Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MÁSTER

COMPARATIVA TEÓRICA Y PRÁCTICA DE MIDDLEWARES MQTT

Alumno/Alumna *Gil, Inchaurza, Gonzalo*
Director/Directora *Higuero, Aperribai, Marivi*
Departamento **Ingeniería de Comunicaciones**
Curso académico **2017/2018**

Bilbao, 24, septiembre, 2018

Resumen trilingüe

Castellano

En este proyecto, en base a un caso de uso dado se realiza una comparación teórica y práctica de middlewares MQTT para conocer y desplegar la solución que garantice el cumplimiento de una serie de prestaciones inicialmente definidas ofreciendo el mejor rendimiento en base a unos indicadores definidos. Primeramente, se define el caso de uso y las prestaciones a cumplir. A continuación, se definen las características mínimas que han de ofrecer los middlewares MQTT para poder garantizar dichas prestaciones. Posteriormente, descartados los middlewares MQTT que no garanticen alguna de las prestaciones se realiza un análisis del rendimiento de los resultantes con el objetivo de conocer aquel que optimice el rendimiento en base a unos indicadores inicialmente definidos. Finalmente, conocida la solución óptima se procede a desplegar la misma en el caso de uso definido.

Euskera

Proiektu honetan, erabilera-kasuetan oinarrituta, MQTT-ren teoriko eta praktika konparatiboak konparatu behar dira, hasiera batean definitutako prestazioen serie bat betetzen dela ziurtatzeko irtenbide bat zabaldu eta zabaldu ahal izateko. Lehenik eta behin, erabilera kaskoa eta prestazioak aitortzea definitzen du. Ondoren, MQTT bitarteko erdiek eskaintzen dituzten gutxieneko ezaugarriak definituko dira, horien prestazioak bermatzeko. Posteriormente, los MQTT de los intermediarios descartados que no garanticen alguna de las prestaciones se realiza un análisis de los resultados de los resultantes con el objetivo de conocer qué se optimiza el rendimiento en base a unos indicadores inicialmente definidos. Azkenean, ezagupenaren optimizazioari dagokionean, berau definitutako erabilera desplegatu behar da.

Inglés

In this project, a theorical and practical comparison of middleware MQTT is made based on a use of case so as to know and deploy the solution that garantees the fulfillment of a group of requirements that have been inittialy defined offering the best performance based on defined indicators. Firstly, a use of case and the requirements to fulfill are defined. Secondly, the mínimum characteristics that MQTT middlewares have to offer in order to guarantee those requirements are defined. Later, once the MQTT middlewares that do not guarantee one of the requirements have been discarded a performance analysis is performed in order to know who optimizes the performance of some indicators defined in the first instance. Finally, if the optimum solution is known, it is deployed in the defined use of case.

Índice

Resumen trilingüe 2
Castellano 2
Euskera 2
Inglés 2
Índice de figuras 6
Índice de tablas 7
Acrónimos 8
1. Introducción 10
1.1. Escenario de desarrollo y operación 11
2. Contexto 13
2.1. Nivel de penetración de escenarios IoT 13
2.2. Tecnologías relacionadas 14
2.2.1. Sistemas distribuidos 14
2.2.2. Sistema de control de intercambio de información. Middleware 14
2.2.3. Paradigma publicación/suscripción 15
3. Objetivos 16
4. Beneficios 18
4.1. Beneficios económicos 18
4.2. Beneficios técnicos 18
4.3. Beneficios sociales 18
5. Análisis del estado del arte 19
5.1. Benchmark of MQTT Servers 19
5.2. Kafka vs rabbitMQ 20
6. Análisis de alternativas 22
6.1. Alternativas para la adquisición de datos 22
6.1.1. XMPP 22
6.1.2. AMQP 23
6.1.3. MQTT 24
6.1.4. Ponderación de las alternativas 25
6.1.5. Solución adoptada 26
6.2. Alternativas para el análisis de rendimiento 26
6.2.1. Equipamiento real 26
6.2.2. Simulación 27
6.2.3. Ponderación de las alternativas 28

6.2.4. Solución adoptada 28
7. Descripción de la solución 29
7.1. Escenario de aplicación 29
7.1.1. Descripción del sistema 29
7.1.2. Formato de los mensajes 30
7.1.3. Protocolo de intercambio de mensajes 30
7.1.4. Topología 31
7.2. Metodología 31
7.2.1. Estado del arte. Middlewares MQTT 32
7.2.2. Aspectos críticos 32
7.2.3. Medidas a realizar 36
7.2.4. Escenario de pruebas 37
7.2.5. Pruebas realizadas 39
7.2.6. Resultados obtenidos 41
7.2.7. Despliegue de la solución adoptada 48
8. Descripción de tareas. Gantt 51
8.1. Diagrama de Gantt 54
9. Coste del trabajo 55
9.1. Horas internas 55
9.2. Amortizaciones 56
9.3. Gastos 56
9.4. Subcontrataciones 56
9.5. Coste total del proyecto 56
10. Conclusiones 57
Bibliografía 58
ANEXOS 60
ANEXO I: Diagramas de flujo 60
Mensajes de control intercambiados para garantizar un nivel de QoS dado 60
Mensajes de control LDAP 62
SSL Handshake 63
Anexo II: Ficheros de configuración 64
Docker 64
Middlewares 68

Índice de figuras

Ilustración 1: Escenario de desarrollo y operación *11*
Ilustración 2: Sistema distribuido *14*
Ilustración 3:Sistema de control de intercambio de información. Middleware *14*
Ilustración 4: Lógica de rutado basada en topic *15*
Ilustración 5: Lógica de rutado basada en contenido *15*
Ilustración 6: Test escenario *20*
Ilustración 7: Topología del escenario de aplicación *31*
Ilustración 8: Maqueta del escenario de pruebas *39*
Ilustración 9: Throughput de salida en función del nº de clientes y el nivel de QoS *41*
Ilustración 10: Latencia en función del nº de clientes y el nivel de QoS *42*
Ilustración 11: Log del middleware EMQ y servidor LDAP. Invalid Credentials *42*
Ilustración 12: Log del middleware RabbitMQ y servidor LDAP. Invalid Credentials *42*
Ilustración 13: Throughput de salida en función del control de acceso implementado *43*
Ilustración 14: Latencia en función del control de acceso implementado *43*
Ilustración 15: Log del middleware EMQ. Certificate Unknown *44*
Ilustración 16: Log del middleware RabbitMQ. Unknown CA *44*
Ilustración 17: Intercambio de mensajes a través de una conexión SSL *45*
Ilustración 18 Mensaje 'Publish Message' cifrado *45*
Ilustración 19: Throughput de salida en función del protocolo de transporte utilizado *46*
Ilustración 20: Latencia en función del protocolo de transporte utilizado *46*
Ilustración 21: Throughput EMQ *48*
Ilustración 22: Latencia EMQ *49*
Ilustración 23: Latencia EMQ (Zoom) *49*
Ilustración 24: Diagrama de Gantt *54*
Ilustración 25: Diagrama de flujo. Qos 0 ..... *60*
Ilustración 26: Diagrama de flujo. QoS 1 ..... *61*
Ilustración 27:Diagrama de flujo. QoS 2 ..... *62*
Ilustración 28: Diagrama de flujo. Mensajes de control LDAP *63*
Ilustración 29: SSL Handshake *64*

Índice de tablas

Tabla 1: Ponderación de las alternativas para la adquisición de datos 26
Tabla 2: Ponderación de las alternativas para el análisis de rendimiento 28
Tabla 3: Estado del arte. Middlewares MQTT 32
Tabla 4: Middlewares MQTT OpenSource 32
Tabla 5: Middlewares MQTT que garantizan gestión de pérdidas 33
Tabla 6: Middlewares MQTT que soportan Clústers 34
Tabla 7: Middlewares MQTT que soportan la implementación de mecanismos de control de acceso mediante servidores LDAP 35
Tabla 8: Middlewares MQTT que soportan la implementación de conexiones SSL seguras 35
Tabla 9: Detalles del Set Up 38
Tabla 10: Topología del escenario real 50
Tabla 11: Tasas horarias de los RRHH 55
Tabla 12: Horas internas del Ingeniero Junior 55
Tabla 13: Horas internas del Ingeniero Senior 55
Tabla 14: Subtotal de las horas internas 55
Tabla 15: Subtotal de amortizaciones 56
Tabla 16: Subtotal de gastos 56
Tabla 17: Coste total del proyecto 56
Tabla 18: Parámetros de configuración del fichero docker-compose.yml 65
Tabla 19: Parámetros de configuración del fichero emq.conf 68
Tabla 20: Parámetros de configuración del fichero rabbitmq.conf 76

Acrónimos

Advanced Research Projects Agency (ARPA)

Advanced Research Projects Agency NETwork (ARPANET)

Internet of Things (IoT)

Messsage Queuing Telemtry Transport (MQTT)

International Organization for Standarization (ISO)

Tecnologías de la Información y Comunicación (TIC)

PriceWaterhouseCoopers (PWC)

Quality of Service (QoS)

Central Processing Unit (CPU)

eXtensible Messaging and Presence Protocol (XMPP)

Advanced Message Queuing Protocol (AMQP)

eXtensible Markup Laguage (XML)

Transmission Control Protocol (TCP)

Transport Layer Secutiry (TLS)

Lightweight Directory Access Protocol (LDAP)

Autoridad Certificadora (AC)

Structured Query Language (SQL)

Secure Sockets Layer (SSL)

Sistema Operativo (SO)

Random Access Memory (RAM)

1. Introducción

El origen de Internet se remonta a 1969 cuando la Agencia de Investigación de Proyectos Avanzados de Estados Unidos o ARPA (Advanced Research Projects Agency) desplegó el primer enlace entre tres universidades de California. Esta red se denominó ARPANET (Advanced Research Projects Agency NETwork) y permitía el envío de mensajes entre los diferentes equipos que lo formaban.

A partir de entonces, el número de nodos que han pasado a formar parte de Internet ha crecido exponencialmente, alcanzando en la actualidad, según el portal de estadísticas Statista simplemente en escenarios IoT (Internet of Things) los 23 billones de unidades. Además, se espera un incremento exponencial llegando a poder alcanzar los 75 billones para 2025. Por otro lado, la localización de estos con el paso del tiempo ha conseguido dar alcance a la totalidad del mundo.

En 2009 surgió el concepto de IoT el cual se refiere a la comunicación de objetos cotidianos (dispositivos) a Internet con el objetivo principal de monitorizar el estado de diferentes componentes que los conforman. Esto, por tanto, supone un gran desafío a la hora de dar soporte, por un lado, al gran número de dispositivos que bajo el concepto de IoT forman o van a formar parte en un futuro de Internet y, por otro lado, al gran número de nuevos mensajes que estos nuevos dispositivos intercambian a través de Internet.

Bajo el nombre de Industria 4.0 el concepto de IoT ha emergido en el sector de la industria. La industria 4.0 se basa en la automatización de los procesos de producción y la mejora de la efectividad de los modelos de administración y control de las maquinarias mediante el procesamiento en tiempo real de gran cantidad de información, utilizando sistemas analíticos que trabajan bajo algoritmos de inteligencia artificial pudiendo generar ahorros millonarios y permitiendo mejorar la efectividad de los negocios.

Uno de los principales desafíos en este contexto es la definición de un protocolo de intercambio de mensajes óptimo que se adecue a las características de los dispositivos, de los mensajes enviados por estos y de las comunicaciones. Por un lado, los dispositivos con los que se trabaja en este contexto disponen de recursos limitados. Por otro lado, los mensajes enviados al contener únicamente información del estado de equipamiento se caracterizan por un tamaño pequeño (p. ej. un único valor asociado a una medida). Finalmente, la monitorización continua de los dispositivos supone una gran cantidad de tráfico a transmitir en escenarios donde el ancho de banda es limitado.

Otro desafío nace ante la necesidad de elegir una lógica de intercambio de mensajes que permita adquirir los datos de los dispositivos para su procesamiento, almacenamiento y futura explotación de forma sencilla, escalable, segura y óptima.

El protocolo más extendido para el intercambio de mensajes en entornos IoT es MQTT (Messsage Queuing Telemtry Transport). Esto se debe en gran medida a su posibilidad de implementación en pequeños dispositivos de bajo coste con poca potencia y memoria. Además, es ideal para redes con pequeño ancho de banda debido al pequeño overhead y tráfico de control que introduce.

MQTT es un protocolo de mensajería estándar de la ISO (International Organization for Standarization) basado en el paradigma publicación suscripción. Este paradigma es muy

utilizado en sistemas distribuidos ya que ofrece facilidad de escalabilidad, sincronismo y abstracción.

En consecuencia, los middlewares más utilizados actualmente en estos escenarios son aquellos que soportan el protocolo MQTT.

1.1. Escenario de desarrollo y operación

El proyecto que se describe en este documento se ha realizado en el centro tecnológico IK4-Tekniker en un escenario de Industria 4.0 basado en el protocolo de mensajería MQTT.

Dicho escenario está formado por los siguientes bloques:

- **Monitorización**: este bloque realiza el proceso de monitorización de diferentes máquinas herramienta. Los resultados son enviados al sistema de almacenamiento a través del sistema de control de intercambio de información para posteriormente ser explotados.
- **Sistema de control de intercambio de información:** red intermediaria que permite él envió de la información recogida durante la monitorización al sistema de almacenamiento.
- **Sistema de almacenamiento:** recibe, procesa y almacena la información enviada a través del sistema de control de intercambio de información por el bloque de monitorización.
- **Sistema de explotación de datos:** a partir de la información almacenada se encarga de dar valor a la misma. En concreto, se representa en serie temporales, se predicen valores futuros y se realiza una gestión de alarmas ante valores fuera de lo normal. La representación de los valores y la gestión de alarmas son funciones que se realizan en tiempo real.

Ilustración 1: Escenario de desarrollo y operación

Partiendo de este escenario, se analizan las diferentes alternativas existentes a la hora desplegar un sistema de control de intercambio de información con el objetivo de conocer y desplegar aquel que se adecúe mejor a un escenario con las siguientes condiciones y exigencias:

- **Despliegue personalizado y económico:** Se requiere de un despliegue flexible, sencillo y económico del sistema de control de intercambio de información.
- **Fiabilidad:** Los mensajes intercambiados no pueden perderse.
- **Disponibilidad:** El sistema de control de intercambio de información debe estar el mayor tiempo posible disponible.
- **Seguridad:**
  - **Control de acceso:** Los recursos del sistema de control de intercambio de información no pueden ser accesibles por usuarios no autorizados.
  - **Confidencialidad:** Los mensajes intercambiados con el sistema de control de intercambio de información no pueden ser legibles por terceros.

- **Rendimiento:** Debido a la necesidad de implementar funciones en tiempo real en el sistema de explotación de datos, el rendimiento del sistema de control de información deberá ofrecer el mejor rendimiento posible.

2. Contexto

En este apartado, se busca contextualizar el proyecto. Primero se analizará el nivel de penetración de los escenarios IoT en la sociedad. Posteriormente, se describirán tecnologías relacionadas con aplicaciones IoT.

2.1. Nivel de penetración de escenarios IoT

Según un estudio de Gartner, el IoT (sin considerar PCs, tablets y teléfonos móviles) crecerá a 26 mil millones de unidades instaladas en 2020, es decir, se producirá un incremento del 3000% con respecto a 2009 donde el número de unidades instaladas alcanzaba los 0,9 mil millones. Este crecimiento, supera al de otros dispositivos como PCs o tablets que llegarán a alcanzar los 7,3 mil millones de unidades para 2020. A su vez, tanto el IoT como producto como los proveedores de servicios llegarán a generar unos ingresos que alcanzarán los 300 mil millones de dólares en 2020 (sobre todo por servicios).

Debido al bajo coste asociado a añadir capacidades IoT a diferentes dispositivos, las empresas han hecho un uso extensivo de la tecnología IoT y, por tanto, han aparecido una amplia gama de productos y servicios tales como, ciudades y hogares inteligentes, sensores y aplicaciones de automatización industrial enfocada a la industria 4.0, sistemas de monitorización de la integridad de diferentes equipos o infraestructuras...

Los hogares inteligentes son un gran ejemplo del crecimiento que está experimentando el IoT. Según un estudio de la consultoría Berg Insight el número de hogares inteligentes en Europa y EE. UU. alcanzó los 17,9 millones en el año 2015. Por un lado, a finales de 2015 en EE. UU. había 12,7 millones de hogares inteligentes. Este estudio estima un crecimiento de hasta 46,2 millones de hogares inteligentes en EE. UU. para 2020 lo que equivale a un 35% de los hogares totales. Por otro lado, en Europa a finales de 2015 había 5,3 millones de hogares inteligentes estimándose para 2020 un crecimiento de hasta 44,9 millones, es decir, un 20% de los hogares en Europa. Según este estudio los equipos con mayor penetración en este campo son los termostatos inteligentes, sistemas de seguridad, equipos de iluminación y bombillas inteligentes y eficientes…

Otra de las aplicaciones emergentes asociadas al concepto de IoT son las ciudades inteligentes. Según el experto Boyd Cohen, las ciudades inteligentes buscan mediante el uso de las Tecnologías de la Información y Comunicación (TIC) ser más inteligentes y eficientes en el uso de recursos, reduciendo costes y ahorrando energía, mejorando los servicios proporcionados y la calidad de vida, y reduciendo la huella medioambiental. Se estima que en el futuro aproximadamente el 70% de la población vivirá en ciudades y, por tanto, es necesario adoptar medidas que permitan mejorar la eficiencia de estas.

En el sector de la industria, el informe Industria 4.0 elaborado por PwC (PriceWaterhouseCoopers) a partir de entrevistas a empresas del sector industrial en 26 países desarrollados diferentes calcula que el 8% de las empresas industriales españolas dispone de un nivel de digitalización avanzado frente al 33% de la media mundial. Para 2020 estima que se alcanzará un 19% frente al 72% de la media mundial.

A nivel mundial, la industria 4.0 tiene un mayor nivel de penetración en sectores como la electrónica (45%), automoción (41%) y fabricación industrial (35%).

Además, según este informe se espera que las empresas industriales digitalmente avanzadas experimenten un incremento adicional de su facturación del 2,9% de media anual.

2.2. Tecnologías relacionadas

Hoy en día, Internet está formado en gran medida por sistemas distribuidos entre los cuales destacan aquellos asociados a aplicaciones IoT. Con el objetivo de entender el modo de comunicación entre los diferentes dispositivos que forman parte de una infraestructura IoT se considera de interés describir las siguientes tecnologías relacionadas.

2.2.1. Sistemas distribuidos

Un sistema distribuido se define como un conjunto de dispositivos separados físicamente entre sí, conectados por una red de comunicaciones a través de la cual intercambian mensajes para comunicarse y sincronizarse, que tienen un objetivo final común pero que pueden desempeñar funciones completamente diferentes y variantes en el tiempo. Un sistema distribuido está compuesto por un gran número de dispositivos.

Ilustración 2: Sistema distribuido

2.2.2. Sistema de control de intercambio de información. Middleware

El sistema de control de intercambio de información es un software que permite el intercambio de mensajes entre los diferentes componentes del sistema distribuido. También se denomina middleware. Este, está compuesto por una lógica de procesamiento y un conjunto de colas mediante las cuales es capaz gestionar toda la información intercambiada entre los dispositivos interconectados a través de él.

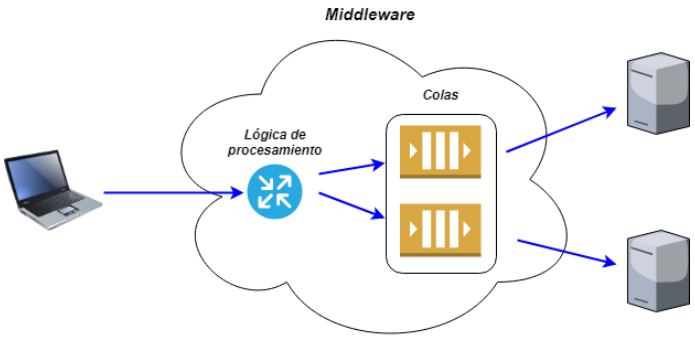

Ilustración 3:Sistema de control de intercambio de información. Middleware

2.2.3. Paradigma publicación/suscripción

Las comunicaciones entre dispositivos a través del middleware pueden realizarse mediante diferentes paradigmas. El paradigma publicación/suscripción es la solución utilizada con más frecuencia para el intercambio de mensajes en sistemas distribuidos donde se requiere:

- **Facilidad de escalabilidad**: El número de equipos que lo componen varía con el tiempo
- **Facilidad de sincronismo:** Los equipos entre los que se intercambia la información no han de estar disponibles y sincronizados en todo momento. Permite un cierto nivel de abstracción.
- **Disponibilidad:** Facilita la continuidad de los servicios de red ante la caída de uno o varios de los equipos.

Bajo este paradigma, los dispositivos cuando quieren enviar un mensaje realizan una publicación con dicho mensaje en el middleware. Este último, en función de una lógica de rutado y de la suscripción realizada por los suscriptores entregará estos mensajes a unos u otros.

Las dos lógicas de rutado más utilizadas en este contexto son las siguientes:

Basada en topic: Se caracteriza por la publicación de mensajes según topics. El middleware en función del topic de publicación y de los topics de suscripción de los suscriptores toma decisiones para encaminar los mensajes a unas colas u otras.

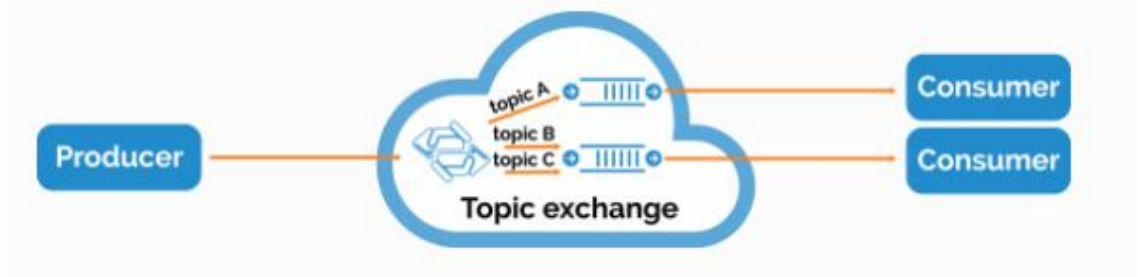

Ilustración 4: Lógica de rutado basada en topic

Basado en contenido: El middleware toma decisiones de encaminamiento en función del contenido de los mensajes ya sean datos o metadatos. En este caso, no se hace uso de una etiqueta u topic para tomar la decisión de encaminamiento.

