Evaluación Tecnológica de Modelos de Comunicación para la plataforma IoT Smart Campus UIS

Diego Fernando Gonzalez Ortiz y Cristian Alejandro Rengifo Mejía

Trabajo de Grado para Optar por el Título de Ingeniero de Sistemas

Director
Gabriel Rodrigo Pedraza Ferreira
PhD. Ciencias de la Computación

Codirector
Henry Andrés Jiménez Herrera
Ingeniero de Sistemas

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería de Sistemas e Informática
Bucaramanga
2023

Dedicatoria

“A mis padres por darme todo su amor y apoyo durante este proceso académico.

A mis abuelos por ese amor y ayuda que me brindaron para que pudiera culminar mis estudios.

A mis hermanos, que me aconsejaron en los momentos en que no quería seguir y me ayudaron a mantenerme firme en mis decisiones.

A los compañeros y amigos que hice en mi proceso y que siempre voy a recordar como grandes personas.

A mis directores de proyecto, el profesor Gabriel y el profesor Henry quienes siempre nos guiaron de la mejor manera durante el desarrollo de este proyecto.**”**

Diego Fernando Gonzalez Ortiz

Dedicatoria

“Dedicado a todo el apoyo recibido por mi familia, amigos y profesores.

Mis padres que nunca dejaron de darme su apoyo y amor.

A mis hermanos, sobre todo mi hermano mayor que siempre estuvo guiando a cómo debía enfocar mis capacidades y que ha sido referente a lo que quiero llegar a ser en la vida.

A los amigos que hice durante el transcurso de la carrera, gente maravillosa que tiene sus propias metas y que deseo apoyarlos al igual que ellos a mí.

A mis directores de proyecto, el Profesor Gabriel que nos dio está oportunidad de trabajar con él y al Profesor Henry al que siempre acudimos en las dudas y nos daba esa solución que buscábamos.**”**

Christian Alejandro Rengifo Mejía

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Lista de Siglas

AMQP: Advanced Message Queuing Protocol

COAP: Constrained Application Protocol

XMPP: Extensible Messaging and Presence Protocol

MQTT: Message Queue Telemetry Transport

P2P: Pear to Pear

M2M: Machine to Machine

IoT: Internet of Things

JSON: JavaScript Object Notation

REST: Representational State Transfer

HTTP: Hypertext Transfer Protocol

TCP: Transmission Control Protocol

SSL: Secure Socket Layers

QoS: Quality of Service

CLI: Command Line Interface

VPN: Virtual Private Network

VM: Virtual Machine

PUB/SUB: Publisher Subscriber

REQ-RES: Request Response

Resumen

Título: Evaluación Tecnológica de Modelos de Comunicación para la Plataforma IoT Smart Campus UIS.*

Autores: Diego Fernando González Ortiz, Cristian Alejandro Rengifo Mejía.**

Palabras Clave: Internet de las Cosas, Campus Inteligente, Modelos de Comunicación, Protocolos de Comunicación

Descripción: Durante el diseño e implementación de una arquitectura IoT se suelen enfrentar retos a la hora de establecer e implementar modelos de comunicación IoT de manera adecuada, en el cual recae la transmisión de la información entre los diferentes componentes de una arquitectura IoT de la mejor forma posible. Por esta razón es importante conocer los diferentes modelos de comunicación y herramientas para la distribución de la información, y así tener una arquitectura adecuada que brinde las capacidades necesarias para el sistema IoT a implementar.

El objetivo de este proyecto es estudiar y comparar diferentes modelos de comunicación usando múltiples protocolos de comunicación tomando como base de estudio una plataforma IoT Smart Campus, evaluando las ventajas y desventajas que presentan implementarlos en diferentes partes de la arquitectura IoT. En este proyecto se estudiaron los modelos de comunicación que constan con una implementan de tres protocolos, MQTT, AMQP y Apache Kafka, los cuales son muy usados en IoT y como resultado se determinó que su funcionamiento es más adecuado en partes específicas de la arquitectura IoT.

Este proyecto es un aporte realizado en trabajos anteriores que elaboraron una Infraestructura IoT Smart Campus, sobre la implementación y uso de los protocolos de comunicación en un Smart Campus.

---

*Trabajo de grado.

**Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira. PhD, Ciencias de la Computación. Gabriel Rodrigo Pedraza Ferreira, PhD, Ciencias de la Computación.

Abstract

Title: Technological Evaluation of Communication Models for the IoT Smart Campus UIS.*

Authors: Diego Fernando González Ortiz, Cristian Alejandro Rengifo Mejía.**

Keywords: IoT, Smart Campus, Communication Models, Communication Protocols

Description: During the design and implementation of an IoT architecture, challenges will be faced when it comes to establishing and implementing IoT communication models in an adequate way, in which the transmission of information between the different components of an IoT architecture falls in the best possible way. For this reason, it is important to know the different communication models and tools for the distribution of information, and thus have an adequate architecture that provides the necessary capabilities for the IoT system to be implemented.

The objective of this project is to study and compare different communication models using multiple communication protocols based on an IoT Smart Campus platform, evaluating the advantages and disadvantages of different parts of the IoT architecture. In this project, the communication models that consist of an implementation of three protocols, MQTT, AMQP and Apache Kafka, which are widely used in IoT and as a result, it is prolonged that their operation is more appropriate in specific parts of the IoT architecture were studied.

This project is a contribution made in previous works that developed an IoT Smart Campus Infrastructure, on the implementation and use of communication protocols in a Smart Campus.

---

*Degree Work.

**Faculty of Physic mechanical Engineering. School of Systems Engineering and Computer Science. Director: Gabriel Rodrigo Pedraza Ferreira, Doctor of Computer Science. Co-director: Henry Andrés Jiménez Herrera, Systems Engineer.

Introducción

Desde las últimas décadas se ha masificado el uso de dispositivos que disponen de conexión a internet, esto se debe en gran parte, al gran salto que tuvo la definición y el constante estudio del concepto de IoT (Internet of Things), el cual describe la forma en que los sistemas de los dispositivos se comunican con el mundo real, mediante sensores que captan estos cambios y los envían como datos. Con esta definición, nacieron otros conceptos que hacen uso del IoT, tales como las Smart Cities y Smart Campus, zonas físicas que captan el estado a tiempo real de dicha zona mediante los sensores, que envían estos datos para ser tratados y analizados por otros sistemas y tomar acciones ante las directivas programadas.

Dichos espacios inteligentes, requieren de un número elevado de sensores, esto incrementa la cantidad de dispositivos conectados simultáneamente, y se requiere de una distribución de los datos de manera adecuada, por eso, se da la necesidad de utilizar una solución que garantice que los mensajes sean enviados de forma adecuada. En la actualidad existen diferentes modelos de comunicación que aplican patrones de mensajería, pero cada uno funciona con unas bases diferentes. El patrón de mensajería más usado para los Smart Campus son los Patrones Publicador-Subscriptor (Publish-Subscribe – Pub/Sub), que describe como los mensajes son enviados constantemente, sin que ambas partes se conozcan, esto facilita el consumo energético que puede llegar a tener un Smart Campus con cientos o miles de dispositivos recolectando datos.

Durante la elaboración de un arquitectura para un Smart Campus, la conexión de entre las diferentes partes de esta conforman una parte esencial durante el desarrollo, las partes más fundamentales que conforman dicha arquitectura son el Backend, el Frontend y los componentes

para la comunicación y que para la elaboración, estos no necesariamente se deban desarrollar en los mismos lenguajes o el conocer partes ajenas a su funcionamiento, esta es la función a cumplir por los modelos de comunicación, que dan las reglas de cómo se debe establecer la conexión entre las partes de la arquitectura sin afectar el rendimiento o interferir durante el funcionamiento de la misma. Pero no todo modelo de comunicación se adecua de manera ideal entre partes de la arquitectura, no es igual la comunicación entre un dispositivo de bajos recursos que toma datos del entorno y los envía hacia un Backend ubicado en un servidor para ser analizados, o un mensaje que viaja entre el Backend que están ubicados en diferentes servidores físicos, en estos casos, un modelo de comunicación funciona idealmente en un caso, pero en el otro puede funcionar no de la manera esperada o de forma errónea, lo que puede causar perdida en los datos o que estos nunca lleguen al destino.

Estas problemáticas se reflejan durante el planeamiento de la arquitectura, donde se determina como funcionaran las partes de esta y como se deben comunicar, por eso, en este proyecto se busca evaluar algunos de los protocolos de comunicación con el patrón Pub/Sub que son activamente usados en los Smart Campus y establecer una serie de diferencias entre ellos además de definir sus fortalezas y debilidades en cada parte de la arquitectura en las que se analizaran y de solventar dudas del funcionamiento de los protocolos de comunicación que se surgen durante la ejecución que no son visibles inicialmente, con el fin, de ayudar a la selección de modelos de comunicación que sirvan en el desarrollo de aplicaciones Smart Campus o similares.

1. Objetivos

1.1. Objetivo General

Realizar una evaluación tecnológica de un conjunto de modelos de comunicación para identificar cómo se adecuan a las necesidades de la plataforma de IoT Smart Campus UIS.

1.2. Objetivos Específicos

- Identificar las necesidades de una infraestructura IoT en términos de comunicación de datos.
- Establecer un marco de evaluación (conjunto de criterios) para evaluar los modelos de comunicación que pueden ser utilizados.
- Seleccionar e implementar un par de modelos de comunicación existentes en el estado del arte en la infraestructura IoT existente.
- Evaluar los modelos implementados utilizando el marco de evaluación definido.

2. Estado del Arte

La comunicación de protocolos en IoT es un área de investigación en constante evolución, con una amplia gama de tecnologías y protocolos que se utilizan para la transferencia de datos entre dispositivos IoT. La elección del protocolo adecuado es crucial para garantizar una comunicación eficiente y segura entre los dispositivos y para permitir una gestión eficaz de los datos generados por el IoT.

A pesar del amplio estudio, se encuentra poca información al respecto cuando se involucra un Smart Campus con modelos de comunicación IoT, tal como explican en el artículo Benchmarking Pub/Sub IoT middleware platforms for smart services (Carlos Pereira, João Cardoso, Ana Aguiar, Ricardo Morla) no hay un estudio sistematizado a la hora de usar con tal de encontrar una comparación de rendimiento, ventajas y desventajas en aplicaciones hechas por middlewares que usen tecnologías IoT, como lo brokers de comunicación.

Comparative Analysis of IoT Communication Protocols (Burak H. Çorak, Feyza Y. Okay, Metehan Güzel, Şahin Murt, Suat Ozdemir, 2018) es un articuló en el que inicialmente se recompila una serie de análisis previos relacionados con protocolos de comunicación IoT con los cuales se realiza una tabla resumen para presentar rápidamente el objetivo y resultados de dicho proyecto. Adicionalmente realizan un análisis en una arquitectura propia para comparar algunos parámetros en la transmisión de datos sobre los protocolos CoAP, MQTT, XMPP.

Se tiene el caso de un estudio que creó su propio bróker para comunicaciones P2P y lo comparó con los resultados contra MQTT presentados por Froiz-Míguez, Fraga-Lamas & Fernández-Caramés (2020). El estudio habla de cómo con los avances actuales, los sistemas descentralizados

han tenido mejoras en la implementación en la computación en el borde y se centran en cómo implementan comunicaciones M2M basadas en Peer-to-Peer (P2P), usando dispositivos de bajo consumo. También se habla de cómo usan el protocolo Pub/Sub en comunicaciones M2M y para finalmente concluir que su bróker da buenos tiempos de respuesta, pero no tan buenos en comparación a MQTT, debido a que este trabaja con bajas latencias y que el protocolo Pub/Sub no está pensado para comunicaciones M2M. Entonces con esto en mente se puede tomar una referencia de hacia dónde se dirige este proyecto al comparar diferentes brokers en un mismo entorno y de cómo se deben hacer las comunicaciones para obtener resultados.

En resumen, la elección del protocolo adecuado en la comunicación de protocolos en IoT es crucial para garantizar una comunicación eficiente, segura y escalable entre los dispositivos IoT. La literatura actual ofrece una amplia gama de protocolos de comunicación y protocolos de seguridad que pueden ser utilizados en diferentes escenarios de IoT, pero se encuentra muy poca información al respecto sobre diferencias y características entre los protocolos de comunicación o brokers, que permitan su selección a la hora de elaborar una aplicación que implemente tecnologías de comunicación IoT. Por lo tanto, es importante realizar una comparación detallada y una evaluación rigurosa de estos protocolos para seleccionar el protocolo adecuado en función de las necesidades específicas de la aplicación de IoT.

3. Marco de Referencia

3.1 Internet of Things

El Internet de las cosas (IoT, por sus siglas en inglés) se define como "una red de objetos físicos, dispositivos, vehículos, edificios y otros elementos que están conectados a través de Internet y pueden intercambiar datos" (Gubbi, Buyya, Marusic, & Palaniswami, 2013, p. 2). En otras palabras, el IoT permite la conexión y comunicación de objetos y dispositivos cotidianos a Internet y a otros dispositivos conectados, lo que permite la creación de sistemas inteligentes y automatizados que pueden recopilar datos, analizarlos y utilizarlos para mejorar la eficiencia, la productividad y la toma de decisiones en diferentes ámbitos.

El IoT tiene muchos usos y aplicaciones en diferentes sectores, entre los cuales se destacan:

- **Hogar inteligente:** dispositivos conectados en el hogar, como termostatos, sistemas de iluminación, electrodomésticos y sistemas de seguridad, que se pueden controlar a través de una aplicación móvil o por voz.
- **Salud y bienestar**: sensores y dispositivos que monitorean la salud y el bienestar de las personas, como relojes inteligentes, monitores de actividad física, medidores de glucosa y dispositivos de seguimiento remoto de pacientes.
- **Ciudades inteligentes:** sensores y dispositivos conectados que ayudan a optimizar la gestión de los recursos y servicios públicos, como el transporte, la energía, el agua y el medio ambiente.
- **Industria:** sensores y dispositivos conectados que permiten la automatización y el control de los procesos de producción, la gestión de la cadena de suministro y la monitorización de la maquinaria y los activos.

- **Agricultura inteligente:** sensores y dispositivos que permiten el monitoreo del clima, la calidad del suelo y la salud de las plantas, lo que ayuda a optimizar el rendimiento y la eficiencia en la producción agrícola.

3.2 Smart Campus

Un Smart Campus es una zona física y digital originada inicialmente para los campus universitarios que se originó de las Smart Cities en década de los 2000, ambos términos son similares, la diferencia está en la aplicación de ambos, las Smart Cities están fuertemente ligadas a las ciudades y como mejorar la calidad de vida de los ciudadanos, mientras que, un Smart Campus esta más enfocado en campus universitarios o zonas residenciales que utilizan las mismas tecnologías.

Con el uso de las tecnologías de comunicación IoT, un Smart Campus es capaz de captar información de las aulas, oficinas o espacios abiertos y ser capaz de determinar niveles óptimos de funcionamiento, reduciendo costos energéticos al ajustarse según los criterios de los administradores o el sistema diseñado, también sirve para la mejorar la seguridad de la zona con una forma inteligente de monitoreo, todo esto con tal de mejorar la calidad de aprendizaje.

3.3 Protocolos de Comunicación en IoT

En el amplio estudio del Internet de las Cosas, los protocolos de comunicación son los estándares a los que se rige una aplicación para el envío de mensajes entre partes del sistema y que se asegura de que estos lleguen al destino final de manera correcta. Los protocolos de comunicación tienen múltiples formas de implementarse y usarse, todo depende del uso y las

condiciones a las que se usara, esto implica que algunos protocolos de comunicación son mejores en ciertas tareas que otro.

