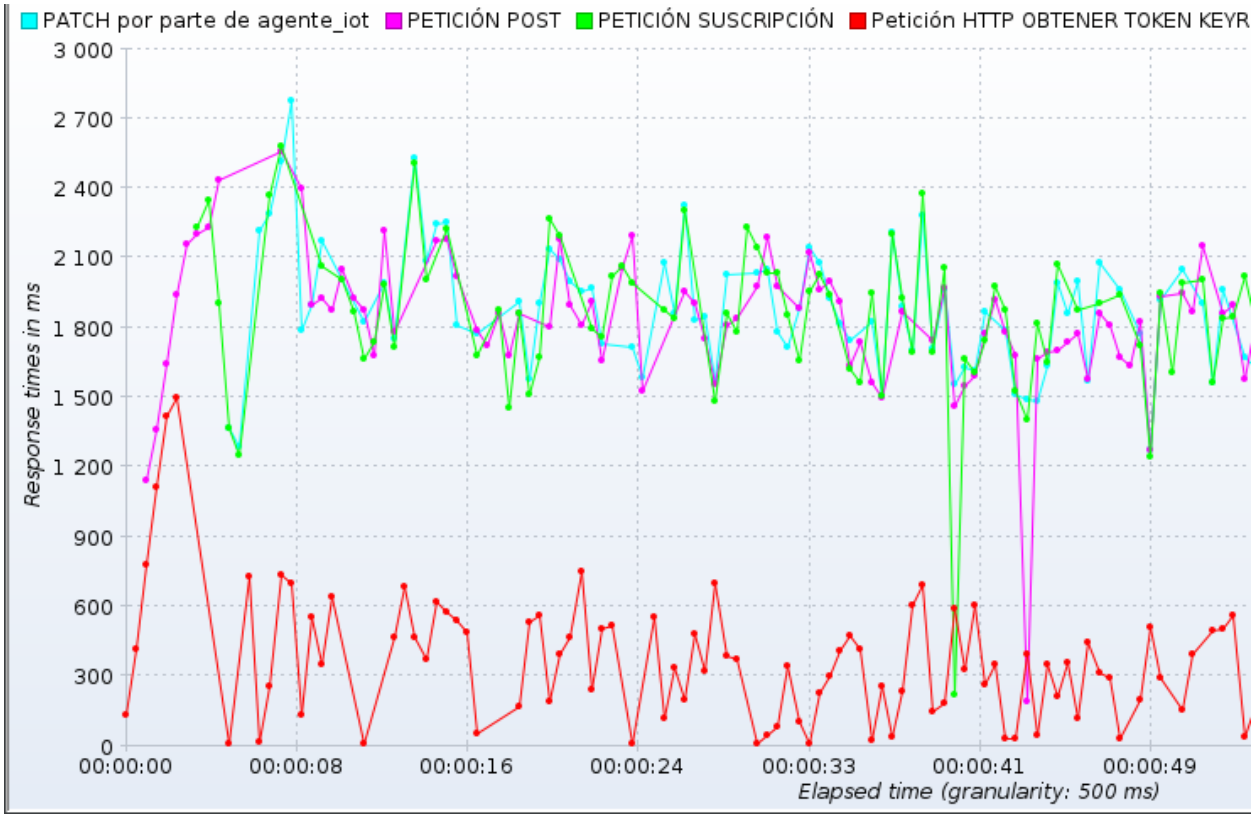


Figura 77. Tiempos 200 Usuarios. Política Básica. Políticas de Authzforce

Política Compleja:

1-User

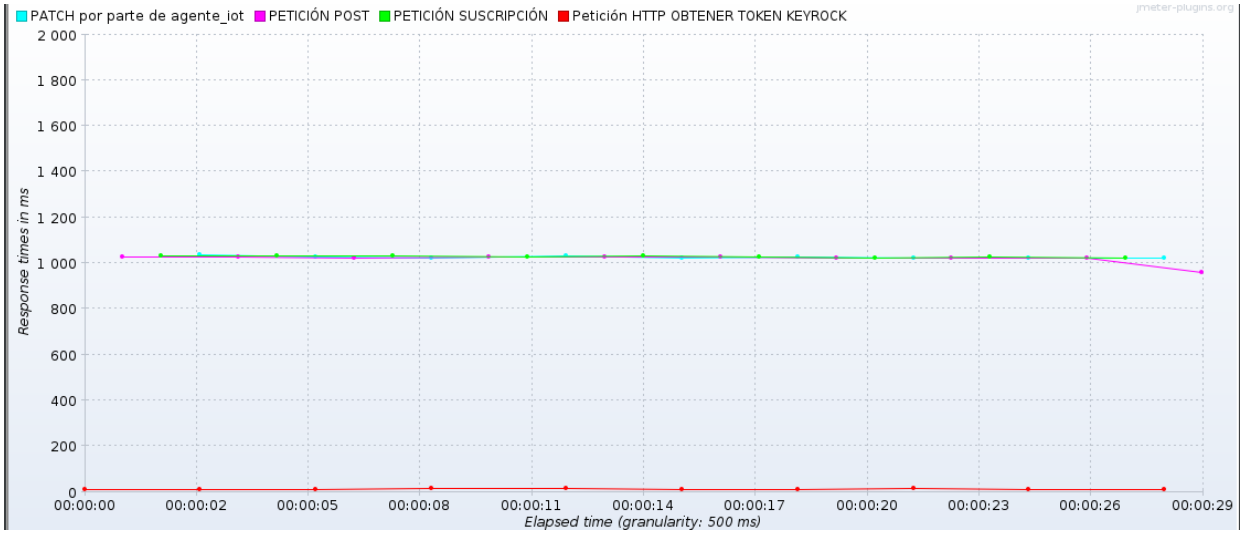


Figura 78. Tiempos 1 Usuario. Política Compleja. Políticas de Authzforce

200-Users

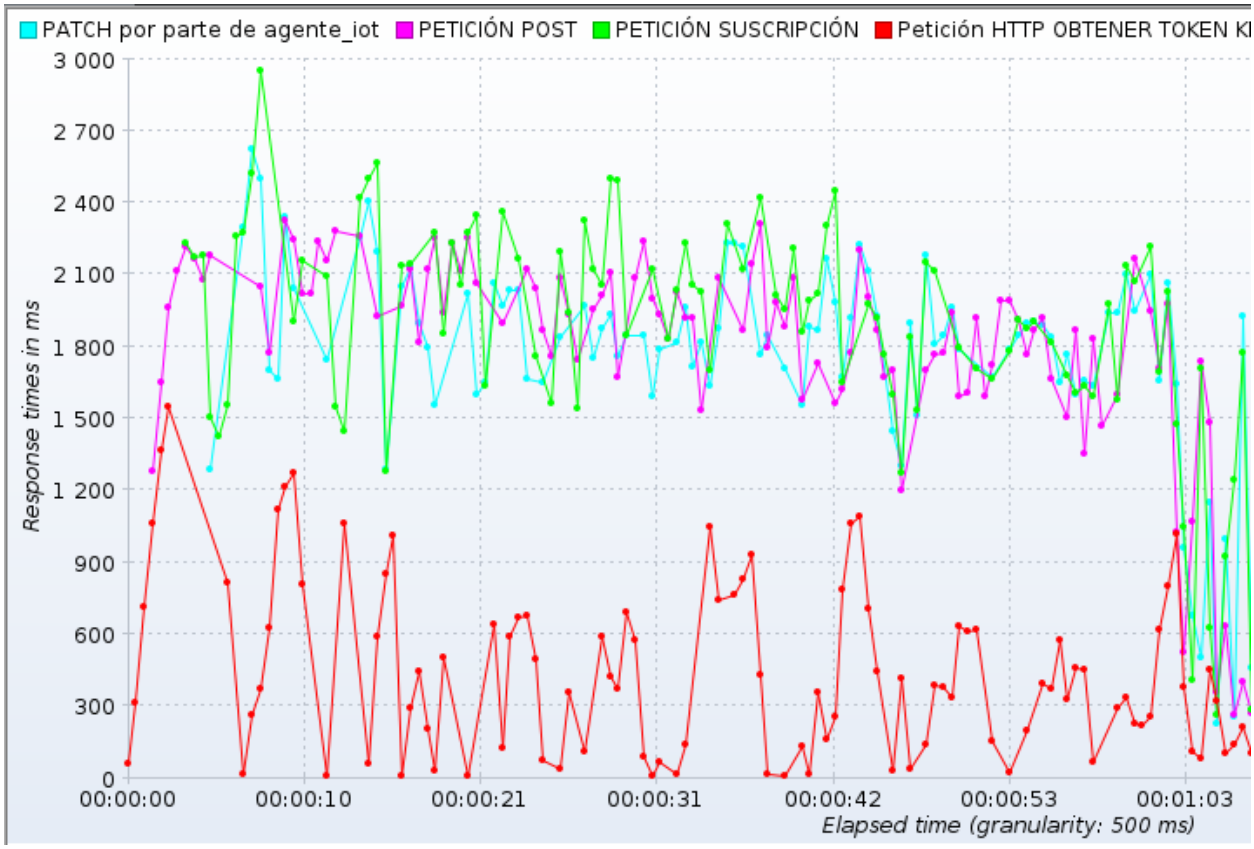


Figura 79. Tiempos 200 Usuarios. Política Compleja. Políticas de Authzforce

En ambos casos podemos observar cómo las latencias no presentan un cambio apreciable. Pese al gran aumento en políticas la media de retardo en respuesta es de 2000 ms en ambos escenarios, no siendo afectado por la magnitud de la política. Consideraremos que el uso de políticas complejas y extensas no será un factor limitante para el rendimiento de nuestro sistema.

4.6 Prueba 5: Pruebas de tensión

Para esta prueba vamos a cargar el sistema con numerosas peticiones, buscando que Wilma PEP Proxy reciba la mayor cantidad de solicitudes. Todas estas serán GET de este modo no cargamos con POST la base de datos MongoDB y no manchamos las pruebas, dado que, en ese caso, el retardo del sistema aumentaría debido a la carga de dicha base de datos y no de la saturación de peticiones. Como hemos visto, por ahora el único factor limitante del sistema realmente es la base de datos MongoDB que aumenta los tiempos de carga significativamente cuantos más sensores existan. Keyrock al recibir una gran cantidad de peticiones también aumenta considerablemente los tiempos de respuesta en otorgar un token, esto es un paso previo a la emisión de la petición a Wilma, pero también puede ser un factor limitante, por lo que haremos las pruebas con diferentes cantidades de usuarios que acceden a la vez al sistema.