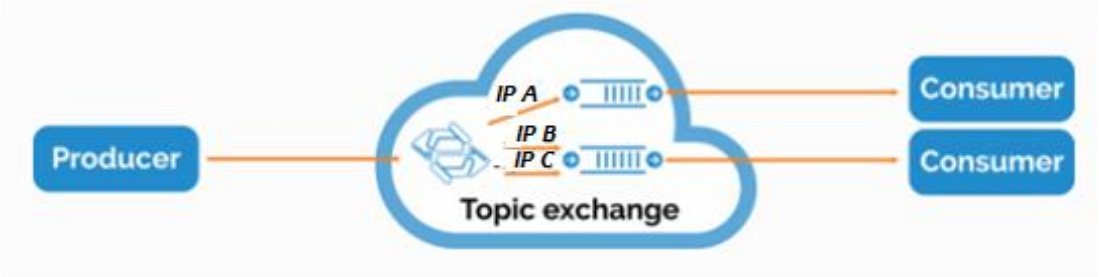

Ilustración 5: Lógica de rutado basada en contenido

3. Objetivos

Los objetivos de este proyecto son, por un lado, la comparación teórica y práctica de un sistema de control de información denominado middleware en cuanto a las siguientes prestaciones:

Despliegue personalizado y económico

Se valorará la capacidad de las soluciones para posibilitar el despliegue de un middleware personalizado, con diferentes funcionalidades de forma sencilla con el menor coste.

Fiabilidad

Los mensajes intercambiados entre el bloque de monitorización, el middleware y el sistema de almacenamiento pueden perderse, llegar duplicados, desordenados o de forma correcta.

Tal y como se ha mencionado en la introducción, los mensajes intercambiados con el middleware no pueden perderse. Por ello, se valorará la capacidad de las diferentes soluciones para garantizar la entrega de los mensajes.

Disponibilidad

El middleware es el nodo central que permite el intercambio de mensajes entre el bloque de monitorización y el sistema de almacenamiento. Una caída del middleware, por tanto, supondrá una desconexión entre la monitorización y el sistema de almacenamiento que provocará pérdidas y la imposibilidad de explotar en tiempo real aquellos datos recogidos durante el tiempo en el que permanezca indisponible con las consecuencias que esto supone (representación irreal de los datos, imposibilidad de detectar errores…).

Por tanto, es necesario que esté disponible el máximo tiempo posible minimizando en todo momento la posibilidad de que permanezca indisponible.

Seguridad

Desde el punto de vista de la seguridad, se valorará la capacidad de los middlewares para ofrecer las siguientes prestaciones:

- **Control de acceso:** El middleware está expuesto a Internet. Por ello, se valorará la capacidad de implementar mecanismos que garanticen el acceso a los recursos a solo aquellos que tengan permiso.
- **Confidencialidad:** El intercambio de mensajes con el middleware se realiza a través de Internet haciendo de este modo, vulnerable la información intercambiada. Esta información puede ser desde datos en crudo hasta credenciales utilizadas para el control de acceso. Para evitar que la información intercambiada pueda ser legible por terceros, se valorará la capacidad de implementar mecanismos que garanticen la confidencialidad de los datos.

Rendimiento

La representación de los datos y la gestión de alarmas del sistema de explotación de datos ha de realizarse en tiempo real. Además, la cantidad de información con la que se trabaja es elevada. Por ello, es necesario que el rendimiento ofrecido por el middleware sea suficiente para poder ofrecer un servicio que ofrezca información actualizada y con unos retardos máximos

inferiores a los perceptibles por el ser humano (se define como admisible retardos inferiores a 1 segundo).

Por otro lado, el segundo objetivo es el despliegue de la solución adoptada en un escenario real basado en el escenario de desarrollo descrito en la introducción que permita validar de forma práctica la forma de comparativa propuesta.

4. Beneficios

4.1. Beneficios económicos

Este proyecto define una metodología que permite conocer las virtudes y deficiencias que presentan los diferentes middlewares en base a los requisitos y características de un escenario dado. De este modo, podrá elegirse la solución más idónea consiguiendo así no tener que centrar esfuerzos en suplir posibles deficiencias que otros middlewares MQTT puedan tener y que desemboquen en un desembolso innecesario de tiempo y dinero.

Por otro lado, va a permitir conocer qué solución o soluciones optimizan el rendimiento total de la plataforma con unos niveles de fiabilidad, disponibilidad y seguridad mínimos definidos. Desde el punto de vista de un proveedor de servicios, va a ser posible ofrecer de este modo un servicio personalizado de intercambio de mensajes de gran calidad y a un mayor número de clientes. Este aumento de la calidad permitirá a su vez, atraer a un mayor número de clientes.

Por último, los resultados obtenidos de los diferentes middlewares MQTT van a poder ser de gran utilidad a futuro para otros posibles situaciones que puedan surgir, y que tengan necesidades o requisitos diferentes. De este modo, no va a ser necesario realizar una nueva inversión para un nuevo estudio o análisis.

4.2. Beneficios técnicos

Este proyecto, va a suponer un punto de inflexión a la hora de escoger un middleware MQTT para una plataforma IoT dada.

Permite de una forma sencilla, siguiendo la metodología dada, a partir del estado del arte en un determinado momento sobre middlewares MQTT y de las necesidades identificadas en un escenario de aplicación dado definir los funciones mínimas requeridas de los middlewares, descartar las soluciones no válidas, identificar las situaciones críticas, realizar una comparativa del rendimiento en dichas situaciones y escoger la solución óptima personalizada.

4.3. Beneficios sociales

El IoT permite mejorar las prestaciones que ofrecen diferentes sectores a la sociedad a un coste menor.

En el caso de sectores como los hogares y ciudades inteligentes, ha permitido mejorar la eficiencia de los recursos disponibles, reduciendo costes, ahorrando energía, mejorando los servicios ofrecidos y la calidad de vida, y reduciendo la huella medioambiental.

Por otro lado, en el caso de la industria, ha permitido la automatización de los procesos de producción y la mejora de la efectividad de los modelos de administración y control de las maquinarias lo que resulta en beneficios millonarios.

Este proyecto permite optimizar las prestaciones ofrecidas por cualquier escenario mediante el despliegue de la infraestructura óptima siguiendo la metodología descrita.

5. Análisis del estado del arte

El concepto de IoT ha empezado a tomar especial relevancia en el mundo de las telecomunicaciones estos últimos años. Además, debido a las características de los escenarios IoT ha sido necesario el desarrollo de nuevos protocolos que se adecuen a estos y permitan el intercambio de mensajes entre los diferentes participantes de forma óptima. En los últimos años, el protocolo MQTT es el que más relevancia ha tomado en el mundo del IoT.

Por un lado, por la poca vida que tiene el concepto de IoT y la tecnología MQTT ha sido realmente difícil encontrar gran cantidad de documentación que permita ayudar en el desarrollo de este proyecto.

Por otro lado, al tratarse de un campo en desarrollo, gran parte de la documentación se centra en casos muy concretos. Además, en muchos casos, por asuntos de confidencialidad no es de interés para los investigadores mostrar el desarrollo completo que se ha realizado y, por tanto, la documentación se encuentra incompleta.

A continuación, se describe la información más relevante que se ha obtenido a cerca de dos metodologías de comparación de middlewares. Estos documentos, han sido seleccionados por la similitud que existe de los escenarios que se plantean con el del proyecto, por el uso de middlewares MQTT y por las comparaciones teóricas y prácticas realizadas.

5.1. Benchmark of MQTT Servers

Objetivo

El objetivo de este documento es conocer la capacidad de escalabilidad que tienen diferentes servidores MQTT cuando el número de publicadores aumenta.

Middlewares

Los servidores MQTT que han sido evaluados en este documento son: ActiveMQ 5.10.0, Apollo 1.7, JoramMQ 1.1.3, Mosquitto 1.3.5 y RabbitMQ 3.4.2.

Test escenario

El escenario de prueba simula un conjunto de dispositivos que publican datos de telemetría a un centro de comando cada segundo con un payload de 64 bytes. Los dispositivos son los publicadores y el centro de comando el suscriptor.

El servidor MQTT esta desplegado en una única instancia en único servidor independiente.

Se define el concepto de instancia de test. Esta instancia de test está constituida por 1.000 publicadores y un suscriptor. Las instancias de test están desplegadas en dos servidores y son independientes entre sí mediante el uso de un conjunto de topics distintos (10.000 topics por instancia de test). En las pruebas realizadas, las instancias de test son incrementadas desde 2 hasta 100 con el objetivo de aumentar la carga.

Los clientes que conforman cada una de las instancias están desplegados en Java.

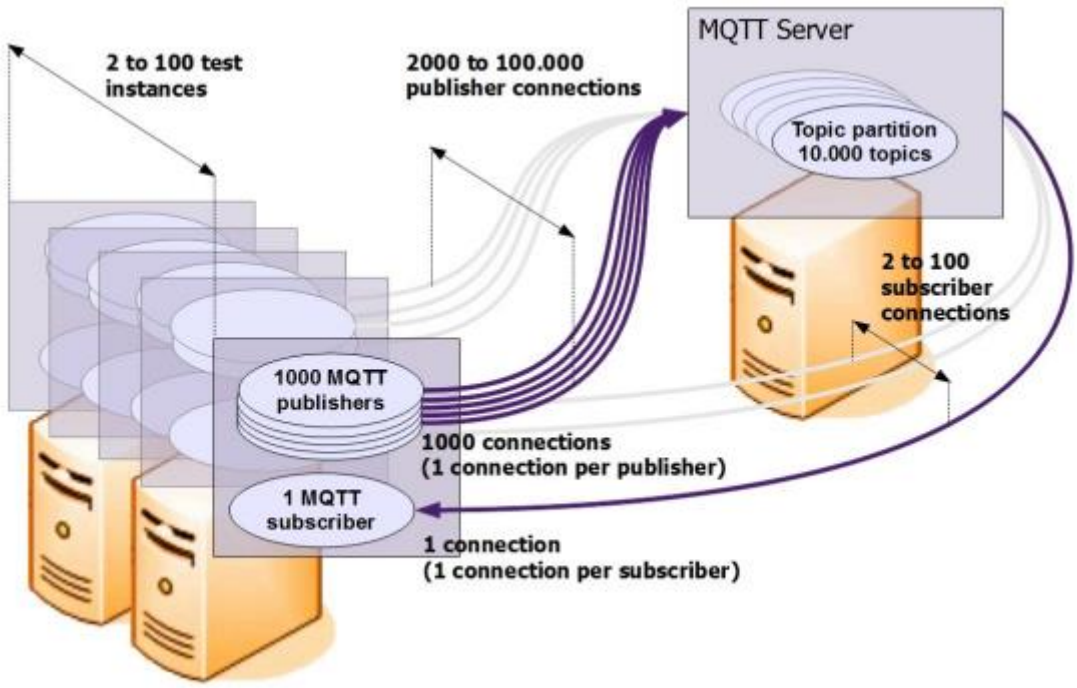

Ilustración 6: Test escenario

Pruebas realizadas

Las pruebas han sido realizadas para un nivel de QoS (Quality of Service) 0 y 1. Siempre manteniendo el mismo nivel de QoS en publicación y suscripción.

Las pruebas están divididas en 3 partes:

1. Establecimiento de la conexión entre los clientes y el servidor MQTT
2. Publicación de los mensajes
3. Desuscripción de los suscriptores y cierre de las conexiones.

Medidas realizadas

Para medir la escalabilidad de cada uno de los servidores MQTT, se ha medido el throughput (tasa de mensajes recibidos en el suscriptor), el uso de la CPU (Central Processing Unit) en el servidor MQTT y el tiempo requerido para transmitir un mensaje del publicador al suscriptor, es decir, la latencia.

El uso de la CPU es medida mediante el comando 'top' de Linux.

Como tanto publicadores como suscriptores están ejecutándose en la misma máquina, el reloj en ambos casos es el mismo y la latencia se calcula mediante la diferencia entre el momento en el que se recibió un mensaje y el momento en el que se publicó. Para ello, en el payload del mensaje se añade un timestamp con el tiempo de publicación.

5.2. Kafka vs rabbitMQ

Objetivo

El objetivo de este documento es realizar una comparativa cualitativa y cuantitativa de frameworks basados en sistemas de publicación/suscripción que permita definir los casos de uso en los que el despliegue de cada uno de ellos es más aconsejable.

Middlewares

Los frameworks que se analizan en este documento son Kafka 0.10.0.1 y RabbitMQ 3.5.3.

Comparación cualitativa

Las características analizadas en la comparación cualitativa son:

- **Tiempo de desacoplo:** Se describe el lugar de almacenamiento de los mensajes y su posible impacto en el rendimiento.
- **Lógica de rutado:** Se describen las lógicas de enrutado que son soportadas.
- **Garantía de entrega:** Se analiza si se permite detectar pérdidas y garantizar la entrega de todos los mensajes.
- **Garantía de orden:** Se analiza si permiten garantizar un orden en los mensajes
- **Disponibilidad:** Se describen los métodos que implementan para aumentar la disponibilidad.
- **Transacciones:** Se analiza si soportan la posibilidad de publicar mensajes en batch.
- **Multicast:** Se analiza si soportan el envió de un mensaje a múltiples destinatarios y en tal caso como.
- **Escalado dinámico:** Se describe el modo de realizar el escalado, así como su facilidad.

Comparación cuantitativa

Test escenario

El escenario utilizado para realizar las pruebas ha sido desplegado en un único servidor y consta de los siguientes participantes:

- **Framework:** Inicialmente, se despliega una única instancia.
- **Publicadores y suscriptores:** están desplegados mediante herramientas de testeo proporcionadas por cada una de las distribuciones. En el caso de RabbitMQ mediante un cliente Java y en el caso de Kafka mediante herramientas propias.

Pruebas realizadas

En las pruebas realizadas, todos los clientes han sido configurados para publicar el máximo de carga posible. En caso de requerir más de una instancia del framework, se despliegan replicas automáticamente en la misma máquina.

Las pruebas se han realizado para diferentes niveles de QoS con el objetivo de conocer que coste tiene el ofrecer ciertas garantías en la entrega de mensajes.

Medidas realizadas

Los parámetros medidos en las diferentes pruebas han sido el throughput (tasa de mensajes recibidos en el suscriptor), la latencia, el consumo de CPU y de memoria.

6. Análisis de alternativas

En este apartado se analizarán diferentes alternativas que han surgido para la realización de diferentes partes del proyecto. En concreto se analizarán dos aspectos: Tecnologías para la adquisición de datos y métodos para el análisis de rendimiento.

6.1. Alternativas para la adquisición de datos

Al inicio del proyecto, MQTT fue definido como el protocolo de mensajería a utilizar para el intercambio de mensajes entre el bloque de monitorización y el sistema de almacenamiento.

Con el objetivo de verificar que MQTT es la alternativa más idónea para el intercambio de mensajes en el contexto de este proyecto, se han estudiado diferentes tecnologías que permiten enviar la información que se genera en el bloque de monitorización al sistema de almacenamiento. En concreto, se han analizado las características de los protocolos de mensajería más extendidos en la actualidad en escenarios con un contexto similar: XMPP (eXtensible Messaging and Presence Protocol), AMQP (Advanced Message Queuing Protocol) y MQTT.

6.1.1. XMPP

XMPP es un protocolo de mensajería instantánea basado en XML (eXtensible Markup Laguage). Los clientes XMPP están organizados en dominios e intercambian mensajes a través de servidores intermedios basándose en un paradigma push/pull. Para su implementación, es necesario definir un conjunto de dominios, desplegar los servidores intermedios asociados a estos y desplegar los clientes XMPP.

Ventajas

Confiable

El protocolo XMPP trabaja sobre TCP (Transmission Control Protocol), de esta manera se garantiza que los mensajes son entregados al destino sin errores y en el mismo orden que se transmitieron.

Fácil de desplegar

Es un protocolo abierto, público, estándar y fácil de entender. El despliegue y configuración tanto de clientes como servidores XMPP es sencillo y se dispone de una gran cantidad de documentación al respecto.

Descentralizado

Existe más de un servidor intermedio a través de los cuales los clientes intercambian los mensajes. Los servidores intermedios en el intercambio de información pertenecen a los dominios del remitente y el destinatario del mensaje.

Asíncrono

Gracias al paradigma push/pull la información puede ser tratada por el destinatario de forma asíncrona. Los clientes XMPP pueden abstraerse del estado del resto del clientes.

Los mensajes son enviados por el remitente con un push sin tener en cuenta el estado de los destinatarios. Serán estos últimos, los que solicitarán los mensajes a los servidores de su dominio cuando sea posible mediante un pull.

Soporta control de acceso

Los clientes XMPP deben presentar unas credenciales para poder intercambiar mensajes. Estas credenciales son configuradas en el servidor del dominio al que pertenezca el cliente en cuestión.

Soporta encriptación TLS (Transport Layer Security)

Las especificaciones XMPP contemplan la implementación del protocolo de encriptación TLS.

Desventajas

Pesado

Por un lado, al tratarse de un protocolo descentralizado, el número de servidores intermedios puede ser elevado.

Por otro lado, los clientes XMPP requieren de cierta capacidad computacional (p. ej. para generar y analizar los XML). No está pensado para su implementación en pequeños dispositivos con pocos recursos.

No fiable

XMPP no asegura la entrega de mensajes. En caso de que fuera necesario, se ha de implementar en un protocolo por encima de la capa XMPP.

6.1.2. AMQP

AMQP es un protocolo de mensajería orientado a mensajes basado en el paradigma publicación/suscripción a través de un equipo intermedio denominado bróker. Para su implementación, es necesario el despliegue de clientes AMQP que obtengan y publiquen la información monitorizada, un bróker y un cliente AMQP que realice la función de suscriptor, recoja, procese y almacene la información en el sistema de almacenamiento.

Ventajas

Confiable

El protocolo AMQP trabaja sobre TCP, de esta manera se garantiza que los mensajes son entregados al destino sin errores y en el mismo orden que se transmitieron.

Fácil de desplegar

Por un lado, en múltiples lenguajes de programación (Java, C, Python…) se permite de forma sencilla la implementación de clientes AMQP a través de librerías.

Por otro lado, existe también una gran variedad de brókers AMQP. Sin embargo, no es tan extensa como en el caso del protocolo MQTT.

Asíncrono

Los mensajes publicados en el bróker son enviados a los suscriptores bajo una previa solicitud de estos. De este modo, la información puede ser tratada de forma asíncrona y los publicadores se pueden abstraer del estado de los suscriptores.

Fiable

AMQP garantiza la entrega de los mensajes mediante el intercambio de mensajes de control.

Contempla el uso de transacciones

AMQP permite el intercambio de mensajes a través de transacciones.

Soporta control de acceso

AMQP soporta el uso de credenciales para restringir el acceso a los recursos del bróker a aquellos clientes que estén autorizados.

Soporta encriptación TLS

Permite la implementación del protocolo TLS para el cifrado de las comunicaciones.

Desventajas

Pesado

Los clientes AMQP requieren de una cierta capacidad computacional. No está pensado para su implementación en pequeños dispositivos con pocos recursos.

Además, debido a la gran cantidad de tráfico de control que se introduce para ofrecer fiabilidad no está pensado para su implementación en redes con bajo ancho de banda.

Centralizado

El paradigma publicación/suscripción se basa en un equipo central (bróker) a través del cual los clientes intercambian toda la información.

Es posible la implementación de clústers. Sin embargo, esto supone un aumento de la complejidad.

Esta característica supone una limitación a la hora de ofrecer disponibilidad.

6.1.3. MQTT

MQTT es un protocolo de mensajería basado en el paradigma publicación/suscripción a través de un equipo intermedio denominado bróker. Para su implementación, es necesario el despliegue de clientes MQTT en el bloque de monitorización que recojan y publiquen la información monitorizada, un bróker y un cliente MQTT que realice la función de suscriptor, procese y almacene la información en el sistema de almacenamiento.

Ventajas

Confiable

El protocolo MQTT trabaja sobre TCP, de esta manera se garantiza que los mensajes son entregados al destino sin errores y en el mismo orden que se transmitieron.

Ligero

MQTT se define como un protocolo de mensajería ligero idóneo para escenarios M2M (machine to machine) donde los recursos disponibles son limitados (ancho de banda bajo, dispositivos con recursos limitados…).

Facilidad de despliegue

Por un lado, múltiples lenguajes de programación (Java, Python …) disponen de una gran variedad de librerías que permiten la implementación de clientes MQTT que realizan funciones de publicador o suscriptor de una forma muy sencilla.

Por otro lado, se dispone de una gran variedad de middlewares que pueden ser desplegados y configurados de forma sencilla.

Además, en ambos casos existe una gran cantidad de documentación que permite facilitar aún más su despliegue y configuración de forma personalizada.

Asíncrono

Los mensajes publicados en el bróker son enviados a los suscriptores bajo una previa solicitud de estos. De este modo, la información puede ser tratada de forma asíncrona y los publicadores se pueden abstraer del estado de los suscriptores.

Fiable

MQTT permite la implementación de diferentes calidades de servicio que permiten identificar pérdidas de mensajes o duplicidad en los mismos.

Soporta control de acceso

MQTT soporta el uso de credenciales para restringir el acceso a los recursos del bróker intermedio a aquellos clientes que estén autorizados.

Encriptación TLS

Permite la implementación del protocolo TLS para el cifrado de las comunicaciones.

Desventajas

Centralizado

MQTT es un protocolo centralizado donde un único equipo realiza el procesamiento de toda la información que se intercambia entre los diferentes participantes.

Es posible el despliegue de réplicas. Sin embargo, esto introduce una complejidad considerable.

Esta característica supone una limitación a la hora de ofrecer disponibilidad.

