UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Universitat Politècnica de València

Departamento de Informática de Sistemas y Computadores

Despliegue de Clusters de Brokers MQTT Usando SBCs de Bajo Coste

TRABAJO DE FIN DE MÁSTER

Máster Universitario en Ingeniería de Computadores y Redes

Autor

Sandra Stefany Rodríguez Vinueza

Director

Dr. Pietro Manzoni - Universitat Politècnica de València

Dedicatoria

El presente proyecto lo dedico a mi familia, en especial a mis padres que son mi pilar fundamental, a mis hermanos y a mi abuelita que siempre estuvieron apoyándome en todos los momentos difíciles impulsándome a seguir adelante, y compartieron junto a mí todos los momentos alegres.

Agradecimientos

Mis más sinceros agradecimientos a Dios en primer lugar por todas las bendiciones y gracias que me da y por permitirme concluir esta etapa de mi vida.

A mis padres y mis hermanos por ser mi apoyo en todo momento y por siempre estar presentes cuando los necesito.

Al Departamento de Informática de Sistemas de Computadores (DISCA) y al Grupo de Redes de Computadores (GRC) de la Universidad Politécnica de Valencia, en especial al Dr. Pietro Manzoni por los grandes conocimientos proporcionados y por su gran apoyo, paciencia y ayuda en todo momento.

Resumen

MQTT es uno de los protocolos de IoT que más está creciendo en los últimos años. Por su diseño de comunicación (tipo estrella), los dispositivo clientes dependen directamente de un único nodo central denominado “broker”. Al crecer el número de clientes es normal que la carga de los brokers no pueda ser gestionada por un único dispositivo y se tengan que crear “clusters de brokers” para poder ofrecer tiempos de respuesta reducidos y la mínima eficiencia necesaria.

En este TFM , nos centramos en estudiar los beneficios y los inconvenientes de desplegar clusters de brokers MQTT usando SBCs de bajo coste, haciendo uso combinado de la tecnología de contenedores Docker.

El uso de esta tecnología permite simplificar tanto la instalación como la puesta en marcha de programas muy complejos, contribuyendo a la gestión y la automatización de los módulos software.

El objetivo es demostrar la conveniencia de esta aproximación que permitiría una evolución más rápida hacia soluciones edge para el IoT.

Palabras claves: MQTT, Clusters de brokers, Iot, Docker, Raspberry.

Abstract

MQTT is one of the IoT protocols that has been growing most in recent years. Due to their communication design (star type), client devices depend directly on a single central node called “broker”. As the number of clients grows, it is normal that the load of the broker cannot be managed by a single device and “brokers clusters” have to be created in order to offer reduced response times and the minimum necessary efficiency

In this TFM, we focus on studying the benefits and inconvenience of deploying MQTT brokers cluster using low-cost SBCs, combined with the use of Docker container technology.

The use of this technology allows to simplify the installation and the start-up of very complex programs, contributing to the management and automation of the software modules.

The objective is to demonstrate the convenience of this approach that would allow a faster evolution towards edge solutions for the IoT.

Keywords: MQTT, Broker clusters, Iot, Docker, Raspberry.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El uso de dispositivos *Internet of Things (IoT)* se encuentra en gran aumento con el desarrollo de las ciudades inteligentes, lo que genera que la comunicación entre estos dispositivos y la transmisión de información a otros para la gestión y procesamiento de los datos se torne muy importante.

Existen varios protocolos que permiten transmitir mensajes entre los dispositivos Internet of Things (IoT) en tiempos cortos; Algunos de los protocolos de mensajería más utilizados en este tipo de comunicaciones son Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP) y Hypertext Transfer Protocol (HTTP) [4], unos exigen mayor cantidad de recursos que otros o utilizan mayor ancho de banda que otros.

Generalmente los mensajes que se transmiten en este tipo de comunicaciones son muy pequeños con peso de pocos bytes y dependiendo del caso son esporádicos, esto produce que el ancho de banda utilizado durante la comunicación sea mínimo.

Para escoger un protocolo de mensajería adecuado para los sistemas de IoT, es recomendable tener claro el objetivo del sistema IoT y sus requerimientos al momento de enviar mensajes / datos.

Uno de los protocolos de comunicación más utilizados por dispositivos IoT es MQTT, por su sencillez, ligereza, bajo consumo de energía, robustez y fiabilidad; Este protocolo utiliza comunicación publicador / suscriptor que ofrece muchas ventajas, se utiliza un ″broker″ como nodo central que gestiona todo la comunicación entre los clientes, el tamaño del mensaje MQTT es muy pequeño por lo que no genera mucha carga en la transmisión, ocupa poco ancho de banda y produce

poca latencia, también consume poca energía debido a que utiliza pocos recursos. La fiabilidad de MQTT es mayor que la de otros protocolos gracias a que utiliza *Quality of service (QoS)*.