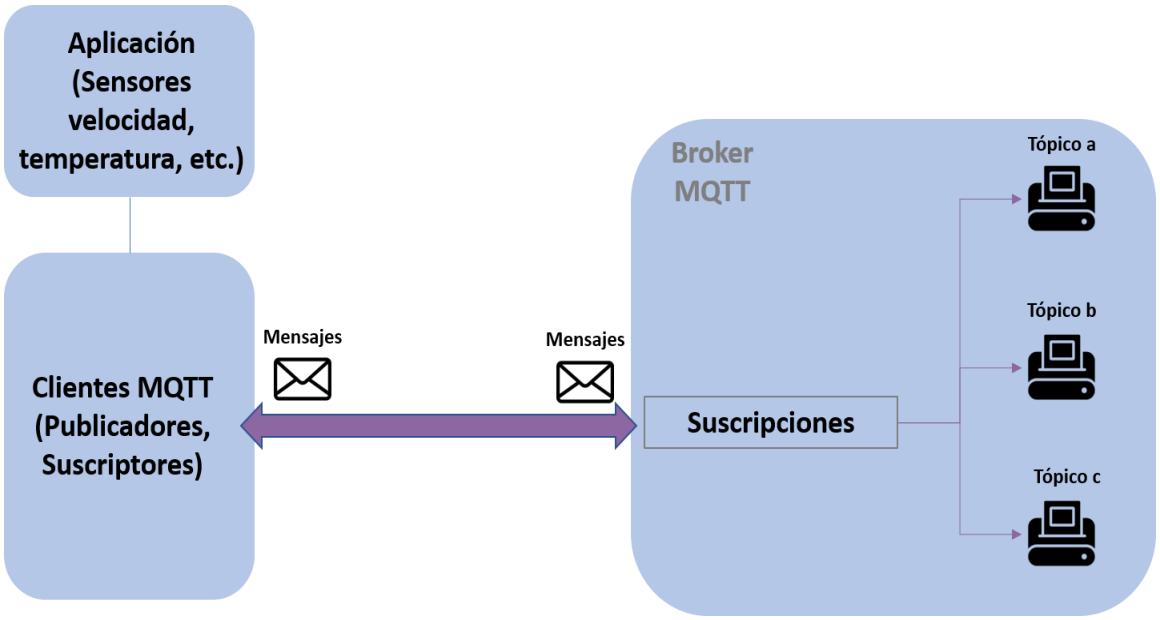
Cuando se pasa a un caso de estudio sobre los Smart Campus un protocolo de mensajería que usen el patrón Request-Response no es el más recomendado, en este caso, los protocolos de comunicación recomendados para el caso son aquellos que implementan un patrón de mensajes Publish-Subscribe, donde los clientes no conocen o están conectados al servidor, si no que un intermediario llamado Broker redirige los mensajes enviados a un tópico hacia el destino que los quiere escuchar. Ante este tipo de protocolos, se dispone de múltiples soluciones que aportan diferentes cambios para hacerlos atractivos según las condiciones requeridas, ejemplo, cuando una aplicación requiere conexiones rápidas y fiables se pueden utilizar protocolos como AMQP y JMS. Cuando la aplicación requiere recolección de datos en redes restringidas se puede usar MQTT y COAP. (Naik, 2017). Para este estudio, se revisó el estado del arte de los protocolos de comunicación más usados en IoT son:

- **HTTP (Hyper Text Transport Protocol):** Según Al-Masri, Kalyanam, Batts, Kim, Singh, Vo, & Yan (2020) y Naik (2017), este es el protocolo de mensajería web predominante. Es un protocolo de petición/respuesta en el cual el cliente envía un mensaje de petición y el host genera un mensaje de respuesta. HTTP usa TCP como protocolo de transporte y TLS/SSL por seguridad.
- **CoAP (Constrained Application Protocol):** Es un protocolo de transferencia de web enfocado para dispositivos en redes restringidas. Este protocolo, tal y como presentan Al-Masri, Kalyanam, Batts, Kim, Singh, Vo, & Yan (2020), fue diseñado para aplicaciones M2M. Similar a HTTP, CoAP utiliza el protocolo petición/respuesta y utiliza conceptos de

la web como identificadores URIs y tipos de medios. Su funcionamiento consta de un Publicador que comparte información con unas URIs específicas y el Suscriptor se subscribe a un recurso específico mediante una URI. Cuando el Publicador comparte datos a la URI, todos los suscriptores que están suscritos a esa URI son notificados de esos nuevos datos compartidos mediante la URI.

- **MQTT (Message Queuing Telemetry Transport Protocol):** Es uno de los protocolos de comunicación más antiguos M2M, fue introducido por primera vez en 1999 y desarrollado por Andy Stanford-Clark de IBM y Arlen Nipper de Arcom Control Systems Ltd, según presenta Naik (2017). Es un protocolo de mensajería Publicador/Suscriptor diseñado para comunicaciones M2M de bajo peso en redes restringidas. El publicador, según Soni & Makwana (2017), comparte datos a un tópico específico, estos datos pasan por un broker de mensajería el cual se encarga de distribuir esos datos a los suscriptores que tengan una suscripción activa a ese tópico en específico.

Figura 1. *Arquitectura del Protocolo MQTT*



Nota. Elaboración propia

- **AMQP (Advanced Message Queuing Protocol):** Es un protocolo de mensajería de bajo peso, pero extensible diseñado para comunicaciones M2M. AMQP es usado generalmente en ambientes corporativos y está enfocado en sistemas interoperables. Es un protocolo que soporta un amplio rango de aplicaciones de mensajería y patrones de comunicación. Al igual que MQTT, AMQP usa el protocolo de mensajería Publicador/Suscriptor además de brindar enrutamiento flexible y transacciones de negocio.

3.4 Brokers de Mensajería

Dependiendo del sistema IoT que se quiera implementar, varía la información que los dispositivos Smart o sensores transmiten para llegar a diferentes usuarios. Por esto, es complicado establecer conexiones con protocolos M2M para cada usuario que requiera acceder a estos datos, es por esto que en las arquitecturas IoT se utilizan broker de mensajería los cuales actúan de

intermediarios validando, transformando y enrutando los datos a los usuarios que los requieran. Ante los protocolos de comunicación, se escogieron los brokers mas usados que los implementan:

- **Mosquitto:** Es un broker de mensajería de código abierto desarrollado por Eclipse y que implementa el protocolo de comunicación MQTT. Es un broker diseñado para usarse desde dispositivos de bajo consumo hasta servidores. Utiliza la metodología pub/sub para recibir y transmitir datos. Mosquitto es muy usado en la implementación de IoT debido a que puede ser usado por dispositivos de bajo consumo como sensores y dispositivos remotos que transmiten información, además de su fácil implementación.
- **RabbitMQ:** Al igual que Mosquitto, RabbitMQ es un broker de mensajería de código abierto pero que implementa el protocolo de comunicación AMQP. Su funcionamiento se basa en implementar colas que almacenan la información compartida por los productores hasta ser consumida. A diferencia de Mosquitto, RabbitMQ posee ventajas como el complejo enrutamiento de mensajes, pero también posee desventajas frente a este como su no tan sencilla implementación en el cliente.
- **Mosca:** Es un bróker de mensajería basado en el protocolo de comunicación MQTT para el desarrollo de infraestructuras IoT. Su desarrollo está basado en JavaScript por lo que es de fácil implementación y uso. Su fuerza esta en el manejo de la característica de QoS de MQTT, siendo QoS0 y QoS1 las que mejor implementa; almacena paquetes fuera de líneas para QoS1. Según Amaguaya (2020), es utilizable en aplicativos Node.js lo que aporta una comunicación en servicios web fuerte. Una de sus principales desventajas es que posee limitaciones en cuanto a escalabilidad, pues requiere de otro agente para poder cumplir con esta característica.

- **ActiveMQ:** Snyder, Bosanac, & Davies (2017) exponen que es un JMS de código abierto orientado MOM de Apache. Provee estándares para la mensajería multilenguaje y multiplataforma con alta disponibilidad, escalabilidad, fiabilidad y seguridad para presentar los mensajes. En cuanto a su conectividad, ActiveMQ dispone de un amplio rango de opciones de conectividad, soportando protocolos como HTTP/S, SSL, Stomp, TCP, UDP, XMPP, entre otros. Dispone de un API para clientes disponible para diferentes lenguajes como C, C++, .NET, Perl, PHP, Python, Ruby, entre otros. Una característica importante de ActiveMQ es que diferentes brokers de ActiveMQ pueden trabajar de la mano para formar una red de brokers con fines de escalabilidad.

- **EMQX:** EMQX es un broker de código abierto basado en MQTT que innova en varios aspectos diferentes a sus competidores. Con un mayor enfoque en la infraestructura Cloud, la principal característica es su escalabilidad, que puede alcanzar hasta 100 millones de conexiones simultáneas por clúster IoT. Su gran fiabilidad en este aspecto lo hace una opción muy popular en servicios con grandes cantidades de datos, siendo mejor que Mosquitto en el mismo protocolo MQTT.

3.5 Apache Kafka

Apache Kafka es un servicio de mensajería open source basado en el sistema Pub/Sub, pero con el añadido de estar más enfocado en el tratamiento de data en tiempo real (Streaming Data). A diferencia de otros brokers, Kafka presta mismamente el entorno para recibir, enviar, tratar y almacenar los datos, esto en una estructura tipo clúster que permite unas mejoras respecto a otros brokers, tales como bajas latencias, resistencia a fallos, escalabilidad horizontal que lo hace ser

más tentador para aplicaciones de tipo Big Data, replicación de datos en todo el clúster, persistencia y más.

3.6 Docker

Docker es un servicio mediante el cual se pueden crear contenedores los cuales pueden ejecutar aplicaciones en entornos independientes y portables en los cuales se incluye todo lo necesario para ejecutar dichas aplicaciones. Docker está compuesto por varias componentes que trabajan juntos para permitir a los desarrolladores crear, distribuir y ejecutar aplicaciones en contenedores. Estas partes son:

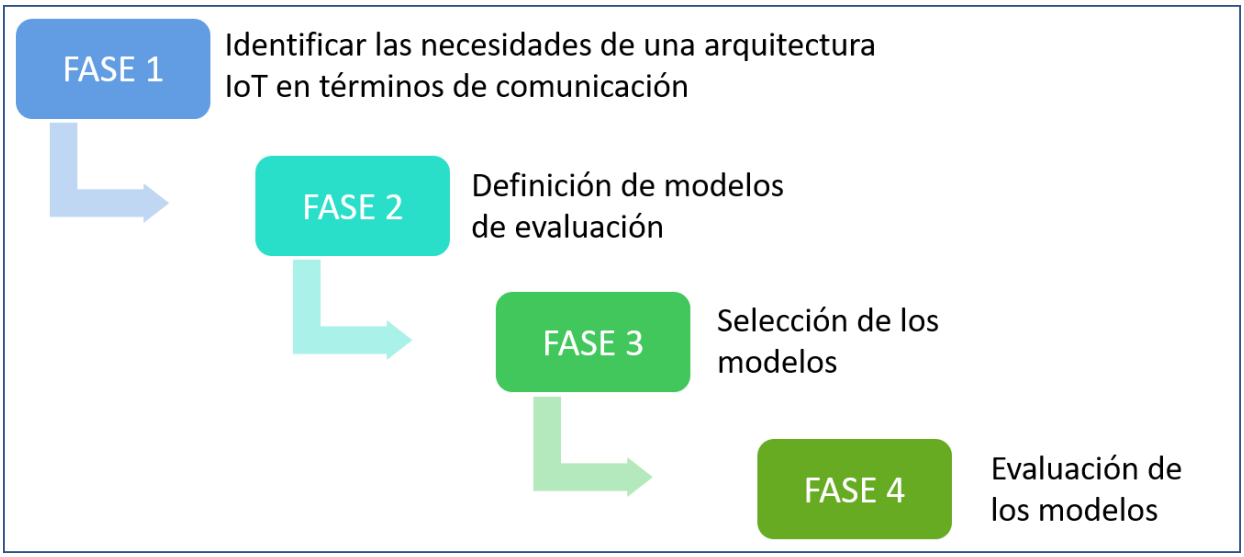
- **Docker Engine**: es el componente central de Docker y se encarga de la creación y ejecución de contenedores. Docker Engine administra los contenedores y una interfaz de línea de comandos (CLI) para que los usuarios puedan interactuar con Docker.
- **Docker Hub:** es un repositorio de imágenes de contenedores que se pueden utilizar para ejecutar aplicaciones. Docker Hub contiene una gran cantidad de imágenes públicas de contenedores que se pueden descargar y utilizar para ejecutar aplicaciones en contenedores.
- **Dockerfile:** es un archivo que contiene las instrucciones para construir una imagen de contenedor. Dockerfile se utiliza para definir el entorno de ejecución de la aplicación y sus dependencias.
- **Docker Compose:** es una herramienta que permite a los usuarios definir y ejecutar aplicaciones compuestas por múltiples contenedores. Docker Compose se utiliza para crear aplicaciones complejas que requieren varios contenedores que trabajan juntos.

3.7 Máquinas Virtuales en la Nube

Mediante las máquinas virtuales se pueden ejecutar distintos sistemas operativos y por ende programas y aplicaciones. Estas máquinas virtuales, como dicen Pahl, Brogi, Soldani, & Jamshidi (2017) se pueden cargar a un servidor para que se pueda acceder a ellas y ser controladas de manera remota desde otro dispositivo. DigitalOcean provee un servicio de contenedores virtuales llamados droplets que son instancias de máquinas virtuales en la nube con las cuales pueden ejecutar aplicaciones y otros servicios desde la nube.

4. Marco Metodológico

Figura 2. *Fases de Desarrollo del Proyecto*



Nota. Elaboración propia.

En la construcción del proyecto, se hace la necesidad de dividir el desarrollo en varias fases, con tal de explorar de forma más detallada cada aspecto del proyecto. Por esto se definió 4 fases en las que se irá construyendo el proyecto de forma lineal, replanteando las definiciones hechas en la fase anterior.

4.1. Identificar las Necesidades de una Infraestructura IoT en términos de comunicación

La fase 1 consta de formar una base de conocimiento acerca de la infraestructura IoT y definir una serie de necesidades que serán la base del desarrollo de las siguientes fases. En esta fase es necesario entender el funcionamiento de las tecnologías de comunicación IoT, una serie de búsquedas y selección de los protocolos de comunicación, frameworks y lenguajes que hay en el mercado y una aplicación de los mismo para aprender la forma en que son aplicados.

Para esta fase se tiene definidas las siguientes actividades:

- Investigación y definición de las necesidades del proyecto ante una infraestructura IoT.
- Selección de la tecnología IoT que se investigara.
- Entendimiento de las herramientas que hacen uso de las tecnologías de comunicación IoT.

4.2. Definición de Modelos de Evaluación

En el marco de evaluación se toman las necesidades definidas de la fase anterior con tal de construir una serie de criterios que son la base para la toma y análisis de resultados con el fin de llegar a una evaluación de los protocolos de comunicación. Estos criterios limitan el alcance al que apunta el proyecto.

Las actividades que se realizaron en esta fase son las siguientes:

- Definición de los criterios de evaluación.
- Explicación de los criterios.

4.3. Selección de los Modelos

Con los primeros acercamientos a la tecnología IoT de la fase 1, se seleccionaron los protocolos de comunicación, servicios y brokers que se adecuaban para el desarrollo del proyecto, esto para listar las tecnologías de comunicación IoT que serán usadas en la creación de experimentos y la toma de datos.

Las actividades que se realizaron en esta fase son las siguientes:

- Definir los filtros para la selección de las tecnologías de comunicación IoT.
- Determinar los mejores modelos de comunicación para las necesidades del proyecto.

4.4. Diseño Experimental

Con las definiciones de los modelos de comunicación y criterios de las fases anteriores, se elaboran las pruebas que se ejercerán sobre los entornos y protocolos de comunicación escogidos, estos siendo evaluados en base a los criterios para su posterior análisis. Las pruebas fueron segmentadas según el entorno al que se evaluara.

Las actividades que se realizaron en esta fase son las siguientes:

- Elaboración de un conjunto de pruebas.
- Configuración de las pruebas.

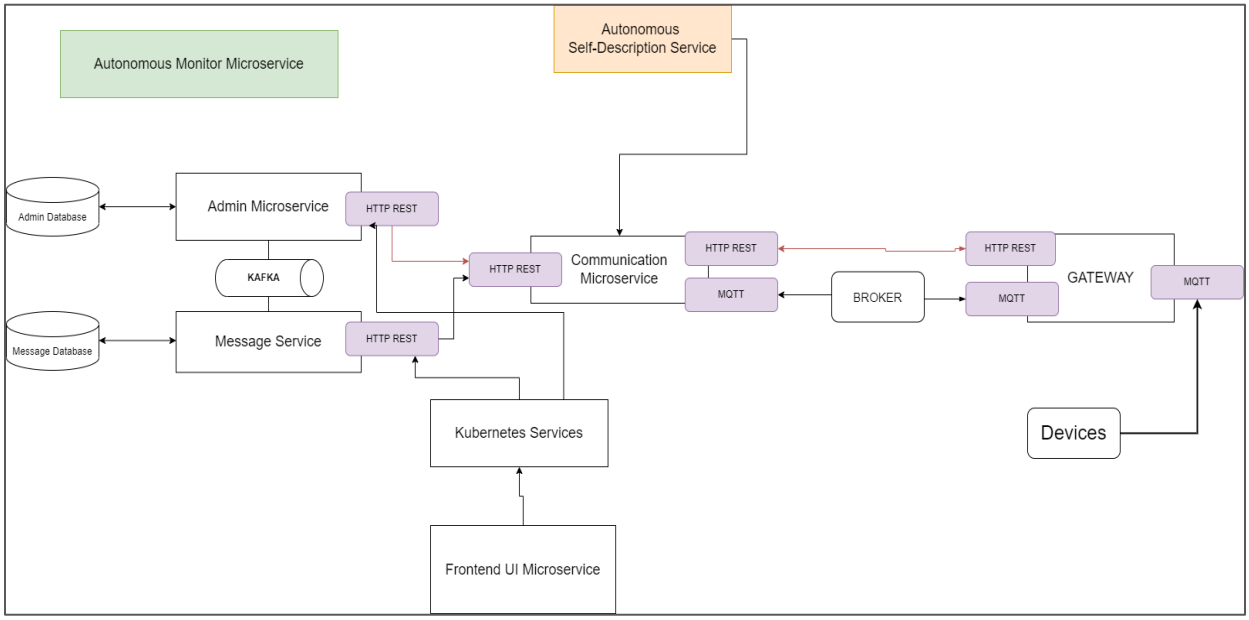
5. Marco Evaluativo

Durante el desarrollo de esta sección se define el planteamiento que conllevó la definición de necesidades, elaboración de los modelos de comunicación y construcción de las pruebas y algunos aspectos que se toman en cuenta en cada una de estas partes.