6.1.4. Ponderación de las alternativas

A la hora de elegir una de las alternativas, se han tenido en cuenta una serie de aspectos. Estos son:

- **Implementación de la solución en dispositivos con recursos limitados:** Los dispositivos donde se quiere desplegar los clientes disponen de recursos limitados. Cuanto menor sean los recursos necesarios mayor será la puntuación.
- **Facilidad de despliegue:** Es importante que la implementación de las alternativas no suponga grandes problemas que puedan repercutir en costes. Una mayor facilidad de implementación tendrá asociado una mayor puntuación en este aspecto.
- **Fiabilidad:** La información monitorizada a de llegar al completo al sistema de explotación. De lo contrario, las funciones realizadas por este no serán eficaces. El uso de mecanismos que garanticen la entrega de los mensajes estará asociado a una mayor puntuación en este aspecto.
- **Disponibilidad:** Algunas de las funciones del sistema de explotación de datos se realizan en tiempo real. Por ello, es necesario minimizar la probabilidad de indisponibilidad del sistema. Cuanto mayor sea la puntuación, mayor será la disponibilidad del sistema.
- **Control de acceso:** Es importante que solo clientes autenticados puedan enviar información o recibirla. De este modo, podrán evitarse ataques de inyección de tráfico, robos de información… Cuanto más robusto sea el mecanismo de control de acceso mayor será la puntuación en este aspecto.
- **Confidencialidad:** Los datos intercambiados entre los clientes contienen información confidencial que no puede ser obtenida por terceros. Cuanto mayor sea el nivel de encriptación de los datos, mayor será la puntuación que se otorgue en este aspecto.

Aspectos por analizar	XMPP	AMQP	MQTT
Implementación en dispositivos con recursos limitados (25%)	3	3	10
Facilidad de despliegue (10%)	8	10	10
Fiabilidad (15%)	3	10	8
Disponibilidad (10%)	9	6	6
Control de acceso (20%)	8	8	8
Confidencialidad (20%)	7	7	7
Resultado final (100%)	**5,9**	**6,85**	**8,3**

Tabla 1: Ponderación de las alternativas para la adquisición de datos

6.1.5. Solución adoptada

En vista de los resultado que se muestran en la tabla 1, se ha verificado que la tecnología más apropiada para la adquisición de datos es MQTT.

6.2. Alternativas para el análisis de rendimiento

Se han estudiado las diferentes alternativas existentes para el despliegue de la infraestructura donde realizar el análisis del rendimiento de los diferentes middlewares. En concreto, se han analizado las ventajas y desventajas que presenta el despliegue de la infraestructura con equipamiento real o mediante simulación.

6.2.1. Equipamiento real

En el caso de realizar un despliegue con equipamiento real, es necesario, por un lado, el despliegue de una red conformada por un conjunto de máquinas herramienta con el software correspondiente que permita la monitorización de estas, el procesamiento de los resultados obtenidos y la publicación de estos en un middleware a través de Internet. Esta red correspondería con el bloque de monitorización.

Por otro lado, es necesario desplegar un bróker MQTT que realice las funciones de sistema de control de intercambio de información y un suscriptor que forme parte del bloque se sistema de almacenamiento. Ambos, deberán desplegarse en un servidor.

Ventajas

Resultados más reales

El uso de equipamiento real va a permitir obtener unos resultados más cercanos a la realidad debido a:

- Similitud en la cadena de procesos que se realizan desde que se monitoriza una variable hasta que esta es recibida por el suscriptor. En definitiva, se tienen en cuenta un mayor número de factores que afectan al rendimiento.
- Capacidades computacionales de los diferentes equipos similares a las de un escenario real.
- Menor número de factores externos asociados a procesos de simulación tenidos en cuenta que realmente en un escenario real no afectan a los resultados

Desventajas

Coste

La compra de máquinas herramienta y el servidor, el mantenimiento de estos… supone un incremento en el presupuesto del proyecto en más de un 100%.

Poca escalabilidad

En relación con la desventaja anterior, el alto coste de los equipos limita las dimensiones que puede alcanzar la infraestructura. Es decir, limita el número de máquinas herramienta que pueden tenerse en cuenta a la hora de realizar el análisis de rendimiento.

Dificultad de escalabilidad

La configuración individual de todos los equipos cuando la infraestructura tiene un gran tamaño aumenta en gran medida la complejidad en el despliegue. Además, supone un coste en tiempo considerable.

6.2.2. Simulación

En caso de realizar un despliegue de la infraestructura mediante simulación, la solución es el despliegue en una misma máquina de:

- Un proceso en Java que simula un conjunto de máquinas herramienta que forman parte del bloque de monitorización. Este proceso genera valores, los procesa y envía al middleware
- El middleware correspondiente al sistema de control de intercambio de información.
- Un proceso en Java encargado de recibir los mensajes. Este proceso formará parte del sistema de almacenamiento.

El uso de Java como lenguaje de programación de los procesos se basa en la facilidad que supone la programación de publicadores y suscriptores mediante librerías externas. Además, existe gran cantidad de documentación que hacen sencilla la implementación de dichas librerías.

Ventajas

Facilidad de despliegue

Tanto la programación en Java como las librerías existentes permiten de una forma muy sencilla la simulación de las máquinas herramienta.

Menor coste

El único coste asociado al despliegue de la infraestructura es el uso de la máquina donde se desplegarán los diferentes procesos y el middleware más el coste de implementación de los procesos y servicios.

El middleware desplegado, el software Java y las librerías utilizadas son Open Source.

Facilidad de escalabilidad

Las librerías utilizadas para el despliegue de procesos que simulan máquinas herramienta permiten de una forma muy sencilla el incremento del número de estas (p. ej. a través de ficheros de configuración).

Facilidad de configuración

Las librerías utilizadas permiten de una forma sencilla configurar diferentes comportamientos de las máquinas herramientas permitiendo de esta manera la prueba de distintos escenarios.

Desventajas

Resultados menos reales

Los resultados obtenidos tendrán una desviación con respecto a los reales debido principalmente a los siguientes factores:

- No se tienen en cuenta los factores propios del equipamiento que afectan al rendimiento de la infraestructura (p. ej. retardos adicionales, errores…)
- Factores externos a la infraestructura relacionados con la máquina pueden afectar al análisis de rendimiento (p. ej. capacidad computacional baja de la máquina, procesos ajenos ejecutándose en segundo plano…)

6.2.3. Ponderación de las alternativas

Los aspectos que se han tenido en cuenta para escoger una de las alternativas son los siguientes:

- **Coste económico:** El coste del proyecto es uno de los factores más importantes a analizar. Cuanto menor sea este, mayor será la puntuación que se otorgue en este aspecto.
- **Facilidad de despliegue:** Un despliegue sencillo evita la aparición de problemas adicionales que pueden tener consecuencias en tiempo y coste. Cuanto mayor sea la sencillez, mayor será la puntuación que obtendrá la alternativa en este punto.
- **Resultados reales:** El proyecto tiene que ser aplicable en un escenario real. Los resultados deben acercarse en la mayor medida posible a los reales. Cuanto más realistas sean los resultados, mayor será la puntuación.
- **Facilidad de escalabilidad:** El proyecto pretende realizar un análisis de rendimiento para un escenario real donde el tamaño es conocido, pero puede variar. Por ello, es necesario que la variación del tamaño de dicho escenario pueda ser modificado de forma sencilla. Cuanto mayor sea la facilidad con la que se puede escalar el escenario, mayor será la puntuación que se dé a la alternativa.

Aspectos por analizar	Equipamiento real	Simulación
Coste económico (40%)	2	10
Facilidad de realización (15%)	4	10
Resultados reales (25%)	9	7
Facilidad de escalabilidad (20%)	3	8
Resultado final (100%)	**4,25**	**8,85**

Tabla 2: Ponderación de las alternativas para el análisis de rendimiento

6.2.4. Solución adoptada

En vista de los resultados obtenidos en la tabla 2 se ha optado por realizar el análisis de rendimiento mediante simulación.

7. Descripción de la solución

7.1. Escenario de aplicación

El escenario de aplicación tiene como funciones finales la representación en tiempo real de los movimientos de un conjunto de máquinas herramienta en series temporales, la detección de errores a partir de esos movimientos y la predicción del estado de las máquinas en un futuro con el objetivo de mejorar la efectividad en el control y la productividad de las máquinas produciendo beneficios para la empresa.

Este proyecto, no entrará en detalle de cómo se realizan las funciones de representación, gestión y predicción a partir de la información almacenada en el sistema de almacenamiento, puesto que queda fuera del ámbito establecido.

7.1.1. Descripción del sistema

Partiendo del escenario descrito en la introducción, los equipos que forman parte del escenario pueden dividirse en función del bloque al que pertenecen:

7.1.1.1. Monitorización

El bloque de monitorización se encuentra en la empresa que contrata el servicio y en él se pueden distinguir los siguientes elementos:

Máquinas herramienta: Conjunto de máquinas que se utilizan para dar forma a piezas sólidas principalmente metales. Para ello, realizan una serie de movimientos sobre un eje o spindle. Estas suelen ser tornos, taladros, fresadoras y pulidoras. Estos movimientos son monitorizados continuamente. Esta monitorización permite conocer el funcionamiento de cada máquina a lo largo del tiempo, detectar y predecir posibles errores.

El proyecto va enfocado a una pequeña o mediana empresa donde el conjunto de máquinas herramienta puede varias entre 1 y 100 unidades localizándose todas en un mismo recinto.

Dispositivo integrado: dispositivo integrado en cada máquina herramienta que realiza las medidas de los diferentes movimientos a lo largo de los ejes o spindles.

Publicador MQTT: Pequeño proceso en Java ejecutado en cada dispositivo integrado que realiza publicaciones en un determinado topic del middleware con los resultados de la monitorización.

En el escenario solo se hace uso de un único topic donde se realizan todas las publicaciones.

7.1.1.2. Sistema de control de intercambio de información

El bloque del sistema de control de intercambio de información está formado únicamente por el middleware.

Middleware: Equipo único que gestiona y envía al sistema de almacenamiento la información publicada por los publicadores MQTT con las medidas realizadas en las máquinas-herramienta por los dispositivos integrados.

El middleware puede encontrarse en cualquier punto de Internet. Es decir, los mensajes intercambiados entre este y los publicadores y suscriptores viajan a través de Internet.

7.1.1.3. Sistema de almacenamiento

El bloque de sistema de almacenamiento se encuentra en el proveedor de servicios y está formado por los siguientes elementos:

Suscriptor MQTT: Pequeño proceso en Java desplegado en un servidor cuya función es la recepción de la información publicada por los diferentes publicadores MQTT en el middleware, el procesamiento y el almacenamiento de esta en una base de datos PostgreSQL. Inicialmente, este proceso realiza una suscripción al topic donde los publicadores MQTT realizan las publicaciones.

En el escenario, un único suscriptor recibe, procesa y almacena toda la información publicada en el middleware.

Base de datos PostgreSQL: Base de datos donde se almacena la información de la monitorización. A partir de esta información, el sistema de explotación de datos es capaz de realizar representaciones en series temporales, predecir el estado de las máquinas herramienta a futuro y detectar posibles anomalías.

En el escenario se va a desplegar una única base de datos.

Tanto el servidor donde esta desplegado el suscriptor MQTT como la base de datos PostgreSQL se encuentran conectadas directamente en una misma subred.

7.1.2. Formato de los mensajes

Los mensajes enviados desde los publicadores a los suscriptores a través del middleware con las medidas realizadas por los dispositivos integrados en las respectivas máquinas herramienta contienen la siguiente información:

- Nombre de la variable medida (p.ej. velocidad)
- Tiempo en el que se realizó dicha medida
- Valor de la medida

Y son enviados con el siguiente formato:

{"varName": "nombre_vaiable", "measures": [{"timestamp": tiempo_milisegundos_since_epoch, "value": valor}]}

Donde “varName” corresponde con el nombre de la variable medida, “timestamp” con un sello de tiempo que identifica el momento en el que se realizó dicha medida en milisegundos en epoch time y “value” con el valor.

Los nombres de las variables y las unidades de medida de cada una de ellas son inicialmente definidas y configuradas en la base de datos. Los nombres de las variables se definen con el objetivo de que sean representativas. Las unidades de medida se definen de tal modo que correspondan con la medida realizada y el rango de valores esperados.

7.1.3. Protocolo de intercambio de mensajes

El protocolo de mensajería utilizado para el intercambio de mensajes entre los publicadores, el middleware y el suscriptor es MQTT.

Este protocolo es ideal para su implementación en redes con pequeño ancho de banda no solo por el pequeño tamaño de sus mensajes debido al poco overhead que se añade en estos sino también por el poco tráfico adicional de control que introduce.

Además, esta ideado para la implementación de clientes en dispositivos pequeños, baratos, con poca memoria y poca potencia como son los dispositivos integrados en las máquinas herramienta.

Finalmente, este protocolo al estar basado en un paradigma publicación/suscripción ofrece fiabilidad, facilidad de escalabilidad y sincronismo.

7.1.4. Topología

La topología del escenario de aplicación es la siguiente:

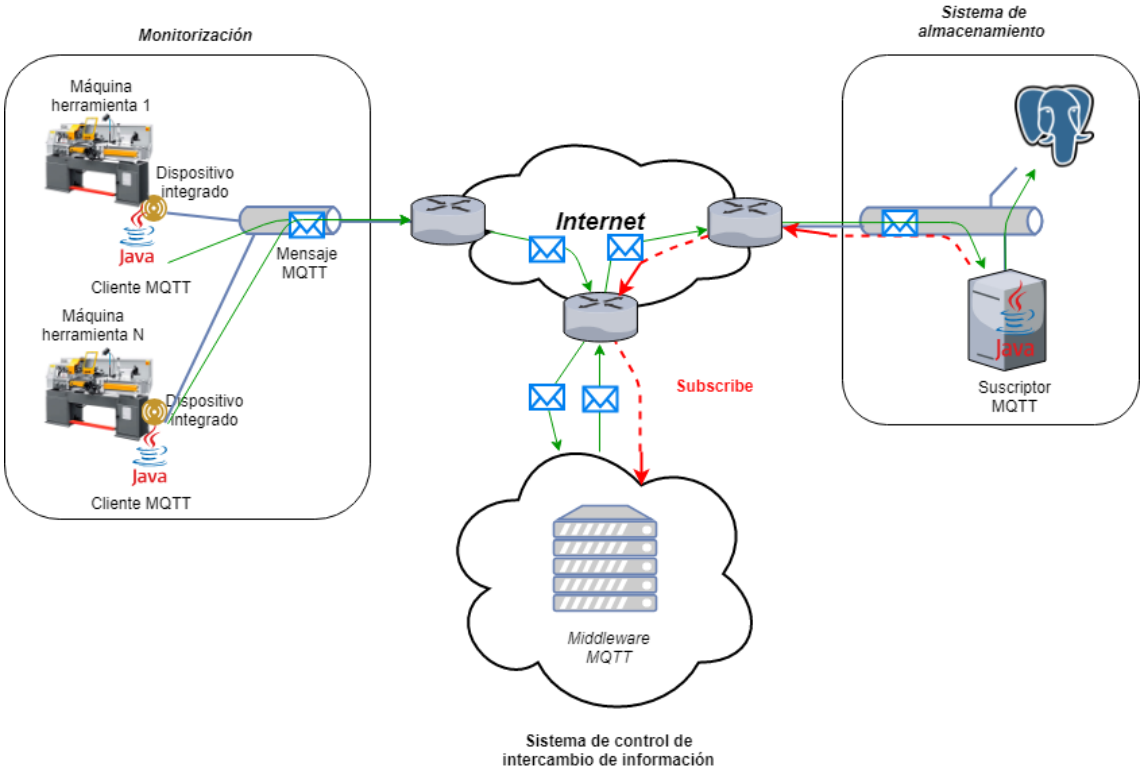

Ilustración 7: Topología del escenario de aplicación

7.2. Metodología

A continuación, se describe la metodología utilizada para escoger el middleware MQTT que se adecue mejor a los requerimientos y características del escenario de aplicación.

Inicialmente, se describen los middlewares MQTT más extendidos actualmente.

Posteriormente, se describen los aspectos más críticos del escenario. Estos aspectos van a definir las funciones mínimas necesarias que deben ofrecer los middlewares MQTT para minimizar la posibilidad de ocurrencia de situaciones indeseables. Todos aquellos middlewares MQTT que no ofrezcan alguna de dichas funcionalidades se descartarán.

A continuación, una vez conocidos los middlewares que ofrecen las funcionalidades inicialmente definidas, se definirán los parámetros de rendimiento más importantes a medir entendiéndose estos como aquellos que nos proporcionen la información más relevante acerca del rendimiento del middleware.

Finalmente, se definirán las pruebas a realizar que permitan conocer la solución que mayor rendimiento ofrezca en las situaciones de interés en cuanto a los parámetros de rendimiento previamente definidos.

7.2.1. Estado del arte. Middlewares MQTT

A continuación, se presenta una tabla con los middlewares MQTT más extendidos hoy en día junto a la versión más reciente de los mismos. Los análisis ser realizarán en todos los casos utilizando las versiones especificadas.

Middleware MQTT	RabbitMQ	VerneMQ	Mosquitto	EMQ	HiveMQ
Versión	3.7.7	1.4.0	1.5	2.3.9	3.4.0

Tabla 3: Estado del arte. Middlewares MQTT

7.2.2. Aspectos críticos

A continuación, se describen para cada una de las prestaciones definidas en los objetivos, las situaciones críticas asociadas y las funcionalidades que han de implementar los middlewares MQTT para evitar las mismas. De lo contrario, se descartarán y no podrán llegar a ser una opción válida.

7.2.2.1. Software personalizado y económico

Con el objetivo de poder desplegar una solución personalizada que permita implementar aquellas funciones que sean de interés para el caso en cuestión en cada momento, es necesario que el software utilizado para el despliegue del middleware sea personalizable.

Por otro lado, también se requiere de un software económico intentando que el coste asociado sea el mínimo posible.

Las soluciones OpenSource permiten realizar despliegues de una forma sencilla pudiendo realizar consultas a otros usuarios y personalizar las funciones implementadas por el middleware a coste 0.

Por ello, se descartarán todas aquellas soluciones que no sean OpenSource.

RabbitMQ	VerneMQ	Mosquitto	EMQ	HiveMQ
X	X	X	X	

Tabla 4: Middlewares MQTT OpenSource

Como se describe en la tabla 4, el middleware HiveMQ es una solución privada que requiere de un desembolso económico para su despliegue, es decir, no es OpenSource y, por tanto, es descartado.

7.2.2.2. Fiabilidad

Cuando se produce un intercambio de mensajes, estos pueden perderse, llegar duplicados, desordenados o con errores.

Por un lado, el protocolo TCP sobre el que trabaja MQTT garantiza que los mensajes no lleguen desordenados o con errores.

Por otro lado, en el escenario de aplicación, una situación en la que los mensajes lleguen duplicados no supone ninguna repercusión. Los mensajes, contienen una medida junto a su valor y el sello de tiempo en el que se realizó dicha medición. Gracias a este sello, desde el

sistema de procesamiento de datos pueden gestionarse de forma sencilla los mensajes duplicados y desordenados.

Sin embargo, una pérdida de varios de los mensajes puede provocar las siguientes situaciones no deseables:

- **Monitorización ineficiente:** La representación de los datos no es real.
- **Predicción ineficiente:** Las predicciones realizadas no contienen todos los datos de los movimientos de las máquinas herramienta. Rangos de valores medidos en un determinado momento pueden ser perdidos, no registrados y, por tanto, no contemplados en la predicción. Esto provoca la no predicción de estas situaciones en un futuro.
- **Errores no detectados por el sistema de gestión de alarmas:** Mensajes perdidos con valores asociados a errores en una de las máquinas herramienta no son detectados por el sistema de gestión de alarmas provocando un aumento del tiempo de indisponibilidad de esta y de aquellas que dependan de esta.

Para evitar las pérdidas es necesaria la implementación de un mecanismo de detección de pérdidas. Para ello, los middlewares deben poder soportar niveles de QoS 1 o 2.

Por un lado, con un nivel de QoS 1 los middlewares mediante el intercambio de un mensaje de control ofrecen garantía de entrega de los mensajes (véase el anexo I). No garantizan que estos vayan a llegar ordenados o sin duplicidad. Por otro lado, con un nivel de QoS 2, los middlewares garantizan que no van a ocurrir ni pérdidas ni mensajes duplicados mediante el intercambio de un conjunto de mensajes de control (véase el anexo I).

Con el objetivo de evitar las pérdidas en el intercambio de mensajes, es necesario descartar aquellas soluciones que no permitan implementar mínimo un nivel de QoS 1.

RabbitMQ	VerneMQ	Mosquitto	EMQ	HiveMQ
X	X	X	X	x

Tabla 5: Middlewares MQTT que garantizan gestión de pérdidas

Como se puede observar en la tabla 5, todos los middlewares MQTT permiten la implementación de QoS 1 o lo que es lo mismo, un mecanismo que garantiza la entrega de los mensajes. Por ello, ninguno de los middlewares será descartado.

7.2.2.3. Disponibilidad

En el escenario planteado, una caída del middleware MQTT provoca la pérdida de todos los mensajes que están publicándose durante ese periodo de tiempo imposibilitando la representación de los datos y la detección de errores en tiempo real. También provocaría una no consideración de los mensajes perdidos en la función de predicción.

El primer y último caso podrían llegar a solucionarse mediante la implementación de un mecanismo en los clientes que detecte la caída del middleware y almacene temporalmente las medidas realizadas durante ese periodo para posteriormente ser publicadas. Se tratan de escenarios menos sensibles al retardo.

Sin embargo, esta solución no evita el segundo de los casos. Durante el periodo de indisponibilidad, las medidas realizadas no serán representadas y posibles errores no podrán ser detectados. Este caso es más sensible al retardo.

Con el objetivo de aumentar el tiempo de disponibilidad del middleware es necesario poder implementar clústers, es decir, réplicas que permitan ante la caída de una el procesamiento del tráfico entrante por parte del resto. Estas replicas son transparentes de cara a los clientes, viéndose estas como un único middleware en todo momento.

Por ello, es necesario descartar aquellas soluciones que no permitan la implementación de clústers.