1.1 Motivación

Con el paso del tiempo y la implementación de ciudades inteligentes alrededor del mundo, el uso de dispositivos IoT se encuentra en gran aumento. La comunicación entre estos dispositivos debe ser ágil y segura ya que en muchas ocasiones transmiten datos de sensores, dispositivos de acceso a algún lugar, entre otros. En MQTT los clientes no tienen comunicación entre ellos si no que la comunicación se realiza a través del broker que es el encargado de la gestión de los mensajes, al ser uno de los protocolos más utilizados para la comunicación las redes que lo utilizan son cada vez más grandes, por lo que si el número de clientes con sesiones simultáneas aumenta se puede generar una degradación del servicio en la comunicación ya que la carga no puede ser gestionada por un único dispositivo. A causa de esto se ve la necesidad de crear clusters de brokers MQTT para la gestión de los mensajes y proporcionar tiempos de transmisión bajos. En la actualidad se tienen varios clusters MQTT que tienen costos elevados a causa de los grandes servidores sobre los que se monta este servicio. También se tienen varios proyectos para hacer clusters propios en los que se requiere un dispositivo de computo para cada broker del cluster lo que lo hace que no sea de acceso a sectores de bajos recursos. En este trabajo se propone realizar un cluster de brokers MQTT, utilizando Raspberry PI como dispositivos de computo de bajo coste [5] e implementado los brokers sobre Docker para poder obtener portabilidad en caso de tener algún daño en los dispositivos. El objetivo de este trabajo es demostrar la conveniencia de esta aproximación que permitiría una evolución más rápida hacia soluciones edge para el IoT.

1.2 Objetivos

Este Trabajo Fin de Máster tiene como objetivos:

- Desplegar un cluster de 3 brokers MQTT usando Single Board Computers (SBCs) de bajo coste y la tecnología de contenedores Docker.
- Validar la efectividad del cluster mediante la conexión de clientes simultáneos a la red.
- Analizar y comparar los tiempos de transmisión de los mensajes con un solo broker y con el cluster en diferentes condiciones.

1.3 Metodología

El despliegue de este proyecto se realizó con computadoras de placa única (*Single Board Computers (SBCs)*) y bajo costo. Se utilizó 4 raspberrys Pi modelo 1 y 1 router para la interconexión dentro de la misma red.

Se implementó en 3 de las placas utilizando la tecnología Docker un broker de Eclipse Mosquitto para activar el cluster de broker's MQTT. En la cuarta placa se implementó con High Availability Proxy (HAProxy) un balanceador de carga como nodo master del cluster.

Se hicieron varias pruebas en las que se realizó variaciones en la cantidad de clientes MQTT simultáneos y en la frecuencia de publicaciones de mensajes para obtener datos de los tiempos de transmisión y perdida de mensajes. Con el análisis de los datos obtenidos en las pruebas se pudo evaluar la conveniencia de esta aproximación.

1.4 Estructura

El presente Trabajo de Fin de Máster consta de cinco capítulos. En el primer capítulo se detalla la problemática sobre la cual se busca trabajar, el objetivo general y los objetivos específicos del trabajo. En el segundo capítulo se menciona el estado del arte del protocolo MQTT, de Docker y Raspberry. En el tercer capítulo se desarrolla el diseño de este trabajo junto con la metodología y los elementos con los que se realizó. En el capítulo cuatro se describe las pruebas, resultados y análisis realizados al diseño del cluster. En el quinto y último capítulo se exponen las conclusiones obtenidas en base al análisis realizado a las resultados de las pruebas.

Capítulo 2

Antecedentes

El número de dispositivos conectados a Internet alcanzó los 22 mil millones en todo el mundo a mediados del presente año, y se espera que alcance los 47 mil millones en los próximos dos años [1]. Gran parte de estos son dispositivos *Internet of Things (IoT)* y se utilizan en varios sectores, entre estos industrias críticas como la atención médica, la aviación, la automoción, etc..

Los dispositivos IoT en su gran mayoría son dispositivos de pocos recursos, lo que influye en limitaciones para el procesamiento de datos, almacenamiento y uso energético, por lo que para poder escoger adecuadamente los diferentes protocolos que se utiliza en IoT se debe tener en cuenta ciertos aspectos como que sea escalable, fiable o que tenga mecanismos de protección para entornos en los que el canal de comunicación no sea el más óptimo, además de que en base a la aplicación que se utilice poder minimizar el tamaño del paquete del mensaje con su cabecera para tener menor consumo de ancho de banda y menores tiempos de transmisión.

Al usar MQTT en grandes redes el número de dispositivos conectados por broker y los mensajes que se registran son por segundo. Esta es exactamente la razón por la que las implementaciones de IoT fijan constantemente la esperanza en los protocolos de mensajería excepcionales como MQTT desarrollados específicamente para la comunicación M2M dentro de las aplicaciones de IoT. Sin embargo, casi todas las aplicaciones de cliente MQTT no están diseñadas para manejar la inmensa cantidad de mensajes MQTT por segundo. Por lo tanto, se sobrecarga debido a la alta frecuencia de mensajes entre los clientes. Además, también, el procesamiento de mensajes de alto rendimiento es necesario para manejar la compleja implementación de IoT [1].