5.1. Arquitectura IoT de Referencia

Para el desarrollo del proyecto se tiene como base la arquitectura IoT Smart Campus. A continuación, se presenta la gráfica de la arquitectura IoT suministrada en la que se trabajó para encontrar las parte donde se puede implementar un modelo de comunicación.

Figura 3. Arquitectura IoT para Smart Campus



Nota. Propiedad de el Codirector del Proyecto.

5.2. Definición de las Necesidades

Antes de establecer algún criterio evaluativo se deben identificar las necesidades que presenta una arquitectura IoT en términos de comunicación de datos, por lo cual se hace uso del diagrama de una arquitectura IoT Smart Campus presentado anteriormente como referencia.

Al analizar la forma en que se transmiten los datos en esta arquitectura se puede identificar las partes donde se hace uso de la tecnología IoT. Este análisis permite plantear las siguientes observaciones respecto a los protocolos de comunicación IoT:

a. El uso más común de un protocolo de comunicación IoT está en la parte de los dispositivos externos al sistema que recolectan información y la envían al Backend para ser procesada. Una necesidad identificada en esta parte de la arquitectura trata sobre la velocidad en la que son enviados los datos.
b. En la comunicación dentro de los microservicios, el uso de un protocolo comunicación IoT no es la primera opción para tomar por los desarrolladores, siendo Res/Req la opción popular. Se plantea esta necesidad con tal de evaluar las propiedades de los protocolos y sus capacidades de envió de información.

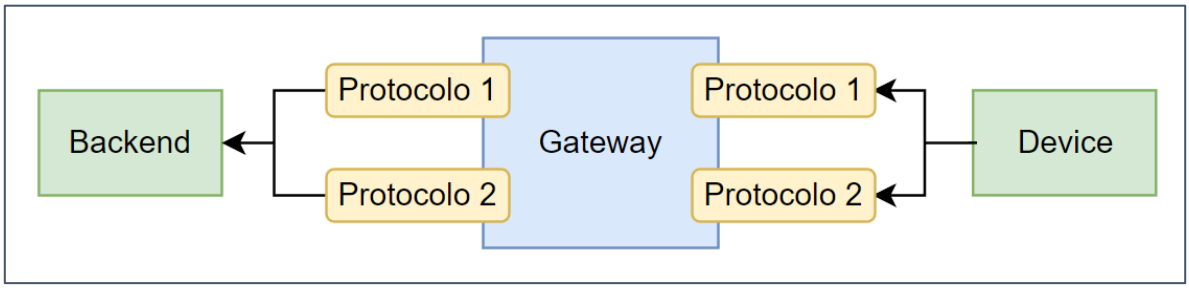
Con base a esta arquitectura se consideran muchos posibles casos de uso de los protocolos de comunicación IoT, pero el enfoque de este proyecto se centra en las siguientes dos necesidades. A continuación, se explica a detalle cada caso.

5.2.1. Comunicación Device to Backend

Los dispositivos o sensores de recolección de datos generalmente poseen recursos limitados en comparación a equipos de tratamiento de datos como computadoras o servidores, se evidencia la

necesidad de que el protocolo sea capaz de ejecutarse en dispositivos de bajos recursos y consumo y que no afecten a su rendimiento.

Figura 4. *Arquitectura de Comunicación Device to Backend*



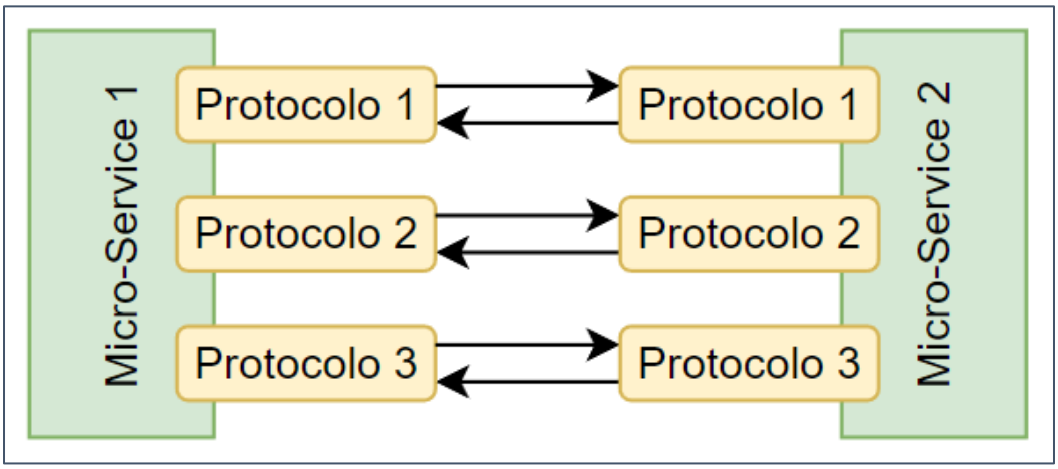
Nota. Elaboración Propia.

Las propiedades para tomar en cuenta en este primer caso, es la capacidad que dispone el broker para comunicar múltiples dispositivos a uno o muchos Backend, siendo esta la escalabilidad que tiene el broker. Como los dispositivos siempre están recolectando información la perdida de datos no es un inconveniente para tener en cuenta.

5.2.2. Comunicación Backend to Backend

La comunicación Backend to Backend resulta ser una necesidad dentro de la arquitectura, mientras que el caso anterior está basado en dispositivos externos con recursos limitados, en este se puede disponer de todos los recursos que se dispongan en la arquitectura, siendo más prioritario la fiabilidad en que los datos son enviados, más que el bajo consumo.

Figura 5. *Arquitectura de Comunicación Backend to Backend*



Nota. Elaboración Propia.

5.3. Definición de Modelos de Evaluación

Las necesidades abren paso para la elaboración de las modelos de comunicación que servirán para la construcción de las evaluaciones que se realizarán a los protocolos, la siguiente fase del desarrollo consiste en la selección de Criterios de la evaluación y la construcción de modelos de comunicación de los brokers.

5.3.1. Criterios de Evaluación

Los criterios de evaluación hacen relación con los objetivos de las evaluaciones de los protocolos, siendo estos los que marcan la dirección a la que enforcar las pruebas que se realizaran a los casos descritos en las necesidades y a analizar los datos obtenidos de estos. De todos los criterios presentados, solo se logro evaluar una parte de los mencionados a continuación:

- **Escalabilidad:**

  La Escalabilidad es la capacidad que tiene el broker para comportarse ante crecientes variaciones en los volúmenes de clientes, los datos que este procesa y el cómo se implementa está característica en cada protocolo. Esto esta inicialmente ligado con el

diseño del protocolo y del sistema y significa como el protocolo se adecua ante estos incrementos. Este criterio fue evaluado durante las pruebas.

    - **Concurrencia:** La Concurrencia es la forma en que se ejecutan procesos de manera simultánea sin afectar el resultado final. Esto significa que el broker es capaz de procesar mensajes de diferentes fuentes y dirigirlos a su respectivo punto de llegada sin la necesidad de entrar en conflicto con otros mensajes. Este criterio fue evaluado durante las pruebas.
    - **Tolerancia a Fallos:** La Tolerancia a Fallos es la capacidad que dispone el broker para asegurar los mensajes ante un fallo de una parte de la arquitectura, incluido el mismo broker. Estos fallos pueden ser la caída de los servicios o mensajes corruptos durante el envío.

- **Persistencia de los Mensajes:** La Persistencia de los Mensajes es una propiedad que presentan los protocolos para almacenar los mensajes en caso de no ser capaz de enviarlos a su destino. Esta propiedad varia en implementación y funcionalidad según el protocolo y el broker.
- **Quality of Service - QoS:** El QoS es un conjunto de instrucciones en los protocolos de mensajería que se encarga de asegurar el envío de los mensajes, así como el rendimiento y el manejo en el uso del ancho de banda disponibles del broker. Este criterio fue evaluado durante las pruebas.
    - **Integridad:** La integridad es la capacidad que dispone el broker de mantener el contenido intacto, tal cual es enviado por el publicador y recibido en el subscriptor. Este criterio fue evaluado durante las pruebas.

- **Latencia:** La latencia es la medida que indica el tiempo que tarda un mensaje en llegar a su destino, después de ser enviado. Este criterio fue evaluado durante las pruebas.
- **Variación de la Latencia:** La variación de la Latencia es una medida que indica la diferencia de tiempo entre mensajes y que el broker se encarga de que los mensajes enviados no sean entregados en orden incorrecto o que lleguen mucho después de ser necesitados.

5.4. Selección de Modelos de Comunicación

Definidos los criterios de cómo se evaluarán los modelos de comunicación, se investiga y selecciona un conjunto de modelos de comunicación los cuales son conformados por servicios, protocolos y brokers, que más adelante serán evaluados según las necesidades. Este conjunto fue escogido según unos filtros que se describen una implementación de un Smart Campus:

- Protocolos enfocados en la comunicación de dispositivos a través de implementaciones en servidores locales o en la nube.
- Ampliamente investigados y con una gran base de información disponibles.
- De un uso institucional u organizacional, sin ser enfocados a lo industrial.
- Protocolos con patrón Publish/Subscribe.

Estos filtros permiten una selección más enfocada a la hora de escoger el conjunto a evaluar.

- Protocolos:
  - MQTT
  - AMQP

- Apache Kafka

- Herramientas:
  - DigitalOcean
  - Docker
- Brokers:
  - Mosquitto
  - RabbitMQ
  - EMQX

5.4.1. Acercamiento a los Protocolos

Inicialmente solo se hicieron experimentos utilizando la herramienta Jmeter y scripts en un entorno de VPN además de que las pruebas solo se ejecutaron una única vez, con una variación en la cantidad de mensajes. Con estos experimentos, se encontraron varios puntos claves a la hora de hacer una evaluación de los protocolos, pero durante el análisis de los resultados se encontraron falencias en los datos, tales como, pérdidas de mensajes, desconexiones a mitad del funcionamiento, rendimiento muy por debajo de lo esperado, ante esta situación se diseñaron los escenarios 1 y 2.

5.5. Diseño Experimental

En esta sección se define como son las pruebas que se realizan a los modelos de comunicación definidos y que tipo de datos y resultados se obtienen de las pruebas.

5.5.1. Planeación de las Pruebas

En la sección 5.2.1 se explican los criterios que son la forma en cómo se evaluaran los protocolos durante las pruebas, estos criterios definen las características y que según cada protocolo es diferente en cada uno, esto da un punto de vista para la obtención de datos según las pruebas y su posterior análisis y conclusiones.

Se elaboró una serie de pruebas que buscan replicar los comportamientos de los protocolos ante situaciones de un entorno real, para esto, se dispone de una serie de escenarios extraídos de la arquitectura IoT (sección 5.1), todo construido sobre servidores de servicios Cloud.

Para el uso de los servicios cloud, está el conjunto de servidores de DigitalOcean que permite la creación de máquinas virtuales que ejecutan un Sistema Operativo Linux, que aporta una implementación de los protocolos estable, permitiendo mejores resultados a la hora ejecución de código.

5.5.2. Elaboración de las Pruebas

Tomando el conjunto de software, este es implementado en las máquinas virtuales que harán la ejecución de las pruebas. Estas pruebas son elaboradas siguiendo un diagrama que enseña cómo están construidas.

- ➔ Entornos
  - ◆ Criterios de Evaluación
  - ◆ Escenarios
    - ● Parámetros
    - ● Experimentos

➔ Resultados

Diagrama de pruebas.

5.5.2.1. Entornos. En la sección 5.1 se definen los entornos que parten de las necesidades definidas, estos entornos son la forma en como interactúan las partes de la arquitectura y como es el trayecto de los mensajes según el cliente y el subscriptor:

- Device to Backend
- Backend to Backend

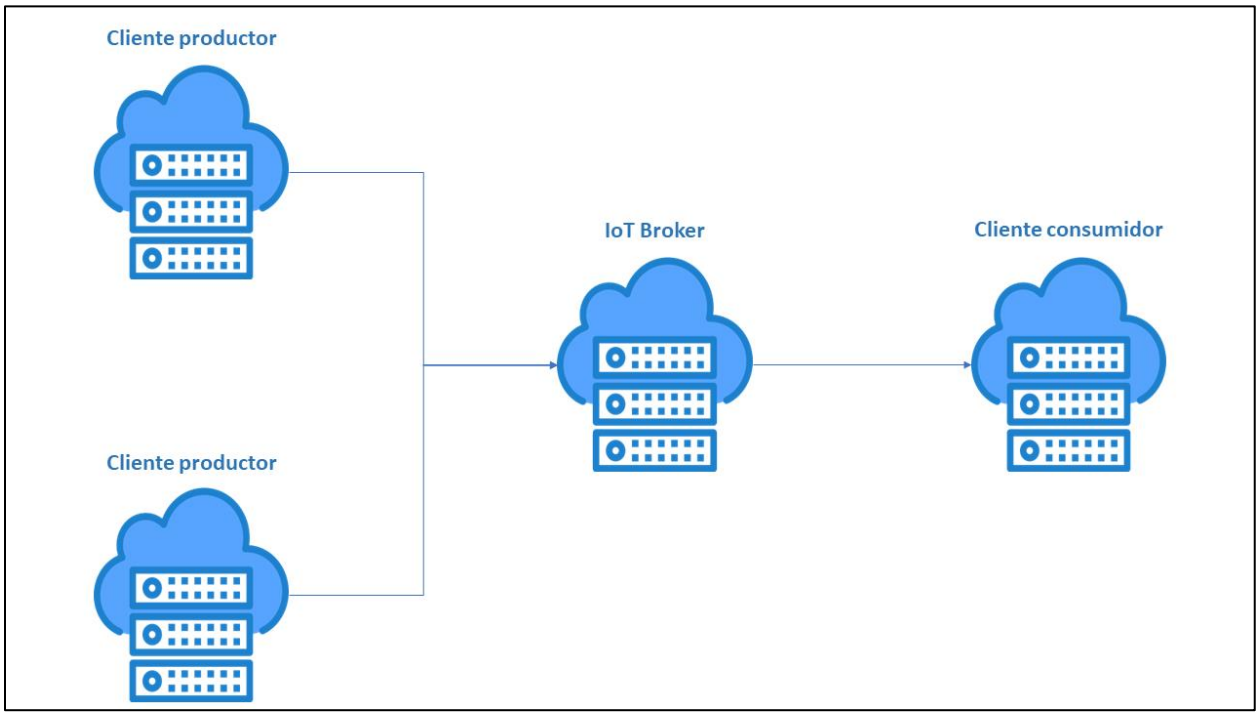
5.5.2.2. Criterios de Evaluación. En la sección 4.2 se definen los criterios para la toma de información de las pruebas, que terminaran en la evaluación de los protocolos y brokers.

5.5.2.3. Escenarios. Los escenarios son las situaciones en las que se están ejecutando las pruebas con tal de obtener información para la evaluación de los protocolos y brokers. Se crearon dos escenarios de uso que hacen esforzar al máximo las máquinas y protocolos.

- **Escenario 1:** Se implementa una comunicación entre tres diferentes máquinas virtuales que esta ubicados en servidores de diferentes zonas geográficas que ofrece DigitalOcean. Los clientes publicadores envían mensajes simultáneamente a través del broker y llegan a un único cliente suscriptor.

Figura 9.