Para realizar esta prueba utilizaremos un único usuario que manda diferentes cantidades de peticiones a la vez, este usuario siempre hace login antes de enviar cada petición. Todas las peticiones van al mismo sensor. Este modelo tan simple nos protege de errores de expiración de token, usuarios que no existan, sensores que no existan… La política activa permite a cualquier usuario hacer un GET, por lo que, si este usuario, que se ha validado y tiene permisos para hacer un GET de dicho sensor, obtiene algún error, será error de sobrecarga

debido a la cantidad de peticiones concurrentes. Vamos a ver cuál es el punto en el que el sistema comienza a colapsar, con pruebas rápidas de tensión buscando los momentos críticos del sistema.

Nº Requests	Error GET	Av Delay	Max Delay	Throughput/sec
200	0.00 %	1514 ms	2750 ms	95.625
300	0.00 %	2255 ms	4162 ms	107.411
400	10.25 %	2671 ms	5153 ms	117.942
500	48.80 %	3807 ms	6616 ms	115.554

Tabla 13. Resultados de pruebas para diferente número de usuarios paralelos. Pruebas de Tensión

Tiempos:

200 usuarios:

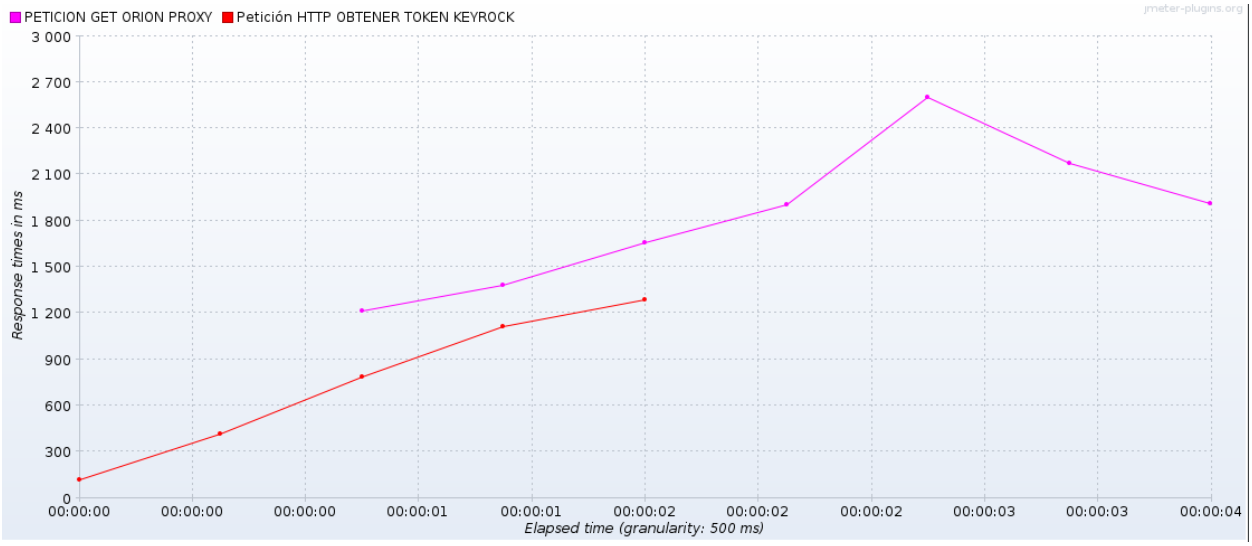


Tabla 14. Tiempos 200 Usuarios. Pruebas de Tensión

300 usuarios:

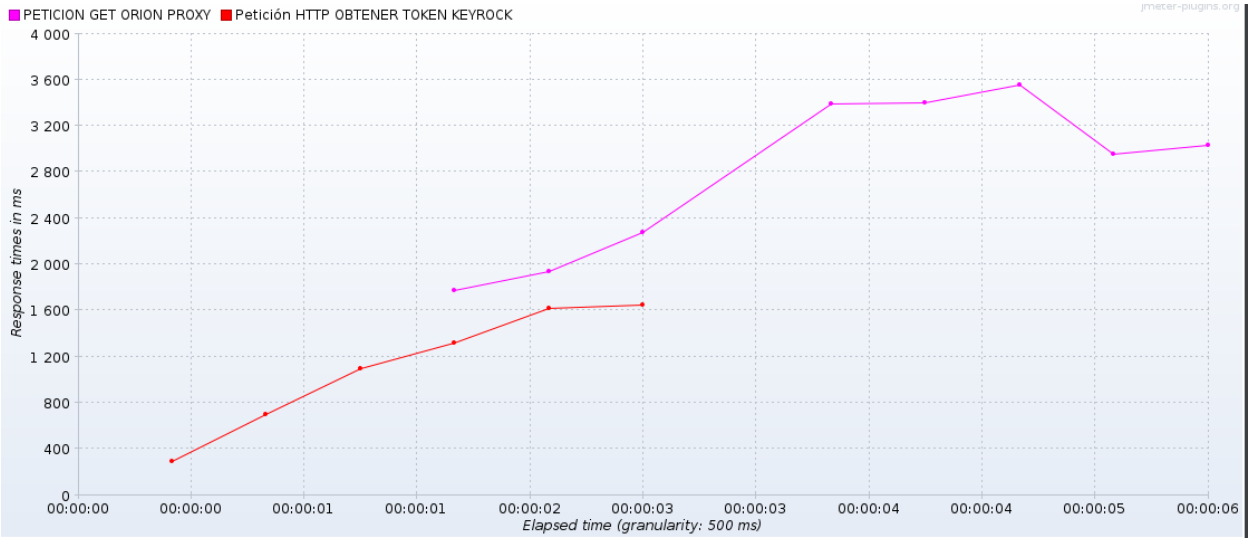


Tabla 15. Tiempos 300 Usuarios. Pruebas de Tensión

400 usuarios:

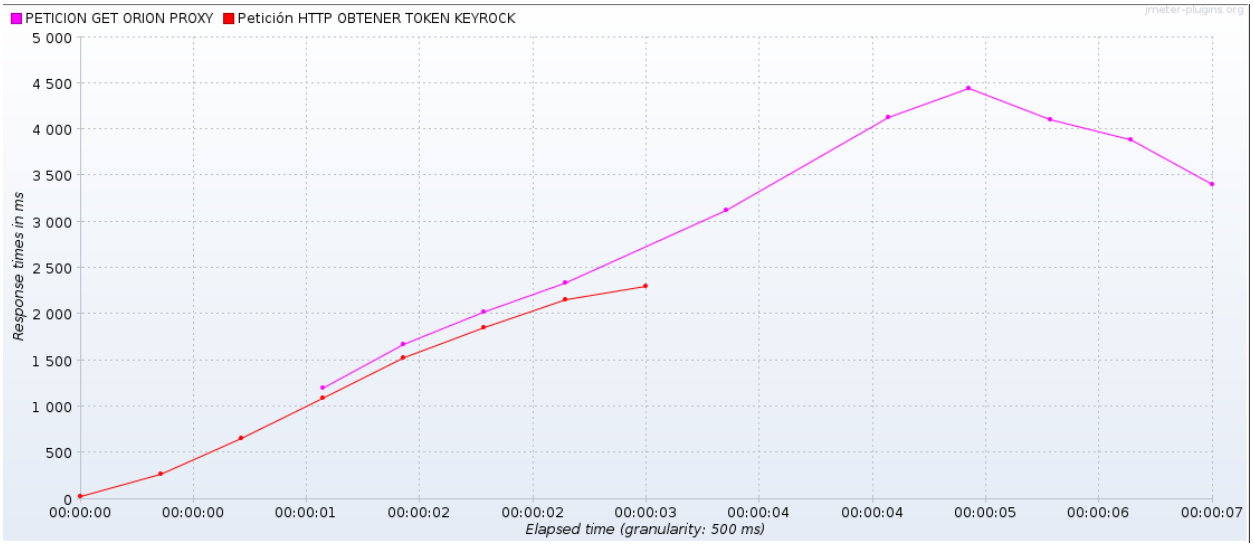


Figura 80. Tiempos 400 Usuarios. Pruebas de Tensión

500 usuarios:

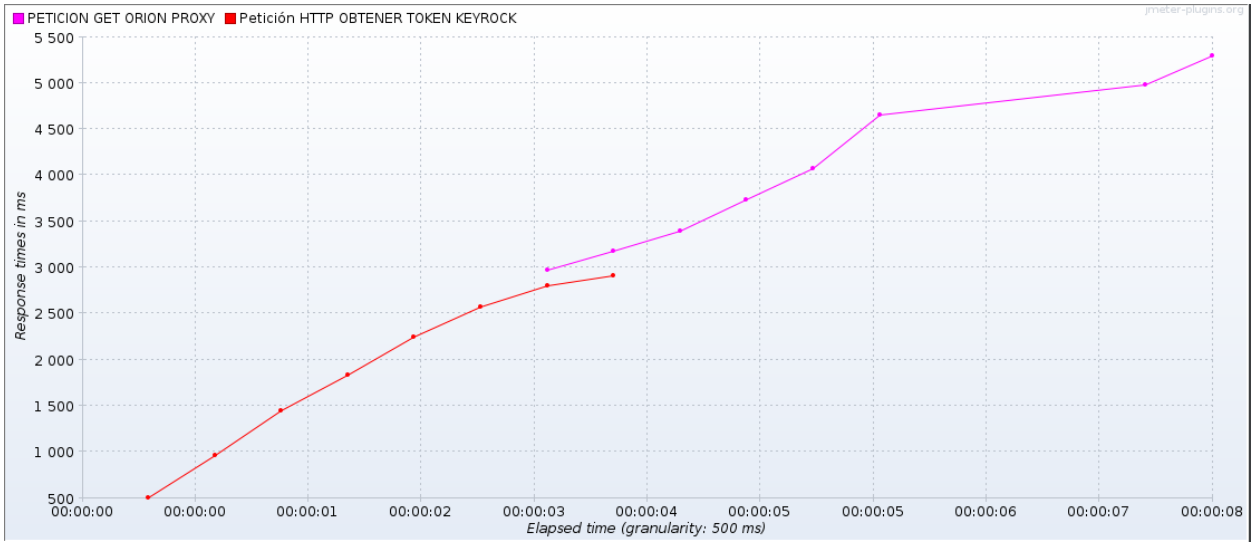


Figura 81. Tiempos 500 Usuarios. Pruebas de Tensión

Como podemos observar en los resultados de las pruebas, cuando superamos los 300-400 usuarios que mandan peticiones de manera consecutiva, es decir las 400 peticiones a la vez, comenzamos a obtener errores. Para estas pruebas se ha utilizado un mismo usuario y un mismo sensor al que se hace GET, este usuario accede de manera paralela mandando peticiones GET a este mismo sensor. El error (Figura 82) que aparece es de usuario no autorizado a ingresar al sistema, no obstante, la política que se encuentra activa permite a cualquier usuario hacer un GET de cualquier sensor, por lo que el sistema colapsa llegado a esa carga y comienza a denegar peticiones debido a que no abarca tantas peticiones a la vez. Las peticiones ejecutadas de manera seguida, una detrás de otra, no presentan ningún tipo de problema ya que se despachan rápidamente y no se satura el sistema.