RabbitMQ	VerneMQ	Mosquitto	EMQ	HiveMQ
X	X		X	X

Tabla 6: Middlewares MQTT que soportan Clústers

A la vista de los resultado de la tabla 6, el middleware Mosquitto es descartado ya que no permite la implementación de réplicas o clústers que garanticen un tiempo de disponibilidad del servicio mayor.

7.2.2.4. Seguridad. Control de acceso

El middleware MQTT está expuesto a Internet. Esto provoca dos situaciones críticas:

1. Cualquier usuario puede realizar publicaciones en el middleware con datos aleatorios provocando, por un lado, alteraciones en los resultados representados y las predicciones realizadas y, por otro lado, detecciones de errores que realmente no han ocurrido.
2. Cualquier usuario puede realizar suscripciones obteniendo de ese modo información acerca del movimiento de las máquinas.

Para evitar estas situaciones, es necesario la implementación de un mecanismo que permita controlar el acceso al middleware. El mecanismo a implementar debe permitir gestionar el acceso al middleware mediante la asignación de credenciales.

Por ello, es necesario descartar aquellas soluciones que no permitan implementar mecanismos de control de acceso a los recursos.

Además, es necesario que los mecanismos de control de acceso a los recursos se implementen mediante el despliegue de un servidor LDAP (Lightweight Directory Access Protocol). Otras posibles implementaciones mediante ficheros internos del middleware o bases de datos SQL (Structured Query Language) presentan deficiencias.

En caso de utilizar fichero internos, el control de acceso estaría configurado en ficheros de los propios middlewares. El control de acceso a los recursos sería bastante complejo y engorroso teniendo que acceder individualmente a cada middleware en caso de que hubiera que realizar modificaciones. Por otro lado, la facilidad con la que terceros pueden realizar modificaciones sobre estos ficheros es grande debido a que estos, por lo general no se encuentran cifrados o únicamente disponen de una pequeña y simple encriptación (solo se necesitaría tener acceso a los mismos).

En caso de utilizar bases de datos SQL, sería necesario establecer una base de datos por empresa (con sus tablas personalizadas asociadas) reduciendo la facilidad de escalabilidad. Además, aumentaría el nivel de complejidad en el control de acceso a los recursos a la hora de añadir nuevos usuarios y permisos.

La implementación de un servidor LDAP va a permitir, por un lado, tener centralizado el acceso a los recursos de todos los middlewares desplegados en las diferentes empresas a las que se les

esté dando soporte. Por otro lado, permite de una forma sencilla la inserción de nuevos usuarios, la asignación de los permisos asociados y sobre todo la inserción de nuevos servicios.

El servidor LDAP se desplegará en la red interna del proveedor del servicio. El control de acceso al servidor LDAP se realiza mediante el uso de credenciales y un firewall. Por un lado, la gestión de los usuarios y los permisos en el servidor LDAP serán realizados por un personal autorizado mediante la asignación de credenciales. Los usuarios y los permisos asociados serán solicitados por la empresa a la que se le esté ofreciendo el servicio y configurados posteriormente. Por otro lado, se configurará el firewall frontera del proveedor del servicio para no permitir el acceso desde el exterior ni al host donde se encuentre desplegado el servidor LDAP ni al puerto 389 donde está expuesto el servicio.

RabbitMQ	VerneMQ	Mosquitto	EMQ	HiveMQ
X		X	X	X

Tabla 7: Middlewares MQTT que soportan la implementación de mecanismos de control de acceso mediante servidores LDAP

Como se puede observar en la tabla 7, el middleware VerneMQ no soporta la implementación de mecanismos de control de acceso a través de servidores LDAP. Por ello, esta opción es descartada.

7.2.2.5. Seguridad. Confidencialidad

El intercambio de mensajes entre los dispositivos, el middleware y el sistema de almacenamiento se realiza a través de Internet. Esto, por tanto, permite a cualquier usuario realizar una escucha de las conversaciones obteniendo de este modo información acerca del estado de las máquinas, credenciales de usuarios…

Para evitar esta situación es necesario intercambiar los mensajes a través de comunicaciones seguras cifradas.

Para ello, se desplegará una autoridad certificadora (AC) en la red interna del proveedor del servicio. El acceso a la AC estará restringido mediante un firewall frontera. Los certificados solo podrán ser emitidos por usuarios autorizados mediante el uso de credenciales previa solicitud de la empresa a la que se le esté ofreciendo el servicio. Una vez emitidos los certificados estos se almacenarán en los equipos a los que correspondan con una serie de permisos de tal modo que no puedan ser leídos ni modificados por terceros no autorizados.

Por ello, es necesario descartar aquellas soluciones que no permitan realizar el intercambio de mensajes mediante conexiones SSL (Secure Sockets Layer) seguras.

RabbitMQ	VerneMQ	Mosquitto	EMQ	HiveMQ
X	X	X	X	x

Tabla 8: Middlewares MQTT que soportan la implementación de conexiones SSL seguras

Como se puede observar en la tabla 8, todas las opciones permiten la implementación de conexiones SSL seguras. Por tanto, ninguna es descartada.

7.2.2.6. Resultados obtenidos

En función del análisis realizado, se han descartado los middlewares MQTT Mosquitto, VerneMQ y HiveMQ.

De este modo, los middlewares MQTT que se analizarán para conocer las prestaciones que ofrecen en cuanto a rendimiento son RabbitMQ y EMQ. Para analizar el rendimiento, a

continuación, se describen las medidas a realizar, las pruebas realizadas, el escenario de pruebas desplegado sobre el cual se han realizado y los resultados obtenidos.

7.2.3. Medidas a realizar

En escenarios donde se realizan representaciones de datos y detecciones de errores en tiempo real es de vital importancia que, en lo que al middleware se refiere, las medidas realizadas lleguen lo antes posible al sistema de almacenamiento para su procesado. De lo contrario, los errores no podrán detectarse a tiempo provocando la indisponibilidad de la máquina o lo que es peor de una línea de producción que dependa de la máquina en dicho momento.

El tiempo necesario para que una medida realizada por un dispositivo sea almacenada depende de la tasa de paquetes con la que el dispositivo realice las publicaciones, del tiempo de transmisión del mensaje a través de Internet, del tiempo necesario para procesar dicho mensaje por el middleware, del tiempo que permanece el mensaje encolado en el middleware, de la tasa con la que el suscriptor recibe mensajes del middleware, del tiempo de procesamiento de los mensajes en el suscriptor y del tiempo necesario para realizar la inserción en la base de datos.

El rendimiento del middleware, por tanto, tendrá repercusión en la tasa de paquetes con la que los dispositivos realicen las publicaciones, en el tiempo de procesamiento de los mensajes en él, en el tiempo que permanecen los mensajes encolados y en la tasa con la que el suscriptor recibe los mensajes del middleware. El tiempo de encolamiento de los mensajes está directamente relacionado con la tasa con la que el suscriptor recibe los mensajes del middleware.

Por ello, con el objetivo de comprobar y comparar el rendimiento de los middlewares en diferentes situaciones se ha decidido realizar medidas sobre los siguientes parámetros:

7.2.3.1. Throughput

El throughput se entiende como la tasa de paquetes que se envían a través de un canal de comunicación. En el caso del middleware este se divide en:

- **Throughput de entrada:** Tasa de paquetes publicados por un cliente (publicador) a través de un canal de comunicación.
- **Throughput de salida:** Tasa de paquetes recibidos por un cliente (suscriptor) a través de un canal de comunicación.

Como se ha comentado con anterioridad, tanto en la publicación como la suscripción en función del nivel de QoS definido en el establecimiento de la conexión, se intercambian una serie de mensajes de control con el objetivo de garantizar cierta fiabilidad.

Suponiendo unas mismas capacidades computacionales tanto de los publicadores como del suscriptor, a un mismo nivel de QoS tanto en la publicación como en la suscripción, el throughput de entrada y salida será el mismo dependiendo este únicamente de la capacidad del middleware para gestionar los diferentes mensajes de control intercambiados.

Ante una mayor capacidad de gestión de los mensajes de control, mayor será tanto el throuhgput de entrada como de salida y, por tanto, mayor será la velocidad con la que los mensajes lleguen al sistema de almacenamiento. Por otro lado, en caso de un encolamiento de mensajes, a mayor throuhgput de salida, mayor será la velocidad con la que las colas se vacían o, dicho de otra manera, menor será el tiempo de encolamiento de los mensajes.

Por ello, sabiendo que tanto el suscriptor como el publicador establecen conexiones con el middleware con un mismo nivel de QoS en los diferentes casos a analizar, será suficiente con conocer uno de los dos throughputs.

En concreto, se analizará el throughput de salida y este se calculará como el número de paquetes totales recibidos entre el tiempo total necesario para recibirlos. El número de paquetes totales se dividirá en 10 grupos con el objetivo de conocer como varía el throughput de salida a lo largo de esos 10 tramos que dura la prueba.

7.2.3.2. Latencia

La latencia se entiende como la diferencia de tiempos entre el momento en el que un mensaje es publicado y el momento en el que dicho mensaje es recibido por el suscriptor previo a ser procesado.

Considerando que el thorughput de entrada y salida es el mismo, que los tiempos de transmisión del mensaje a través de la red son despreciables (en el escenario de pruebas todos los equipos se encuentran en la misma máquina) y trabajando en un escenario estable donde el tráfico entrante no sobrecarga el middleware ni se producen caídas de la red o del suscriptor (tiempo de espera en cola despreciable), la latencia permitirá conocer la capacidad que tienen los middlewares a la hora de ejecutar la lógica de encaminamiento. Es decir, permitirá conocer el tiempo de procesamiento de los mensajes en el middleware.

A menor latencia, la capacidad del middleware para ejecutar la lógica de encaminamiento será mayor y, también, la velocidad con la que los mensajes llegan al sistema de almacenamiento.

La latencia se calculará como la diferencia de tiempos entre el momento que el suscriptor recibe un mensaje y el tiempo en el que se ha realizado la publicación de este. Para ello, se asociará un sello de tiempo al payload del mensaje con el momento en el que se realizó la publicación. Al igual que en el caso del throughput de salida, el número de paquetes totales se dividirá en 10 tramos con el objetivo de conocer como varia la latencia a lo largo de dichos 10 tramos. En cada tramo la latencia se calcula como la media de las latencias de todos los mensajes.

7.2.4. Escenario de pruebas

7.2.4.1. Set Up

El escenario se ha desplegado sobre una Máquina Virtual con un sistema operativo (SO) Ubuntu 16.04 de 6Gb de RAM (Random Access Memory) y 1 único procesador.

Los clientes MQTT se han desarrollado en Java utilizando la librería Eclipse Paho Client en el entorno de desarrollo Eclipse.

Los middlewares y el servidor LDAP se ha desplegado mediante contenedores de Docker.

La autoridad certificadora se ha desplegado en la propia máquina virtual de Ubuntu.

SO	Ubuntu 16.04
Procesador	Intel® Core™ i5-7500 CPU @ 3.40GHz
RAM	6 GB
Versión Java	1.8.0_171
Versión Eclipse Paho Client	1.2.0
Versión Docker	18.03.1-ce
Versión RabbitMQ	3.7.7 (Docker image: library/rabbitmq)

Versión EMQ	2.3.9 (Docker image: machinedata/emq)

Tabla 9: Detalles del Set Up

A continuación, se describen cada uno de los elementos que conforman la maqueta donde se han realizado las pruebas.

7.2.4.2. Maqueta

La maqueta desplegada tiene como objetivo analizar únicamente el rendimiento del middleware. Para ello, se han contemplado los procesos desde que se realizan las publicaciones hasta que los mensajes son recibidos por el suscriptor. Los procesos de realización de medidas por parte de los dispositivos y de procesamiento de los mensajes e inserción en la base de datos por parte del suscriptor no han sido contemplados ya que no afectan al rendimiento del middleware.

La maqueta en el cual van a realizarse las pruebas consta de los siguientes actores:

Publicador/es: proceso en Java que implementa varios clientes MQTT que realizan publicaciones en un único topic del middleware de forma síncrona tan rápido como le sea posible. Es decir, cada cliente realizará una publicación cuando el cliente anterior haya realizado una de sus publicaciones correctamente.

No se contempla la posibilidad de publicar en paralelo. Esto se debe a dos factores. Por un lado, a la necesidad de ejecutar procesos en paralelo lo cual requiere un mayor número de recursos del set up donde se está ejecutando también el middleware y, por tanto, afectaría al rendimiento del middleware. Por otro lado, dado el escenario de aplicación, el throughput de publicación realizando publicaciones síncrona es suficiente.

Esto, sin embargo, no es un problema. Por un lado, la repercusión del número de conexiones establecidas en el rendimiento del middleware ya se contempla mediante la implementación de varios clientes, aunque no se ejecuten en paralelo. Por otro lado, no es necesario conocer cómo se produce el encolamiento de los mensajes ante una sobrecarga de tráfico entrante cuando hay muchos procesos en paralelo. Lo importante es que el throughput de salida sea alto para que los encolamientos no se produzcan o se produzcan durante el menor tiempo posible.

Middleware MQTT: sistema de control de intercambio de información que se encarga de recibir los mensajes publicados por el/los publicador/es, procesarlos y enviarlos a el suscriptor.

Cuando se implemente el control de recursos realizará un intercambio de mensajes con el servidor LDAP para verificar las credenciales presentadas por los publicadores y el suscriptor.

En primera instancia, será RabbitMQ y, en segunda instancia EMQ.

Suscriptor MQTT: proceso en Java que implementa un cliente MQTT que realiza inicialmente la función de suscribirse al topic del middleware en el que se está trabajando y, posteriormente la de recibir los mensajes entregados por el middleware.

Servidor LDAP: Servicio que permite la gestión de usuarios y permisos para la implementación en el middleware de un mecanismo de control de acceso mediante el uso de credenciales. Se ha desplegado un servidor openLDAP.

Autoridad Certificadora: Servicio que permite la emisión de certificados y claves para los clientes MQTT y el middleware para el intercambio de mensajes a través de conexiones SSL seguras cifradas. Ubuntu permite el despliegue de forma nativa de una autoridad certificadora.

A continuación, se muestra una imagen de la maqueta desplegada para la realización de pruebas:

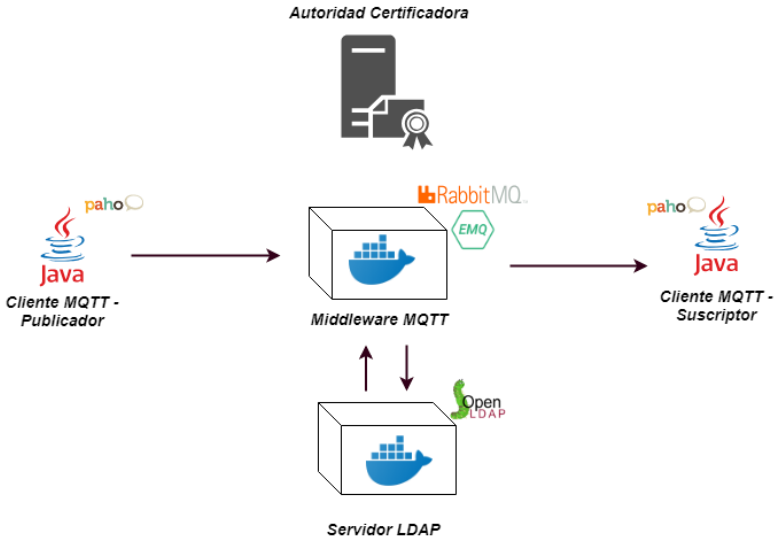

Ilustración 8: Maqueta del escenario de pruebas

7.2.5. Pruebas realizadas

En este apartado, se describen las diferentes pruebas que se han decidido realizar en los middlewares para conocer el rendimiento de estos cuando estos ofrecen las prestaciones definidas en los objetivos.

7.2.5.1. Fiabilidad

Con el objetivo de evitar la pérdida de mensajes es necesario ofrecer un nivel de QoS 1 o 2. En el caso de RabbitMQ no se contempla la posibilidad de ofrecer un nivel de QoS 2. Por ello, la comparativa se ha realizado con un nivel de QoS 1.

Tal y como se comentó en la descripción del equipamiento, el proyecto está enfocado a una empresa donde el número de máquinas puede oscilar entre 1 y 100. Además, el número de máquinas puede ser variante con el tiempo. Por ello, el middleware debe ser capaz de ofrecer un rendimiento mínimo para poder explotar datos en tiempo real tanto con el mínimo número de máquinas (1) como con el máximo estimado (100).

Para conocer el rendimiento de los middlewares cuando se ofrece un nivel de QoS 1 para un número de clientes diferentes, se han realizado una serie de pruebas donde se han definido 10, 50 y 100 publicadores publicando 500, 100 y 50 mensajes respectivamente en una única instancia con una QoS de 1 sin mecanismos que ofrezcan control de acceso ni confidencialidad.

El número de mensajes por publicador se ha definido sabiendo que a partir de los 5.000 mensajes de entrada totales al middleware el comportamiento de este es estable y, con el objetivo de igualar las pruebas realizadas en cuanto al número total de mensajes publicados se refiere. De este modo, se podrá conocer cómo afecta la gestión del número de clientes al rendimiento del middleware.

El nivel de QoS se ha definido con el objetivo de garantizar la entrega de los mensajes.

No se han implementado varias instancias del middleware ni mecanismos que ofrezcan control de acceso o confidencialidad para evitar que estos afecten a los resultados.

7.2.5.2. Disponibilidad

El aspecto más crítico a la hora de garantizar una alta disponibilidad son los ataques externos. Para evitar los mismos, se han implementado mecanismos de control de acceso y confidencialidad. Estos, se explican en los apartados siguientes.

Se considera que la probabilidad de que el middleware se caiga sin acciones de terceros es muy remota. Por esto mismo, inicialmente, no se ha considerado el despliegue de múltiples instancias del middleware y, por tanto, el rendimiento en función del número de instancias no será objeto de estudio.

7.2.5.3. Seguridad. Control de acceso

Con el objetivo de controlar el acceso a los recursos del middleware se ha desplegado un servidor LDAP donde se han configurado una serie de credenciales. De este modo, el middleware solo aceptará conexiones de aquellos usuarios que presenten unas credenciales válidas.

Para ello, se han registrado en el servidor LDAP dos nuevos usuarios “publicador” y “suscriptor” con sus contraseñas asociadas. Además, se ha realizado la configuración necesaria en los middlewares de tal modo que, implementen el mecanismo de autenticación únicamente a través del servidor LDAP.

La implementación de un mecanismo de control de acceso externo supone la necesidad de realizar una consulta al mismo por parte del middleware cada vez que se realiza una publicación con el objetivo de verificar las credenciales del publicador. Esto, por tanto, supondrá un retardo adicional que afectará tanto al throughput como a la latencia. Por ello, es importante comprobar cómo afecta la implementación de este mecanismo al rendimiento del middleware con el objetivo de conocer que solución optimiza en mayor medida el despliegue de este nuevo servicio manteniendo la explotación de datos en tiempo real.

Por un lado, para verificar que los recursos son solo accesibles previa presentación de credenciales válidas, se han hecho un conjunto de pruebas donde se ha intentado realizar publicaciones y suscripciones con credenciales no configuradas en el servidor LDAP.

Por otro lado, para conocer cómo afecta la implementación de un control de acceso mediante un servidor LDAP en el rendimiento de los middlewares, se han realizado una serie de pruebas donde el número de publicadores se ha definido en 50 (media de los publicadores definidos en el escenario de aplicación), el número de mensajes en 100 (se considera el comportamiento del middleware estable a partir de un total de 5.000 mensajes publicados), el nivel de QoS en 1 (definido para garantizar la entrega de los mensajes), el número de instancias en 1 y no se ha desplegado ningún mecanismo que ofrezca confidencialidad.

El número de instancias definido y la no implementación de un mecanismo que garantice confidencialidad se deben a la necesidad de evitar que estos afecten a los resultados.

7.2.5.4. Seguridad. Confidencialidad

Con el objetivo de garantizar confidencialidad, se ha hecho uso de certificados y claves que permitan el establecimiento de conexiones SSL entre el middleware y los clientes a través de las cuales se realiza el intercambio de mensajes de forma segura.

Para ello, se ha desplegado una autoridad certificadora, se han generado y emitido los certificados tanto al middleware como a los clientes y se ha realizado la configuración oportuna

de tal modo que el middleware sea solo accesible a través del puerto habilitado para conexiones SSL.

La implementación de conexiones SSL supone la necesidad de realizar funciones de cifrado y descifrado tanto en los clientes como en el middleware. Esto, por tanto, añadirá un retardo adicional que afectará al throughput y a la latencia. Por ello, es importante conocer cómo afectan las funciones de cifrado y descifrado al rendimiento del middleware y si se sigue manteniendo la explotación de datos en tiempo real.

Por un lado, para verificar que el middleware es solo accesible vía conexiones SSL a través del puerto habilitado para las mismas, se han realizado una serie de pruebas donde se ha intentado establecer conexiones TCP contra dicho puerto.

Por otro lado, para conocer cómo afecta el establecimiento de conexiones SSL en el rendimiento de los middlewares, se han realizado una serie de pruebas donde el número de publicadores se ha definido en 50, el número de mensajes en 100, el nivel de QoS en 1, el número de instancias en 1 y no se ha desplegado ningún mecanismo de control de acceso.

El número de instancias definido y la no implementación de un mecanismo que ofrezca control de acceso se deben a la necesidad de evitar que estos afecten a los resultados.

7.2.6. Resultados obtenidos

Una vez definidas las pruebas y medidas se han llevado a cabo las mismas con el objetivo de comparar el rendimiento de ambos middlewares en cada caso.

A continuación, se desglosan los resultados obtenidos para cada una de las pruebas realizadas.