Escenario 1

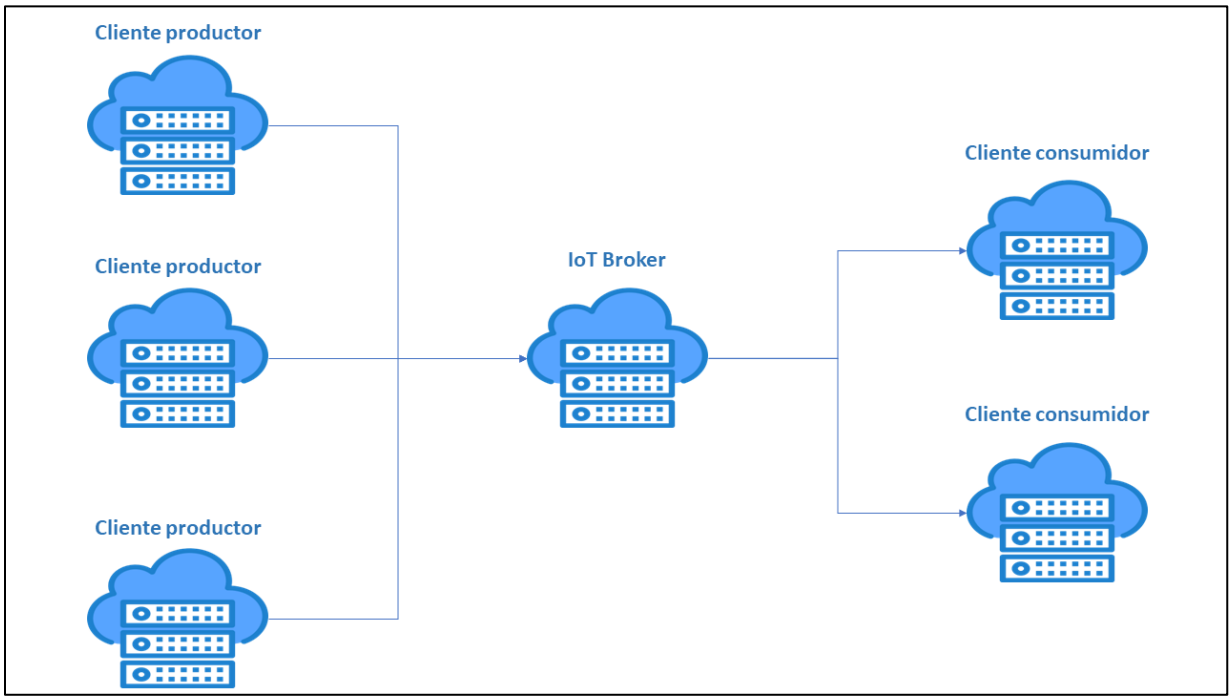


Nota. Elaboración propia.

- **Escenario 2:** Este escenario es similar al anterior con la diferencia del agregado de dos máquinas virtuales extra, una de publicador y una de suscriptor con tal de ejercer mayor carga en los brokers y su posterior análisis de comportamiento. Los tres clientes publicadores envían mensajes simultáneamente que pasan a través del broker y llegan a los dos clientes suscriptores en un solo tópico.

Figura 10.

Escenario 2



Nota. Elaboración Propia

5.5.2.4. Parámetros. Los parámetros son las características que cambian según el experimento durante las pruebas, estos siguen lo establecido con los criterios y es la principal fuente de información que se usará para el análisis de datos.

Parámetros de Escalabilidad:

- Número de Mensajes enviados Simultáneamente
- Número de Mensajes Exitosos

Parámetros de QoS:

- Orden de Llegada
- Quality of Service [0-2] (solo MQTT)

5.5.2.5. Experimentos. Los experimentos son la combinación de las configuraciones anteriores para la elaboración de las pruebas. Cada experimento cuenta con una configuración de protocolo, cantidad de mensajes, escenario de ejecución y en caso de MQTT su QoS, adicionalmente cada experimento se realiza 5 veces para mejor análisis estadístico.

5.5.2.6. Resultados. Con los datos extraídos de los experimentos, se procede a hacer el análisis estadístico para la obtención de resultados de los protocolos ante las situaciones establecidas, con esto se realizan las evaluaciones de los protocolos en los diferentes entornos y obtener un comportamiento aproximado como si estuvieran siendo usados en una arquitectura IoT.

6. Implementación de la Evaluación

En esta sección se habla sobre todas las configuraciones de las pruebas, en los que se aplicará todo el proceso de metodología previamente mencionado, con tal de evaluar los protocolos MQTT, AMQP y Kafka.

Todos los experimentos correrán sobre los escenarios descritos en la sección 6.3.2.3, esto con el fin de obtener una gran cantidad de resultados que den una idea del funcionamiento de los protocolos en dichas situaciones.

6.1. Equipos

Se dispone de 6 máquinas virtuales en la nube que tendrán un rol diferente dependiendo del experimento, las especificaciones de cada máquina estarán redactadas a continuación:

Tabla 1. Equipos

Uso	Equipo	CPU	RAM	Almacenamiento	S.O.
Cliente IoT	VM-1	CPU 2 Core	4Gb DDR4	50Gb SSD	Ubuntu 22.10
Cliente IoT	VM-2	CPU 2 Core	4Gb DDR4	50Gb SSD	Ubuntu 22.10
Gateway IoT	VM-3	CPU 2 Core	4Gb DDR4	50Gb SSD	Ubuntu 22.10
Cliente IoT	VM-4	CPU 2 Core	4Gb DDR4	50Gb SSD	Ubuntu 22.10
Cliente IoT	VM-5	CPU 2 Core	4Gb DDR4	50Gb SSD	Ubuntu

					22.10
Cliente IoT	VM-6	CPU 2 Core	4Gb DDR4	50Gb SSD	Ubuntu 22.10

Nota. Elaboración propia.

Las máquinas virtuales fueron creadas usando el servicio Cloud de DigitalOcean, este permite crear diferentes máquinas virtuales con variaciones en las especificaciones requeridas, aunque de forma sencilla explica únicamente está información y no va más a detalle los componentes.

6.2. Implementación

Para la parte de la implementación se usó una implementación de código propio que ajusta los protocolos a los escenarios mencionados.

6.2.1. Scripts

En la implementación están las características mencionadas, que fueron creadas bajo el framework de JavaScript, NodeJS, mediante el cual, se usaron las librerías de los brokers y protocolos para una implementación más enfocada en las necesidades.

En estos scripts, se define la configuración de conexión hacia el Gateway, la cantidad de mensajes a enviar y las configuraciones propias de los protocolos de cómo se envían los mensajes. Para una aplicación completa de los experimentos se dispuso con una forma de almacenar en un archivo los datos enviados y recibidos para su análisis respectivo.

Los scripts se ejecutan en máquinas virtuales se localizan en la nube, servidores ubicados en diferentes ubicaciones físicas diferentes.

6.3. Tablas de Experimentación

Se da inicio con la planeación de los experimentos que se realizaron, variando los aspectos que definieron previamente:

Tabla 2. *Experimentos*

Experimentos	Escenario	# Mensajes	Broker	QoS
Exp 1	1	10.000	Mosquitto	0,1,2
Exp 2	1	25.000	Mosquitto	0,1,2
Exp 3	1	50.000	Mosquitto	0,1,2
Exp 4	1	10.000	RabbitMQ	-
Exp 5	1	15.000	RabbitMQ	-
Exp 6	1	20.000	RabbitMQ	-
Exp 7	1	10.000	Apache Kafka	-
Exp 8	1	25.000	Apache Kafka	-
Exp 9	1	50.000	Apache Kafka	-
Exp 10	2	10.000	Mosquitto	0,1,2
Exp 11	2	25.000	Mosquitto	0,1,2
Exp 12	2	50.000	Mosquitto	0,1,2
Exp 13	2	10.000	RabbitMQ	-
Exp 14	2	25.000	RabbitMQ	-
Exp 15	2	30.000	RabbitMQ	-

Exp 16	2	10.000	Apache Kafka	-
Exp 17	2	25.000	Apache Kafka	-
Exp 18	2	50.000	Apache Kafka	-

Nota. Elaboración propia.

6.4. Ejecución de los Experimentos

En términos generales el desarrollo de los experimentos planteados en las secciones anteriores está dado en dos pasos: La ejecución de los scripts desarrollados y la generación de los datos.

6.4.1. Ejecución de los Scripts

El envío de mensajes desde los productores se realiza de la siguiente manera:

1. Se divide el envío de mensajes en cierta cantidad de paquetes, de manera que no se sobre carguen los productores al enviar muchos mensajes
2. Los mensajes se envían en el transcurso de 10 segundos
3. Una vez finalizado el envío de mensajes, se almacenan en un Excel para su posterior análisis.

Cabe resaltar que las máquinas en las que se ejecutan los scripts publicadores deben permanecer activas o en línea hasta que se haya completado la carga o envío de datos para no generar pérdidas de información y fallos en las pruebas.

6.4.2. Toma de Datos

Una vez terminada la ejecución de los scripts, se habrán generado las tablas con la información relevante de la prueba realizada, con estos datos se puede realizar el respectivo análisis de la prueba

y verificar que todo haya salido bien. Para cuestiones de veracidad de la información, cada prueba se realiza un total de 10 veces y se promedian los resultados para de esta manera trabajar con datos más veraces.

7. Resultado de los Experimentos

Comenzando la sección, se presentan los resultados y análisis realizados durante la toma de datos de los experimentos mencionados previamente en la Tabla 3 de la sección 7.4, así mismo, en la sección 7 está la descripción de cómo se realizan los experimentos.

7.1. Tratamiento de datos

Los experimentos del Escenario 1 constan de tres clientes y un Gateway broker, dos clientes hacen de publicadores y uno de subscriptor que recibe los mensajes. Los dos publicadores envían mensajes al mismo tiempo, mientras el broker los procesa y redirige hacia el cliente 3 que hace de subscriptor, con esto es el broker quien ejerce todo el trabajo en estos experimentos.

Ambos clientes registran él envió de datos y la llegada de datos, con esto se comprueba que no haya datos perdidos durante el experimento. Durante el tratamiento de datos, previo al análisis, se procede a usar los tiempos que se registran en la parte del subscriptor, el cual recibe el tiempo en el que fue enviado el mensaje por parte del publicador y registra al momento de llegar su propio tiempo, esto, para calcular el tiempo que demoro el mensaje en llegar desde que fue enviado creando una variable Delta, que representa la diferencia del tiempo de llegada con el Latencia.

Todos los experimentos se repitieron 5 veces para obtener un comportamiento estadístico más realístico y al final los Deltas de las 5 pruebas fueron promediadas, sin tener en cuenta los valores ceros, que para este proyecto indican que el mensaje nunca fue enviado, por lo que no puede ser registrado. Al final, se obtienen un Delta promedio por cada cliente, que posteriormente es graficado para analizar su comportamiento.

Finalmente, en la parte del tratamiento de los datos, se entablan los promedios de los porcentajes de mensajes que llegaron exitosamente enviados de cada experimento y los promedios de tiempo que tardo cada mensaje en ser enviado.

7.2. Experimentos en el Escenario 1.

Obteniendo todos los Deltas de los experimentos, las gráficas y las tablas, se procede a analizar el comportamiento de los protocolos, con tal de determinar su funcionamiento ante las cargas de envío de datos.

7.2.1. Experimentos en Mosquitto

Comenzando con los experimentos en MQTT usando el protocolo de Mosquitto, estos se realizaron en 3 experimentos, usando cantidades de datos de 10.000, 25.000 y 50.000 datos enviados secuencialmente. Estos experimentos se variaron usando los diferentes QoS de MQTT, nivel 0, 1 y 2, para un total de 9 pruebas realizadas y se repitieron 5 veces cada una.

Empezando con la tabla de resumen de las pruebas (tabla 4), se encontró un comportamiento único en este proyecto al respecto de MQTT, donde con un QoS de nivel 0, los datos enviados por el publicador llegan sin problemas al subscriptor y no hay mucha diferencia en los tiempos en los experimentos 2 y 3 con 25.000 y 50.000 datos respectivamente.

Tabla 3. *Experimento 1,2,3 - Mosquitto Escenario 1*

Experimento	Cantidad de Mensajes	QoS	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
1	10.000	0	1.4097	0.0683	0.4843	100%
1	10.000	1	9.1776	2.3414	6.7312	11.39%
1	10.000	2	24.6034	5.7696	15.9455	7.5%
2	25.000	0	1.36	0.0687	0.5894	100%
2	25.000	1	11.0601	3.1909	7.7113	4.95%
2	25.000	2	23.634	11.4491	18.2832	3.04%
3	50.000	0	1.8905	0.0686	0.8838	100%
3	50.000	1	10.2291	4.1197	7.845	2.38%
3	50.000	2	23.9536	11.5543	18.6024	1.59%

Nota. Elaboración propia.

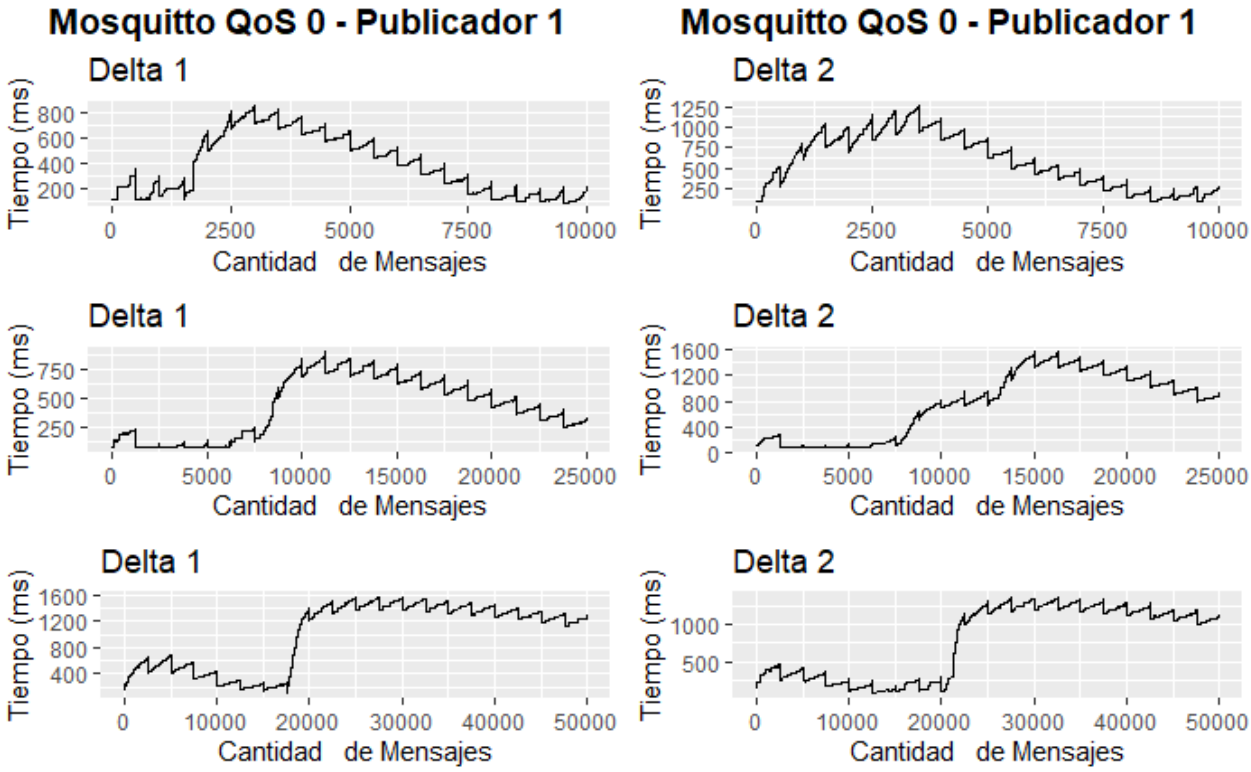
Caso contrario en los QoS de nivel 1 y 2, donde presentan un aumento en el tiempo gigantesco y una pérdida de datos significativa, al analizar los datos se observa que en el experimento 1, 2 y 3 en estos QoS la cantidad de mensajes por prueba es muy similar, siendo de un aproximado de 1.200 en nivel 1 y 750 en nivel 2 en los tres experimentos, eso indica que el máximo teórico que soporta Mosquitto en mensajes de QoS nivel 1 y 2 antes de colapsar y dejar de enviar mensajes.

Continuando con las gráficas, estas muestran los Delta previamente descritos en la sección 8.1. Las gráficas están ordenadas por pares desde arriba hacia abajo con un Delta 1 y un Delta 2 por cada experimento. Los picos en las gráficas son debidos a la forma de envió de los datos, que no se envían todos en secuencia, si no, que se separan en paquetes dependiendo de la cantidad de

mensajes para un envió más sistemático y que el broker no los reciba de uno en uno, esto con tal de observar el comportamiento del broker en el escenario.

Grafica 1.