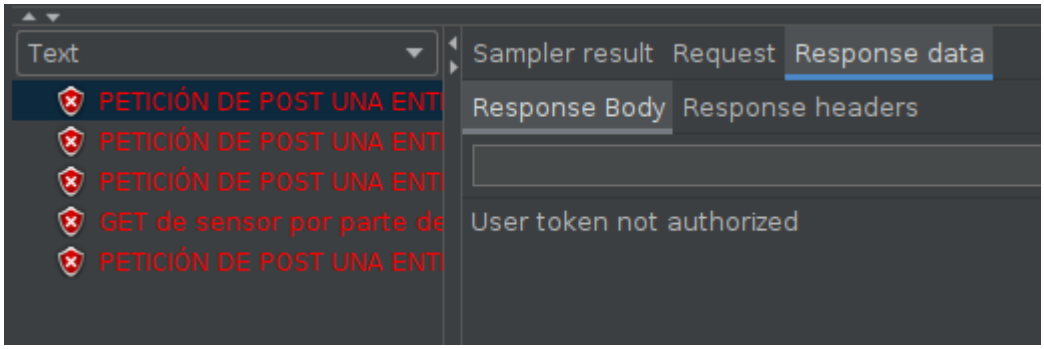


Figura 82. Error User Token not authorized

5 CONCLUSIONES Y POSIBLES MEJORAS

Antes de proceder con las conclusiones del trabajo conviene destacar que todas las pruebas han sido realizadas en una máquina virtual en la cual instalamos la plataforma utilizando Docker y luego disparamos las pruebas utilizando JMeter en la misma máquina, por lo que nos encontramos limitados por la capacidad de esta máquina. En un caso real en que nuestro servidor fuera mucho más potente, confiamos que el rendimiento del sistema perduraría en estado óptimo con una carga mayor, pero esta prueba y análisis a escala menor nos sirve para encontrar cuáles son las partes del sistema cuya carga afectaría mayormente al rendimiento en cualquier servidor.

5.1 Resultados finales del proyecto

Con las pruebas realizadas, hemos obtenido los resultados que se muestran en la Tabla 14 en cuanto a límites y la manera en que afecta al sistema el sobrepasarlos.

Módulo	Cargas	Límite	Afecta al rendimiento	Efecto
Keyrock	2.000 usuarios 5.000 usuarios 8.500 usuarios 10.000 usuarios	Con 9.000 usuarios enviando peticiones en paralelo comienzan a aparecer errores.	Si	Peticiones con respuesta errónea y retardos significativos.
MySQL	1 usuario 10.000 usuarios	Hasta 10.000 usuarios registrados no afecta al rendimiento.	No	No
MongoDB	10.000 entidades 50.000 entidades	Con 50.000 entidades empeora significativamente.	Si	Retardos significativos, pero sin errores.
Authzforce	Política básica 200 líneas. Política compleja de 46.000 líneas	Sin límite, hasta 10 políticas con 10 reglas para cada una comprobado.	No	No
Wilma PEP Proxy	200 usuarios 300 usuarios 400 usuarios 500 usuarios	400 usuarios en paralelo	Si	Errores, pero con bajo incremento de retardo.

Tabla 16. Resumen de los resultados. Límites del sistema.

Estos resultados nos otorgan un rango de carga bajo el cual aseguramos que el sistema funcionará correctamente.

Obtenemos que el principal problema se haya cuando se disparan peticiones a la vez con varios usuarios enviando peticiones paralelas. Es ahí cuando el rendimiento del sistema se ve afectado. Esto sucede principalmente al enviar a nuestro proxy peticiones paralelas, ya que son rechazadas, a pesar de que deberían aceptarse. Este límite de 400 usuarios emitiendo peticiones en paralelo es el que resulta más limitante, no obstante, no empeora drásticamente el retardo. Por otro lado, si tenemos muchas entidades almacenadas (50.000) comprobamos que el sistema se inestabiliza debido a lo lento que se vuelve.

Encontramos limitaciones tanto en el número de usuarios que acceden al sistema como a la cantidad de entidades almacenadas.

Cuando cargamos MongoDB, podemos pensar que la culpa del retardo es de Orion Context Broker en vez de MongoDB, no obstante, mediante una prueba de carga lineal directamente a la base de datos MongoDB comprobamos cómo conforme más se carga, más tarda en realizar cada petición a la base de datos, como vemos en el ejemplo de la Figura 83 en el que emitimos 30.000 peticiones POST de manera lineal, una tras otra, y el rendimiento empeora conforme avanzamos, ya que hay más entidades y tarda más en procesar las peticiones, ya sean POST o GET.

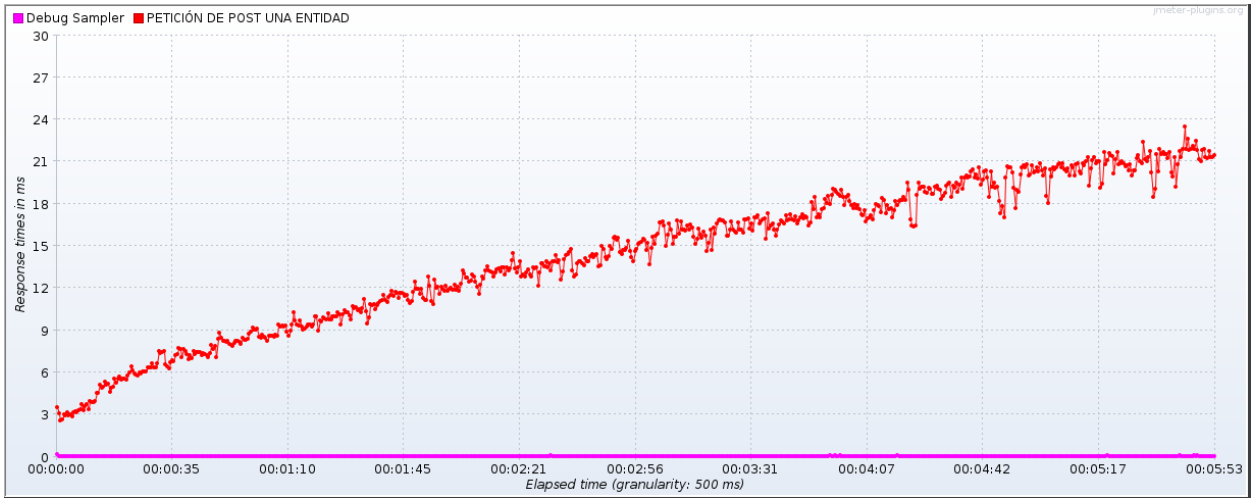


Figura 83. Evolución tiempos con carga de POST directa a MongoDB

Ahora vemos cómo la petición GET también ha empeorado, siendo normalmente el tiempo de GET directo a MongoDB de 5 ms, ahora es 26 ms, demostrando que la carga de esta base de datos MongoDB empeora el rendimiento de manera drástica (Figura 84).

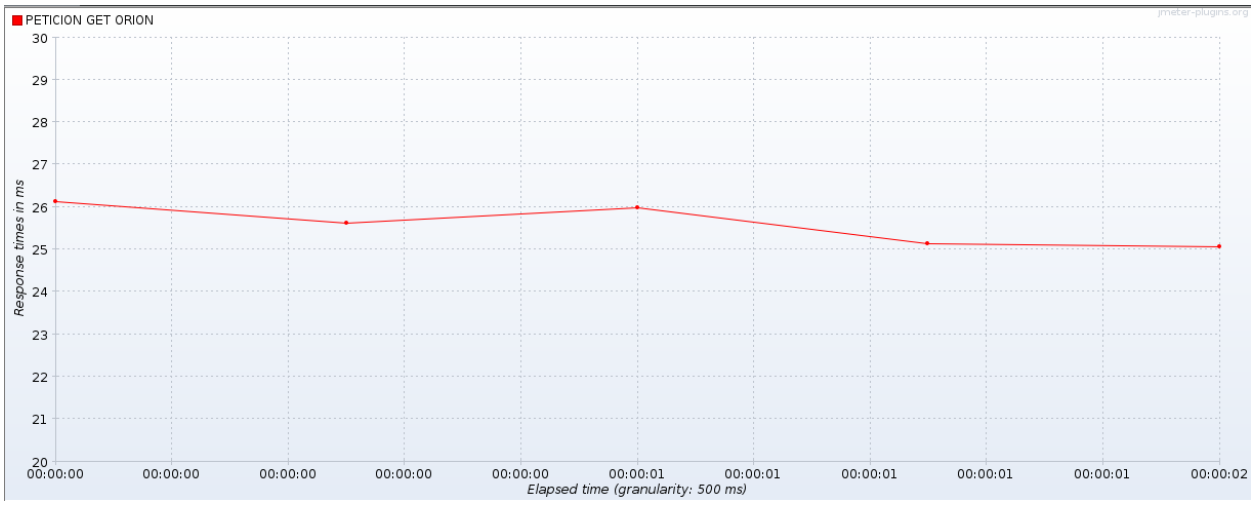


Figura 84. Tiempos GET ORION con MongoDB cargado

5.2 Posibles mejoras para el proyecto.

Pensando en un caso real, por ejemplo, en el Hospital Universitario Virgen Macarena de Sevilla, si tenemos 100 médicos encargados de las constantes vitales recogidas a 50 pacientes para cada médico, tendremos 5000 pacientes registrados como agentes IoT y 100 médicos registrados que consultarán la información de estos pacientes mediante una aplicación que muestra la información actualizada para las constantes de cada paciente, haciendo un GET cada 60 segundos. Los Agentes IoT actualizarán con un PATCH cada uno su información cada 30 segundos para asegurar que se recoge la información actualizada en los GET de los médicos.

En este caso tendremos 3 peticiones por minuto por cada usuario del sistema (paciente + médicos) por lo que llegarán 15300 peticiones por minuto. Suponiendo que el 50% de las peticiones PATCH llegan con 30 segundos entre ellas y que esto es tiempo suficiente para ser procesadas, no supondrán un problema, por lo que realmente confluyen una media de 5000 peticiones. Esto, pese a ser un número mucho menor que 15.300, sigue siendo insoportable para el sistema ya que muchas de estas peticiones, si se lanzan a la vez, serán rechazadas, por lo que habría que proponer una solución para este problema.