7.2.6.1. Fiabilidad

7.2.6.1.1. Throughput de salida

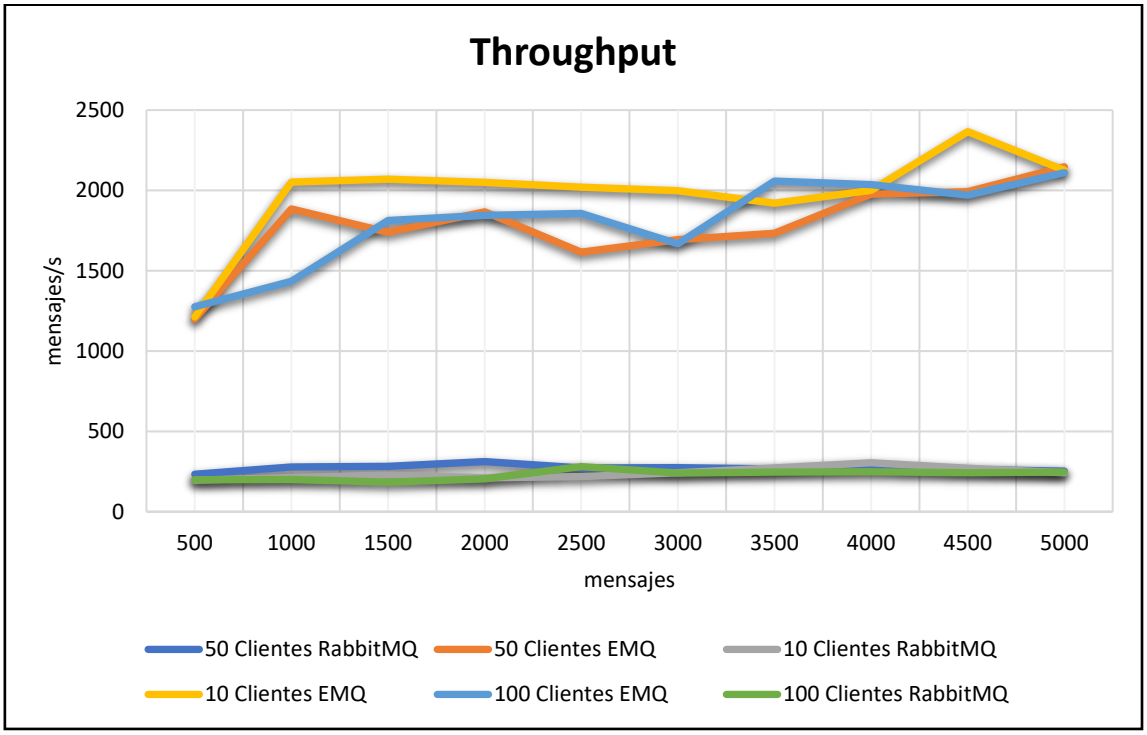

Ilustración 9: Throughput de salida en función del nº de clientes y el nivel de QoS

7.2.6.1.2. Latencia

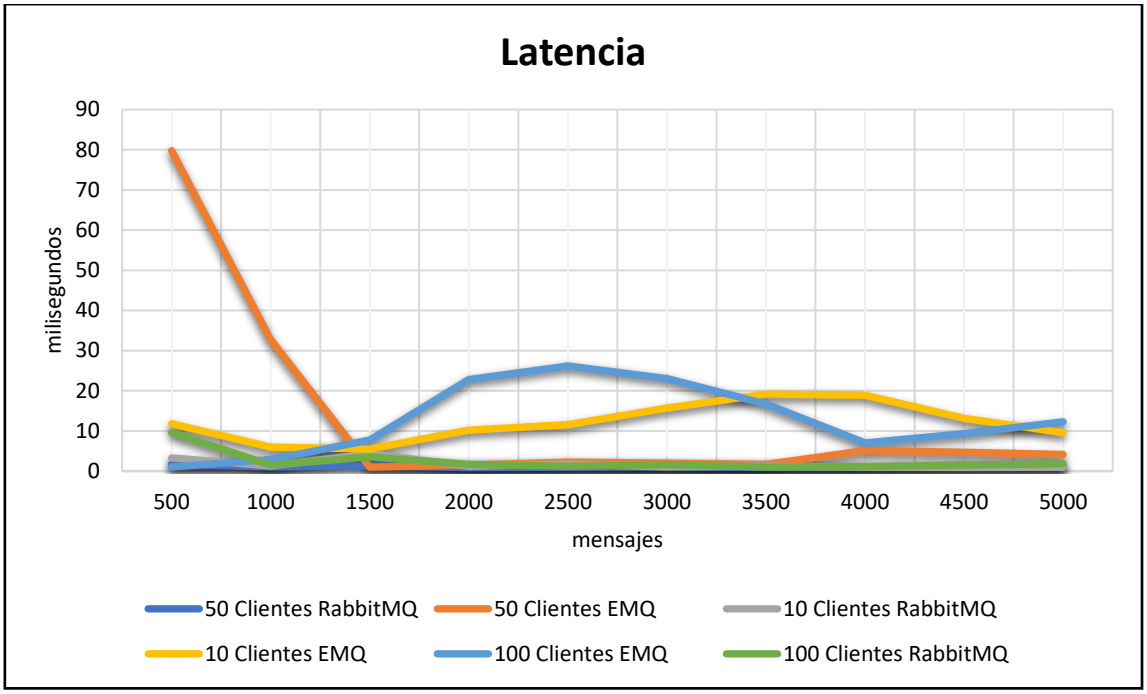

Ilustración 10: Latencia en función del nº de clientes y el nivel de QoS

7.2.6.2. Seguridad. Control de acceso

7.2.6.2.1. Publicación de credenciales no válidas

En la siguientes imágenes se observa la respuesta ofrecida por los middlewares y el servidor openLDAP al realizar una publicación con credenciales no válidas o sin ellas. En primera instancia se observan los resultados cuando se despliega el middleware EMQ. En segunda instancia cuando se despliega el middleware RabbitMQ.

```
openldap | 5b5705b2 conn=1003 fd=14 ACCEPT from IP=172.18.0.3:36134 (IP=0.0.0.0:389)
openldap | 5b5705b2 conn=1003 op=0 BIND dn="cn=noValidAuth,ou=Users,dc=tekniker,dc=es" method=128
openldap | 5b5705b2 conn=1003 op=0 RESULT tag=97 err=49 text=
emq      | 10:55:46.400 [error] Client(publicador_0@172.18.0.1:38756): Username 'noValidAuth' login failed for invalid_credentials
openldap | 5b5705b6 conn=1002 fd=13 closed (connection lost)
openldap | 5b5705b7 conn=1003 fd=14 closed (connection lost)
```

Ilustración 11: Log del middleware EMQ y servidor LDAP. Invalid Credentials

```
rabbitmq | 2018-09-06 15:10:58.344 [info] <0.755.0> LDAP CHECK: login for noValidAuth
rabbitmq | 2018-09-06 15:10:58.344 [info] <0.755.0>        LDAP filling template "cn=${username},ou=Users,dc=tekniker,dc=es" with
rabbitmq |           [{username,<<"noValidAuth">>}]
rabbitmq | 2018-09-06 15:10:58.344 [info] <0.755.0>        LDAP template result: "cn=noValidAuth,ou=Users,dc=tekniker,dc=es"
rabbitmq | 2018-09-06 15:10:58.344 [info] <0.755.0>    LDAP connecting to servers: ["openldap"]
openldap | 5b914382 conn=1000 op=1 RESULT tag=97 err=49 text=
rabbitmq | 2018-09-06 15:10:58.344 [info] <0.746.0>    LDAP network traffic: bind request = {'BindRequest',3,"cn=xxxx,ou=xxxx,dc=xxxx,dc=xxxx",{si
mple,"xxxx"}}
rabbitmq | 2018-09-06 15:10:58.345 [info] <0.746.0>    LDAP network traffic: bind reply = {ok,{'LDAPMessage',1,{bindResponse,{'BindResponse',inval
idCredentials,[],[],asn1_NOVALUE,asn1_NOVALUE}},asn1_NOVALUE}}
rabbitmq | 2018-09-06 15:10:58.345 [info] <0.373.0>    LDAP bind returned "invalid credentials": cn=xxxx,ou=xxxx,dc=xxxx,dc=xxxx
rabbitmq | 2018-09-06 15:10:58.345 [info] <0.755.0> LDAP DECISION: login for noValidAuth: denied
rabbitmq | 2018-09-06 15:10:58.345 [error] <0.750.0> MQTT login failed for "noValidAuth" auth_failure: Refused
```

Ilustración 12: Log del middleware RabbitMQ y servidor LDAP. Invalid Credentials

Se verifica que no es posible realizar publicaciones en dichas circunstancias.

7.2.6.2.2. Throughput de salida

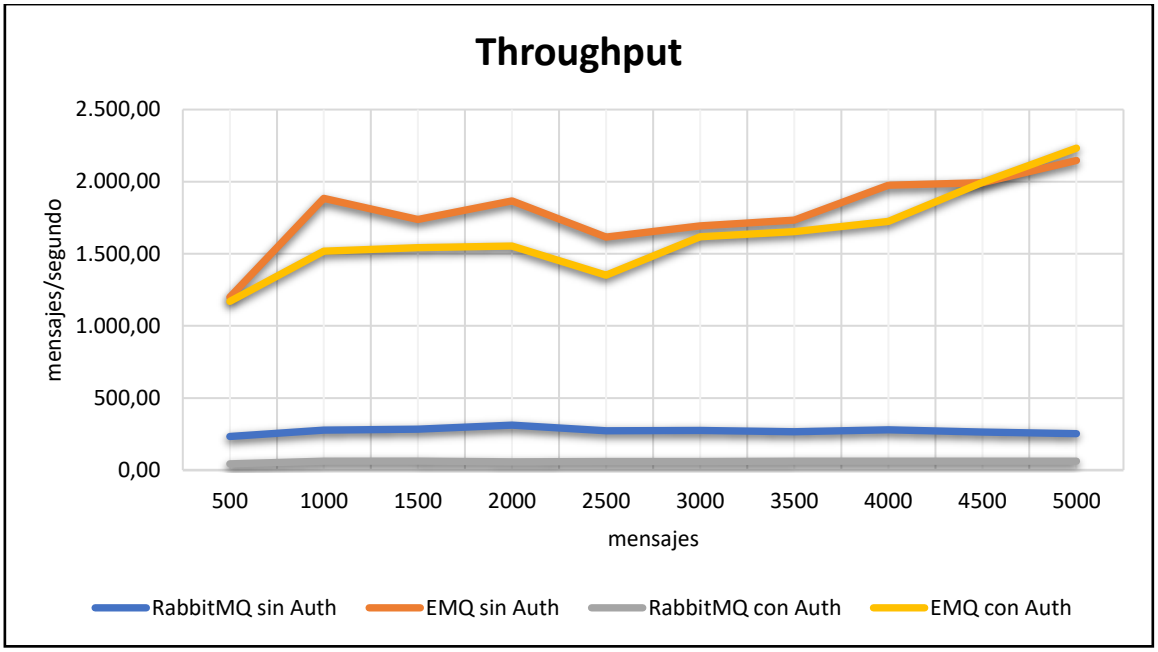

Ilustración 13: Throughput de salida en función del control de acceso implementado

7.2.6.2.3. Latencia

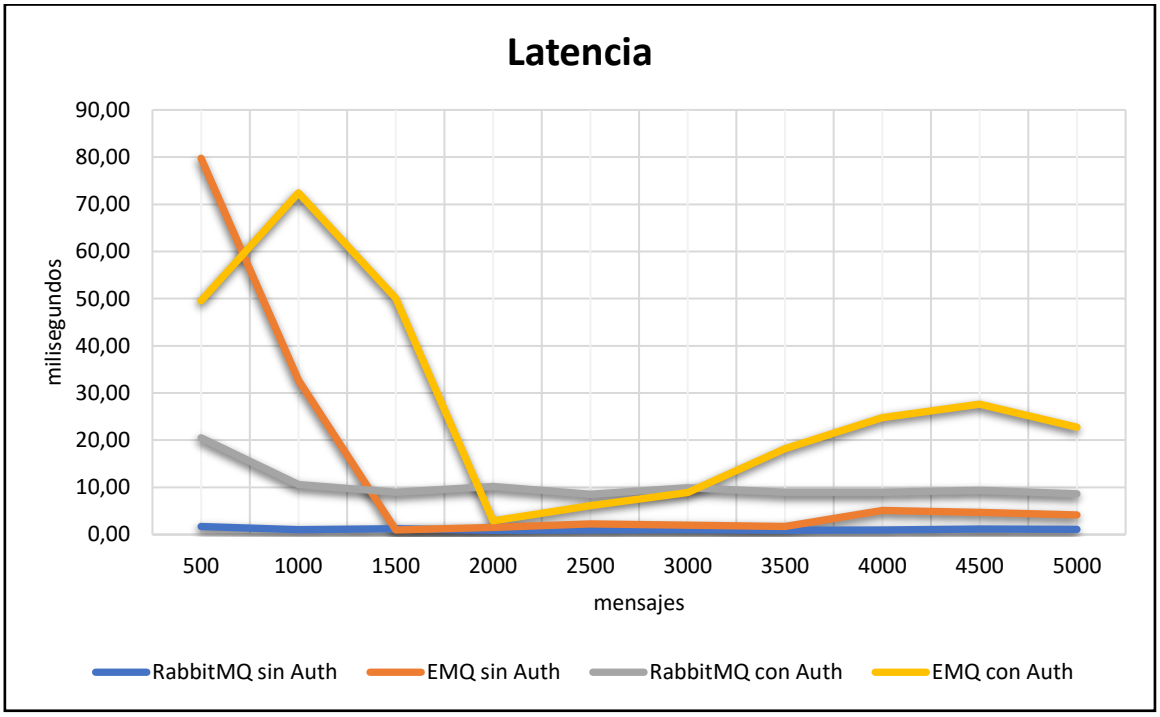

Ilustración 14: Latencia en función del control de acceso implementado

7.2.6.3. Seguridad. Confidencialidad

7.2.6.3.1. Publicación con certificados no válidos

A continuación, se observa la respuesta ofrecida por el middleware EMQ y RabbitMQ respectivamente al intentar un establecimiento de conexión con certificados no válidos o sin ellos.

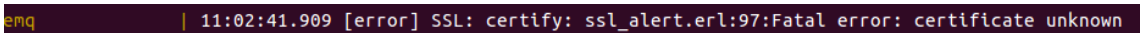

Ilustración 15: Log del middleware EMQ. Certificate Unknown

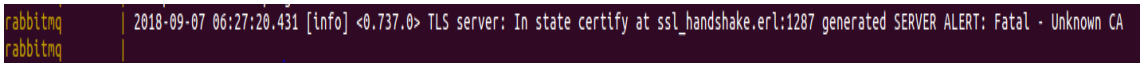

Ilustración 16: Log del middleware RabbitMQ. Unknown CA

Se verifica que no es posible realizar publicaciones en dichas circunstancias.

7.2.6.3.2. Comunicaciones cifradas

A continuación, se muestra en primera instancia el intercambio de mensajes derivado de la publicación por parte de un cliente MQTT de un mensaje con QoS 1. En la imagen se puede observar, por un lado, el SSL Handshake donde cada participante verifica la identidad del otro y negocian el algoritmo de encriptación. Por otro lado, se observan los mensajes asociados directamente a la publicación cifrados.

En segunda instancia, puede observarse el payload de uno de los mensajes. Se verifica que está cifrado.

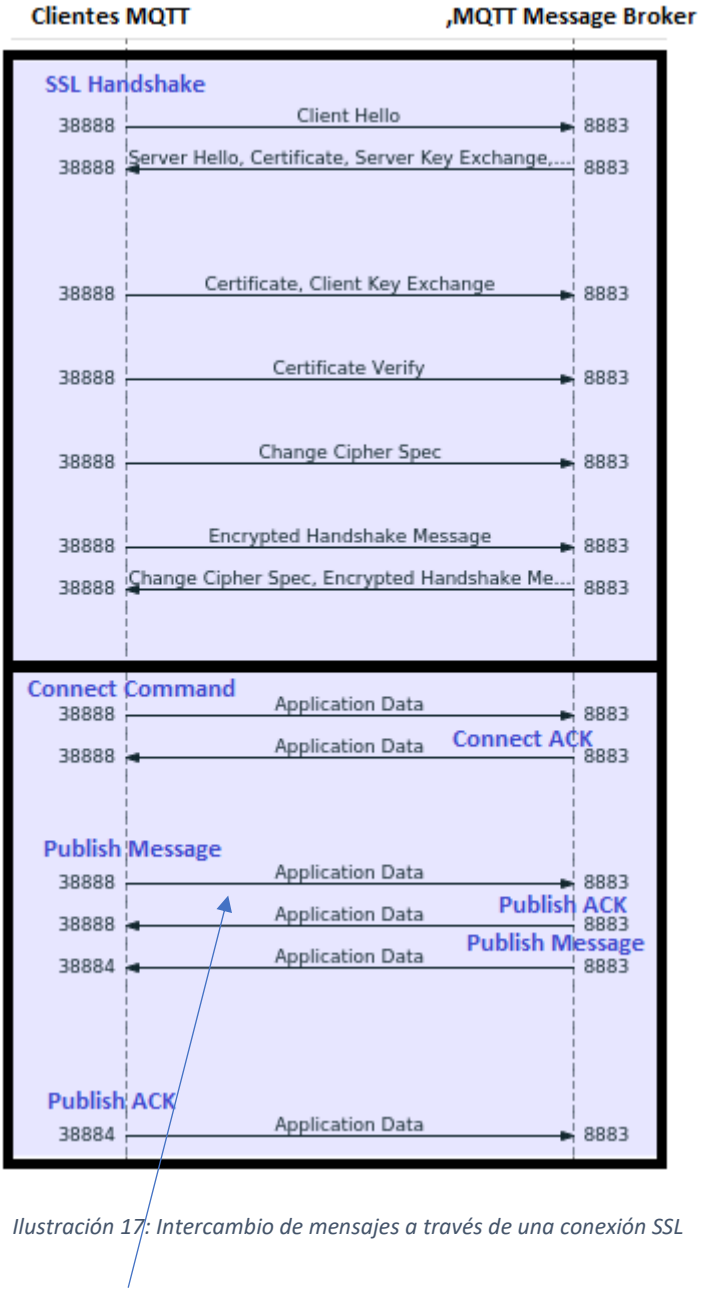

Ilustración 17: Intercambio de mensajes a través de una conexión SSL

```
    71 0.507819435   172.18.0.1          172.18.0.3          TLSv1.2   169 Application Data
▸ Frame 71: 169 bytes on wire (1352 bits), 169 bytes captured (1352 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 172.18.0.1, Dst: 172.18.0.3
▸ Transmission Control Protocol, Src Port: 38888, Dst Port: 8883, Seq: 1541, Ack: 2167, Len: 101
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Application Data Protocol: mqtt
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 96
      Encrypted Application Data: 3786a46a78e9d015a42852faf7ae19411d01104879b35d39...
```

Ilustración 18 Mensaje 'Publish Message' cifrado

7.2.6.3.3. Throughput de salida

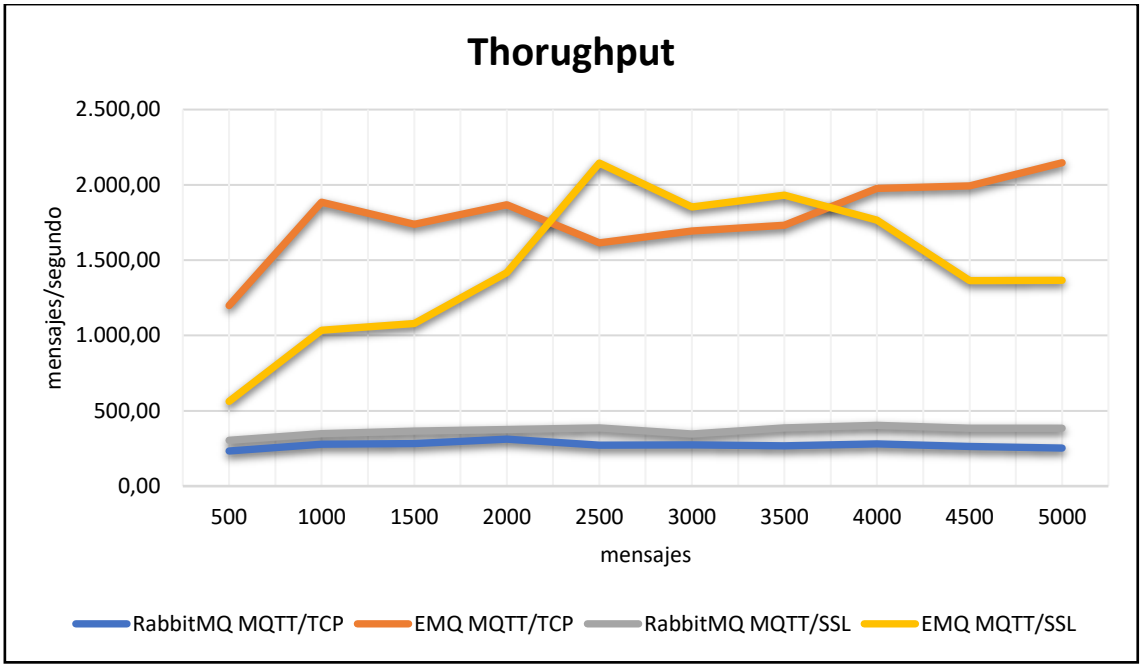

Ilustración 19: Throughput de salida en función del protocolo de transporte utilizado

7.2.6.3.4. Latencia

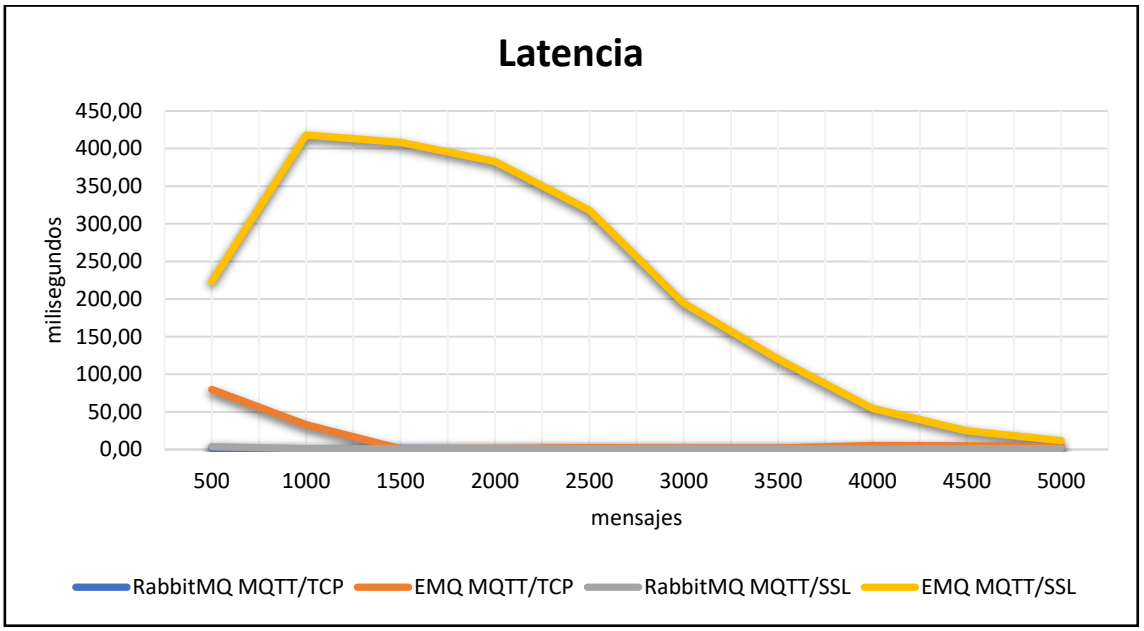

Ilustración 20: Latencia en función del protocolo de transporte utilizado

7.2.6.4. Análisis de los resultados

A la vista de los resultados obtenidos en el análisis de rendimiento, se pueden extraer las siguientes conclusiones:

EMQ ofrece un throughput muy superior a RabbitMQ

Bajo unas mismas circunstancias, EMQ ofrece un mayor rendimiento en cuanto a thoruhgput de salida y, por tanto, throughput de publicación. Por ello, con EMQ atendiendo únicamente al throughput de salida, el tiempo necesario para almacenar una determinada medida será mucho

menor. Además, en caso de producirse un encolamiento de mensajes, un mayor throughput de salida reduce los tiempos de encolamiento.