Experimentos 1, 2 y 3 – Mosquitto QoS 0 - Escenario 1



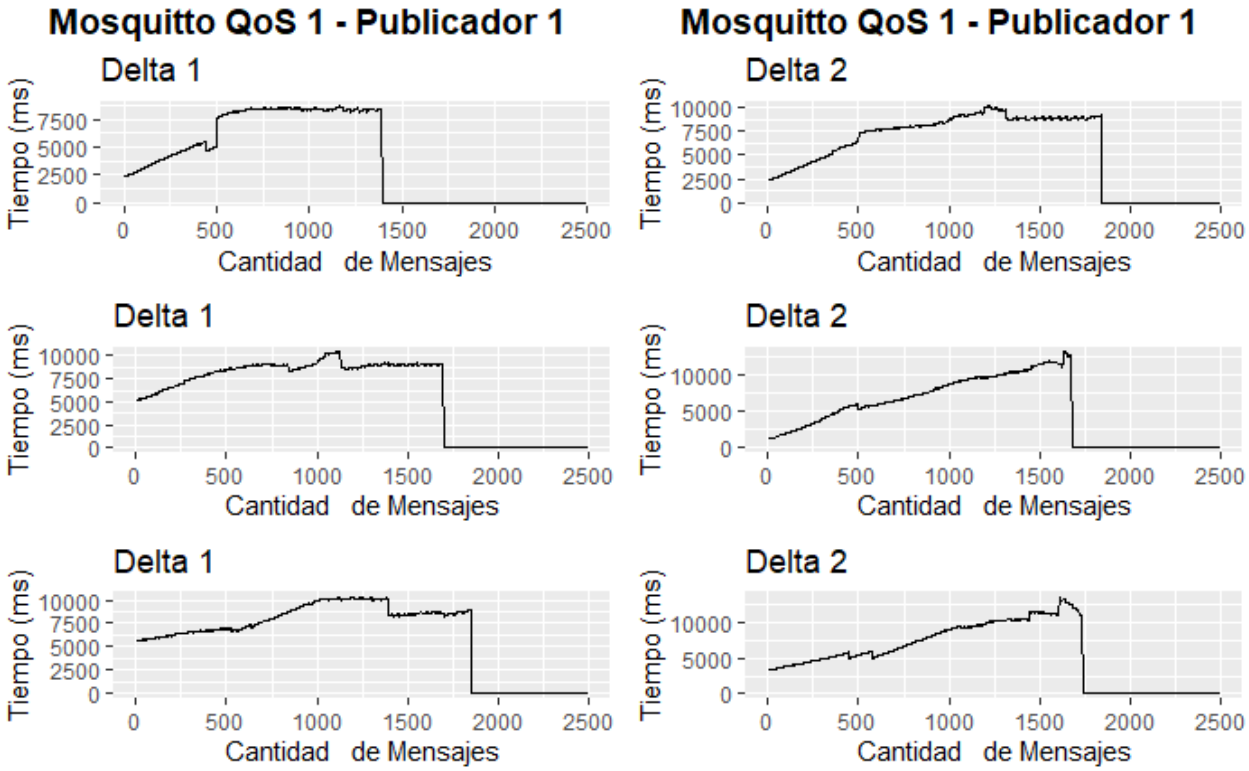
Nota. Elaboración propia.

Se aprecia el comportamiento pausado al inicio de los datos que va aumentando a medida que hay más mensajes por ser procesados, como el broker está procesando los mensajes al mismo tiempo de los dos publicadores, la carga en él va aumentando, por lo que los tiempos aumentan, siendo su punto máximo en el experimento 1 de entre 2.500 y 3.000 mensajes enviados, en el

experimento 2 de unos 10.000-15.000 y en el experimento 3 de 20.000-22.000, esto se nota en los publicadores, donde el tiempo máximo es mayor en uno que en el otro.

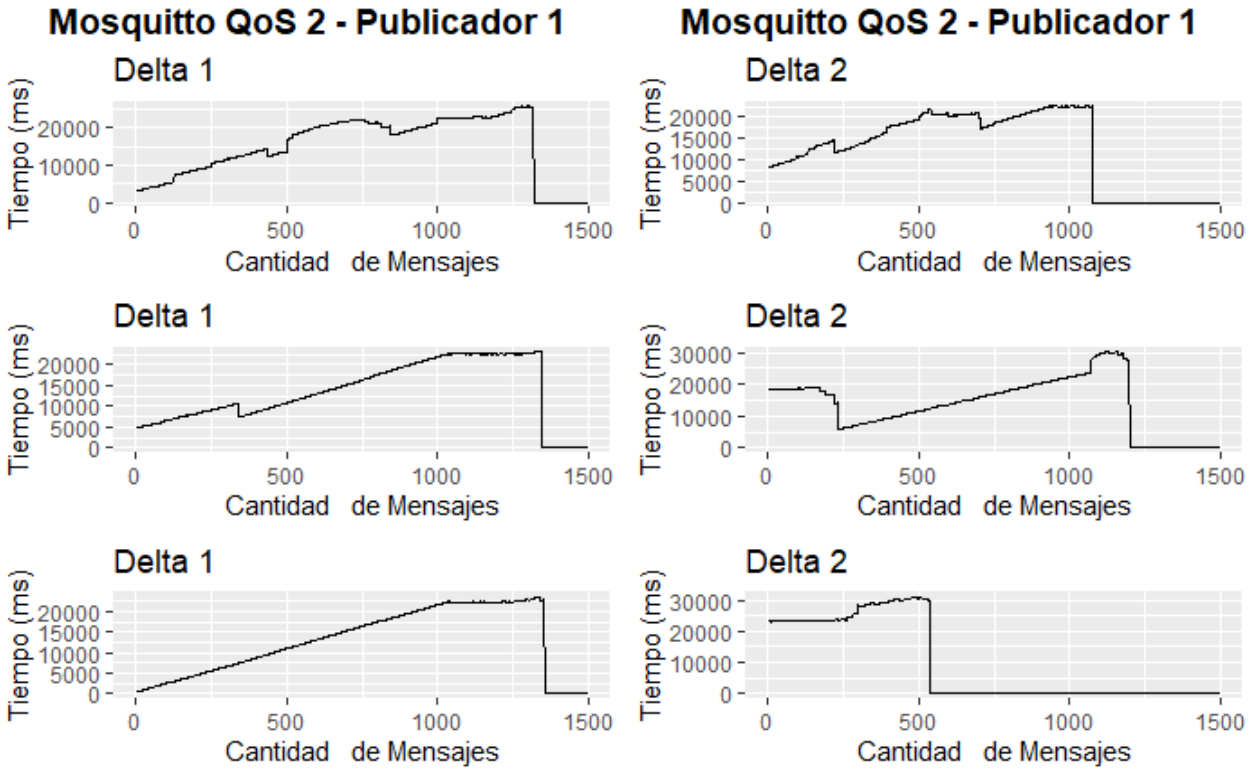
Al analizar las gráficas de los QoS de nivel 1 y 2, (Grafica 2 y 3), el comportamiento de los datos es diferente a los del nivel 0, en este caso, los datos tienen un comportamiento lineal al inicio que va en aumento con el tiempo que demoran en llegar los mensajes, al final, el broker no es capaz de seguir procesado los mensajes en estos niveles y termina por colapsar y el script se detiene por completo, en tal caso, los mensajes del publicador fueron todos enviados, pero al no haber un broker que los redirija, estos se pierden.

Grafica 2. *Experimentos 1, 2 y 3 – Mosquitto QoS 1 - Escenario 1*



Nota. Elaboración propia.

Grafica 3. *Experimentos 1, 2 y 3 – Mosquitto QoS 2 - Escenario 1*



Nota. Elaboración propia.

7.2.2. Experimentos en RabbitMQ

Pasando a los experimentos con AMQP usando RabbitMQ como broker, estos a diferencia de Mosquitto, no tienen un QoS, por lo que las pruebas únicamente fueron los 3 experimentos para 10.000, 15.000 y 20.000 datos respectivamente. Se disminuyó la cantidad de mensajes en Rabbit debido a cómo funciona el protocolo, al intentar aumentar el número de datos el cliente que hace de subscriptor colapsaba, lo cual no es lo buscado en estas pruebas. Igualmente, cada experimento se repitió 5 veces.

Se tiene la tabla con los datos de Rabbit (tabla5), se encuentran datos muy similares a los datos obtenidos de Mosquitto con el QoS de nivel 0, ya que, en gran medida, ambos trabajan de manera similar, la única gran diferencia son las colas en AMQP que también funcionan como sistema para guardar los mensajes, cosa que MQTT no dispone.

Tabla 4. Experimento *4,5,6 - RabbitMQ Escenario 1*

Experimento	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
4	10.000	1.3824	0.0752	0.4198	100%
5	15.000	0.501	0.0721	0.1187	100%
6	20.000	1.3106	0.0752	0.5062	100%

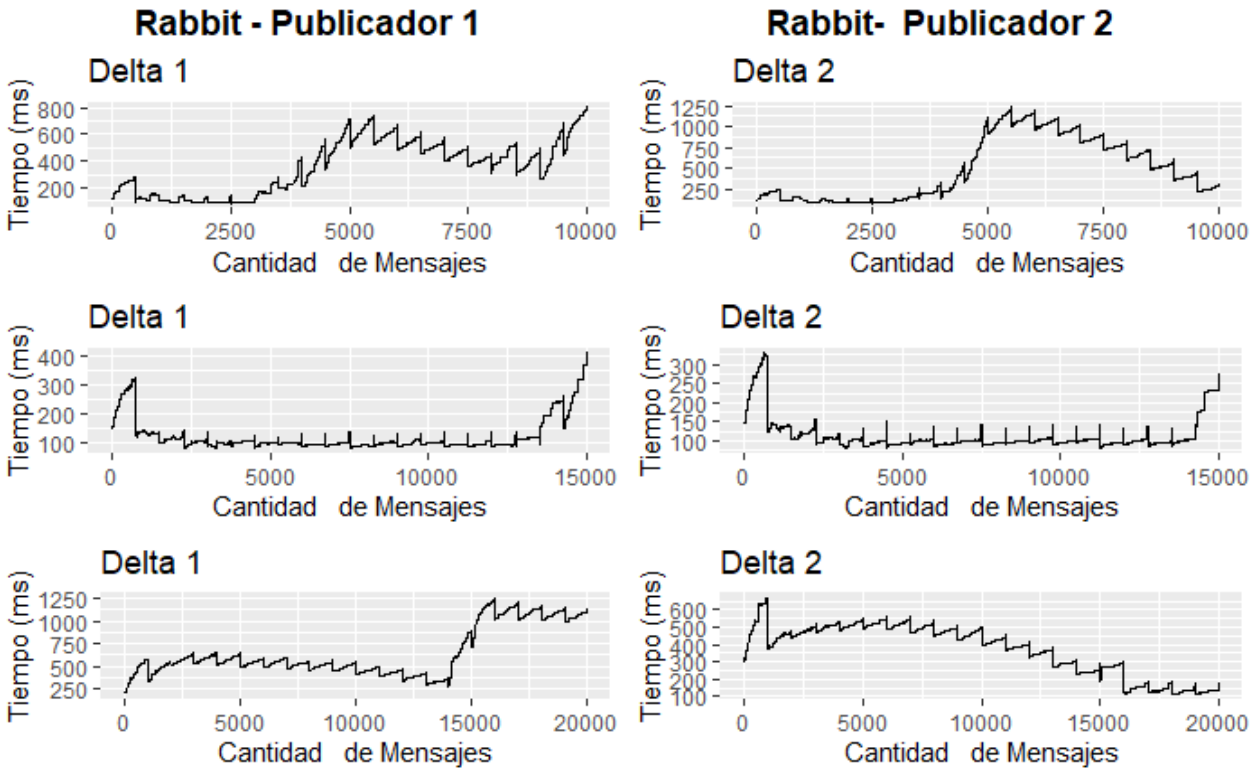
Nota. Elaboración propia.

Los tiempos respecto a Mosquitto son similares, con alguna que otra diferencia, pero debido a la diferencia en la cantidad de mensajes, no es factible una comparación tan directa, al final, los mensajes en los 3 experimentos demuestran que los mensajes llegan sin ningún problema al destino.

Pasando a las gráficas, se encuentra un comportamiento similar a Mosquitto, donde los primeros mensajes que llegan son bien recibidos en bajos tiempos en el experimento 4, al pasar al experimento 5 y 6 hay un pico de datos al principio, muy probablemente debido al choque de los primeros mensajes son enviados al iniciar las pruebas, pero con el progreso, el broker se estabiliza y llega presentar incluso tiempos de mensajes menores que en Mosquitto.

Grafica 4.

Experimentos 4, 5 y 6 – Rabbit - Escenario 1



Nota. Elaboración propia.

Ya en el experimento 5, se evidencia una estabilidad de los mensajes siendo casi constante, siendo esos pequeños picos los primeros datos de un paquete en curso, además, está el claro bajón en los tiempos de este experimento siendo el que menos tardó en completarse.

Para el experimento 6, el comportamiento toma una forma más controlada al principio, siendo en el publicador 2, donde los primeros paquetes de datos, demora más que los últimos, mientras que el publicador 1, va creciendo el tiempo de los mensajes.

7.2.3. Experimentos en Apache Kafka

Para finalizar con los experimentos en el escenario 1, están los referentes a Apache Kafka, el cual presenta un sistema de enviado de mensajes diferente a MQTT y AMQP, pero que sigue bajo el patrón Pub/Sub.

En la tabla de Kafka (tabla 6), se evidencia un comportamiento muy diferente de los vistos previamente en los experimentos 8 y 9, mientras que en el experimento 7, los resultados son muy similares. A medida que aumentan los datos, Kafka se toma más tiempo en procesarlos y enviarlos, siendo el tiempo máximo el mensaje del último mensaje en ser enviado.

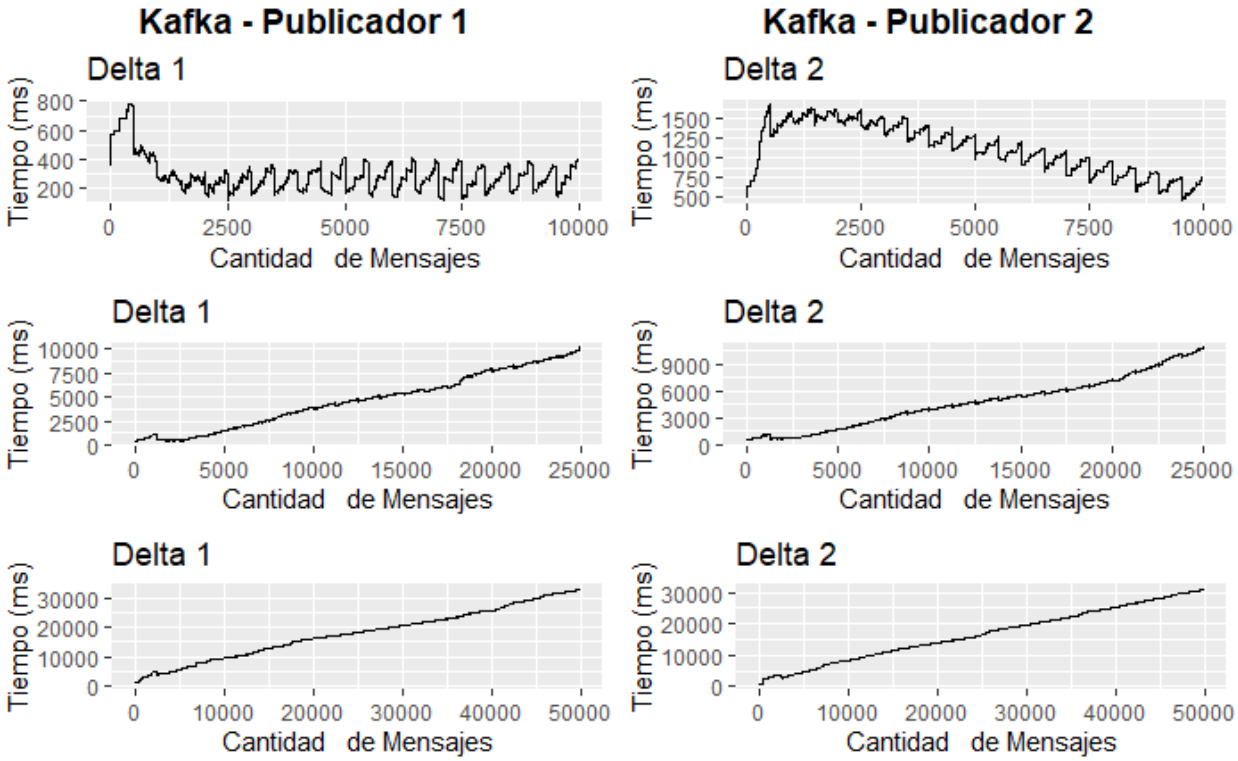
Tabla 5. *Experimento 7,8,9 – Apache Kafka - Escenario 1*

Experimento	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
7	10.000	1.1428	0.076	0.4129	100%
8	25.000	10.5353	0.1563	4.6949	100%
9	50.000	31.9718	0.4483	17.2814	100%

Nota. Elaboración propia.