2.1 Modelo de comunicación Publicador / Suscriptor

Está basado en la comunicación máquina a máquina donde se tiene un cliente (publicador) que envía un mensaje a uno o a varios clientes (suscriptores), dando una alternativa al modelo cliente / servidor. Esta comunicación se realiza sin que los clientes tengan conocimiento el uno del otro, ya que se tiene un nodo central que gestiona toda la comunicación y redirige los mensajes a los suscriptores correspondientes, todos los clientes (publicadores y suscriptores) solo conocen al broker. El hecho de que los clientes que publican y reciben los mensajes no tengan conocimiento unos de otros dota a los sistemas publicador / suscriptor de:

- Desacoplamiento en el tiempo, por lo que no tienen que ejecutarse a la vez.
- Desacoplamiento de sincronismo, por lo que los clientes no tienen que detener las operaciones que estén realizando.

Por estos motivos el modelo de comunicación publicador / suscriptor se adapta en gran medida a los requerimientos que suelen presentar los sistemas de IoT, por ello muchos de los protocolos IoT se basan en este modelo. [6]

2.2 MQTT

Message Queue Telemetry Transport (MQTT) (Transporte de telemetría de cola de mensajes) es un protocolo de comunicación ideado por IBM que es de libre uso. Su tipo de conectividad es *Machine to Machine (M2M)* (máquina a máquina), tiene bajo consumo de energía y muy poco consumo de ancho de banda por lo que puede ser utilizado en la mayoría de dispositivos empotrados con pocos recursos, lo que lo hace muy recomendable para comunicación entre dispositivos de IOT.

Cuando se utiliza la definición tiempo real en MQTT se habla de segundos, debido a que las aplicaciones que lo utilizan como protocolo de comunicación tienden a ser más lentas.

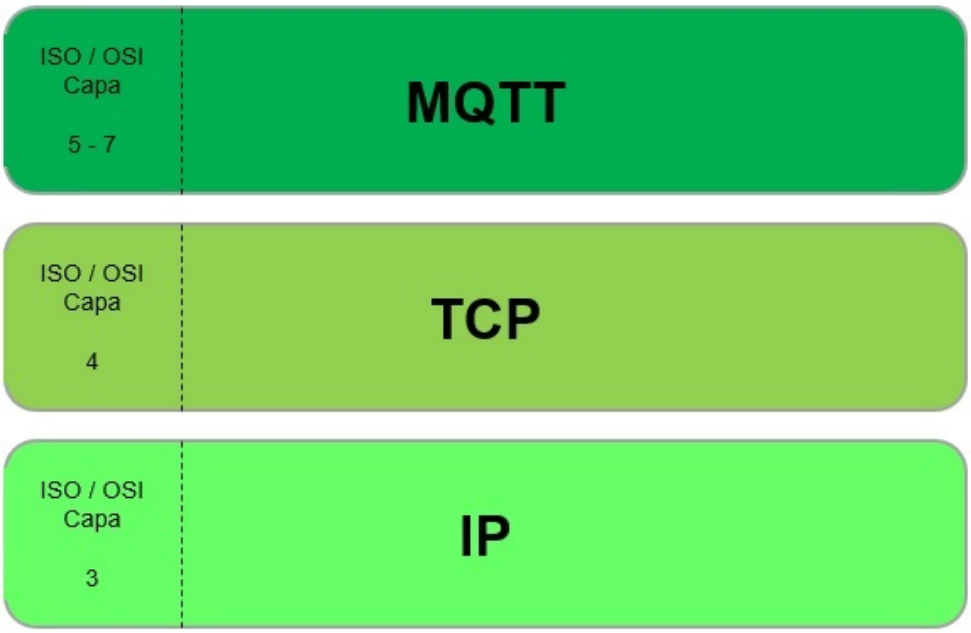

Figura 1: MQTT relacionado al modelo OSI

El protocolo MQTT se ubica en las capas superiores del modelo OSI, y se fundamenta en TCP/IP como se puede observar en la figura (1)

Una sesión MQTT se divide en cuatro etapas: conexión, autenticación, comunicación y terminación. Un cliente comienza creando una conexión TCP/IP con el broker utilizando un puerto estándar o un puerto personalizado definido por los operadores del broker. Al crear la conexión, es importante reconocer que el servidor puede continuar una sesión antigua si se le proporciona una identidad de cliente reutilizada.

Los puertos estándar son el 1883 para la comunicación no cifrada y el 8883 para la comunicación cifrada mediante SSL/TLS. Durante el handshake SSL/TLS, el cliente valida el certificado para autenticar el servidor. MQTT tiene la capacidad de establecer comunicaciones cifradas, lo que aporta una capa extra de seguridad [7].