Por otro lado, si consideramos un despliegue para varios hospitales simultáneamente, probablemente se superen las 50.000 entidades registradas, por lo que habría que optimizar el sistema de alguna forma para que sea capaz de mantener su rendimiento pese a la amplia cantidad de sensores.

En el siguiente apartado se proponen soluciones para superar estas limitaciones y poder hacer que el sistema sea escalable manteniendo su rendimiento.

5.2.1 Separación de Peticiones

Según podemos apreciar en la Figura 85, una petición aislada al sistema de POST de una entidad tarda entre 1000 y 1200 ms en ser completada.

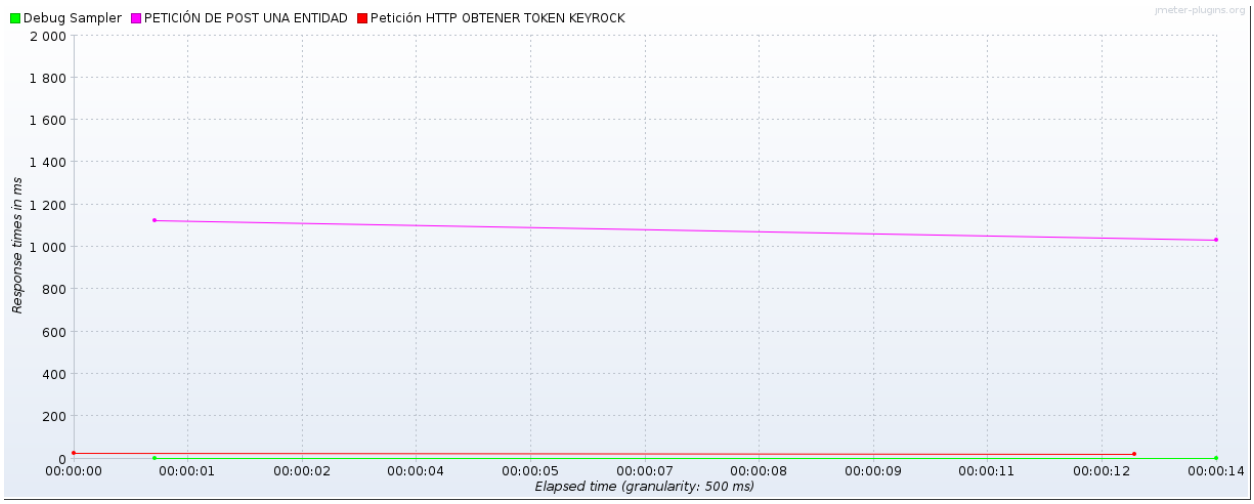


Figura 85. Tiempos petición POST aislada

No obstante, como se aprecia en el apartado 4.4 con 10.000 entidades almacenadas, la media por petición es de 8253 ms para una media de 300 usuarios emitiendo peticiones dado que cada 10 segundos se lanzan 100,150,200… usuarios de manera ascendente.

Esto se debe a que la carga del sistema es fundamentalmente afectada por la existencia de peticiones que llegan a la vez, ya que, de manera lineal con una mínima separación entre peticiones, el sistema no se satura. Por lo que, separando las peticiones para que se realicen una detrás de otra, programando los agentes IoT con un pequeño retardo cada uno y cuadrando los tiempos, podríamos hacer que el sistema mejorara en cuanto a tiempos de respuesta y errores por peticiones paralelas, pudiendo recoger los datos para los médicos cada 2 minutos por ejemplo tendríamos un amplio intervalo para añadir retardos para las peticiones, sacrificando velocidad en la obtención de la información para la aplicación de los médicos.

5.2.2 Duplicación base de datos MongoDB

Ahora trataremos de proponer una solución al problema de carga de MongoDB que hace que se retrase la respuesta a las peticiones. Por suerte, las bases de datos MongoDB basadas en el modelo NoSQL, presentan soluciones que las hacen altamente escalables, como por ejemplo la duplicación de la base de datos, técnica mediante la cual se pueden crear copias de una base de datos. Entre estas réplicas la carga se distribuye y hace que mejore el rendimiento, aumentando la capacidad del sistema en cuanto a entidades almacenadas y velocidades de respuesta.

Las bases de datos con capacidad de distribución suelen ser caras, por ello conviene buscar soluciones que no supongan un alto coste. Escalando horizontalmente nuestro sistema, es decir, duplicando instancias del mismo, en este caso MongoDB, y distribuyendo la carga, se puede alcanzar la escalabilidad que deseamos para mejorar el sistema. Esta escalabilidad estaría basada en un sistema de nodos comunicados entre ellos ampliando con ello la cantidad de puntos que pueden verse afectados por algún error.

MongoDB no solo es NoSQL, tecnología que facilita la creación de réplicas y la cooperación entre estas, sino que además es de código abierto, por lo que simplifica la tarea en cuestión.

5.2.3 Cola de Peticiones

En caso de necesitar abarcar una cantidad mayor de peticiones simultáneas por motivo de un aumento en la masa de agentes IoT que envían información, una solución, para que las peticiones no sean rechazadas por sobrecarga del sistema, sería añadir un proxy que actúe como cola suplementaria, que sea capaz de acumular peticiones con un timer, el cual expire en caso de sobrepasar determinado límite de espera, y así no manchar los datos en caso de sobrecarga, pero haciendo que el sistema sea capaz de soportar una mayor cantidad de peticiones paralelas.

Este proxy podría ser un servicio de colas con reintentos, en caso de que el sistema se cargue demasiado y las peticiones permanezcan demasiado tiempo en la cola, llegando a expirar, se pediría a los agentes IoT que reenvíen las últimas muestras de nuevo, las cuales se almacenarían y quedarían todos los datos registrados sin pérdidas. En caso de que las pérdidas no sean realmente importantes en determinado caso, con el timer sería suficiente, dando mayor fluidez a la cola y no permitiendo que se llene.

ANEXO A: INSTALACIÓN Y CONFIGURACIÓN DE DOCKER Y DOCKER-COMPOSE

Siendo Docker la herramienta utilizada para desplegar esta plataforma, se muestran a continuación los pasos para su instalación, configuración y solución de algunos problemas que pueden aparecer:

Lo primero es actualizar la máquina:

```
sudo apt-get update
sudo apt-get upgrade
```

Ahora procedemos a instalar el protocolo seguro https junto con ciertas dependencias:

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
```

Ahora debemos importar el comando mientras usamos la clave GPG correcta con el comando Curl. Para esto solo debemos escribir el siguiente comando:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Ahora procedemos a añadir el repositorio Docker APT a nuestro sistema de repositorios:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
```

A continuación instalamos Docker en la versión docker-ce:

```
sudo apt-get install docker-ce
```

Es necesaria la instalación de Docker-compose para componer nuestra aplicación completa (cuidado con las comillas y guiones al copiar y pegar):

```
sudo curl -L https://github.com/docker/compose/releases/download/1.21.2/docker-compose-‘uname -s’
‘uname -m’ -o /usr/local/bin/docker-compose
```

Para terminar la instalación de docker-compose otorgamos permisos de ejecución al binario.

```
sudo chmod +x /usr/local/bin/docker-compose
```

A continuación comprobamos las versiones de ambos servicios instalados ejecutando:

```
docker-compose –version
docker --version
```

Para lo que obtendremos la última versión disponible en el repositorio, que es la que hemos instalado.

Es habitual encontrar problemas de memoria utilizando Docker y más aún en un proyecto de pruebas en el cual cargamos de información las bases de datos, mandamos un gran volumen de peticiones, etc. Hay que tener en cuenta que en Docker por más que borremos el contenedor tras la ejecución, los datos quedan almacenados igualmente en el disco. Por ello cabe tener cuidado con la memoria disponible ya que de lo contrario nos veremos envueltos en un problema. Añado este apartado debido a que mi máquina virtual se saturó debido a las pruebas de carga y al funcionamiento de Docker, quedando inutilizada y perdiendo bastante tiempo en recuperarla.

Para ello reinstalaremos Docker, eliminando la carpeta en la que almacena todos los datos de cada ejecución:

```
sudo apt-get remove docker-ce
sudo rm -rf /var/lib/docker
sudo apt-get install docker-ce
```

Otra opción, para liberar memoria sin borrar Docker, sino a través de los comandos ofrecidos por Docker:

Primero mostramos los volúmenes existentes:

```
docker volume ls -f dangling=true
```

Para borrarlos todos los inactivos, o uno por uno:

```
docker volume rm $(docker volume ls -f dangling=true -q)
ó
docker volume rm [ Identificador del volumen a borrar]
```

Lo mismo se puede hacer para listar las imágenes y posteriormente borrarlas:

```
docker images -f dangling=true
docker rmi [ Identificador de la imagen a borrar]
```

Y para listar y borrar contenedores, lo mismo:

```
docker ps -a
docker rm [Id del contenedor]
```

ANEXO B: INSTALACIÓN Y CONFIGURACIÓN DE LOS COMPONENTES DE LA PLATAFORMA

Para entender el funcionamiento de la plataforma y el estado inicial de esta, y así preparar la carga de los diferentes módulos para las posteriores pruebas, ha sido necesario el estudio de diferentes ficheros de configuración del sistema, que se exponen a continuación.