Estos resultados se evidencian mejor en las gráficas, el experimento 7 tiene un comportamiento similar a los experimentos anteriores, más concretamente, en los experimentos de Rabbit, donde los tiempos son muy similares.

Grafica 4. *Experimentos 7, 8 y 9 – Kafka - Escenario 1*



Nota. Elaboración propia.

Ya pasando a los experimentos 8 y 9, el asunto cambia aún más, debido a que los mensajes tienen un comportamiento lineal, casi que idóneo, esto debido a como se envían los mensajes de manera secuencia, se espera que el mensaje 2 demore más que el mensaje 1, pero en los experimentos previos, se nota que no siempre es el caso.

7.3. Experimentos en el Escenario 2

Con los experimentos del escenario 1 terminados, se procede a continuar con el escenario 2, en el cual se usan ahora cinco clientes y un gateway broker, tres de los clientes son publicadores y los otros dos son subscriptores. En este escenario se busca estresar aún más al broker pidiéndole que

los mensajes de los tres clientes sean redirigidos a los dos subscriptores. El tratamiento de datos en este escenario es idéntico al del primer escenario, solo que la cantidad de datos a tratar aumento.

7.3.1. Experimentos en Mosquitto

Comenzando con Mosquitto, la información a analizar ha aumentado, al tener ahora dos subscriptores estos toman los datos de los tres publicadores y son analizados en conjunto según su nivel de QoS para una mejor interpretación, comenzando con el nivel 0 en la siguiente tabla 7.

Tabla 6. *Experimento 10,11,12 - Mosquitto QoS 0 Escenario 2*

Experimento	Cliente Subscriptor	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
10	1	10.000	0.5456	0.072	0.1542	100%
10	2	10.000	0.4596	0.0676	0.1360	100%
11	1	25.000	0.7805	0.0727	0.2302	100%
11	2	25.000	0.7689	0.0669	0.2241	100%
12	1	50.000	1.439	0.0843	0.35	100%
12	2	50.000	1.7894	0.1922	0.8426	100%

Nota. Elaboración propia.

Los datos encontrados en QoS nivel 0, indican un comportamiento esperado si se toma en cuenta respecto a los datos en el Escenario 1, al tener dos subscriptores recibiendo los mismos datos de los tres publicadores, la carga disminuye y se aprecia la disminución en los tiempos que tarda en llegar el mensaje respecto al escenario 1.

Tabla 7. *Experimento 10,11,12 - Mosquitto QoS 1 Escenario 2*

Experimento	Cliente Subscriptor	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
10	1	10.000	9.1903	4.3696	7.0826	7.51%
10	2	10.000	9.2444	4.3896	7.1547	7.49%
11	1	25.000	8.9250	4.614	7.1915	3.07%
11	2	25.000	7.5044	3.671	7.3183	3.03%
12	1	50.000	8.9310	4.1085	7.2511	1.53%
12	2	50.000	9.0625	4.1381	7.3146	1.53%

Nota. Elaboración propia.

Respecto al QoS nivel 1, encontramos datos muy similares entre los experimentos, esto es de esperar, y da a entender que no importa el escenario, la cantidad máxima de mensajes que tolera QoS 1 es de unos pocos más de 1.000 antes de colapsar.

Tabla 8. *Experimento 10, 11, 12 - Mosquitto QoS 2 Escenario 2*

Experimento	Cliente Subscriptor	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
10	1	10.000	25.0448	10.628	18.0078	5.13%
10	2	10.000	24.9359	10.6953	18.0134	5.12%
11	1	25.000	27.7835	14.2694	18.1713	2.36%
11	2	25.000	28.0091	14.4864	18.445	2.35%
12	1	50.000	76.4844	14.7125	44.8617	2.93%
12	2	50.000	76.5028	14.7568	44.9426	2.94%

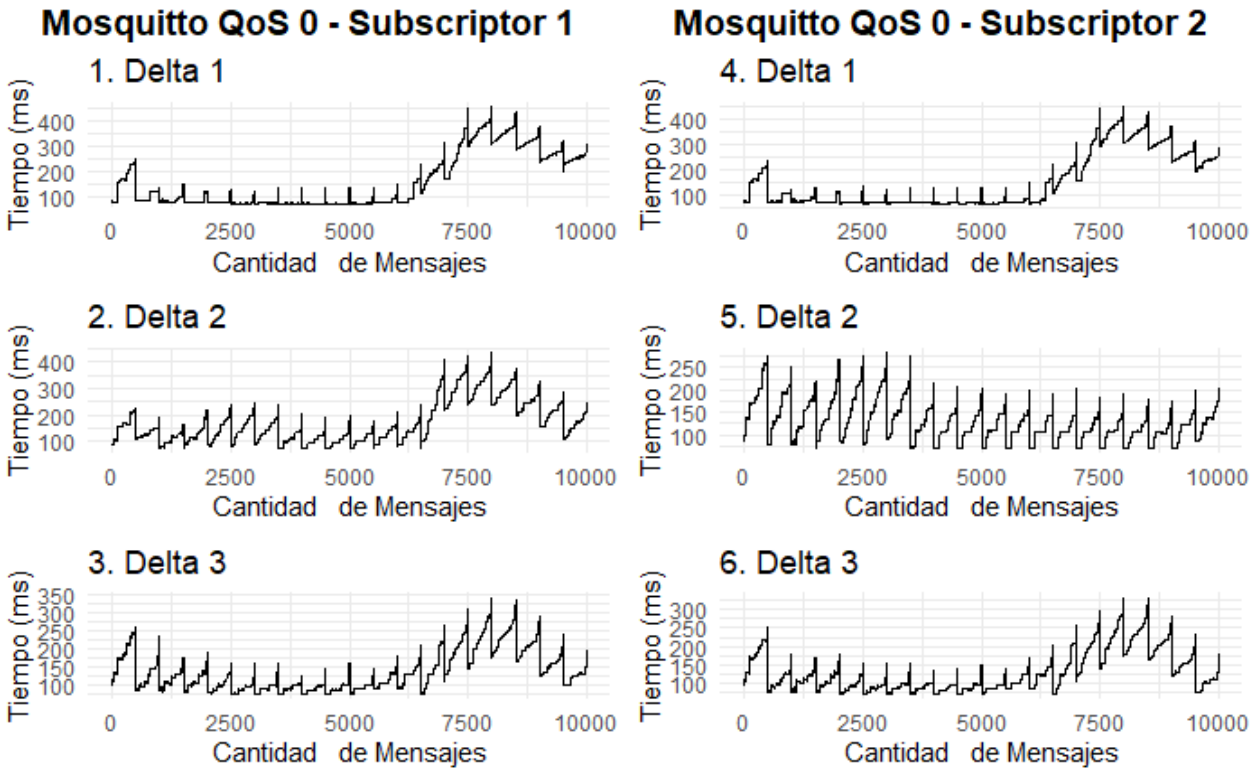
Nota. Elaboración propia.

Para finalizar con las tablas, en QoS nivel 2, se repite un mismo patrón como en QoS 1, donde los datos entre subscriptores son similares, a diferencia del escenario 1, el tiempo se ve incrementado, no por mucho, aunque en el experimento 12, hay un incremento enorme en comparación al escenario 1.

Continuando con las gráficas, como ahora estas se presentan por subscriptor, en el que cada Delta representa los datos de su publicador respectivo, con esto se comparan frente a frente los datos de cada subscriptor y se procede a analizar.

En el caso de QoS 0, vemos un comportamiento un tanto similar al escenario 1, pero este tiende a ser más alargado y los aumentos en la carga del broker se ven menos que los experimentos anteriores, siendo casi en los últimos mensajes. En el caso del publicador 2 del subscriptor 2, los datos son más cíclicos, únicamente variando los tiempos máximos de cada paquete.

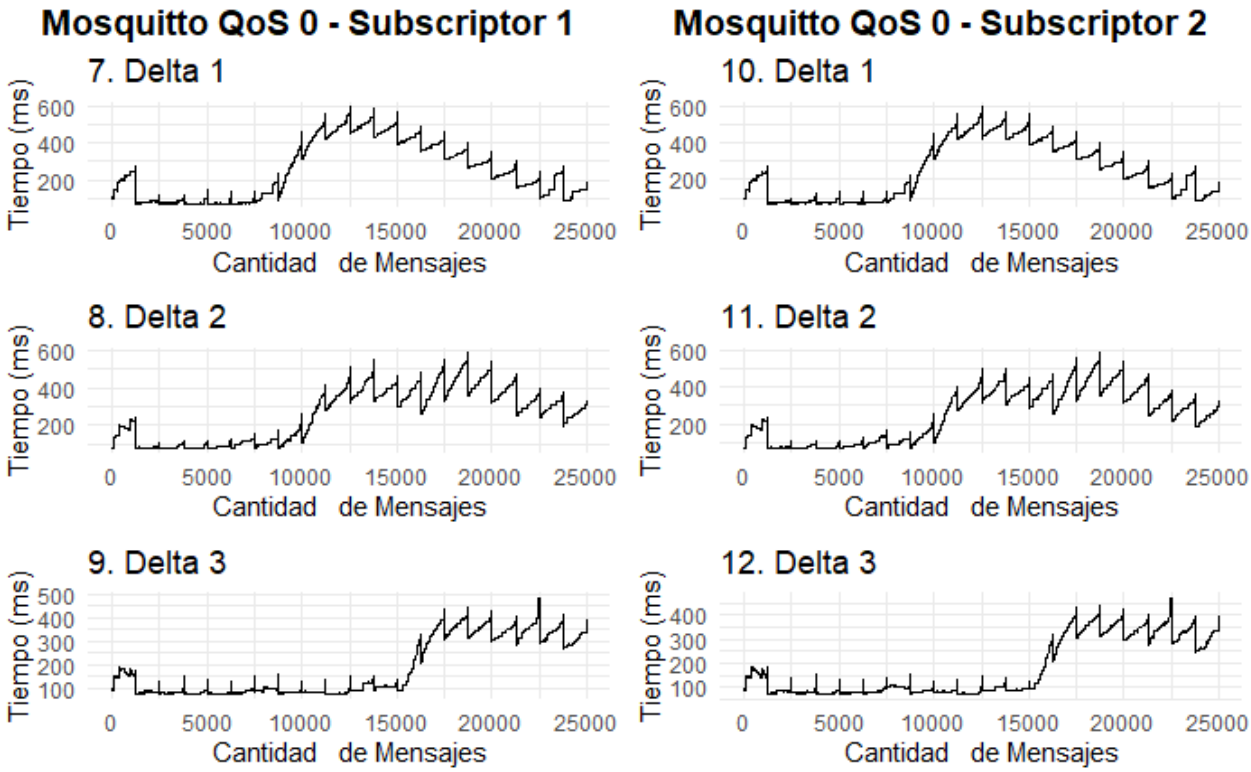
Grafica 5. *Experimento 10 – Mosquitto QoS 0 - Escenario 2*



Nota. Elaboración propia.

En comparación del experimento 10, a medida que aumenta los datos enviados, la curva se va haciendo más suave, este es el comportamiento que se analiza en el experimento 11, aunque la curvatura sigue manejándose de una manera similar.

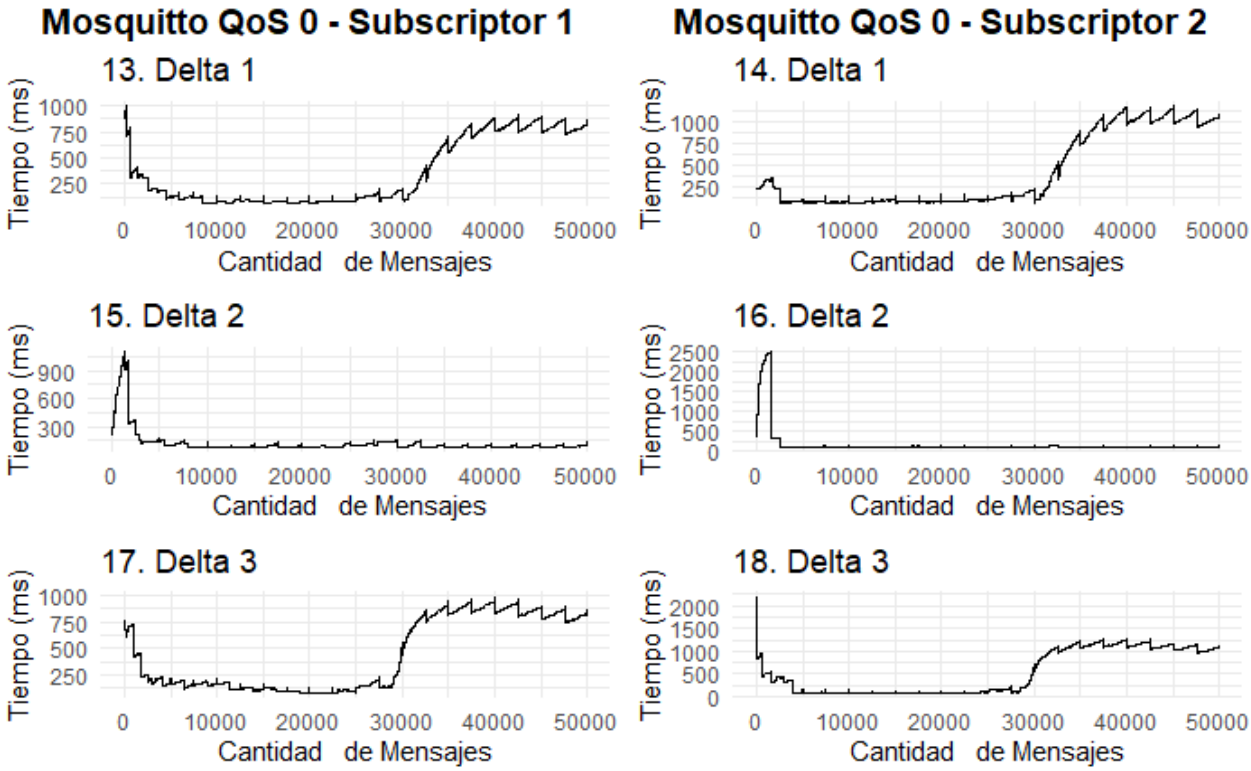
Grafica 6. *Experimento 11 – Mosquitto QoS 0 - Escenario 2*



Nota. Elaboración propia.

De igual manera, en el experimento 12, las curvaturas de los datos se van suavizando más, aunque en este caso los tiempos son mayores respecto a los experimentos 10 y 11. En el Delta 2, que hace referencia al publicador 2, esta toma un comportamiento visto previamente en el experimento 5, donde los tiempos son los mínimos posibles en el envío de los datos.

Grafica 6. *Experimento 12 – Mosquitto QoS 0 - Escenario 2*



Nota. Elaboración propia.

Se entiende entonces que, al disponer de más subscriptores, este divide la carga entrante y permite un envío de los mensajes más rápido, aun así, todo esto es lo observado en QoS nivel 0, para las gráficas de los QoS nivel 1 y 2, no serán analizadas debido a la poca información que se obtienen de ellas, al final tienen el mismo comportamiento que las vistas en el escenario 1 y no cambian en nada al añadir un publicador y subscriptor extra.

7.3.2. Experimentos en RabbitMQ

En Rabbit nuevamente tenemos un cambio respecto a la cantidad de mensajes que se envían, a diferencia del escenario 1, en este caso, la cantidad de mensajes se pudo aumentar, pero se encontró un límite en los 30.000 mensajes.