2.2.1 Arquitectura

MQTT usa topología publicador/suscriptor extremadamente ligera basada en *Transmission Control Protocol/Internet Protocol (TCP/IP)* lo que le permite tener re-

conocimiento de la sesión con los clientes de manera continua. La cual sigue el modelo de una topología estrella ya que utiliza un nodo central conocido como "broker", el cual tiene una capacidad máxima de 10 000 clientes.

Con el fin de tener bajo costo de transmisión de datos MQTT maneja su comunicación por medio de eventos.

El broker es el servidor encargado de direccionar todos los mensajes dentro de la red y enviarlos a los clientes que los requieran, motivo por el cual los clientes no se conocen entre ellos ya que únicamente tienen comunicación con el broker como se observa en la figura (2).

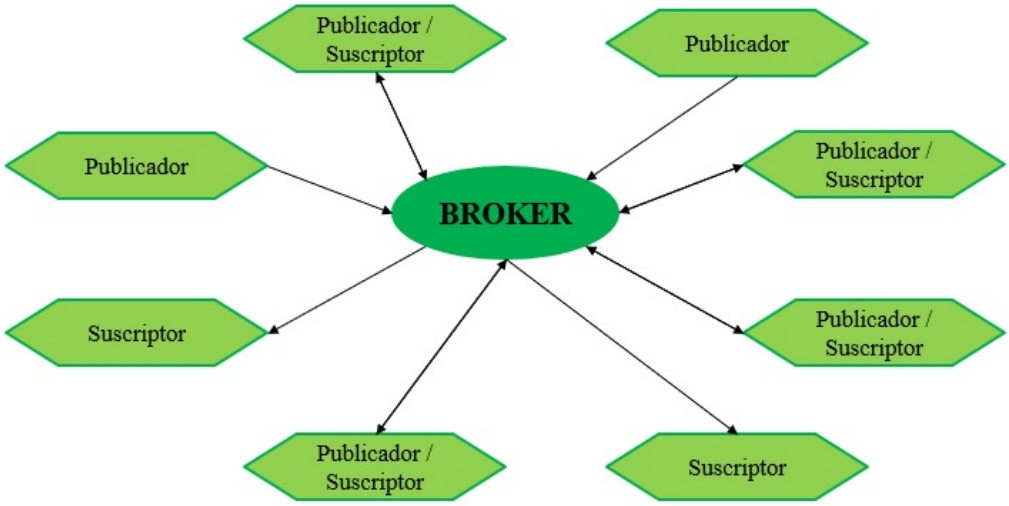

Figura 2: Clientes (Publicadores y Suscriptores) conectados al broker

Durante la fase de comunicación el cliente puede suscribirse a diferentes topics, realizar publicaciones, ping y cancelar la subscripción de alguno o de todos los topics. En la publicación se envía un bloque binario de datos, a todos los clientes suscritos al topic definido por el publicador.

En la comunicación se utiliza "topics"(temas) de manera que los clientes que requieran información de algún nodo o sensor deben suscribirse al topic del mismo para que cuando el nodo publicador envíe datos al broker, este los reenvíe a todos los clientes que estén suscritos a ese topic como se visualiza en la figura (3).

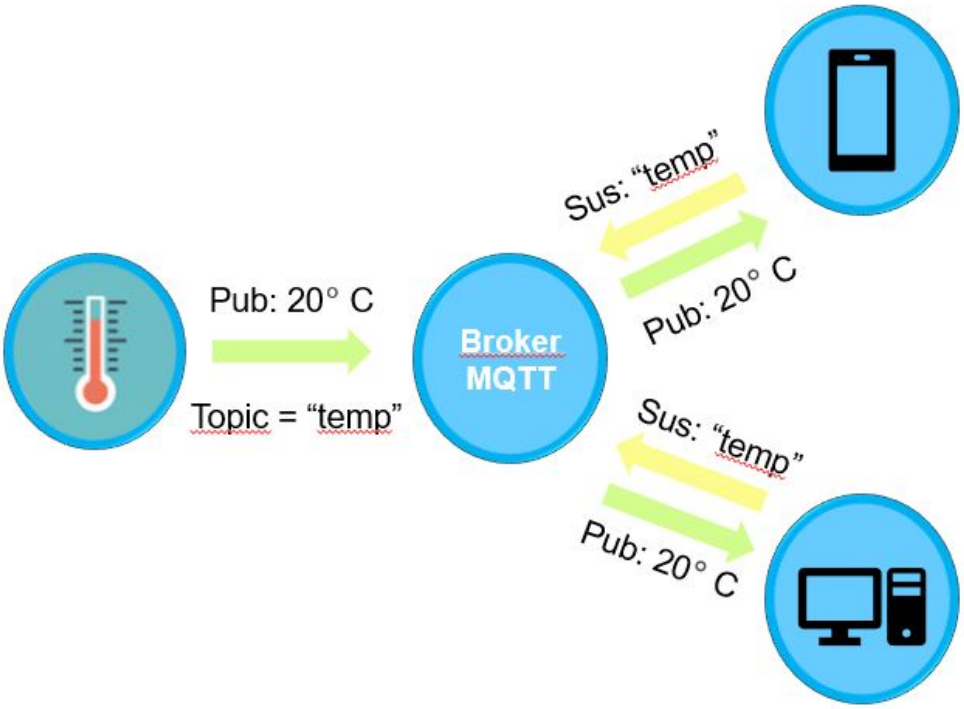

Figura 3: Comunicación entre publicador y suscriptor en MQTT

Con esto para que 2 o más clientes puedan tener interacción entre ellos deben estar suscritos al mismo topic; La comunicación es bidireccional, es decir que los clientes pueden ser suscriptores y publicadores al mismo tiempo; De igual manera la comunicación se puede realizar de un cliente a otro, o de un cliente a muchos, como en la figura (3) que es de 1 cliente publicador a 2 clientes suscriptores.