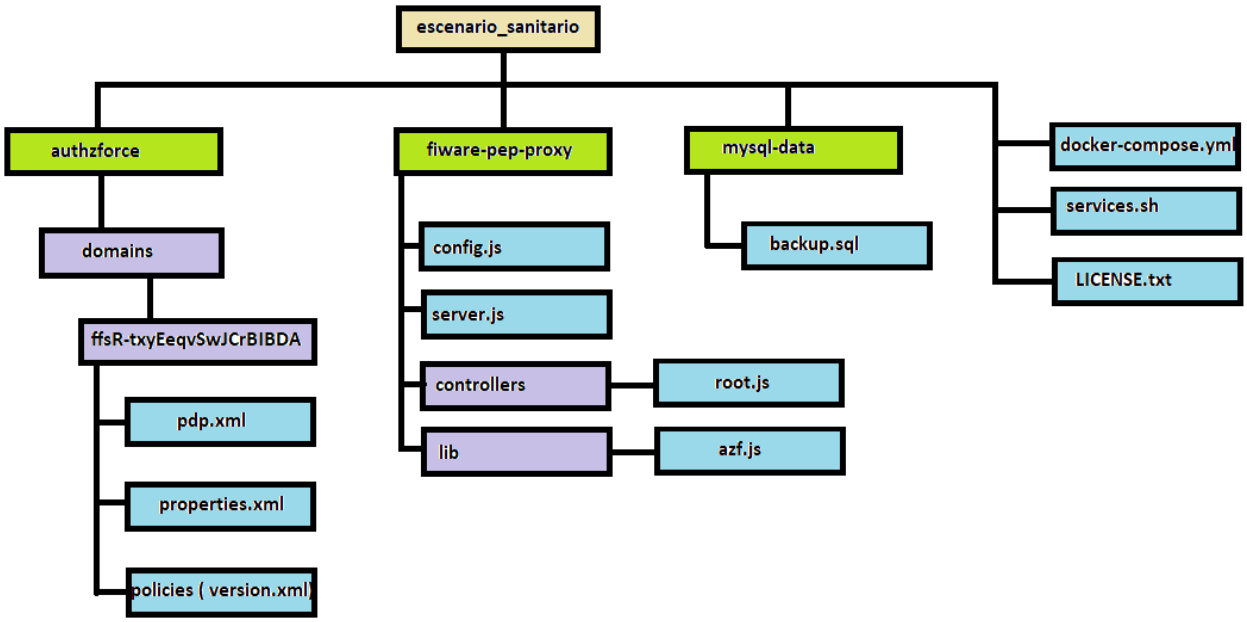


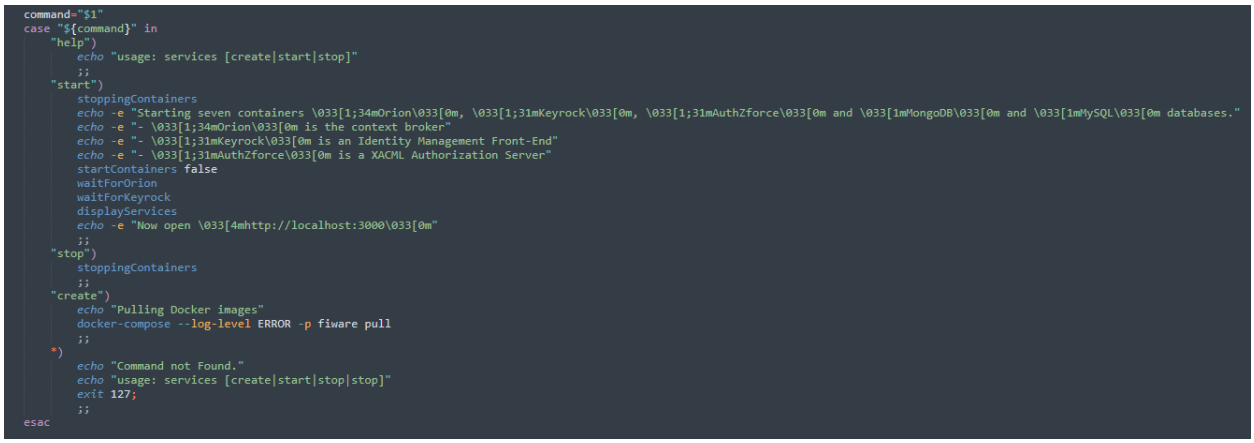
Figura 86. Distribución directorios y ficheros del sistema

La carpeta **authzforce** contiene los ficheros necesarios para el funcionamiento de dicho servicio. Dentro se encuentra la carpeta **domains**, que contiene los diferentes dominios de políticas. En este caso solo se ha utilizado uno, ya que a Wilma se le debe indicar cuál va a ser utilizado, por lo que en la base de datos existirá este identificador de dominio que se indica en la imagen: **ffsR-txyEeqvSwJCrBIBDA**. Dentro del dominio en cuestión, se encuentra el fichero **pdp.xml** que indica el número de PolicySets y VariableDefinition que pueden existir en una política, luego en el fichero **properties.xml** se indican el máximo de políticas por dominio y máximo de versiones por política. Para las pruebas del escenario y las de carga posteriores se han utilizado diversas versiones de política para ir cambiándola de manera sencilla. Cuando se añade una versión de número mayor que la anterior, es esta última la que se tiene en cuenta.

En el directorio **fiware-pop-proxy** encontramos los archivos de configuración del proxy, como **config.js** y server.js. También encontramos el controlador, llamado root.js que define el funcionamiento de Authzforce. Este fichero ha sido utilizado para evaluar la posibilidad de introducir nuevos parámetros a evaluar en las políticas. El fichero **azf.js** contiene las funciones a las que Wilma llamará para interactuar con Authzforce.

En **mysql-data** encontramos el fichero de base de datos backup.sql que contiene la situación inicial que tendrá la base de datos, tanto las tablas como valores iniciales. En nuestro caso, hemos cargado la base de datos partiendo de esta situación inicial, utilizando JMeter para conectarnos a las bases de datos y llenarlas con peticiones, utilizando ficheros CSV para rellenar las peticiones con ciertos datos (nombres, DNI, email…) generados a partir de scripts en Python.

Como parte del estudio del despliegue de la plataforma, cabe destacar el papel del fichero **services.sh** y **docker-compose.yml**. El fichero **services.sh** es el que se utiliza para iniciar los diferentes servicios que se despliegan en contenedores de Docker. Este script puede ser lanzado en diferentes modos, en "create" para hacer pull de las imágenes, en "start" para iniciar los containers, o en "stop" siendo este último modo el encargado de parar los containers.



```
command="$1"
case "${command}" in
    "help")
        echo "usage: services [create|start|stop]"
        ;;
    "start")
        stoppingContainers
        echo -e "Starting seven containers \033[1;34mOrion\033[0m, \033[1;31mKeyrock\033[0m, \033[1;31mAuthZforce\033[0m and \033[1mMongoDB\033[0m and \033[1mMySQL\033[0m databases."
        echo -e "- \033[1;34mOrion\033[0m is the context broker"
        echo -e "- \033[1;31mKeyrock\033[0m is an Identity Management Front-End"
        echo -e "- \033[1;31mAuthZforce\033[0m is a XACML Authorization Server"
        startContainers false
        waitForOrion
        waitForKeyrock
        displayServices
        echo -e "Now open \033[4mhttp://localhost:3000\033[0m"
        ;;
    "stop")
        stoppingContainers
        ;;
    "create")
        echo "Pulling Docker images"
        docker-compose --log-level ERROR -p fiware pull
        ;;
    *)
        echo "Command not Found."
        echo "usage: services [create|start|stop|stop]"
        exit 127;
        ;;
esac
```

Figura 87. Opciones script services.sh

En el fichero **Docker-compose.yml** se especifica el despliegue de cada contenedor:

- Orion Context Broker:

```
orion:
  image: fiware/orion:2.4.0
  hostname: orion
  container_name: fiware-orion
  depends_on:
    - mongo-db
  networks:
    default:
      ipv4_address: 172.18.1.9
  expose:
    - "1026"
  ports:
    - "1026:1026"
  command: -dbhost mongo-db -logLevel DEBUG
  healthcheck:
    test: curl --fail -s http://localhost:1026/version || exit 1
```

Figura 88. Docker-compose. Orion Context Broker

- **image**: se indica la imagen Docker a descargar junto con la versión requerida.
- **hostname**: define el nombre dentro de la red creada.
- **depends_on**: indica las dependencias de arranque de servicios. Antes de poner en marcha este contenedor, se iniciarán las dependencias. Para cerrar los contenedores, se cierran antes los que dependen de otros.
- **networks:** establece la subred donde se encontrará el contenedor.
- **expose:** establece puertos que solo estarán disponibles dentro de los servicios definidos, no en la máquina donde se ejecuta.
- **ports:** puertos que se van a abrir de cara al exterior, para que sean accesibles desde fuera
- **healthcheck:** establece un comando para determinar si el contenedor está correctamente desplegado

Keyrock

```
keyrock:
  image: fiware/idm:7.8.1
  container_name: fiware-keyrock
  hostname: keyrock
  networks:
    default:
      ipv4_address: 172.18.1.5
  depends_on:
    - mysql-db
    - authzforce
  ports:
    - "3005:3005"
  environment:
    - DEBUG=idm:*
    - IDM_DB_HOST=mysql-db
    - IDM_DB_PASS=secret
    - IDM_DB_USER=root
    - IDM_HOST=http://localhost:3005
    - IDM_PORT=3005
    - IDM_ADMIN_USER=fernando_admin_aplicacion
    - IDM_ADMIN_EMAIL=fernando-the-admin@tfg.com
    - IDM_ADMIN_PASS=test
    - IDM_PDP_LEVEL=advanced
    - IDM_AUTHZFORCE_ENABLED=true
    - IDM_AUTHZFORCE_HOST=authzforce
    - IDM_AUTHZFORCE_PORT=8080
  healthcheck:
    test: curl --fail -s http://localhost:3005/version || exit 1
```

Figura 89. Docker-compose. Keyrock

- **image:** se utiliza la versión del contenedor 7.8.1
- **depends_on:** depende de MySQL, para almacenar información sobre credenciales, aplicaciones registradas, usuarios registrados, etc.
- **ports:** Keyrock está disponible a través del puerto 3005, tanto para la conexión con el PEP Proxy para solicitar tokens de acceso.
- **IDM_DB_USER:** especifica el usuario con acceso a la base de datos.
- **IDM_ADMIN_USER:** nombre del administrador de Keyrock.
- **IDM_ADMIN_EMAIL:** email con el que acceder a la configuración de Keyrock
- **IDM_ADMIN_PASS:** contraseña definida para acceder a la configuración de Keyrock junto con el email.
- **IDM_PDP_LEVEL:** especifica qué tipo de nivel va a ser el PDP, con “advanced” indicamos que Authzforce va a actuar como PDP, y no Keyrock.
- **IDM_AUTHZFORCE_ENABLED:** Indicamos si la conexión Keyrock y Authzforce está o no habilitada.