Tabla 9. *Experimento 13,14,15 – RabbitMQ - Escenario 2*

Experimento	Cliente Subscriptor	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
10	1	10.000	0.5036	0.0712	0.1762	50%
10	2	10.000	0.5067	0.0722	0.1780	50%
11	1	25.000	0.9206	0.0779	0.2793	50%
11	2	25.000	0.9161	0.0765	0.2805	50%
12	1	30.000	0.7691	0.0732	0.3180	50%
12	2	30.000	0.7588	0.0715	0.3189	50%

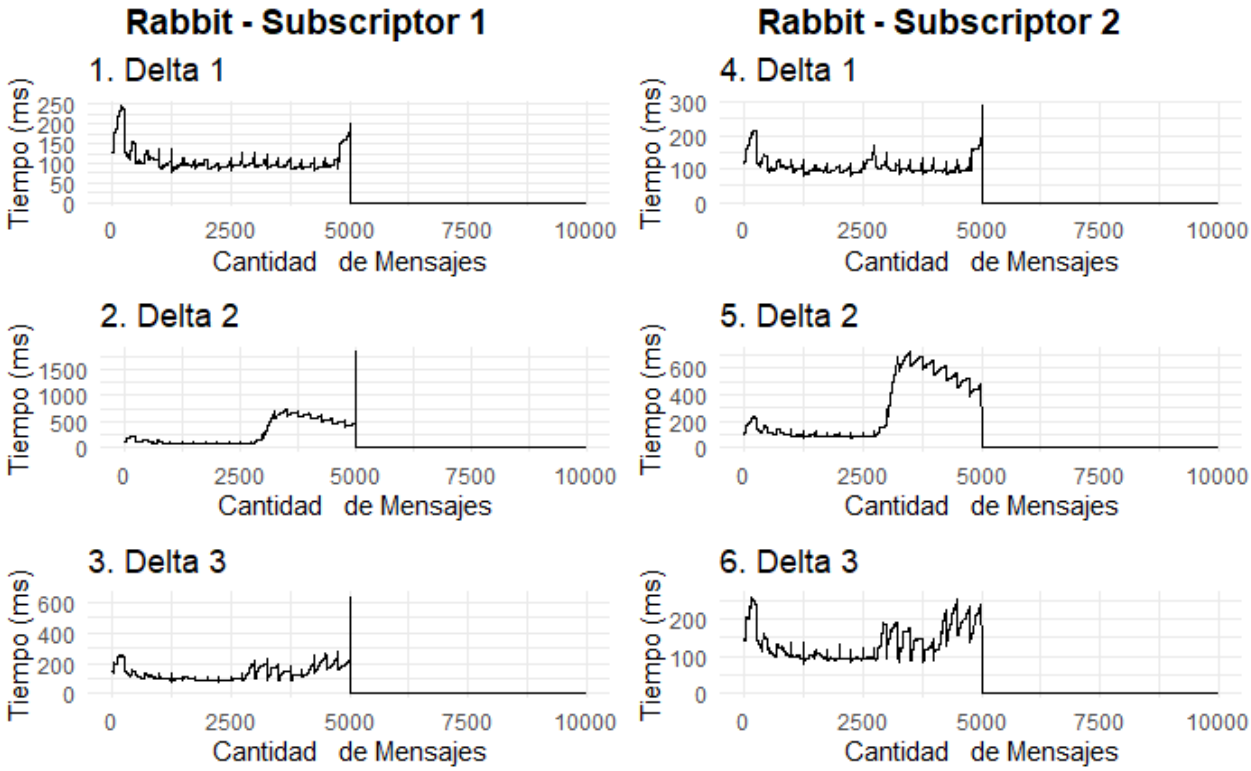
Nota. Elaboración propia.

Para el escenario 2 con RabbitMQ, se encuentra un cambio que difiere totalmente del escenario 1 y es la cantidad de mensajes exitosos que reciben ambos subscriptores, esto es debido a la configuración por defecto que posee RabbitMQ, de que al tener 2 o más subscriptores, este reparte los mensajes entre esa cantidad de subscriptores que escuchan a la cola, entonces los mensajes fueron divididos en tres para este caso. A diferencia de Mosquitto o Kafka, estos mensajes no están duplicados en ambos subscriptores.

Respecto a los tiempos no hay mucha diferencia alguna y mantienen un comportamiento muy estable en los tiempos de envío. Para la parte de las gráficas se nota lo hablado previamente de los mensajes divididos.

Las gráficas en estos experimentos están cortadas por la mitad debido a que cada subscriptor solamente recibió la mitad de los datos de cada publicador. Sobre los datos observados, es posible notar un comportamiento similar en cada Delta, siendo las gráficas de ambos subscriptores bastante idénticas con algún que otro aumento en los tiempos respecto a la otra grafica.

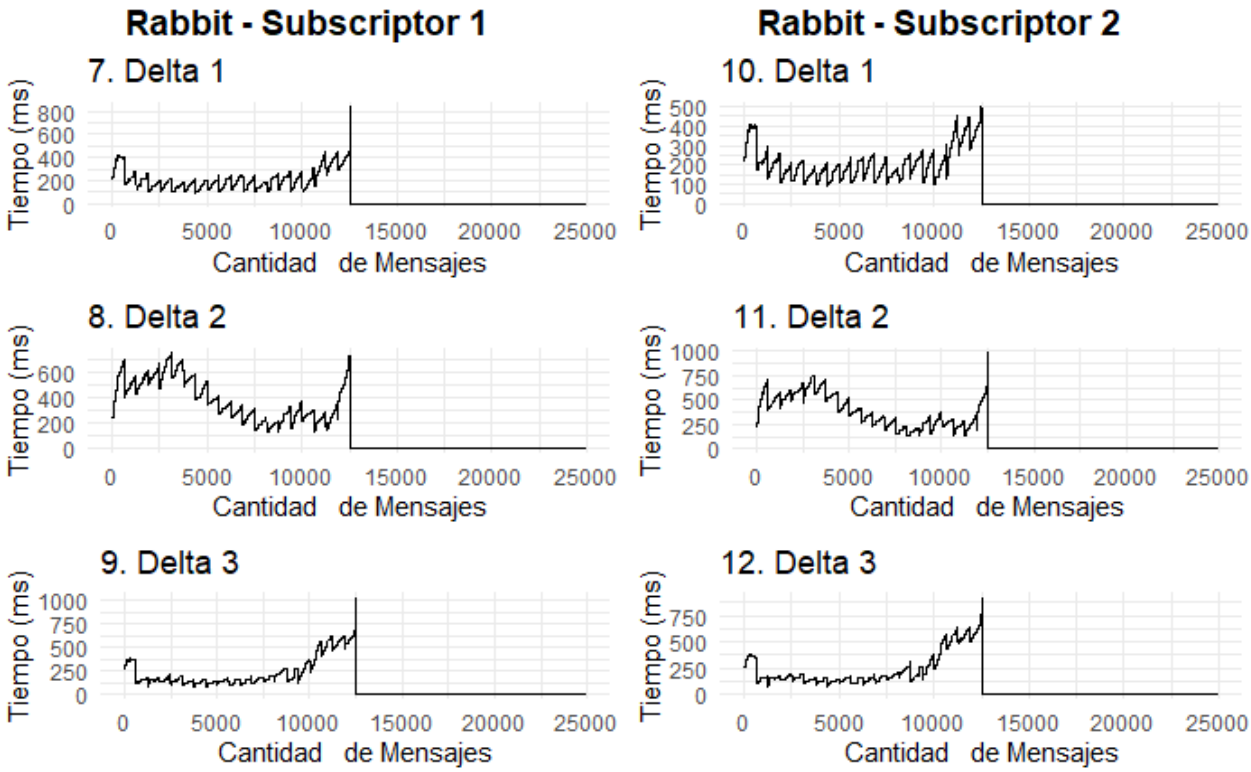
Grafica 7. *Experimento 13 – Rabbit - Escenario 2*



Nota. Elaboración propia.

Con esto, se identifica que las gráficas son similares a sus respectivos experimentos en el escenario 1, aunque el experimento 14 posee más datos en comparación al experimento 5, las gráficas muestran un comportamiento similar, donde hay picos al inicio y al final de los experimentos.

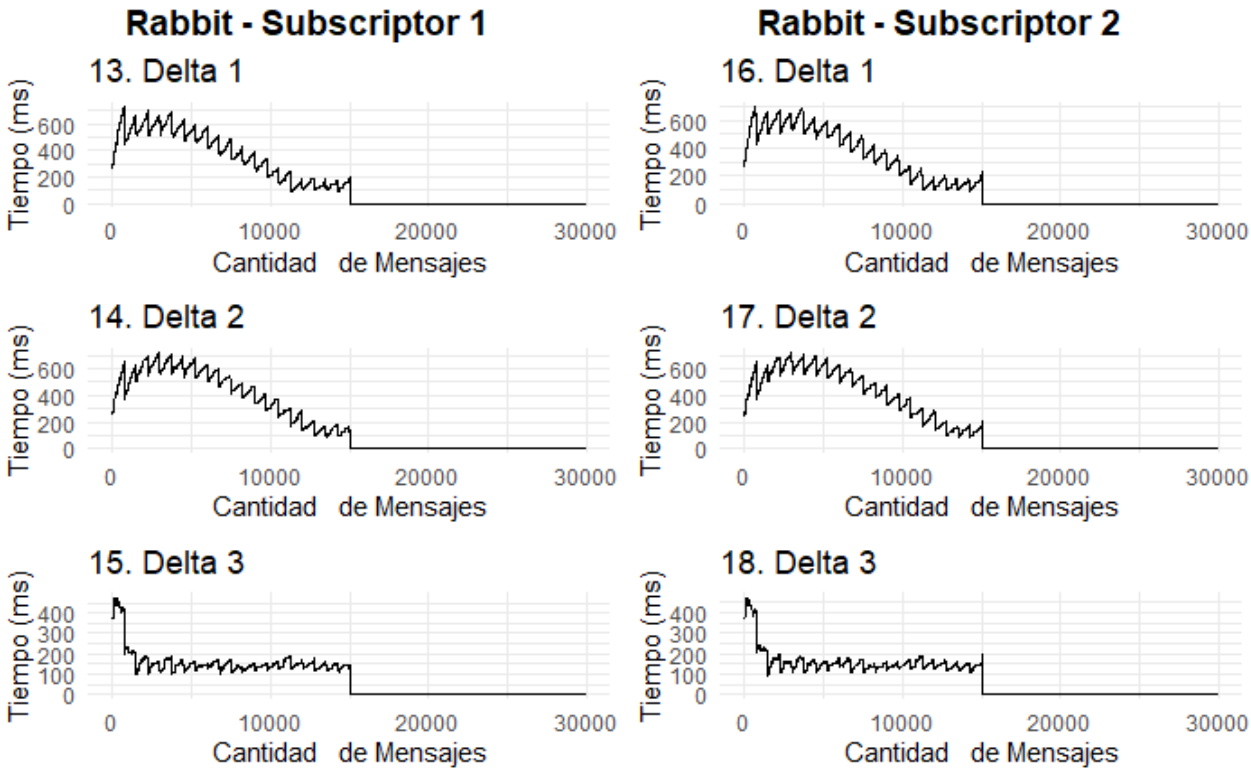
Grafica 8. *Experimento 14 – Rabbit - Escenario 2*



Nota. Elaboración propia.

En el experimento 15, se encuentra un caso similar al experimento 14, donde sus graficas son similares en comportamiento a las realizadas en el escenario 1, con la diferencia de que los tiempos de los mensajes son menores en este escenario.

Grafica 9. *Experimento 15 – Rabbit - Escenario 2*



Nota. Elaboración propia.

Con esto se identifica que para RabbitMQ al tener más subscriptores se puede dividir la carga de recibir mensajes de diferentes fuentes y replicarlos en todos los subscriptores que escuchen esos mensajes, esto da menores tiempos y menores presiones en el protocolo.

7.3.3. Experimentos en Apache Kafka

Finalizando los experimentos del escenario 2, están los resultados obtenidos de los experimentos en Kafka, iniciando con la tabla (tabla 11), se encontró que los datos presentan un ligero incremento en los tiempos en alguno de los subscriptores, como se había analizado previamente en el escenario 1, entre más datos se le dan a Kafka, esta toma más tiempo en procesarlos y al incluirle un publicador más, es notable el incremento del tiempo.

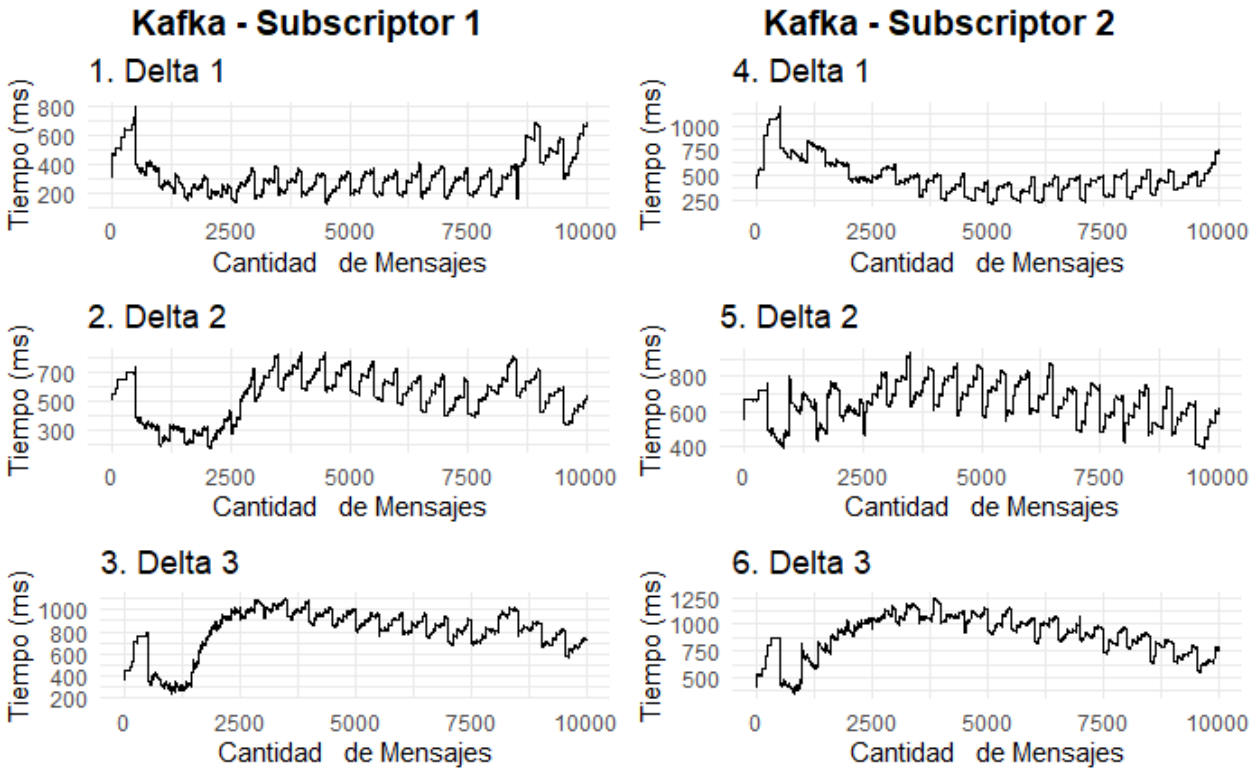
Tabla 10. *Experimento 16,17,18 – Apache Kafka - Escenario 2*

Experimento	Cliente Subscriptor	Cantidad de Mensajes	Tiempo Máximo (s)	Tiempo Mínimo (s)	Promedio del Tiempo (s)	% de Mensajes Exitosos
10	1	10.000	1.332	0.1057	0.5487	100%
10	2	10.000	1.4522	0.115	0.6754	100%
11	1	25.000	9.0152	0.2180	4.9081	100%
11	2	25.000	8.9167	0.2706	4.7862	100%
12	1	50.000	30.5094	0.3535	18.1776	100%
12	2	50.000	30.5425	0.661	18.2783	100%

Nota. Elaboración propia.

Las gráficas del experimento 16, se asemejan a su homólogo del escenario 1, este comportamiento grafico va disminuyendo a medida que entran más publicadores al broker, aumentando considerablemente los tiempos.

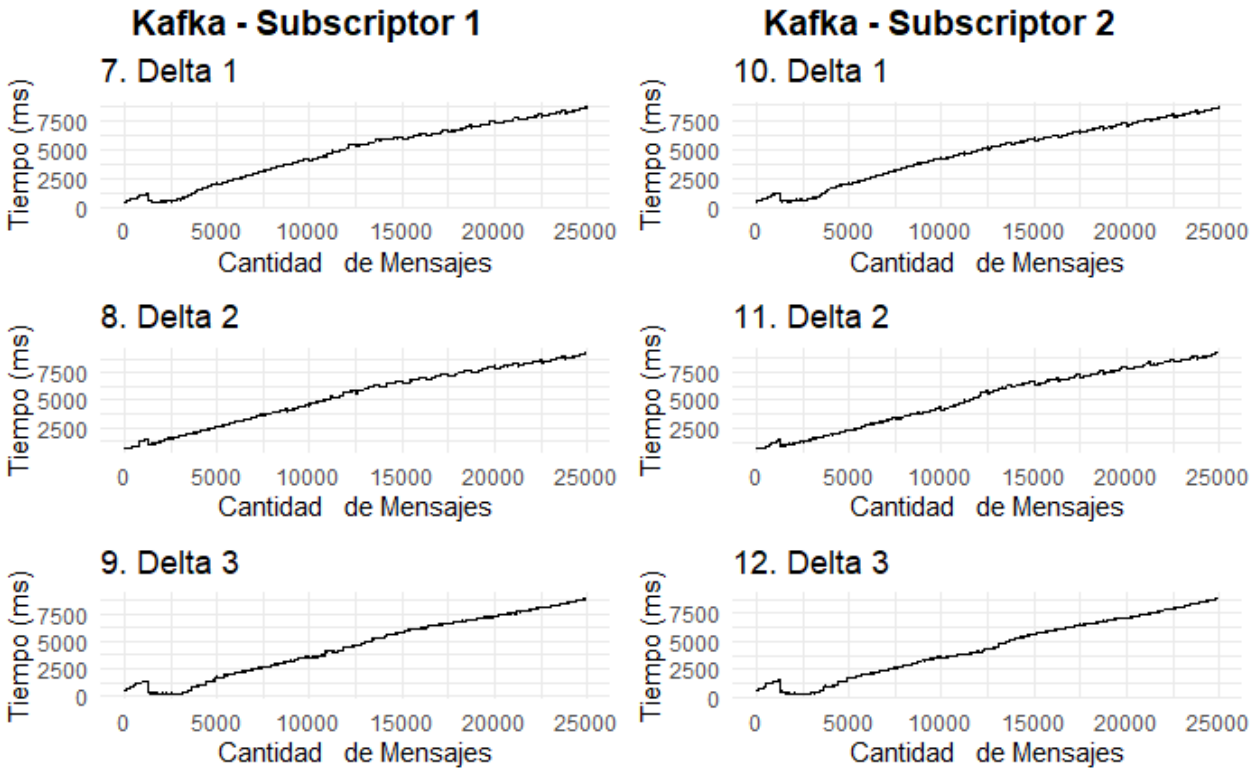
Grafica 10. *Experimento 16 – Kafka - Escenario 2*



Nota. Elaboración propia.