En la suscripción o publicación se distingue mayúsculas y minúsculas, los topics tienen una estructura jerárquica en la que cada nivel se separa con `"/"` y asemeja a un mapa conceptual como se observa en la figura (4).

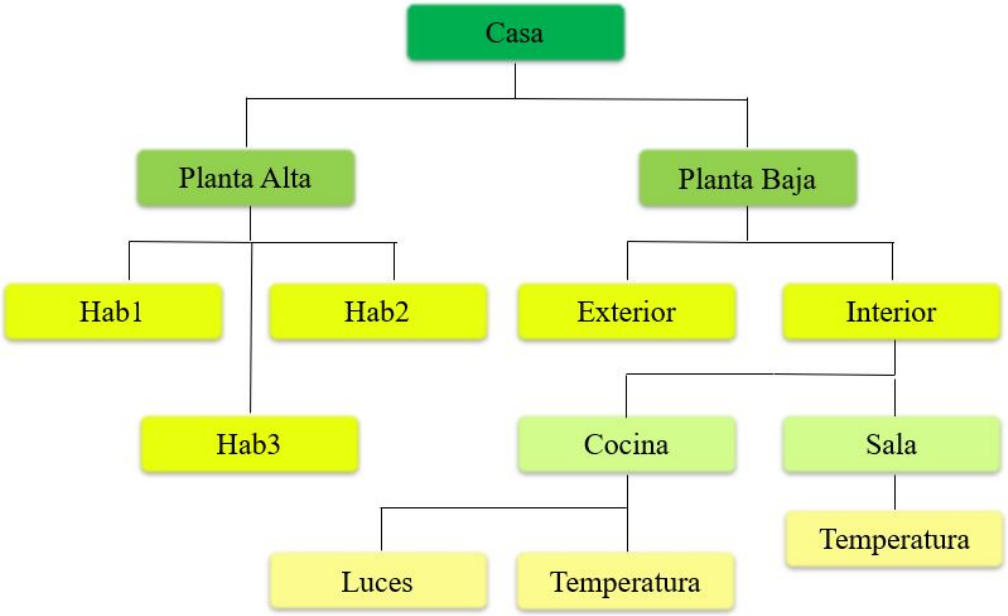

Figura 4: Esquema jerárquico de topic

Un cliente puede suscribirse a un topic específico o también puede suscribirse a varios topics simultáneamente utilizando comodines en el momento de la suscripción. Los comodines pueden ser utilizados únicamente para la suscripción pero no para la publicación de datos.

Existen dos comodines en MQTT que se pueden usar para la suscripción a varios topics:

1. **Comodín de nivel simple + :** este comodín reemplaza un nivel del topic, al utilizarlo, el cliente se suscribe a todos los topics que estén en el mismo nivel donde se coloca el comodín. Por ejemplo, con respecto a la figura (4) si el cliente se suscribe al topic *"Casa/Planta Baja/Interior/+/Temperatura"* va a recibir todas las publicaciones con los topics *"Casa/Planta Baja/Interior/-Cocina/Temperatura"* y *"Casa/Planta Baja/Interior/Sala/Temperatura"*, ya que estos dos topics están en el mismo nivel bajo *Interior* y que como siguiente nivel tienen *Temperatura*.

2. **Comodín multinivel # :** este comodín reemplaza varios niveles del topic y se coloca al final, al utilizarlo el cliente se suscribe a todos los topics que estén bajo el último nivel de topic que se detalló. Por ejemplo, con respecto a la figura (4) si el cliente se suscribe al topic *"Casa/Planta Baja/# "*

va a recibir todas las publicaciones con los topics *"Casa/Planta Baja/Exterior"*, *"Casa/Planta Baja/Interior/Cocina/Luces"*, *"Casa/Planta Baja/Interior/Cocina/Temperatura"* y *"Casa/Planta Baja/Interior/Sala/Temperatura"*, ya que todos estos topics se encuentran bajo *Planta Baja* que es el último nivel antes del comodín.

2.2.2 Estructura del mensaje MQTT

MQTT es un protocolo ligero. Cada mensaje consta de una cabecera fija, una cabecera variable (opcional), y la carga útil de mensaje (opcional) que esta compuesta por el mensaje y el *Quality of service (QoS)*, tal como se muestra en la figura (5).