- Authzforce:

```
authzforce:
  image: fiware/authzforce-ce-server:release-8.1.0
  hostname: authzforce
  container_name: fiware-authzforce
  networks:
    default:
      ipv4_address: 172.18.1.12
  ports:
    - "8080:8080" # localhost:8080
  volumes:
    - ./authzforce/domains:/opt/authzforce-ce-server/data/domains
  healthcheck:
    test: curl --fail -s http://authzforce:8080/authzforce-ce/version || exit 1
```

Figura 90. Docker-compose. Authzforce

- **Volumes**: En el primer argumento indicamos la ruta donde queremos que se almacenen las políticas, el segundo argumento es la ruta donde se monta el archivo o directorio en el contenedor.
- MongoDB y MySQL:

```
# Databases
mongo-db:
  image: mongo:3.6
  hostname: mongo-db
  container_name: db-mongo
  ports:
    - "27017:27017" # localhost:27017
  expose:
    - "27017"
  networks:
    - default
  command: --bind_ip_all --smallfiles
  volumes:
    - mongo-db:/data

mysql-db:
  restart: always
  image: mysql:5.7
  hostname: mysql-db
  container_name: db-mysql
  expose:
    - "3306"
  ports:
    - "3306:3306" # localhost:3306
  networks:
    default:
      ipv4_address: 172.18.1.6
  environment:
    - "MYSQL_ROOT_PASSWORD=secret"
    - "MYSQL_ROOT_HOST=172.18.1.5" # Allow Keyrock to access this database
  volumes:
    - mysql-db:/var/lib/mysql
    - ./mysql-data:/docker-entrypoint-initdb.d/:ro # Preload Keyrock Users
```

Figura 91. Docker-compose. MongoDB y MySQL

- **volumes** (en MySQL): se especifica el fichero que va a exponer el estado inicial de la base de datos. Se explicará con detalle dicho estado inicial en el siguiente apartado
- **ports**: Importante en este caso consultar este fichero para saber a dónde dirigir las peticiones para cargar las bases de datos.
- **Environment**: Indicamos password y host para así permitir el acceso directo de Keyrock.

- PEP Proxy Wilma:

Este componente no será desplegado con Docker, lo instalaremos por código fuente. Para ello lo primero que se debe hacer es instalar nodejs y npm:

```
curl -sL https:// deb.nodesource.com/setup_13.x | sudo -E bash –
sudo apt-get install nodejs
sudo apt install npm
```

En caso de obtener el error: 'Cannot find module xmlhttprequest':

```
npm install xmlhttprequest
```

Al igual que anteriormente, comprobamos que se ha instalado mirando la versión:

```
node -v
npm -v
```

Para iniciar el servidor, ejecutamos:

```
sudo node server
```

ANEXO C: INSTALACIÓN DE POSTMAN

Tanto para las primeras pruebas realizadas en la plataforma, comprobando las respuestas de las llamadas a las funciones de la API de cada servicio y corroborando su funcionamiento, como para el cambio entre políticas para las pruebas de carga posteriores, se ha utilizado Postman.

Para instalar este servicio es conveniente tener previamente instalado snapd:

```
sudo apt-get install snapd
```

Y a continuación instalaremos postman con dicho servicio:

```
sudo snap install postman
```

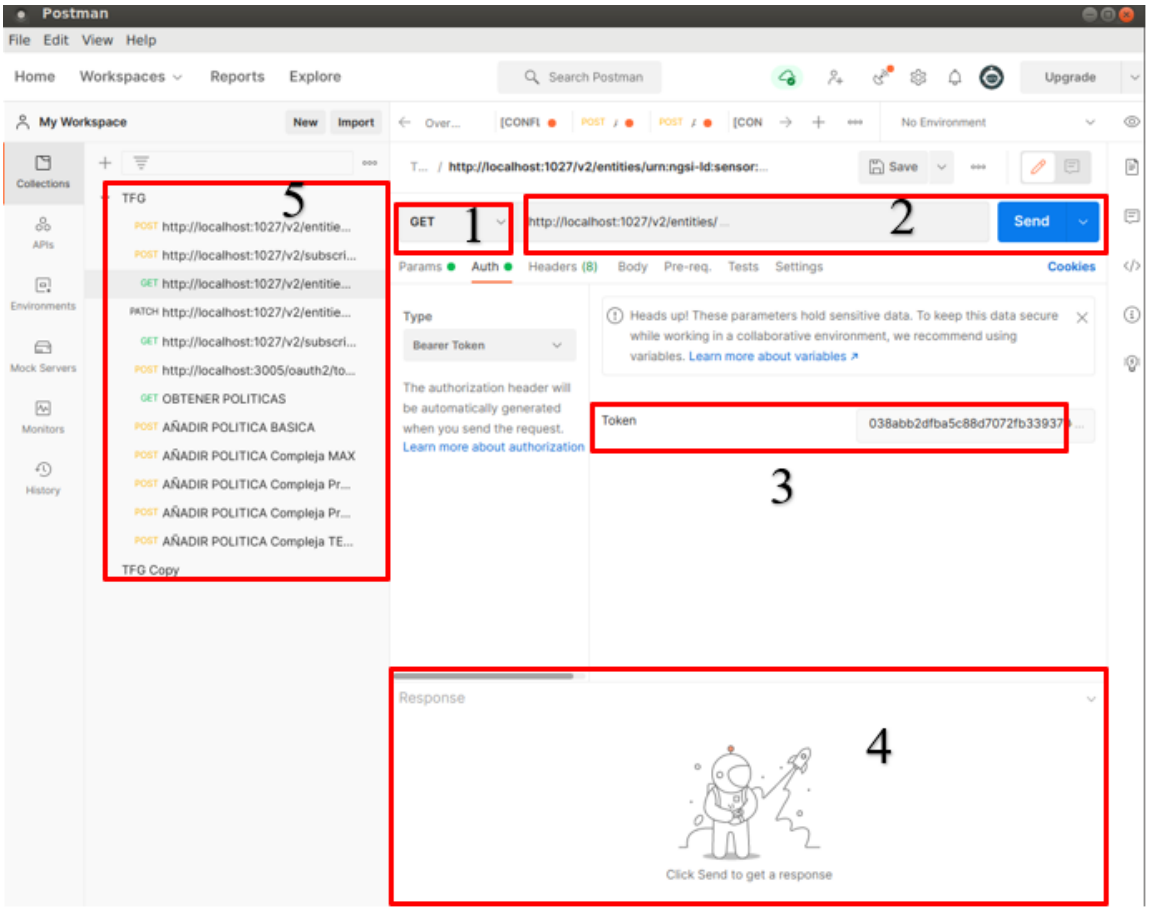


Figura 92. Interfaz de Postman

1. Aquí se selecciona la acción a llevar a cabo.
2. Indicamos la url a la que mandamos la petición y pulsamos en send.
3. El token viaja en la petición, en el apartado Auth.
4. La respuesta se muestra en este recuadro.
5. En este apartado se muestran las peticiones previamente guardadas.

ANEXO D: INSTALACIÓN Y CONFIGURACIÓN DE JMETER:

El software de la carga y pruebas, JMeter, será instalado descargando en su versión 5.2.1, ya que la que se haya en el repositorio APT, obtenida con apt-get, es una versión previa, que ofrece menores posibilidades de cara a las pruebas.

Primero obtenemos el paquete de la versión que nos interesa:

```
wget http://apache.ip-connect.vn.ua//jmeter/binaries/apache-jmeter-5.2.1.tgz
```

Segundo, la extraemos:

```
tar xf apache-jmeter-5.2.1.tgz
```

Ahora movemos la carpeta a la dirección en la que se encuentran las aplicaciones reconocidas por el sistema, y por tanto haremos posible que aparezca en la barra de aplicaciones “Programación”:

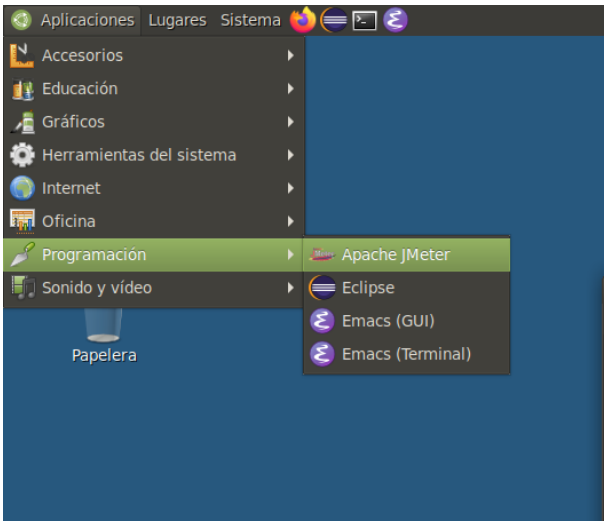


Figura 93. Acceso directo JMeter

Para ello ejecutamos:

```
sudo mv apache-jmeter-5.2.1 /usr/share/jmeter
```

Ahora creamos un enlace simbólico desde la carpeta de binarios al ejecutable de JMeter:

```
ln -s /usr/share/jmeter/bin/jmeter /usr/bin/jmeter
```

JMeter ofrece la siguiente interfaz:

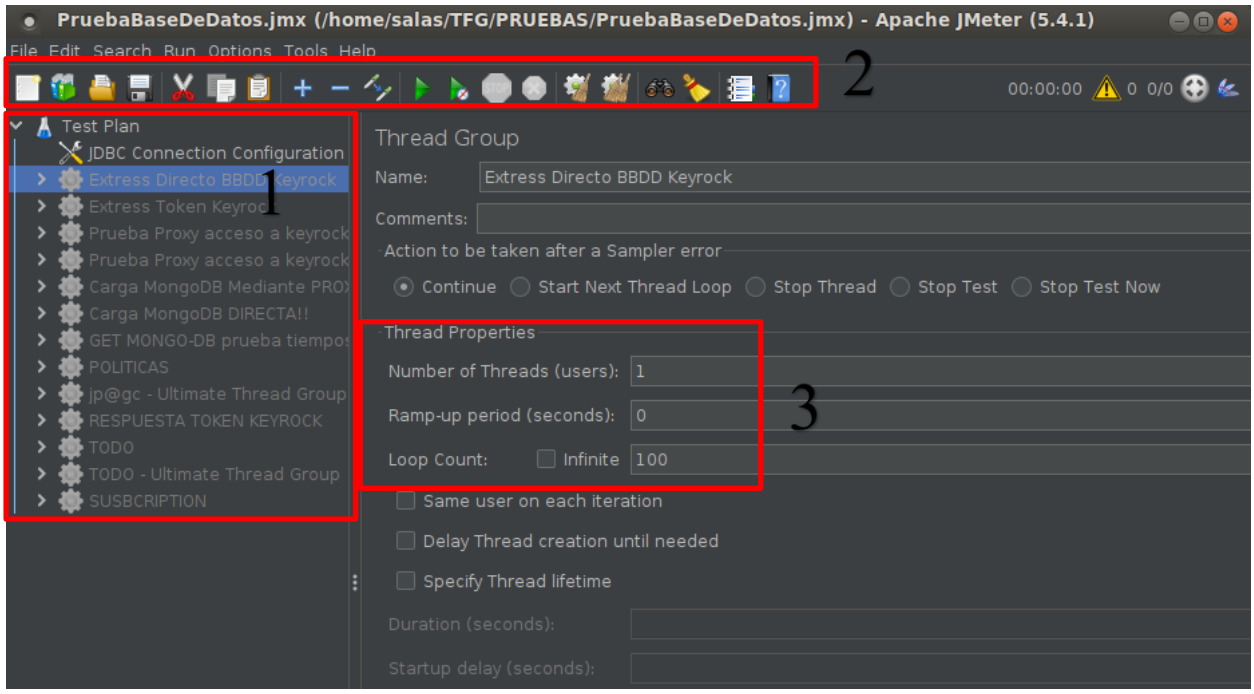


Figura 94. Interfaz JMeter

1. Lo primero que se observa es un Test Plan, en el que definiremos las pruebas a ser realizadas cuando iniciemos la ejecución. Solo se ejecutarán aquellas que hayan sido habilitadas previamente.