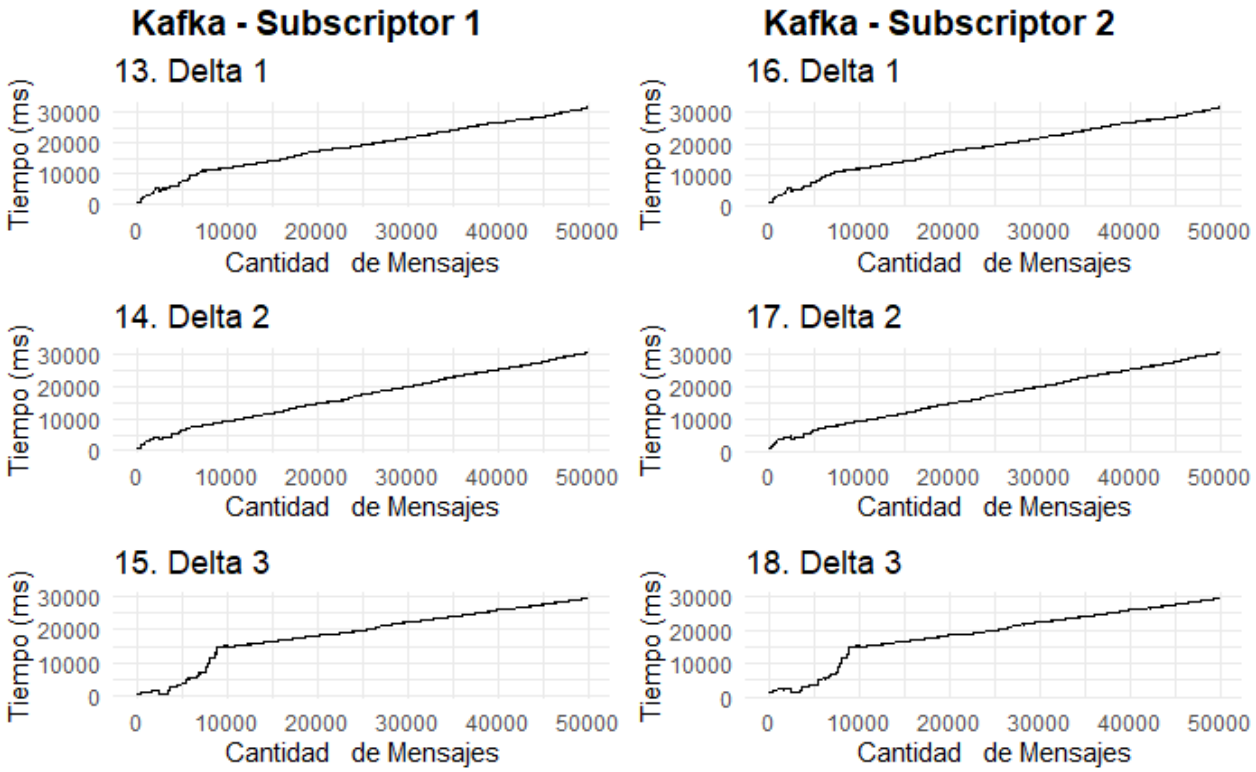
Para el experimento 17 y 18, volvemos a lo mismo visto previamente en el escenario 1, donde la gráfica comienza a tomar un comportamiento lineal, pero, al aumentar la cantidad de publicadores, se aprecia que este último es que toma una deformación más notada al inicio de las gráficas en comparación de los primeros dos publicadores.

Grafica 11. *Experimento 17 – Kafka - Escenario 2*



Nota. Elaboración propia.

Grafica 12. *Experimento 18 – Kafka - Escenario 2*



Nota. Elaboración propia.

7.4. Observaciones

En este apartado se habla de las observaciones que se encuentran al analizar las tablas y las gráficas de los respectivos protocolos, esto dará paso a la elaboración de las conclusiones.

7.4.1. Observaciones de Mosquitto

- Mosquitto tiene un patrón de comportamiento donde a mitad del procesado de los mensajes, este presenta unos aumentos en los tiempos, indicio de la carga ejercida al broker que demora en procesar mensajes que van llegando.
- Los QoS 1 y 2 están hechos para asegurar el envío del mensaje y que el mensaje no llegue repetido de forma respectiva, por ende, al estar bombardeando los brokers

con muchos mensajes, no da espacio a hacer todo el trayecto que toma el mensaje en estos QoS.

- A diferencia de QoS 1 y 2, QoS 0 envía el mensaje y no se preocupa de que le sucedió, en caso de desconectar ya sea el broker o el subscriptor, los mensajes se darán como perdidos de inmediato.
- Para el escenario 2, los datos mantienen un comportamiento similar a los vistos en el escenario 1, pero cada publicador presenta una forma diferente respecto a los demás publicadores, esto se debe a que la carga es distribuida para procesar cada publicador y enviar los mismos mensajes en ambos subscriptores.

7.4.2. Observaciones de RabbitMQ

- La mayor diferencia entre RabbitMQ y Mosquitto es que RabbitMQ utiliza un sistema de Exchange y colas que permiten que los mensajes se almacén en estas colas, permitiendo un almacenamiento de los mensajes. En caso de que alguna parte del sistema se cayera, las colas seguirán enviando los mensajes en ellas hasta que la parte que recibe el mensaje lo marque como leído, esto da paso a que los mensajes terminen siendo repetidos.
- En el Escenario 2 se habla de que, por defecto, RabbitMQ distribuye los mensajes entrantes cuando hay más de un subscriptor y que mediante esta configuración los mensajes enviados no son repetidos en cada subscriptor que llega, dependiendo de la situación puede ser útil o no y RabbitMQ aporta otras soluciones además de esta, una que envía el mismo mensaje en ambos subscriptores similar a Mosquitto y Kafka.

- Aparte de estas diferencias, RabbitMQ trabaja muy similar a como lo hace Mosquitto, siendo una alternativa muy popular y que se diferencia con estas configuraciones extras que aporta RabbitMQ.

7.4.3. Observaciones de Apache Kafka

- Kafka es el más diferente de los otros dos protocolos anteriores, este implementa su propia solución al patrón de envío de mensajes Publicador/Subscriptor, es por eso, por lo que los resultados son diferentes al aumentar la cantidad de datos a procesar.
- Si bien Kafka presenta tiempos elevados en los envíos de los datos, este se encargar de que los mensajes no lleguen de manera repetida igual que los QoS de Mosquitto de forma nativa, sin llegar a fallar a la hora de procesar grandes cantidades de datos.

8. Trabajo Futuro

A pesar del desarrollo tomado durante el proyecto, siempre hay factores posibles para mejoras, en ese caso, se puede enlistar varios aspectos a mejorar del proyecto:

- Mejoras en las capacidades de los Droplets cloud.
- Encontrar el punto donde los brokers no son capaces de seguir procesando información y compararla según los recursos que dispone la máquina virtual o PC que lo ejecuta.
- Variación en las opciones de configuración de AMQP y su Exchange.
- Mejorar la ejecución simultánea de los mensajes.
- Evaluar escenarios donde los publicadores envían lotes de datos en lugar de toda la información de golpe.
- Aumento de los protocolos evaluados.

9. Análisis de Resultados

9.1. Análisis de Comparativo

A continuación, se presenta una tabla en la que se presentan las ventajas y desventajas de los protocolos evaluados previamente con base en los resultados obtenidos.

Tabla 11. *Tabla Comparativa de Protocolos de Comunicación.*

Protocolo	Ventajas	Desventajas
MQTT	1. Funciona bien en dispositivos con bajos recursos de hardware. 2. Posee altas velocidades de transmisión de datos.	1. No dispone de colas de almacenamiento, por lo que los mensajes deben ser consumidos al instante. 2. Su QoS 1 y 2 no es recomendado cuando se requiere un rápido envío de llegada de mensajes. 3. Posee una baja escalabilidad en comparación a AMQP.
AMQP	1. Los mensajes recibidos por el broker son almacenados en una cola para su posterior consumo. 2. La cola de mensajería de AMQP es bastante configurable y se adapta a cualquier necesidad, además de que el funcionamiento de la misma presenta un aseguramiento de los mensajes.	1. Es un protocolo que demanda mayor capacidad de hardware, pero no tanta como Kafka. 2. Si los mensajes no son leídos, seguirá enviando el mismo mensaje hasta que se marque como leído, esto hace que sea más probable que los mensajes se dupliquen.

	3. Presenta una mayor escalabilidad y opciones a la hora del envío de los mensajes.	
Kafka	1. Presenta gran concurrencia de datos. 2. Se asegura de que los mensajes lleguen al destino a pesar de que se sacrifica la latencia. 3. Presenta una tolerancia a fallos y variación en las latencias más altas que en MQTT y AMQP.	1. Su cola de mensajería almacena todos los mensajes que le llegan y los mensajes no son enviados hasta que son solicitados. 2. El subscriptor es el encargado de solicitar los mensajes y de mantener la lectura de estos. 3. Requiere mayor capacidad de procesamiento que MQTT y AMQP

Nota. Elaboración propia.

9.2. Selección de protocolos apropiados

Ya al disponer de un análisis de resultados y un cuadro comparativo presentados en las secciones anteriores, se puede discutir acerca de que protocolo es el adecuando para usar en una arquitectura IoT, y es que no se puede hablar de una única solución genera para cualquier arquitectura, ya que la selección de un protocolo adecuado depende de los requerimientos de la misma. A continuación, se presentan tres ejemplos de requerimientos de arquitecturas IoT con el fin de establecer qué protocolo y broker serían los adecuados para dicha arquitectura:

a) Se requiere una arquitectura IoT no muy robusta en la cual se puedan transmitir datos de forma rápida y sin mucha exigencia de hardware.

En este caso se pueden hablar de dos brokers que serían adecuados para este tipo de arquitecturas IoT que son Mosquitto y RabbitMQ, ya que en ambos se asegura un gran rendimiento en cuanto a velocidad para volúmenes de datos no muy grandes.

b) Se requiere una arquitectura IoT no muy robusta en la cual se puedan transmitir datos de forma rápida, pero teniendo en cuenta que en el caso de que no se puedan consumir estos datos por algún cliente en el instante, estos datos deben persistir durante un tiempo antes de ser eliminados.

En este caso se pueden hablar de un broker implementado con RabbitMQ, ya que este dispone de un rendimiento de transmisión de datos además de que también dispone de una cola por la cual pasan los datos que puede ser configurada para distribuir los datos sobre los clientes consumidores.

c) Se requiere una arquitectura IoT robusta y escalable en la cual se puedan transmitir grandes volúmenes datos de forma segura, siendo estos almacenados temporalmente para ser consumidos por grandes cantidades de clientes.

En este caso se pueden hablar de un broker implementado con Apache Kafka, ya que este funciona a manera de broker soportando grandes volúmenes de datos que son almacenados temporalmente por el mismo y siendo distribuidos posteriormente de manera segura.

9.3. Conclusiones

La evaluación de modelos de comunicación implementados de tres de los protocolos de comunicación IoT más populares en el estado del arte, se logró mediante la selección de los criterios que permitieron evaluar los aspectos de los protocolos de comunicación en el caso de uso de un Smart Campus.

Se logró identificar las ventajas y desventajas de los protocolos de comunicación IoT evaluados con lo que se pudo determinar que no existe un protocolo que se ajuste a las todas necesidades de comunicación en una infraestructura IoT, por el contrario, este dependerá de las mismas.

Se estableció una arquitectura para las pruebas en un entorno Cloud con contenedores de igual capacidad de hardware el cual fue adecuado ya que mediante este se pudo tener un ambiente de pruebas más controlado, descartando que factores como rendimientos de procesamiento de datos pudieran influir negativamente en los resultados de las pruebas.

Se logró comprobar mediante las pruebas la eficiencia de los modelos de comunicación implementados, pudiendo realizar una comparación de sus ventajas y desventajas, así como establecer casos de uso en los cuales sería adecuada la implementación de alguno.

Referencias Bibliográficas

Al-Masri, E., Kalyanam, K., Batts, J., Kim, J., Singh, S., Vo, T., & Yan, C. (2020). Investigating messaging protocols for the Internet of Things (IoT). 8, 94880-94911.

Al-Sarawi, S., Anbar, M., Alieyan, K., & Alzubaidi, M. (2017). Internet of Things (IoT) communication protocols. In 2017 8th International conference on information technology (ICIT) (pp. 685-690).

Amaguaya, R. (2020). Análisis comparativo a nivel transaccional de brokers MQTT (Mosquitto, Mosca y Emq) con respecto a la disponibilidad en infraestructuras IoT ante ataques DDoS.

Amaya, E. (2018). Redes de computadoras. Introducción a las redes, necesidad de una red, tipo y equipos de redes, topología de una red, diseño de redes, instalación y administración de redes LAN.

Carlos Pereira, João Cardoso, Ana Aguiar, Ricardo Morla (2018). Benchmarking Pub/Sub IoT middleware platforms for smart services

Burak H. Çorak, Feyza Y. Okay, Metehan Güzel, Şahin Murt, Suat Ozdemir. (2018). Comparative Analysis of IoT Communication Protocols

De la Cuadra, E. (1996). Internet: Conceptos Básicos. Cuadernos de Documentación Multimedia, 5, 35.

Froiz, I., Fraga, P., Fernández, T. (2020) Decentralized P2P Broker for M2M and IoT Applications. Proceedings. 54(1):24. https://doi.org/10.3390/proceedings2020054024

Garg, N. (2013). Apache kafka. Birmingham: Packt Publishing. (pp. 30-31).

Glover, B., & Bhatt, H. (2006). RFID essentials. " O'Reilly Media, Inc.".

Gómez, C., & Paradells, J. (2010). Wireless home automation networks: A survey of architectures

and technologies. IEEE Communications Magazine, 48(6), 92-101.

Gómez, C., Veras, J., Vidal, R., Casals, L., & Paradells, J. (2019). A sigfox energy consumption model. Sensors, 19(3), 681.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7)

Jiménez, H., Cárcamo, E., & Pedraza, G. (2020). Plataforma Software Extensible para Campus Inteligente Basada en Microservicios. Revista Ibérica de Sistemas e Tecnologias de Informacion, 270–282.

Kumar, N., & Mallick, P. K. (2018). The Internet of Things: Insights into the building blocks, component interactions, and architecture layers. Procedia computer science, 132, 109-117.

Leiner, B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., & Wolff, S. (1999). Una breve historia de Internet. Revista Novática. (130), 131.

Madakam, S., Lake, V., Lake, V., & Lake, V. (2015). Internet of Things (IoT): A literature review. Journal of Computer and Communications, 3(05), 164.

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In 2017 IEEE international systems engineering symposium (ISSE) (pp. 1-7). IEEE.

Naranjo, D., Buenaño, D., & Mejía, I. (2016). Evolución de la tecnología móvil. Camino a 5G. Revista Contribuciones a las Ciencias Sociales.

Olsson, J. (2014). 6LoWPAN demystified. Texas Instruments, 13.

Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2017). Cloud container technologies: a state-of the-art review. IEEE Transactions on Cloud Computing, 7(3), 677-692

Rao, T., & Haq, E. (2018). Security challenges facing IoT layers and its protective measures. International Journal of Computer Applications, 179(27), 31-35.

Snyder, B., Bosanac, D., & Davies, R. (2017). Introduction to apache activemq. Active MQ in action, 6-16.

Soni, D., & Makwana, A. (2017). A survey on mqtt: a protocol of internet of things (iot). In International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017) (20).

Weinstein, R. (2005). RFID: a technical overview and its application to the enterprise. IT professional, 7(3), 27-33.