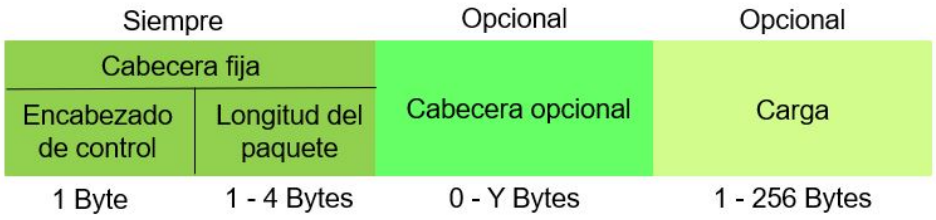

Figura 5: Estructura del mensaje MQTT

MQTT no usa metadatos en la cabecera lo que hace que el mensaje sea más ligero; Soporta objetos binarios grandes (*Binary Large Objects (BLOBS)*) de mensajes con un tamaño de hasta 256 MB de información, esto varia dependiendo de la aplicación y del usuario [8].

El código de control depende del tipo de mensaje enviado, se tiene varios dependiendo de la comunicación, por ejemplo para la suscripción a topics se utilizan paquetes SUBSCRIBE/SUBACK, para eliminar la suscripción UNSUBSCRIBE/UNSUBACK, se utiliza paquetes PINGREQ/PINGRESP hacia el broker, la función de esta operación es mantener la sesión activa, también se tienen paquetes como CONNECT, CONNACK, PUBLISH, DISCONNECT entre otros.

2.2.3 Calidad de Servicio MQTT (QoS)

MQTT utiliza niveles de Calidad de Servicio (QoS) para compensar que las redes de comunicación sean poco fiables; El QoS es un acuerdo entre el publicador y

el suscriptor para garantizar el envío de mensajes específicos, ya que es el que garantiza la entrega del mensaje. MQTT soporta 3 niveles de QoS:

1. **Al menos una vez (QoS 0)** en este nivel el mensaje se envía una vez pero no se confirma si hay recepción por parte del broker o del suscriptor, esto depende del entorno de comunicación lo que puede generar pérdidas de mensajes ya que no se tiene retransmisiones.

2. **Como máximo una vez (QoS 1)** en este nivel el mensaje se envía hasta que se confirme que hay recepción por parte del broker o del suscriptor, el mensaje se reenvía cada determinado tiempo hasta asegurar la recepción de por lo menos 1 mensaje, en este caso se puede dar la posibilidad de que se tenga mensajes duplicados.

3. **Exactamente una vez (QoS 2)** en este nivel el mensaje se envía hasta que se confirme que haya un único mensaje recibido por parte del broker o del suscriptor, ya que no se quiere mensajes duplicados, se debe tener en cuenta que esto genera una sobrecarga en la comunicación.

Al momento de escoger un nivel de servicio para nuestra comunicación se debe tener en cuenta que en MQTT la comunicación se divide en 2 tramos, del publicador al broker y del broker al suscriptor; En el primer tramo el publicador es quien asigna el QoS al mensaje que envía y el broker lo recibe con el mismo nivel asignado, en el segundo tramo el broker envía el mensaje al suscriptor con el QoS que cada suscriptor escogió al momento de la suscripción al topic, es decir que si el suscriptor escoge un QoS menor que el del publicador, el broker lo va a enviar con el QoS menor.

La latencia es proporcional al QoS, es decir que a mayor QoS se tiene mayor latencia y menor rendimiento de la red, por lo que para escoger el QoS más apropiado para cada aplicación se debe verificar si es imprescindible o no que se tenga perdidas de mensajes, ya que de no ser así se recomienda escoger el QoS bajo para tener menor latencia.

2.2.4 Ventajas de MQTT

El uso del protocolo MQTT tiene varias ventajas como

- Es asíncrono, tiene escalabilidad, los clientes no se encuentran acoplados entre ellos

- Usa calidad de servicio lo que ayuda en los momentos en los que la conexión a internet no es muy confiable

- Es un protocolo ligero, su comunicación es por medio de mensajes cortos, por lo que tiene bajo consumo de ancho de banda
- No requiere de muchos recursos para su ejecución por lo que tiene bajo consumo de energía
- Tiene cifrado de datos en la comunicación y protección con usuario y contraseña

2.2.5 Brokers MQTT

Se tiene la necesidad de seleccionar el Broker MQTT perfecto que pueda manejar la frecuencia de mensajes para diferentes verticales y realice un procesamiento de mensajes confiable. Un análisis reciente más profundo de todos los principales Brokers MQTT en el mercado hecho por la Universidad de Szeged, Hungría, realizado entre Mosquitto, ActiveMQ, HiveMQ y Bevywise MQTTRoute [1].