2. En este bloque encontramos los diferentes botones que ofrecen un abanico de acciones a realizar, vamos a describir aquellos más interesantes y desconocidos.

Figura 95. Botones Interfaz JMeter

   a. Este botón se utiliza para habilitar o deshabilitar un grupo de hilos previamente seleccionado de los que se encuentran en el Test Plan.

   b. Estos botones lanzan la ejecución de las pruebas del Test Plan, el izquierdo lo hace en modo "sin pausa" no parándose en ningún momento hasta terminar la simulación.

   c. Los botones de pausa, el izquierdo paraliza la ejecución mientras que el derecho apaga todos los hilos en ejecución.

   d. Aquí tenemos los botones de limpieza, el izquierdo se encarga de borrar todo lo referido a las pruebas seleccionadas, y el derecho borra absolutamente todos los datos simulados de todo el Test Plan.

   e. Estos son los botones de búsqueda por palabras, el derecho borra la búsqueda realizada.

3. Tenemos seleccionado un "Grupo de Hilos" dentro del Test Plan, dentro del grupo de hilos podemos seleccionar cuántos hilos se ejecutarán, y cuántas ejecuciones realizará cada uno de las acciones que situamos dentro del grupo de hilos, entre otras acciones.

Un ejemplo de un grupo de hilos que contiene diferentes acciones a ser realizadas sería el siguiente:

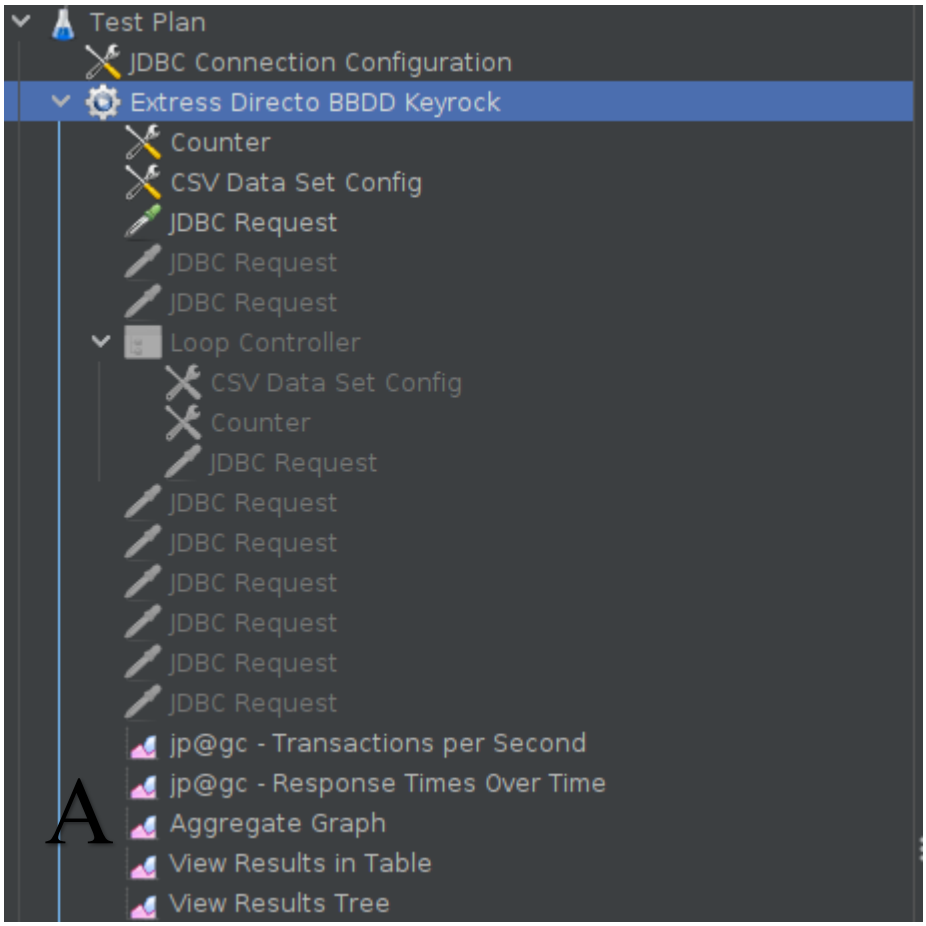


Figura 96. Grupo de Hilos JMeter

Aquí podemos apreciar una serie de peticiones que los hilos realizarán de manera consecutiva y cuyas respuestas serán manejadas por los elementos señalados por el símbolo “A”. Estos son “Listeners” que se encargan de mostrar los resultados en tablas, gráficas…

Para añadir un Grupo de hilos:

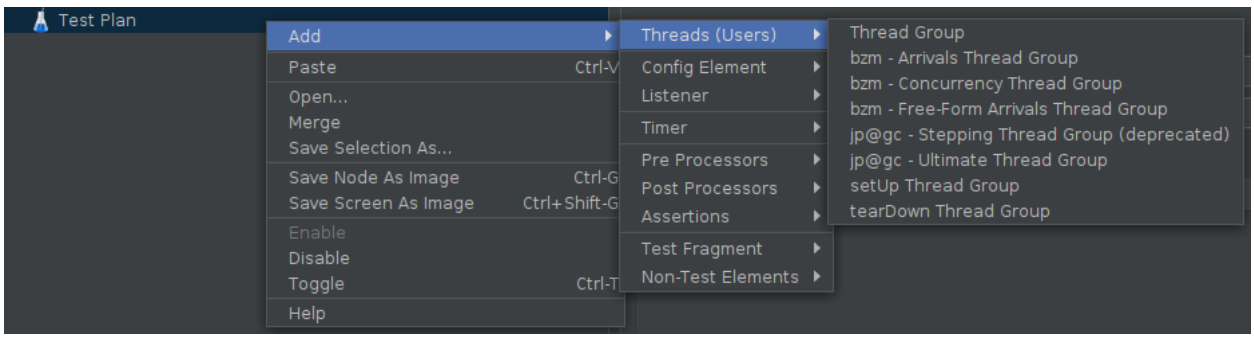


Figura 97. Añadir grupo de hilos JMeter

Para añadir un elemento al grupo de hilos:

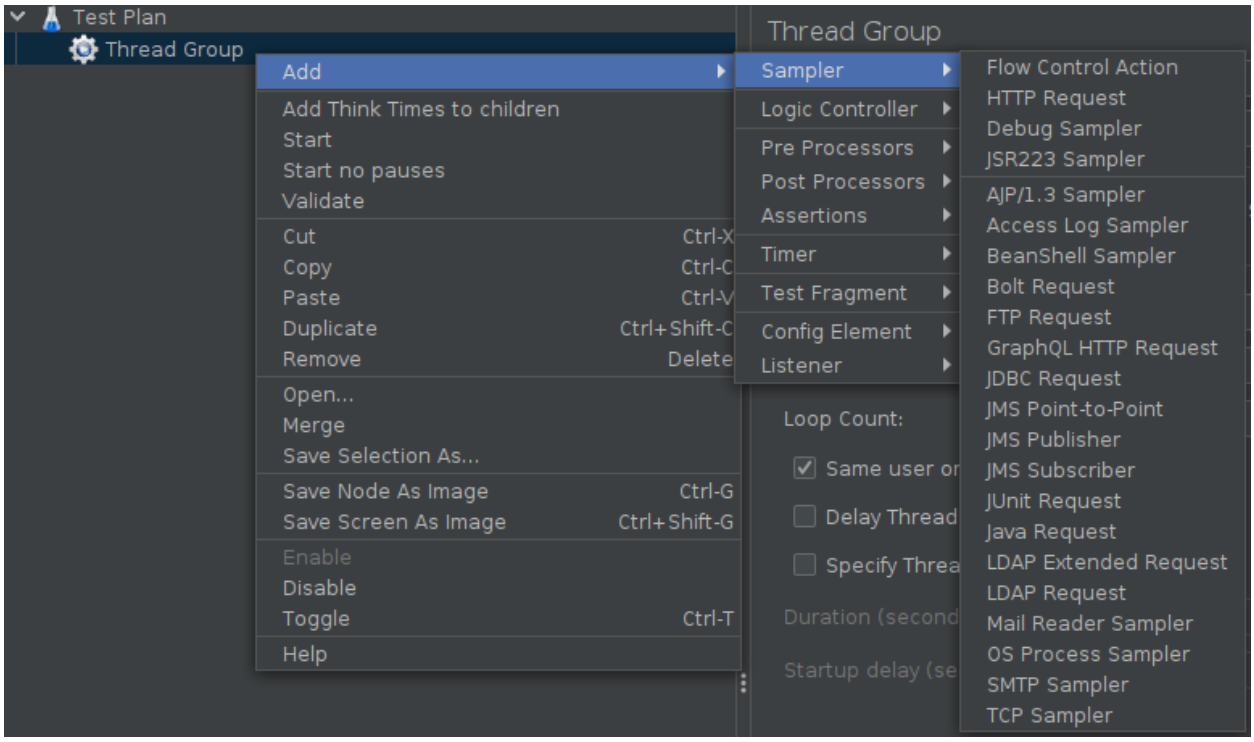


Figura 98. Añadir elementos JMeter

De este modo se pueden añadir una gran cantidad de elementos al grupo de hilos que se pueden consultar en [15].