Tanto EMQ como RabbitMQ ofrecen unos valores de latencia óptimos y similares

Tanto en EMQ como en RabbitMQ la latencia observada en todos los casos es mínima y similar, no llegando a alcanzar un valor de 1 segundo. En EMQ en el escenario donde se implementan conexiones SSL es cierto que se observa una latencia inicial alta superior al resto de casos, sin embargo, esta converge a valores mínimos a medida que la prueba avanza debido al gran rendimiento que ofrece en cuanto a throughput de salida.

El rendimiento del middleware no depende del número de clientes

La variación del número de clientes o, dicho de otra manera, el número de conexiones establecidas con el middleware no afecta al rendimiento del middleware ni en throughput ni en latencia.

En EMQ el throughput se ve afectado en menor medida por la implementación de un mecanismo de control de acceso mediante un servidor LDAP

La reducción del throughput se debe a la necesidad de realizar un intercambio de mensajes entre el middleware y el servidor LDAP cada vez que se realiza una publicación con el objetivo de verificar las credenciales presentadas por el publicador correspondiente. Por ello, por cada mensaje se requiere de un tiempo adicional para realizar su procesamiento y, por tanto, el número de mensajes que un mismo cliente puede publicar en un determinado tiempo es menor.

En EMQ el throughput se ve reducido en un 8% mientras que en RabbitMQ se ve reducido en 78%.

En ambos casos la latencia se ve afectada en igual medida por la implementación de un mecanismo de control de acceso mediante un servidor LDAP

El aumento de la latencia se debe a la necesidad de un mayor número de recursos por parte del middleware para realizar las funciones relacionadas con la implementación del mecanismo de control de acceso. Debido a esto, los recursos disponibles para ejecutar la lógica de encaminamiento son menores y, por tanto, el tiempo necesario para procesar un mensaje en el middleware también.

Tanto en EMQ como en RabbitMQ la latencia aumenta en la misma medida, entorno a un 8%.

En ambos casos el rendimiento no se ve afectado por la implementación de conexiones SSL seguras

Teóricamente, la implementación de conexiones SSL seguras tiene repercusión en el rendimiento. Esto se debe a la necesidad de realizar las funciones de cifrado y descifrado por cada mensaje intercambiado. Sin embargo, dependerá en gran medida de la capacidad computacional tanto del middleware como de los clientes.

En las pruebas realizadas la repercusión es mínima. Esto se debe a la gran capacidad computacional de los clientes y del middleware. Sin embargo, en un escenario donde los clientes estuvieran implementados en dispositivos integrados con capacidades computacionales bajas, la implementación de conexiones SSL seguras reducirán en mayor medida el rendimiento y, por tanto, deberán ser un factor para tener en cuenta.

7.2.7. Despliegue de la solución adoptada

En función de las conclusiones realizadas en el análisis de resultados, la solución adoptada para el despliegue en el escenario de aplicación es EMQ.

7.2.7.1. Rendimiento

Partiendo de la maqueta definida en el escenario de pruebas, con el objetivo de conocer el rendimiento final ofrecido por el middleware EMQ cuando se garantizan las exigencias definidas al inicio se ha hecho un análisis del rendimiento para un número medio de publicadores (50) publicando 1000 mensajes (se quiere comprobar el comportamiento durante un periodo de tiempo suficiente), ofreciendo garantía en la entrega de mensajes (QoS 1) e implementando un mecanismo de control de acceso (servidor LDAP) y un mecanismo que garantiza la confidencialidad (conexiones SSL).

Los resultados obtenidos son los siguientes:

7.2.7.1.1. Throughput

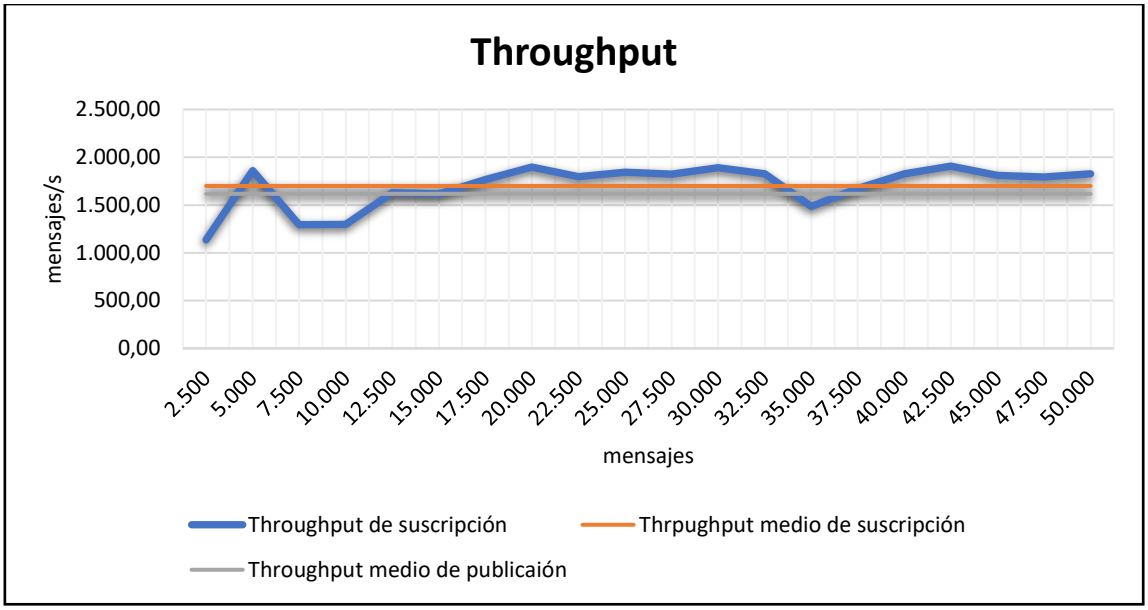

Ilustración 21: Throughput EMQ

7.2.7.1.2. Latencia

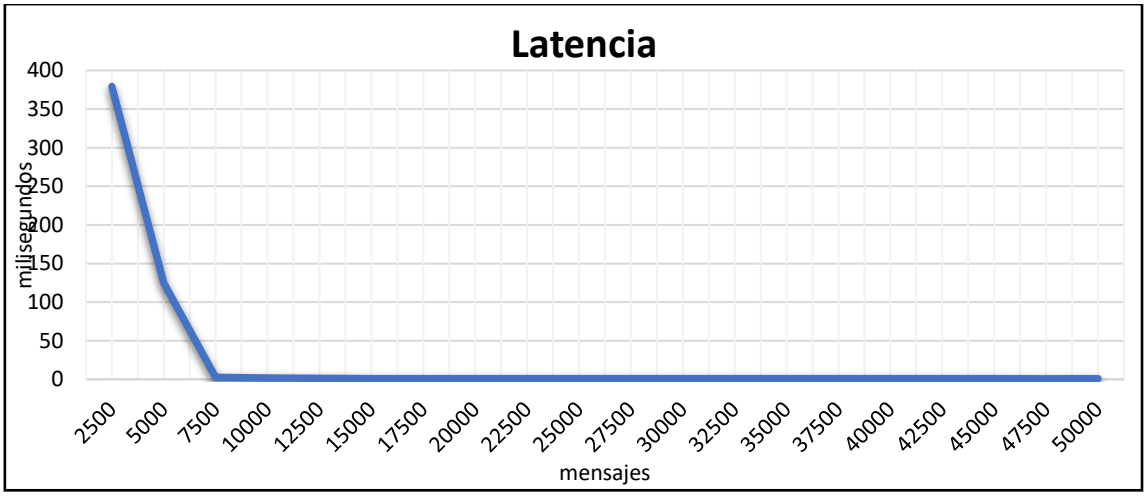

Ilustración 22: Latencia EMQ

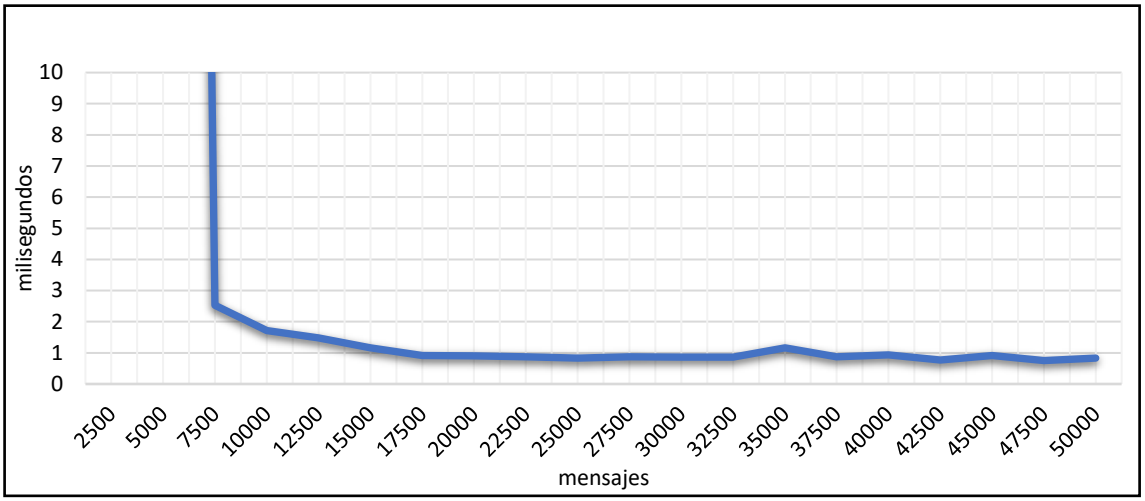

Ilustración 23: Latencia EMQ (Zoom)

7.2.7.2. Despliegue en un escenario real

A continuación, se ilustra la topología del escenario real donde se ha desplegado el middleware EMQ. En este escenario, se le ofrece el servicio a una empresa A con N máquinas herramienta a monitorizar.

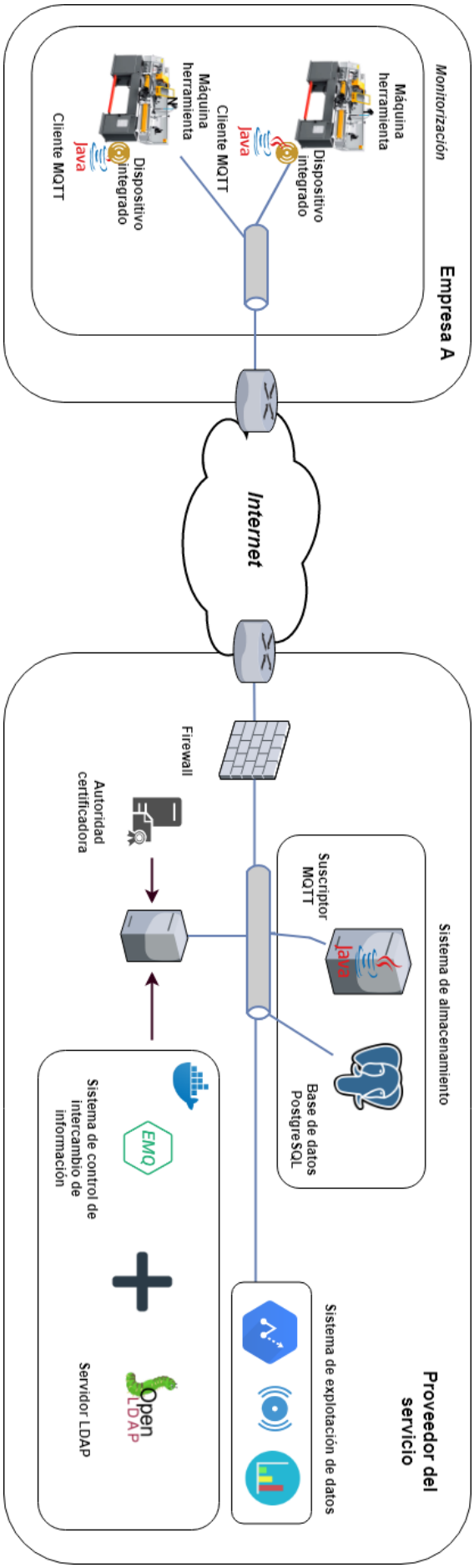

Tabla 10: Topología del escenario real

8. Descripción de tareas. Gantt

En esta apartado, se desglosarán las diferentes fases en las que se ha dividido el proyecto con el fin de poder llevarlo a cabo. Además, quedarán también reflejados los diferentes hitos del proyecto.

Definición del proyecto: Fase inicial del proyecto. En esta fase, se definen las especificaciones de este y se hace un análisis del estado del arte y de las diferentes alternativas existentes para la realización del proyecto. Esta fase está formada por las siguientes tareas:

1. **Definición de los objetivos y las especificaciones:** En esta tarea inicial, se definen los objetivos, el alcance y la planificación del proyecto. Además, se define el escenario de desarrollo.
2. **Estudio del estado del arte:** En esta tarea se hace una búsqueda inicial para conocer comparativas realizadas en cuanto a middlewares que sirvan de punto de partida para el desarrollo de la metodología comparativa que se describe en este proyecto. En esta tarea, se analizan los escenarios desplegados en cada caso, las pruebas realizadas, los parámetros medidos…
3. **Estudio de alternativas:** En esta tarea, se hace un estudio de las diferentes alternativas existentes para la adquisición de datos del bloque de monitorización y el análisis de rendimiento.

Selección de los middlewares: En esta fase, en base a los objetivos previamente definidos, se escogen aquellos middlewares que van a ser comparados.

1. **Estudio del estado del arte. Middlewares MQTT:** En esta tarea se hace una búsqueda de los middlewares MQTT más extendidos en la actualidad.
2. **Definición de los aspectos críticos:** En esta tarea, en base a los objetivos previamente definidos, se definen las situaciones menos deseables, los aspectos críticos y las funciones que han de implementar los middlewares para evitar dichas situaciones. Todas aquellas soluciones que no implementen una de las funciones no serán objeto de análisis posteriormente.

Diseño y puesta en marcha del Escenario I: En esta fase, se realiza el diseño y configuración de un escenario IoT que implemente un middleware MQTT RabbitMQ. Este escenario se diseña y configura con el objetivo de poder realizar un análisis de rendimiento posterior. Esta fase está formada por las siguientes tareas:

1. **Preparación del entorno de trabajo**: En esta tarea, se instalan las herramientas necesarias para poder desplegar de forma correcta el escenario IoT implementando el middleware MQTT RabbitMQ. En concreto, se instala la plataforma Docker y se descarga del repositorio de DockerHub el software correspondiente para el despliegue de un middleware RabbitMQ mediante un contenedor de Docker. Por otro lado, se descarga el entorno de desarrollo Eclipse y se instala el software Java para la implementación de los clientes. A su vez, se hace uso de la librería en Java Eclipse Paho Client que permite la implementación del protocolo MQTT en los diferentes clientes.
2. **Estudio de las herramientas a utilizar**: En esta tarea, se estudia, por un lado, las diferentes posibilidades que ofrece tanto la plataforma Docker como el software correspondiente al middleware RabbitMQ para su óptima configuración. Por otro lado,

se hace un estudio de las características que ofrece tanto el software Java como la librería Eclipse Paho Client para el desarrollo de clientes que hagan uso del protocolo de mensajería MQTT.

3. **Diseño de red**: En esta tarea, se diseña el escenario IoT a partir del cual se realizan las pruebas y medidas correspondientes al análisis de rendimiento. En concreto, se define el comportamiento tanto del middleware RabbitMQ como el de los diferentes clientes MQTT.
4. **Configuración de los equipos de la red**: En esta tarea, se configuran los diferentes equipos de la red para que sigan el comportamiento esperado. Por un lado, en el middleware RabbitMQ se configuran aquellos servicios o funciones que sean necesarios implementar y, por otro lado, en los clientes MQTT se configuran los parámetros necesarios para que implemente el protocolo de mensajería MQTT según lo previsto.

Diseño y puesta en marcha del Escenario II: En esta fase, se realiza el diseño y configuración de un escenario IoT que implemente un middleware MQTT EMQ. Este escenario se diseña y configura del mismo modo que el escenario anterior con el objetivo de poder sacar conclusiones a la hora de realizar una comparativa de los resultados obtenidos en los análisis de rendimiento. Esta fase está formada por las siguientes tareas:

1. **Preparación del entorno de desarrollo:** En esta tarea se descarga del repositorio de DockerHub el software correspondiente necesario para el despliegue de un middleware EMQ mediante un contenedor de Docker. Se hace uso también de la plataforma Docker, del entorno de desarrollo Eclipse, del software Java y de la librería Eclipse Paho Client previamente ya instaladas.
2. **Estudio de las herramientas a utilizar:** En esta tarea, la herramienta adicional utilizada es el software necesario para el despliegue del middleware EMQ. Por lo tanto, en esta tarea, se hace un estudio de las diferentes posibilidades que este nos ofrece a la hora de configurar el comportamiento del middleware EMQ.
3. **Diseño de la red:** En esta tarea, se diseña el escenario IoT a partir del cual se realizan las pruebas y medidas correspondientes al análisis de rendimiento. Con el objetivo de que puedan extraerse conclusiones de los resultados obtenidos en el este escenario y el anterior se define un comportamiento del middleware EMQ y de los clientes MQTT similar a la del middleware RabbitMQ
4. **Configuración de los equipos de red:** En esta tarea se realiza la configuración del middleware EMQ para que siga el comportamiento esperado. Los clientes MQTT utilizados en el escenario anterior se reutilizarán para este.

Análisis de rendimiento: En esta fase, se definen y realizan las pruebas y medidas de rendimiento en cada uno de los escenarios IoT desplegados. Esta fase está constituida por las siguientes tareas:

1. **Diseño de las pruebas a realizar:** En esta tarea, se realiza un estudio de las diferentes situaciones de interés que pueden presentarse. En base a dichas situaciones, se definen una serie de pruebas a realizar con el objetivo de poder posteriormente mediante medidas conocer el comportamiento de cada uno de los escenarios ante dichas situaciones.
2. **Diseño de las medidas a realizar:** Una vez definidas las pruebas a realizar en los diferentes escenarios, en esta fase, se hace un estudio de las posibles medidas a realizar. En base a estas y a las especificaciones del proyecto, se escogerán aquellas que nos

permitan posteriormente mediante una comparativa de resultados extraer las conclusiones más relevantes.

3. **Realización de las pruebas de rendimiento en el escenario IoT I:** En esta tarea, se realizan las pruebas y medidas definidas en las fases anteriores sobre el escenario que implementa el middleware MQTT RabbitMQ
4. **Realización de las pruebas de rendimiento en el escenario IoT II:** En esta tarea, se realizan las pruebas y medidas definidas en las fases anteriores sobre el escenario que implementa el middleware MQTT EMQ.
5. **Análisis de los resultados:** En base a las pruebas y medidas realizadas con anterioridad, se realiza una comparativa con el objetivo de conocer la solución que se adecua en mayor medida a las necesidades del proyecto.

Comprobación de la viabilidad de la solución adoptada: En esta fase, se realiza el despliegue de la solución adoptada para verificar que es una solución viable en el escenario definido en este proyecto.

1. **Despliegue de la solución adoptada:** Se configura y despliega la solución adoptada de tal modo que implemente las funciones que garantizan el cumplimiento de los objetivos.
2. **Realización de las pruebas de rendimiento:** Se realizan las pruebas y medidas definidas sobre el escenario donde esta implementada la solución adoptada.
3. **Análisis de los resultados:** En base a los resultados obtenidos de las pruebas y medidas, se verifica que la solución es viable en el contexto definido.

Gestión del proyecto: Esta fase consiste en el seguimiento del proyecto desde su puesta en marcha hasta el cierre, así como su documentación. Tiene la misma duración del proyecto y está formada por las siguientes tareas:

1. **Puesta en marcha del proyecto:** En esta tarea, se realiza una planificación del proyecto.
2. **Seguimiento del proyecto**: En esta tarea se realiza una verificación de la correcta realización de las diferentes tareas a realizar en tiempo y coste.
3. **Cierre del proyecto**: En esta tarea, se verifica que se han cumplido los objetivos del proyecto en tiempo y coste establecidos.
4. **Documentación:** En esta tarea, se realiza la memoria final del trabajo de fin de máster en base al proyecto realizado.