La prueba se realizó poniendo estos brokers en situaciones de estrés en un escenario de pruebas real; Las métricas que se utilizaron para la comparación son CPU, latencia, tiempos de transmisión de mensajes, y las condiciones de las pruebas incluyen nivel de QoS, transferencia de los mensajes por cliente, tamaño de la carga del mensaje, etc. Para los resultados se analizó el rendimiento de los brokers proyectando la tasa de mensajes utilizando el 100 % de la CPU y el tiempo promedio de transmisión de los mensajes. El broker debe ser capaz de enviar mensajes seguros en condiciones de red poco fiables.

QoS	Observaciones	Mosquitto 1.4.15	ActiveMQ 5.15.8	HiveMQ CE 2020.2	Bevywise MQTT Route 2.0
QoS0	Tasa de mensajes (msgs/sec)	32,016	573	249	32839
	Uso promedio de la CPU (%), en relación a la tasa de mensajes	84,29	110,44	96,68	97,93
	Latencia promedio (ms)	1,655	1,508	2,74	1,137
	Tasa de mensajes proyectada usando el 100% de la CPU	37,983	518,8	257,5	33533
QoS1	Tasa de mensajes (msgs/sec)	9488	363	118	3542,49
	Uso promedio de la CPU (%), en relación a la tasa de mensajes	89	108,82	104,16	95,79
	Latencia promedio (ms)	0,742	1,782	3,062	0,96
	Tasa de mensajes proyectada usando el 100% de la CPU	10660	333,57	113,28	3697,67
QoS2	Tasa de mensajes (msgs/sec)	6585	293	99	2649
	Uso promedio de la CPU (%), en relación a la tasa de mensajes	96,73	104,36	102,28	98,202
	Latencia promedio (ms)	1,383	2,148	3,665	1,534
	Tasa de mensajes proyectada usando el 100% de la CPU	6807,6	280	96,7	2697,5

Figura 6: Comparativa de los resultados de las pruebas [1]

Como se observa en la figura (6) se tiene como resultado que respecto a la capacidad de procesamiento de mensajes con utilización del 100 % de la CPU en todas las categorías del QoS, Mosquitto es el mejor broker, también se visualiza que tiene la menor latencia en entrega de los mensajes [1].

2.2.6 Eclipse Mosquitto

Mosquitto es un servidor de mensajes (broker) con código abierto que implementa el protocolo MQTT. Mosquitto es liviano y adecuado para su uso en todos los dispositivos, desde computadoras de placa única de baja potencia hasta servidores completos.

El broker Mosquitto provee librerías en lenguaje C para implementar clientes MQTT [9]

2.3 Docker

Docker es una plataforma de código abierto (open source) donde se puede realizar pruebas o implementación de aplicaciones software de forma rápida y sencilla. Docker utiliza contenedores que son ligeros donde se empaqueta todo lo necesario para que pueden ejecutarse distintas aplicaciones en dispositivos que tengan instalado Docker independiente de su sistema operativo, lo que facilita el uso o implementación de alguna aplicación específica en diferentes dispositivos sin preocuparse de actualizaciones o aplicaciones adicionales ya que dentro del contenedor que se ejecuta ya se encuentran todas las librerías y todo lo que la aplicación necesite para su funcionamiento [10].

Docker es muy recomendado para entornos de prueba a causa de los pocos recursos que utiliza, lo que permite ejecutar varios contenedores al mismo tiempo ya que comparte el kernel del sistema operativo. Adicional a que son sencillos de crear y de borrar lo que facilita su implementación, identificación de problemas y poder retornar a una fase anterior de ser necesario para poder solventar algún problema, también su portabilidad lo hace muy bueno para este tipo de entornos [11].

Los contenedores se crean con una imagen como estado inicial, los contenedores incluyen librerías, herramientas del sistema, código y tiempo de ejecución.

Una imagen es como una plantilla que pueden contener el sistema operativo, servidores, aplicaciones entre otros; Existen muchas imágenes disponibles en la nube que se pueden descargar o también se puede crear una imagen propia desde cero o en base a otras existentes dependiendo de los requerimientos del usuario [12].

2.4 Raspberry Pi

La Raspberry Pi es un ordenador simple de pequeño tamaño y bajo coste al cual se pueden conectar diferentes periféricos para poder utilizarlo de igual manera que un ordenador normal que ejecuta Linux como sistema operativo desde la tarjeta SD donde almacena toda la información, lo que permite cambiarla de manera rápida únicamente cambiando la tarjeta SD [13].

La Raspberry Pi puede realizar las mismas funciones que una maquina Linux pero con un costo más bajo.

Esta compuesto de un SoC, CPU, memoria RAM, puertos de entrada, salida de

audio y vídeo, puerto de conectividad de red, ranura SD, reloj, puerto de alimentación, y diversos puertos para conectores de periféricos. Utiliza lenguajes de programación de alto nivel como C++, Java y Python.