Además de los elementos que aporta JMeter, para este proyecto se han utilizado algunos añadidos de manera extra instalando un complemento llamado “Plugins Manager”, para instalar un nuevo plugin se siguen los siguientes pasos:

Primero descargamos el Plugins Manager, en este caso la versión 1.6 ya que es la última disponible, debemos colocarlo en la carpeta lib/ext :

```
cd /usr/share/jmeter/lib/ext
curl -O https://repo1.maven.org/maven2/kg/apc/jmeter-plugins-manager/1.6/jmeter-plugins-manager-
```

Ahora reiniciamos JMeter, y nos aparecerá la siguiente pestaña:

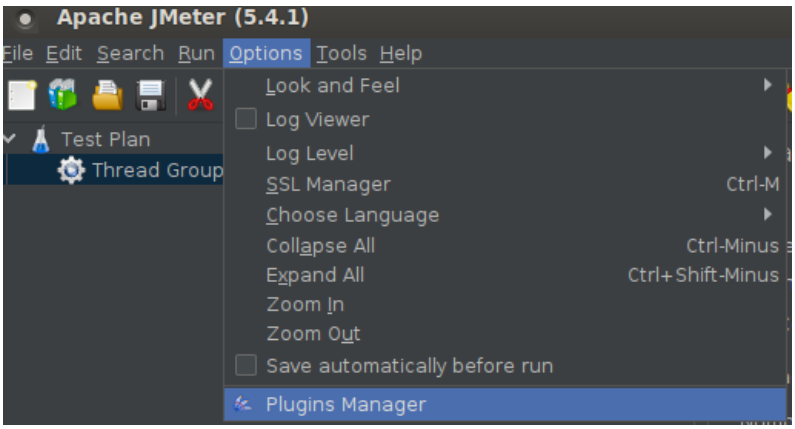


Figura 99. Pestaña Plugins Manger JMeter

Si accedemos a esta pestaña, tendremos dentro la interfaz de descarga de plugins de JMeter en la que se pueden ver los plugins utilizados en este proyecto, existen una infinidad que se pueden añadir para diferentes tipos de pruebas, esto es una ventaja de utilizar este software de pruebas de código abierto, ya que nosotros mismos podemos desarrollar tipos de pruebas y complementos para la aplicación:

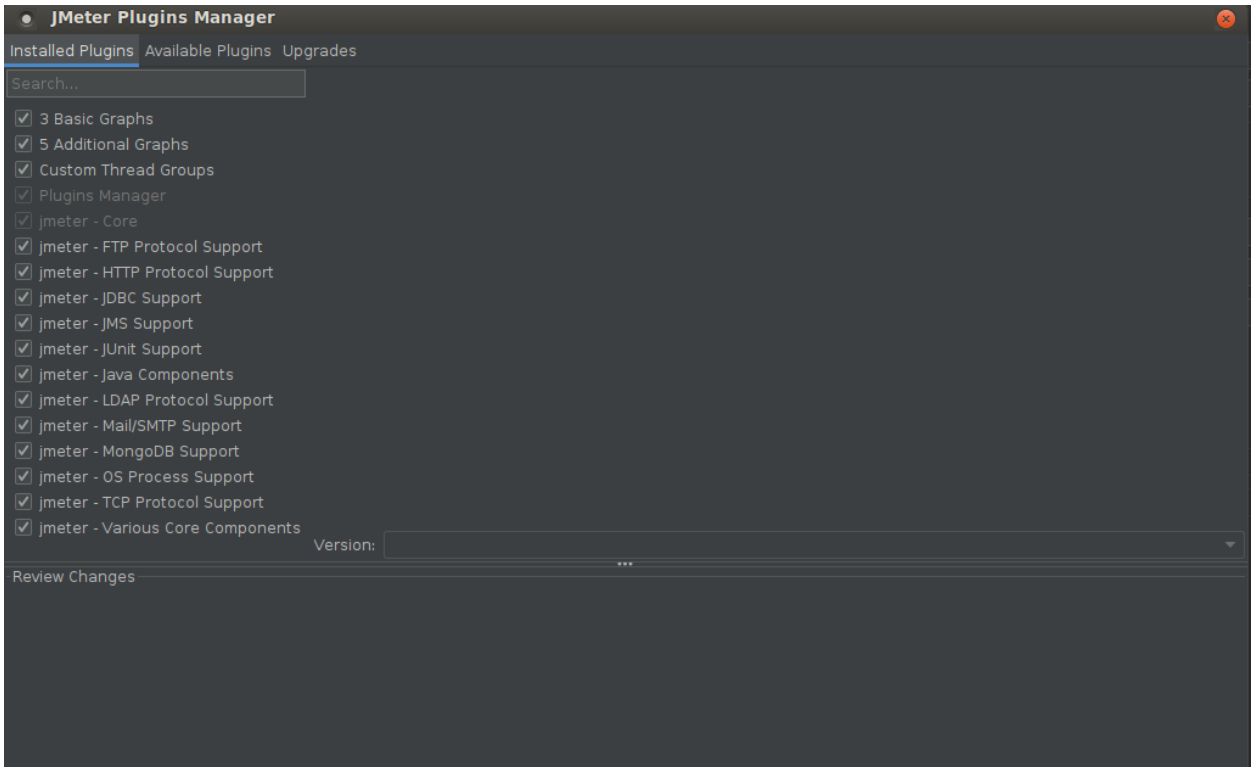


Figura 100. Plugins Manager Menu JMeter

Para la utilización del conector a la base de datos MySQL ha hecho falta utilizar un controlador JDBC, instalado de la siguiente manera:

```
cd /usr/share/jmeter/lib

curl -O https://jar-download.com/artifacts/mysql/mysql-connector-java/8.0.23/source-code/mysql-connector-java-8.0.23.jar
```

Quedando ubicado en la carpeta de librerías de JMeter que, tras su reinicio, tendrá disponible dicho conector y podremos conectarnos a la base de datos MySQL, con la configuración precisa para ello (Figura 32).

```
-rw-r--r-- 1 salas salas 2415192 abr  7 15:27 mysql-connector-java-8.0.23.ja
-rw-r--r-- 1 salas salas 4580832 ene  2  1970 neo4j-java-driver-4.2.0.jar
-rw-r--r-- 1 salas salas   65261 ene  2  1970 oro-2.0.8.jar
-rw-r--r-- 1 salas salas 1302550 ene  2  1970 ph-commons-9.5.1.jar
-rw-r--r-- 1 salas salas  506329 ene  2  1970 ph-css-6.2.3.jar
-rw-r--r-- 1 salas salas   11369 ene  2  1970 reactive-streams-1.0.3.jar
-rw-r--r-- 1 salas salas 1315838 ene  2  1970 rhino-1.7.13.jar
-rw-r--r-- 1 salas salas 1196575 ene  2  1970 rsyntaxtextarea-3.1.1.jar
-rw-r--r-- 1 salas salas 5506669 ene  2  1970 Saxon-HE-9.9.1-8.jar
-rw-r--r-- 1 salas salas  276420 ene  2  1970 serializer-2.7.2.jar
-rw-r--r-- 1 salas salas   41472 ene  2  1970 slf4j-api-1.7.30.jar
-rw-r--r-- 1 salas salas  301161 ene  2  1970 svgSalamander-1.1.2.3.jar
-rw-r--r-- 1 salas salas  708157 ene  2  1970 tika-core-1.24.1.jar
-rw-r--r-- 1 salas salas 1336431 ene  2  1970 tika-parsers-1.24.1.jar
-rw-r--r-- 1 salas salas 3154938 ene  2  1970 xalan-2.7.2.jar
-rw-r--r-- 1 salas salas 1386397 ene  2  1970 xercesImpl-2.12.0.jar
-rw-r--r-- 1 salas salas  220536 ene  2  1970 xml-apis-1.4.01.jar
-rw-r--r-- 1 salas salas  671727 ene  2  1970 xmlgraphics-commons-2.3.jar
-rw-r--r-- 1 salas salas    7188 ene  2  1970 xmlpull-1.1.3.1.jar
-rw-r--r-- 1 salas salas   24956 ene  2  1970 xpp3_min-1.1.4c.jar
-rw-r--r-- 1 salas salas  627848 ene  2  1970 xstream-1.4.15.jar
salas@linux:/usr/share/jmeter/lib$
```

Figura 101. Conector mysql JMeter. Carpeta de plugins

Para que JMeter sea capaz de conectarse a la base de datos es necesario que posea un usuario válido, para solucionar esto he escrito un script que primero crea un usuario llamado “usernameall” accediendo con las credenciales de root, y luego permite el acceso a este usuario:

```
1   #!/bin/bash
2
3   mysqlContainerName=db-mysql
4   mysqlRootUsername=root
5   mysqlRootPassword=secret
6   mysqlUsername=usernameall
7   mysqlPassword=ThePassword
8   mysqlDb=project_db
9
10
11  docker exec $mysqlContainerName bash -c "mysql -u$mysqlRootUsername -p$mysqlRootPassword
12  -e \"CREATE USER '$mysqlUsername'@'%' IDENTIFIED BY '$mysqlPassword'; \"; exit;"
13
14  docker exec $mysqlContainerName bash -c "mysql -u$mysqlRootUsername -p$mysqlRootPassword
15  -e \"grant all on *.* to '$mysqlUsername'@'%'; \"; exit;"
```

Figura 102. Script conexión base de datos MySQL

Este script se ejecuta como cualquier otro, necesitando permisos de root:

```
sudo ./UserBBDDauth
```

REFERENCIAS

[1] «Página web oficial de FIWARE,» [En línea]. Available: https://www.fiware.org.

[2] «Orion Context Broker,» [En línea]. Available: https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/ocb/.

[3] «Identity Manager - Keyrock,» [En línea]. Available: https://fiware-idm.readthedocs.io/en/latest/.

[4] «PEP Proxy - Wilma,» [En línea]. Available: https://fiware-pep-proxy.readthedocs.io/en/latest/.

[5] «Authzforce,» [En línea]. Available: https://authzforce-ce-fiware.readthedocs.io/en/latest/.

[6] «Ubuntu 20.04.0 LTS,» [En línea]. Available: https://releases.ubuntu.com/20.04/.

[7] «VMware Workstation Player 16,» [En línea]. Available: https://www.vmware.com/es/products/workstation-player/workstation-player-evaluation.html.

[8] «Postman,» [En línea]. Available: https://www.postman.com/.

[9] «Docker,» [En línea]. Available: https://www.docker.com/.

[10] «Docker-Compose,» [En línea]. Available: https://docs.docker.com/compose/.

[11] «MongoDB,» [En línea]. Available: https://www.mongodb.com/es.

[12] «MySQL,» [En línea]. Available: https://www.mysql.com.

[13] «Wireshark,» [En línea]. Available: https://www.wireshark.org/.

[14] «JMeter,» [En línea]. Available: https://jmeter.apache.org/.

[15] «Plugins JMeter,» [En línea]. Available: https://jmeter-plugins.org/wiki/Start/.