Además de las diferentes fases y tareas, se han definido una serie de hitos:

1. **Inicio del proyecto:** Se establecen los objetivos y el alcance del proyecto.
2. **Selección de los middlewares a comparar:** Se definen los middlewares que cumplen con las funcionalidades exigidas y van a ser objeto de análisis.
3. **Demostración del correcto funcionamiento del escenario IoT I:** Se verifica que tanto el escenario como los diferentes equipos que lo componen tienen el comportamiento esperado.
4. **Demostración del correcto funcionamiento del escenario IoT II:** Se verifica que tanto el escenario como los diferentes equipos que lo componen tienen el comportamiento esperado.
5. **Fin del análisis de rendimiento:** Este hito corresponde con el fin de la realización de las pruebas y del análisis de resultados.
6. **Demostración del correcto funcionamiento del escenario IoT con la solución adoptada:** Se verifica que el escenario desplegado con la solución final adoptada tiene el comportamiento esperado y cumple con las especificaciones del proyecto.

7. **Fin del proyecto:** Hito correspondiente al fin del proyecto.

8.1. Diagrama de Gantt

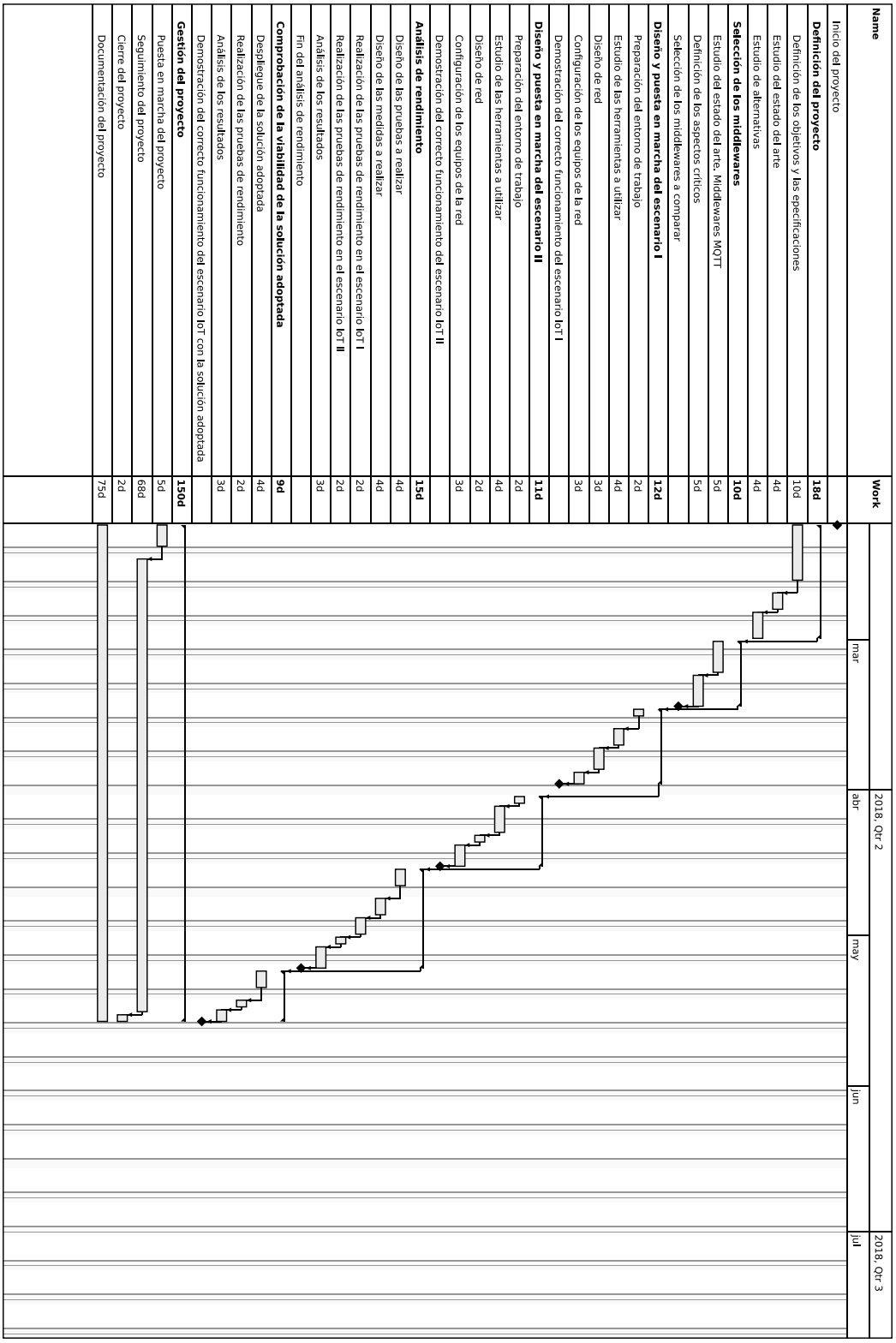

Ilustración 24: Diagrama de Gantt

9. Coste del trabajo

A continuación, se presenta el coste total del proyecto dividido en las diferentes partidas que lo componen: Horas internas, amortizaciones, gastos, subcontrataciones y costes indirectos.

9.1. Horas internas

La realización de este proyecto ha sido llevada a cabo por un Ingeniero Senior y un Ingeniero Junior. La tasa horaria de cada uno de ellos queda desglosada en la siguiente tabla:

Nombre	Responsabilidad	Tasa horaria
Marivi Higuero Aperribai	Ingeniera Senior (directora del proyecto)	50€/h
Gonzalo Gil Inchaurza	Ingeniero Junior	20€/h

Tabla 11: Tasas horarias de los RRHH

Las horas de trabajo dedicadas por cada uno de los trabajadores a los diferentes paquetes de trabajo quedan especificadas en las siguientes tablas:

Paquete de trabajo	Responsable	Número de horas
Definición del proyecto	Ingeniero Junior	120h
Selección de los middlewares	Ingeniero Junior	64h
Diseño y puesta en marcha del escenario IoT I	Ingeniero Junior	80h
Diseño y puesta en marcha del escenario IoT II	Ingeniero Junior	64h
Análisis de rendimiento	Ingeniero Junior	96h
Comprobación de la viabilidad de la solución adoptada	Ingeniero Junior	56h
Documentación del proyecto	Ingeniero Junior	120h
SUBTOTAL		**600h**

Tabla 12: Horas internas del Ingeniero Junior

Paquete de trabajo	Responsable	Número de horas
Supervisión	Ingeniero Senior	150h
SUBTOTAL		**150h**

Tabla 13: Horas internas del Ingeniero Senior

En función de las horas definidas, el coste total derivado de las horas internas asociado al proyecto es el siguiente:

Nombre	Responsabilidad	Tasa horaria	Número de horas	Coste
Marivi Higuero Aperribai	Ingeniera Senior (directora del proyecto)	50€/h	150h	7.500 €
Gonzalo Gil Inchaurza	Ingeniero Junior	20€/h	600h	12.000 €
SUBTOTAL				**19.500 €**

Tabla 14: Subtotal de las horas internas

9.2. Amortizaciones

Las amortizaciones correspondientes al proyecto son el equipo con el que se ha llevado a cabo el despliegue de los escenarios IoT y las pruebas de rendimiento y el software necesario para la documentación del proyecto. A continuación, se desglosa el coste asociado a cada uno de ellos:

Concepto	Valor inicial	Valor residual	Vida útil	Tasa horaria	Tiempo de utilización	Coste
Ordenador	600 €	100 €	6 años	6,94 €/mes	6 meses	41,64 €
Microsoft Office 365 Hogar	99 €	0 €	1 año	8,25 €/mes	3 meses	24,75 €
SUBTOTAL						**66,39 €**

Tabla 15: Subtotal de amortizaciones

9.3. Gastos

Los gastos atribuibles a este proyecto se desglosan en la siguiente tabla:

Concepto	Coste
Material de oficina	50 €
Electricidad	20 €
Conexión a Internet	80 €
SUBTOTAL	**150 €**

Tabla 16: Subtotal de gastos

9.4. Subcontrataciones

Este proyecto no requiere de la subcontratación de terceros para su realización. Por ello, los costes asociados a subcontrataciones en este proyecto son de 0 €.

9.5. Coste total del proyecto

El cálculo del coste total del proyecto se realiza mediante la suma de los costes de las partidas de horas internas, amortizaciones, subcontrataciones, gastos y costes indirectos.

En este proyecto, los costes indirectos se han definido como un 0% de la suma de costes del resto de partidas.

A continuación, se desglosan los costes asociados a las diferentes partidas, así como el coste total del proyecto.

Concepto	Subtotal
Horas internas	19.500 €
Amortizaciones	66,39 €
Subcontrataciones	0 €
Gastos	150 €
Costes indirectos (0%)	0 €
TOTAL	**19.716,39 €**

Tabla 17: Coste total del proyecto

10. Conclusiones

Por un lado, se han conseguido alcanzar los objetivos definidos para este Trabajo de Fin de Máster en plazos y según los costes previstos.

Por otro lado, para el caso de uso dado, se ha visto que los middlewares EMQ y RabbitMQ son unas alternativas sencillas de desplegar, escalables, fiables y seguras.

En lo que a rendimiento se refiere, primeramente, se ha observado que el middleware EMQ bajo mismas circunstancias ofrece un mayor throughput frente al middleware RabbitMQ. En consecuencia, el middleware EMQ permitirá, la publicación y entrega de mensajes en un tiempo menor. De este modo, esta solución ofrece mayor garantías en escenarios que requieren del procesamiento de información en tiempo real.

Posteriormente, en ambos middlewares se han observado unos valores de latencia similares y buenos para ofrecer un servicio en tiempo real.

Por otro lado, se ha observado una correlación entre la implementación de mecanismos de control de acceso y la reducción del throughput en ambos casos. Cabe destacar que en ambos casos el rendimiento en términos de throughput no se ve afectado de la misma manera. Mientras que en el middleware EMQ el throughput apenas se ve afectado, en el middleware RabbitMQ el throughput se ve reducido en un 78%.

Sin embargo, se ha visto que no hay repercusión en ninguno de los casos en la latencia cuando se implementa un mecanismo de autenticación.

Por último, la implementación de conexiones SSL seguras no tiene repercusión ni en términos de throughput ni de latencia para ninguno de los casos. Se ha de destacar que los recursos disponibles son superiores a los que pueda haber en un escenario real sobre todo en el lado de los clientes MQTT. Por ello, en un escenario donde estos últimos presentan limitaciones en cuanto a capacidades computacionales se refiere, la implementación de conexiones SSL seguras puede llegar a tener una repercusión relevante en el rendimiento que ha de tenerse en cuenta.

Como conclusión, se puede decir que el middleware EMQ permite la adquisición de una cantidad elevada de datos, de forma sencilla, escalable, fiable y segura con un rendimiento superior al del resto de middlewares MQTT. Por tanto, esta solución es idónea para aquellos escenarios donde se necesita explotar una gran cantidad de datos en tiempo real o en el menor tiempo posible.

Bibliografía

Origen de Internet. https://es.wikipedia.org/wiki/ARPANET

Statista. Number of connected devices. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

IoT. https://es.wikipedia.org/wiki/Internet_de_las_cosas

Industria 4.0. https://es.wikipedia.org/wiki/Industria_4.0

MQTT. https://mqtt.org/

Gartner. *Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020.* https://www.gartner.com/newsroom/id/2636073

Hogartec. Hogares inteligentes en EE. UU y Europa. http://hogartec.es/hogartec2/ya-hay-18-millones-de-hogares-inteligentes-en-eeuu-y-europa/

Ecointeligencia. Smart cities. https://www.ecointeligencia.com/2013/07/que-servicios-ofrece-una-smart-city-a-sus-ciudadanos-1/

PwC. Industria 4.0. https://informes.pwc.es/industria40/assets/industry4.0-building-your-digital-enterprise-april-2016.pdf

Sistema distribuido.

ScalAgent. Benchmark of MQTT servers. http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

P. Dobbelaere and K. Sheykh Esmaili. (2017). Kafka versus RabbitMQ. http://www-inf.telecom-sudparis.eu/COURS/CILS-IAAIO/Articles/debs_kafka_versus_rabbitmq.pdf

Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP. https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html

XMPP. https://en.wikipedia.org/wiki/XMPP

AMQP. https://es.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

HiveMQ. https://www.hivemq.com/

RabbitMQ. https://www.rabbitmq.com/

VerneMQ. https://vernemq.com/

EMQ. http://emqtt.io/

Mosquitto. https://mosquitto.org/

ANEXOS

ANEXO I: Diagramas de flujo

Mensajes de control intercambiados para garantizar un nivel de QoS dado

A continuación, se muestran los mensajes intercambiados entre los clientes MQTT (publicador y suscriptor) y el MQTT Message Broker en función del nivel de QoS definido.

En todos los casos, los mensajes intercambiados corresponden con un establecimiento de conexión inicial por parte de ambos clientes MQTT con el MQTT Message Broker, la publicación de un único mensaje en el MQTT Message Broker y su posterior recepción en el suscriptor.

QoS 0

Los puertos 54006 y 54016 corresponden con el suscriptor y publicador respectivamente.

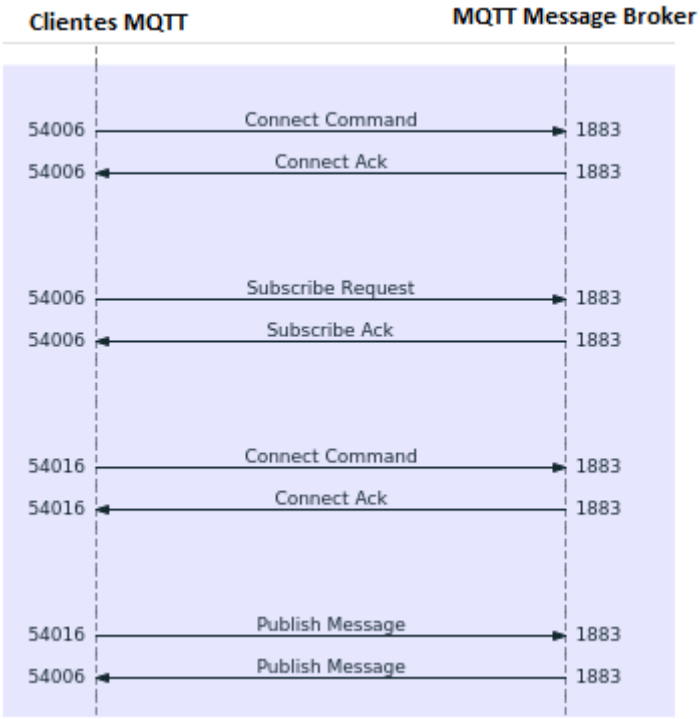

Ilustración 25: Diagrama de flujo. Qos 0

QoS 1

Los puertos 54056 y 54060 corresponden con el suscriptor y el publicador respectivamente.

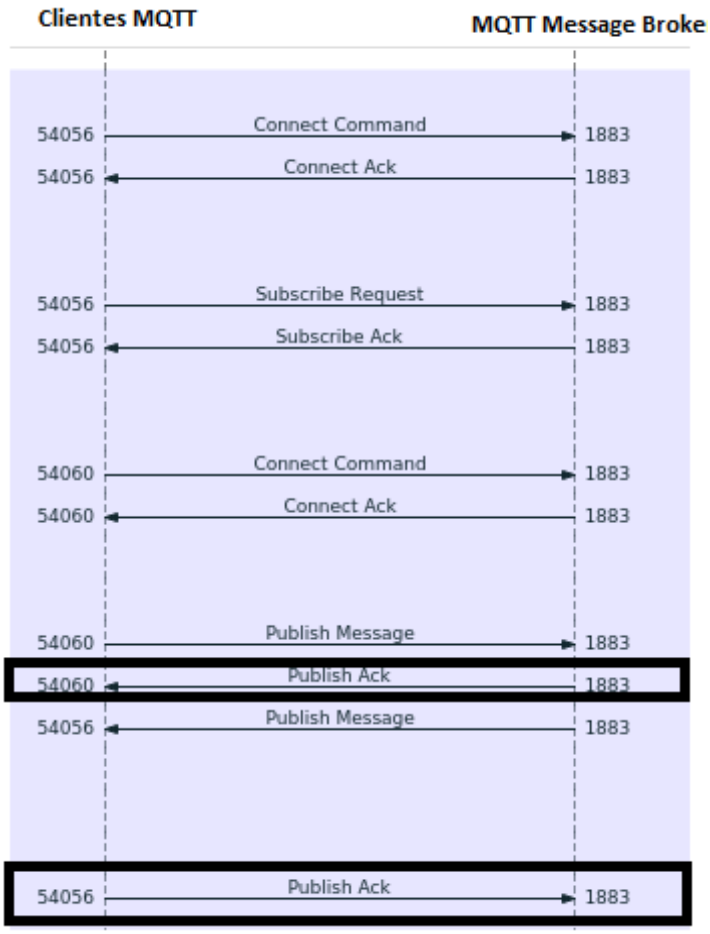

Ilustración 26: Diagrama de flujo. QoS 1

QoS 2

Los puertos 54080 y 54084 corresponden con el suscriptor y publicador respectivamente.

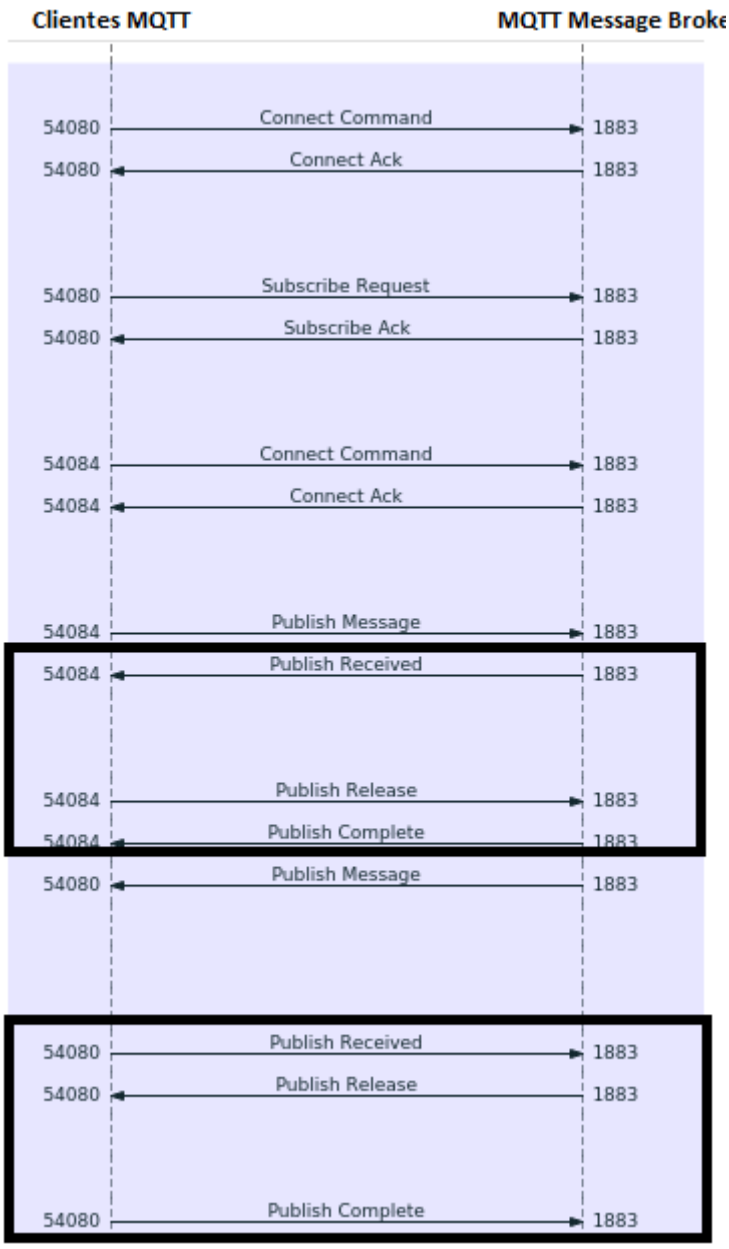

Ilustración 27:Diagrama de flujo. QoS 2

Mensajes de control LDAP

A continuación, se muestra un ejemplo con el intercambio de mensajes realizado entre el MQTT Message Broker y el servidor LDAP para comprobar las credenciales presentadas por dos clientes MQTT (suscriptor y publicador respectivamente) en el establecimiento de la conexión con el MQTT Message Broker.

Corresponde con el caso anterior donde las publicaciones se realizan con QoS 2.

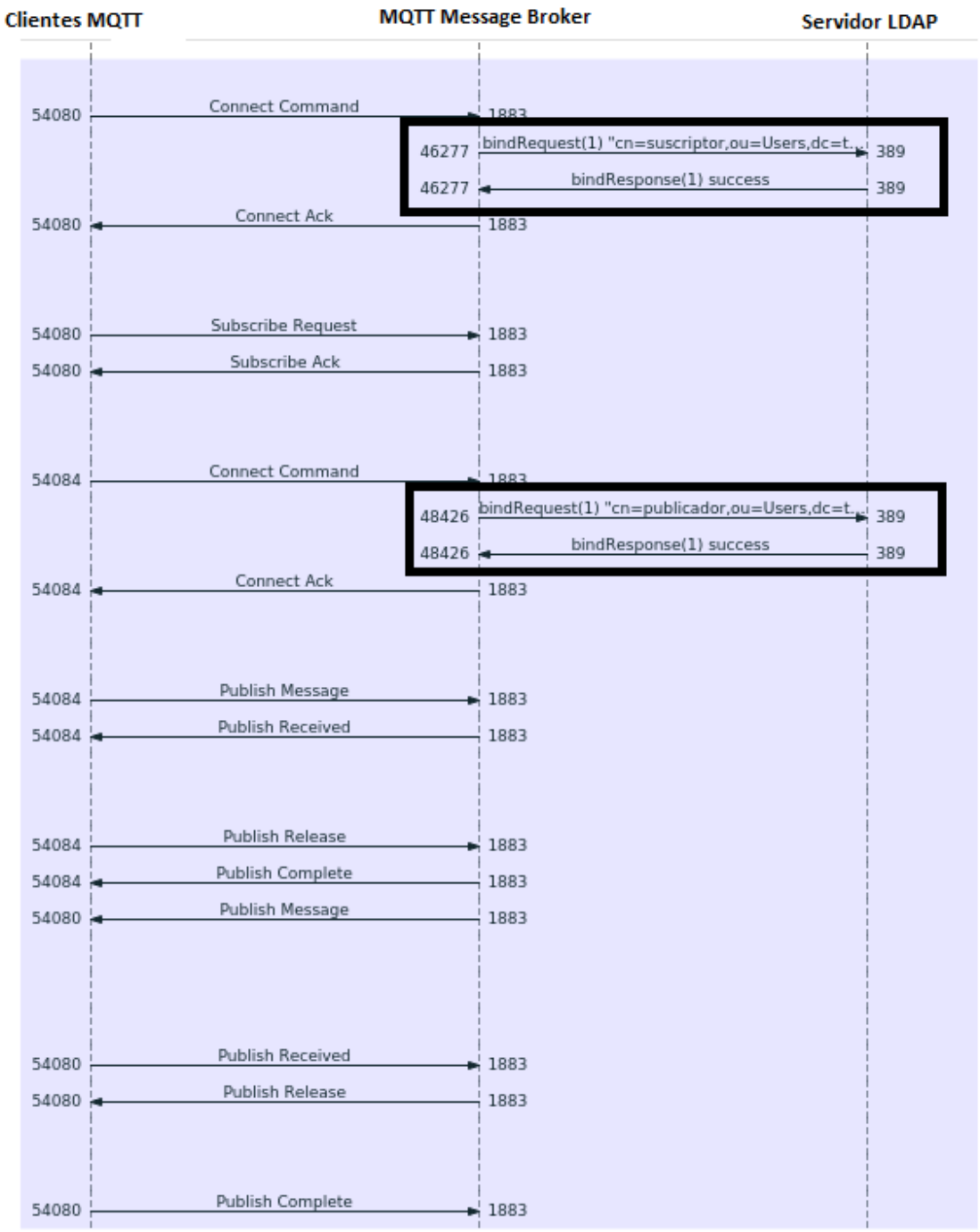

Ilustración 28: Diagrama de flujo. Mensajes de control LDAP

SSL Handshake

A continuación, se muestra el intercambio de mensajes realizado para el intercambio de claves entre un cliente MQTT y el MQTT Message Broker previo al establecimiento de una conexión SSL segura cifrada. Tanto el establecimiento de la conexión como el posterior intercambio de mensajes (de aplicación y control) se realizará a través de conexiones SSL seguras cifradas.

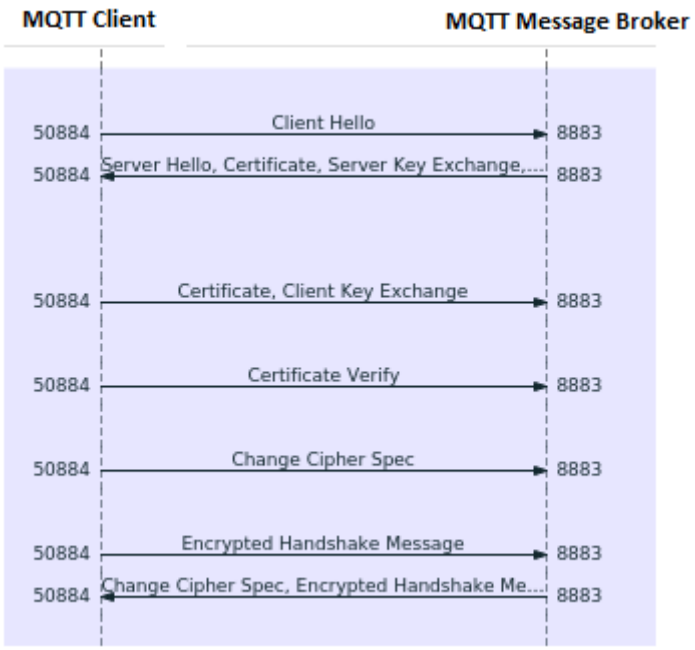

Ilustración 29: SSL Handshake

Anexo II: Ficheros de configuración

Docker

A continuación, se desglosan los ficheros de configuración docker-compose utilizados para el despliegue de ambos middlewares mediante de la herramienta Docker. En ellos, caben destacar los siguientes parámetros:

Image	Permite definir la imagen de Docker que se va a utilizar para el despliegue del servicio. Una imagen podría definirse como una plantilla de un servicio con una configuración dada. Puede estar descargada en el repositorio local o no. En este último caso, en caso de existir, se descarga automáticamente del repositorio Docker Hub.
Volumes	Docker utiliza virtualizaciones para el despliegue de los servicios. Este campo, permite el mapeo de ficheros del host a las virtualizaciones. Normalmente, los ficheros de configuración de los servicios son almacenados en el host y mapeados a través de este campo al directorio correspondiente de la virtualización. De este modo, si se cae el servicio, este puede ser restaurado con la configuración más actualizada. Por ejemplo, en el caso de EMQ, los ficheros de configuración son almacenados en el directorio **/home/ubuntu/Desktop/TFM/MQTTMessageBrokers/emq/configuration** del host y son mapeados al directorio donde se encuentran los ficheros de configuración del EMQ en la virtualización: **/opt/emqttd/etc**

Ports	Permite mapear los puertos de las virtualizaciones con los del host. De este modo, los servicios expuestos en las virtualizaciones son accesibles a través del host desde fuera. Por ejemplo, en el caso de EMQ, en el puerto 8883 de la virtualización está expuesto el servicio del bróker, pero no es accesible desde fuera del host. A través de este parámetro, podemos permitir la accesibilidad desde fuera al servicio a través del puerto 8883 del propio host.
Environment	Permite la configuración de parámetros del servicio.
Links	Permite el establecimiento de enlaces entre diferentes virtualizaciones desplegadas en una misma máquina.

Tabla 18: Parámetros de configuración del fichero docker-compose.yml

EMQ

Docker-compose.yml

```
version: '2.0'

services:

 emq:

    restart: always

    image: machinedata/emq

    hostname: emq

    container_name: emq

    volumes:

      -
/home/ubuntu/Desktop/TFM/MQTTMessageBrokers/emq/configuration:/opt/emqttd/etc

    ports:

      - "8883:8883"

      - "18083:18083"

    environment:

      -
EMQ_LOADED_PLUGINS="emq_auth_ldap,emq_recon,emq_modules,emq_retainer,emq_dashboard"

 openldap:

   restart: always

   image: osixia/openldap

   hostname: openldap

   container_name: openldap
```

volumes:

- /home/ubuntu/Desktop/TFM/openldap/slapd.d:/etc/ldap

- /home/ubuntu/Desktop/TFM/openldap/ldap:/var/lib/ldap

- /home/ubuntu/Desktop/TFM/openldap/container:/container

ports:

- "389:389"

phpLDAPadmin:

image: osixia/phpldapadmin

hostname: phpLDAPadmin

container_name: phpLDAPadmin

environment:

- PHPLDAPADMIN_LDAP_HOSTS=openldap

- PHPLDAPADMIN_HTTPS=false

links:

- openldap:serverldap

ports:

- "0.0.0.0:8081:80"

RabbitMQ

Docker-compose.yml

version: "2"

services:

rabbitmq:

restart: always

image: rabbitmq

hostname: rabbitmq

container_name: rabbitmq

volumes:

-
/home/ubuntu/Desktop/TFM/MQTTMessageBrokers/rabbitmq/configuration:/etc/rabbitmq #config files: /etc/rabbitmq/rabbitmq.conf /etc/rabbitmq/advanced.config

- /var/lib/rabbit:/var/lib/rabbit #home dir: /var/lib/rabbitmq, database dir: /var/lib/rabbitmq/mnesia/rabbitmq

-
/home/ubuntu/Desktop/TFM/Certificados/MQTTMessageBroker:/opt/rabbitmq/certificados #Certificado RabbitMQ

- /home/ubuntu/Desktop/TFM/Certificados/CA/cacert.pem:/opt/CA/cacert.pem #Certificado de la CA

ports:

- "8883:8883" #Puerto para conexiones MQTT over SSL

- "15672:15672" #Puerto para conexiones a la UI de management

environment:

- RABBITMQ_NODENAME=rabbitmq #nodo rabbitmq_rabbitmq_1

openldap:

restart: always

image: osixia/openldap

hostname: openldap

container_name: openldap

volumes:

- /home/ubuntu/Desktop/TFM/openldap/slapd.d:/etc/ldap

- /home/ubuntu/Desktop/TFM/openldap/ldap:/var/lib/ldap

- /home/ubuntu/Desktop/TFM/openldap/container:/container

ports:

- "389:389"

phpLDAPadmin:

image: osixia/phpldapadmin

hostname: phpLDAPadmin

container_name: phpLDAPadmin

environment:

- PHPLDAPADMIN_LDAP_HOSTS=openldap

- PHPLDAPADMIN_HTTPS=false

links:

- openldap:serverldap

ports:

- "0.0.0.0:8081:80"

Middlewares

A continuación, se desglosan los ficheros más relevantes a la hora de configurar ambos middlewares.

EMQ

A continuación, se desglosa el fichero de configuración del middleware EMQ. En el caben destacar las siguientes líneas:

log.error.file = log/error.log	Permite definir el directorio de almacenamiento del fichero de logs. Muy útil para la monitorización del middleware.
mqtt.allow_anonymous = false	Permite la restricción de acceso a usuarios autenticados.
mqtt.listener.ssl = 8883	Permite definir el puerto donde se quiere desplegar el servicio para conexiones SSL seguras.
mqtt.listener.ssl.tls_versions = tlsv1.2,tlsv1.1,tlsv1 **mqtt.listener.ssl.handshake_timeout = 15s** **mqtt.listener.ssl.keyfile = etc/certs/key.pem** **mqtt.listener.ssl.certfile = etc/certs/cert.pem** **mqtt.listener.ssl.cacertfile = etc/certs/cacert.pem** **mqtt.listener.ssl.verify = verify_peer** **mqtt.listener.ssl.fail_if_no_peer_cert = true**	Permite realizar la configuración de un conjunto de parámetros relacionados con las conexiones SSL. Por ejemplo, el directorio de almacenamiento de los certificados y la clave
mqtt.plugins.etc_dir =etc/plugins/	Permite definir el directorio de almacenamiento de los ficheros asociados a los distintos plugins. Por ejemplo, para la implementación de un mecanismo de control de acceso a los recursos vía servidor LDAP.

Tabla 19: Parámetros de configuración del fichero emq.conf

Emq.conf

```
#====================================================================

## EMQ Configuration R2.1
```

```
##============================================================
##------------------------------------------------------------
## Node Args
##------------------------------------------------------------
## Node name
node.name = emq@172.18.0.4
## Cookie for distributed node
node.cookie = emq_dist_cookie
## SMP support: enable, auto, disable
node.smp = auto
## vm.args: -heart
## Heartbeat monitoring of an Erlang runtime system
## Value should be 'on' or comment the line
## node.heartbeat = on
## Enable kernel poll
node.kernel_poll = on
## async thread pool
node.async_threads = 32
## Erlang Process Limit
node.process_limit = 2097152
## Sets the maximum number of simultaneously existing ports for this system
node.max_ports = 1048576
## Set the distribution buffer busy limit (dist_buf_busy_limit)
node.dist_buffer_size = 32MB
## Max ETS Tables.
## Note that mnesia and SSL will create temporary ets tables.
node.max_ets_tables = 2097152
## Tweak GC to run more often
node.fullsweep_after = 1000
## Crash dump
node.crash_dump = log/crash.dump
```

Distributed node ticktime

node.dist_net_ticktime = 60

Distributed node port range

node.dist_listen_min = 6369

node.dist_listen_max = 6369

##--

Log

##--

Set the log dir

log.dir = log

Console log. Enum: off, file, console, both

log.console = console

Syslog. Enum: on, off

log.syslog = on

syslog level. Enum: debug, info, notice, warning, error, critical, alert, emergency

log.syslog.level = error

Console log level. Enum: debug, info, notice, warning, error, critical, alert, emergency

log.console.level = error

Console log file

log.console.file = log/console.log

Error log file

log.error.file = log/error.log

Enable the crash log. Enum: on, off

log.crash = on

log.crash.file = log/crash.log

##--

Allow Anonymous and Default ACL

##--

Allow Anonymous authentication

mqtt.allow_anonymous = false

Default ACL File

```
mqtt.acl_file = etc/acl.conf

## Cache ACL for PUBLISH

mqtt.cache_acl = true

##--------------------------------------------------------------------

## MQTT Protocol

##--------------------------------------------------------------------

## Max ClientId Length Allowed.

mqtt.max_clientid_len = 10240

## Max Packet Size Allowed, 64K by default.

mqtt.max_packet_size = 64KB

##--------------------------------------------------------------------

## MQTT Connection

##--------------------------------------------------------------------

## Force GC: integer. Value 0 disabled the Force GC.

mqtt.conn.force_gc_count = 0

##--------------------------------------------------------------------

## MQTT Client

##--------------------------------------------------------------------

## Client Idle Timeout (Second)

mqtt.client.idle_timeout = 50s

## Enable client Stats: on | off

mqtt.client.enable_stats = on

##--------------------------------------------------------------------

## MQTT Session

##--------------------------------------------------------------------

## Upgrade QoS?

mqtt.session.upgrade_qos = off

## Max Size of the Inflight Window for QoS1 and QoS2 messages

## 0 means no limit

mqtt.session.max_inflight = 0

## Retry Interval for redelivering QoS1/2 messages.
```

```
mqtt.session.retry_interval = 20s

## Client -> Broker: Max Packets Awaiting PUBREL, 0 means no limit

mqtt.session.max_awaiting_rel = 0

## Awaiting PUBREL Timeout

mqtt.session.await_rel_timeout = 20s

## Enable Statistics: on | off

mqtt.session.enable_stats = on

## Expired after 1 day:

## w - week

## d - day

## h - hour

## m - minute

## s - second

mqtt.session.expiry_interval = 2h

##--------------------------------------------------------------------

## MQTT Queue

##--------------------------------------------------------------------

## Type: simple | priority

mqtt.queue.type = simple

## Topic Priority: 0~255, Default is 0

## mqtt.queue.priority = topic/1=10,topic/2=8

## Max queue length. Enqueued messages when persistent client disconnected,

## or inflight window is full.

mqtt.queue.max_length = infinity

## Low-water mark of queued messages

mqtt.queue.low_watermark = 0%

## High-water mark of queued messages

mqtt.queue.high_watermark = 100%

## Queue Qos0 messages?

mqtt.queue.qos0 = true

##--------------------------------------------------------------------
```

MQTT Broker and PubSub

##--

System Interval of publishing broker $SYS Messages

mqtt.broker.sys_interval = 60

PubSub Pool Size. Default should be scheduler numbers.

mqtt.pubsub.pool_size = 500

mqtt.pubsub.by_clientid = true

Subscribe Asynchronously

mqtt.pubsub.async = true

##--

MQTT Bridge

##--

Bridge Queue Size

mqtt.bridge.max_queue_len = 1000000

Ping Interval of bridge node. Unit: Second

mqtt.bridge.ping_down_interval = 10

##--

MQTT Plugins

##--

Dir of plugins' config

mqtt.plugins.etc_dir =etc/plugins/

File to store loaded plugin names.

mqtt.plugins.loaded_file = data/loaded_plugins

##--

MQTT Listeners

##--

TCP Listener: 1883, 127.0.0.1:1883, ::1:1883

mqtt.listener.tcp = 1883

Size of acceptor pool

mqtt.listener.tcp.acceptors = 64

Maximum number of concurrent clients

mqtt.listener.tcp.max_clients = 1000000

Rate Limit. Format is 'burst,rate', Unit is KB/Sec

mqtt.listener.tcp.rate_limit = 100,10

TCP Socket Options

mqtt.listener.tcp.backlog = 1024

mqtt.listener.tcp.recbuf = 4096

mqtt.listener.tcp.sndbuf = 4096

mqtt.listener.tcp.buffer = 4096000

mqtt.listener.tcp.nodelay = true

SSL Listener: 8883, 127.0.0.1:8883, ::1:8883

mqtt.listener.ssl = 8883

Size of acceptor pool

mqtt.listener.ssl.acceptors = 32

Maximum number of concurrent clients

mqtt.listener.ssl.max_clients = 500000

Rate Limit. Format is 'burst,rate', Unit is KB/Sec

mqtt.listener.ssl.rate_limit = 100,10

Configuring SSL Options. See http://erlang.org/doc/man/ssl.html

TLS only for POODLE attack

mqtt.listener.ssl.tls_versions = tlsv1.2,tlsv1.1,tlsv1

mqtt.listener.ssl.handshake_timeout = 15s

mqtt.listener.ssl.keyfile = etc/certs/key.pem

mqtt.listener.ssl.certfile = etc/certs/cert.pem

mqtt.listener.ssl.cacertfile = etc/certs/cacert.pem

mqtt.listener.ssl.verify = verify_peer

mqtt.listener.ssl.fail_if_no_peer_cert = true

HTTP and WebSocket Listener

mqtt.listener.http = 8083

mqtt.listener.http.acceptors = 16

mqtt.listener.http.max_clients = 250000

HTTP(SSL) Listener

mqtt.listener.https = 8084

mqtt.listener.https.acceptors = 4

mqtt.listener.https.max_clients = 64

mqtt.listener.https.handshake_timeout = 15

mqtt.listener.https.keyfile = etc/certs/key.pem

mqtt.listener.https.certfile = etc/certs/cert.pem

mqtt.listener.https.cacertfile = etc/certs/cacert.pem

mqtt.listener.https.verify = verify_peer

mqtt.listener.https.fail_if_no_peer_cert = true

##--

System Monitor

##--

Long GC, don't monitor in production mode for:

https://github.com/erlang/otp/blob/feb45017da36be78d4c5784d758ede619fa7bfd3/erts/emulator/beam/erl_gc.c#L421

sysmon.long_gc = false

Long Schedule(ms)

sysmon.long_schedule = 240

8M words. 32MB on 32-bit VM, 64MB on 64-bit VM.

sysmon.large_heap = 8MB

Busy Port

sysmon.busy_port = false

Busy Dist Port

sysmon.busy_dist_port = true

Emq_auth_ldap.conf

A continuación, se desglosa el fichero de configuración asociado al plugin que permite la implementación del mecanismo de control de acceso a los recursos via servidor LDAP.

##--

LDAP Auth Plugin

##--

auth.ldap.servers = openldap

auth.ldap.port = 389

auth.ldap.timeout = 30

auth.ldap.user_dn = cn=%u,ou=Users,dc=tekniker,dc=es

auth.ldap.ssl = false

RabbitMQ

A continuación, se desglosa el fichero de configuración del rabbitMQ. En él, caben destacar los siguientes parámetros:

Rabbitmq_management	Permite definir parámetros de configuración de la interfaz de usuario de gestión. Por ejemplo, el puerto donde se expone.
Auth_backends	Permite definir el backend utilizado para la implementación del mecanismo de control de acceso a los recursos.
Ssl_options	Permite la configuración de diferentes parámetros asociados al establecimiento de conexiones SSL seguras. Por ejemplo, el directorio donde se almacenan los certificados y la clave
Rabbitmq_mqtt	Permite la configuración de diferentes parámetros del protocolo MQTT. Por ejemplo, el puerto utilizado para el despliegue del servicio vía SSL.
Rabbitmq_auth_backend_ldap	Permite la configuración de diferentes parámetros del backend utilizado para la implementación del mecanismo de control de acceso. Por ejemplo, las reglas de acceso a los diferentes recursos del middleware

Tabla 20: Parámetros de configuración del fichero rabbitmq.conf

Advanced.config

{rabbit, [{hipe_compile,false},

{**rabbitmq_management**, [{listener, [{**port, 15672**}]}]},

{**auth_backends**, [**rabbit_auth_backend_ldap**]},

{**ssl_options**, **[{cacertfile, "/opt/CA/cacert.pem"},**

{certfile, "/opt/rabbitmq/certificados/cert.pem"},

{keyfile, "/opt/rabbitmq/certificados/key.pem"},

```
            {verify,     verify_peer},
            {fail_if_no_peer_cert, false}]}]},
 {rabbitmq_mqtt, [{vhost,          <<"/">>},
            {exchange,       <<"amq.topic">>},
            {subscription_ttl, undefined},
            {prefetch,       10},
            {retained_message_store, rabbit_mqtt_retained_msg_store_dets},
            {tcp_listeners,    [1883]},
            {ssl_listeners, [8883]}]},
 {rabbitmq_auth_backend_ldap,
  [ {servers,            ["openldap"]},
    {user_dn_pattern,      "cn=${username},ou=Users,dc=tekniker,dc=es"},
    {use_ssl,            false},
    {port,               389},
    {log,               true},
    {log,               network},
    {vhost_access_query,    {exists,
                        "cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}},
    {resource_access_query,
     {for, [{permission, configure, {exists,"ou=Groups,dc=tekniker,dc=es"}},
          {permission, write,
           {for, [{resource, queue,    {exists,
"cn=Publishers,cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}},
                  {resource, exchange, {exists,
"cn=Publishers,cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}}]}},
          {permission, read,
           {for, [{resource, exchange, {exists,
"cn=Subscribers,cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}},
                  {resource, queue,    {exists,
"cn=Subscribers,cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}}]}}
         ]
     }},
```

```
    {topic_access_query,

     {for, [{permission, write, {exists,
"cn=Publishers,cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}},

          {permission, read, {exists,
"cn=Subscribers,cn=MqttClients,ou=Groups,dc=tekniker,dc=es"}}

                              ]}},

    {tag_queries,          [{administrator, {exists, "ou=Groups,dc=tekniker,dc=es"}},

                      {management,    {exists, "ou=Groups,dc=tekniker,dc=es"}}]}

  ]

 }

].
```

Enabled.plugins

A continuación, se desglosa la configuración necesaria para la implementación de los plugins que permiten la implementación del mecanismo de control de acceso a través de un servidor LDAP, el despliegue de la interfaz de usuario de gestión y el protocolo MQTT para el intercambio de mensajes respectivamente.

```
[rabbitmq_auth_backend_ldap,rabbitmq_management,rabbitmq_mqtt].
```