Por lo que Raspberry se hace recomendable para proyectos donde se requiera sistema Linux y suma la ventaja de que también tiene conexión a internet. Adicional Raspberry PI utiliza un microprocesador lo que le permite ejecutar una variedad de aplicaciones genéricas y que utilizan gran cantidad de recursos

Figura 7: Raspberry Pi modelo 1 [2]

Capítulo 3

Diseño y Metodología de Desarrollo

Se ha diseñado un cluster de brokers MQTT que es capaz de gestionar mayor cantidad de mensajes en menor tiempo que si se tuviera un solo broker. El cluster tiene una capacidad de hasta 30 000 clientes ya que como vimos cada broker soporta 10 000 clientes.

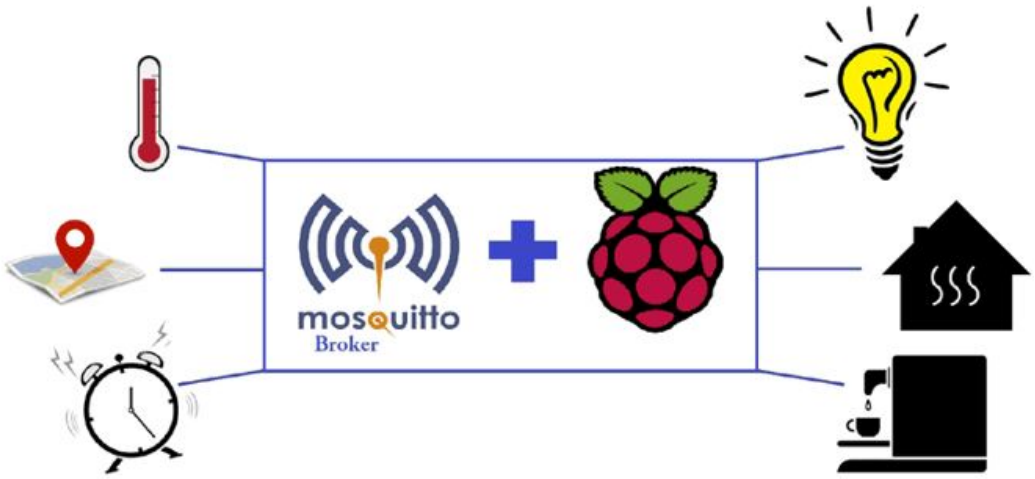

Figura 8: MQTT implementado sobre Raspberry PI con servidor Mosquitto [3]

Se utilizó un router para generar una red interna y así poder asignar parámetros fijos a los nodos del cluster para facilitar la comunicación entre ellos.

Como nodos del cluster se utilizó 4 módulos Raspberry Pi modelo 1, la versión del sistema operativo que se escogió para el desarrollo del cluster es Buster de

Raspbian debido a que es la última versión que se tenía hasta Febrero de 2020.

Para tener una mejor gestión de los nodos del cluster se asignó en cada una de las 4 raspberrys una IP fija y un nombre de tal manera que el manejo de la red se realice de forma más amigable al usuario. La asignación realizada en los nodos es:

	IP	NOMBRE
Nodo 1	192.168.2.101	cluster1
Nodo 2	192.168.2.102	cluster2
Nodo 3	192.168.2.103	cluster3
Nodo 4	192.168.2.104	cluster4

Tabla 3.1: Asignación de IP y nombre a los nodos del cluster

Uno de los nodos (cluster4) se lo utilizó como nodo master del cluster implementando HAProxy para que los clientes (publicadores o suscriptores) se conecten a través del nodo master al cluster, de forma que para todos los usuarios la gestión interna sea transparente y ellos lo puedan visualizar como un solo broker. Adicional con HAProxy se obtiene un mejor balanceo de la carga entre los nodos del cluster.

La ejecución de los brokers se realizó en contenedores de Docker. En los 3 nodos restantes se instaló la versión 19.03.9 de Docker para asegurar la portabilidad de los contenedores en los que se implementó el broker entre los nodos del cluster.

El broker que se escogió para el cluster es Mosquitto de Eclipse, al inicio se utilizó la imagen oficial de eclipse-mosquitto para la ejecución del broker sobre un contenedor de Docker en cada Raspberry, al ejecutar el cluster con los 3 brokers activos se visualizó que la conexión de los clientes (publicadores / suscriptores) se distribuía entre los 3 nodos que es lo deseado para un buen balanceo de la carga, pero al momento de realizar publicaciones no se tenia comunicación entre los 3 brokers, por lo que cada uno funcionaba como broker independiente y no como uno solo lo que generó que solo se tuviera comunicación entre los clientes conectados al mismo broker pero no entre todos los clientes suscritos al mismo topic.

Para solucionar esto se utilizó la funcionalidad bridging de MQTT [12], la que se implementó editando el archivo de configuración de cada broker, por lo que se creó una imagen en base a la imagen oficial de eclipse-mosquitto para la ejecución de cada broker con el contenedor de Docker, usando un archivo de configuración

diferente para cada nodo, de tal manera que se conecten los 3 brokers entre sí, y con esto lograr que todos los clientes, sin importar el broker al cual estén conectados, tengan comunicación con los demás clientes que estén conectados al mismo topic.

La conexión bridging de los brokers se realizó siguiendo el esquema de la figura (9).

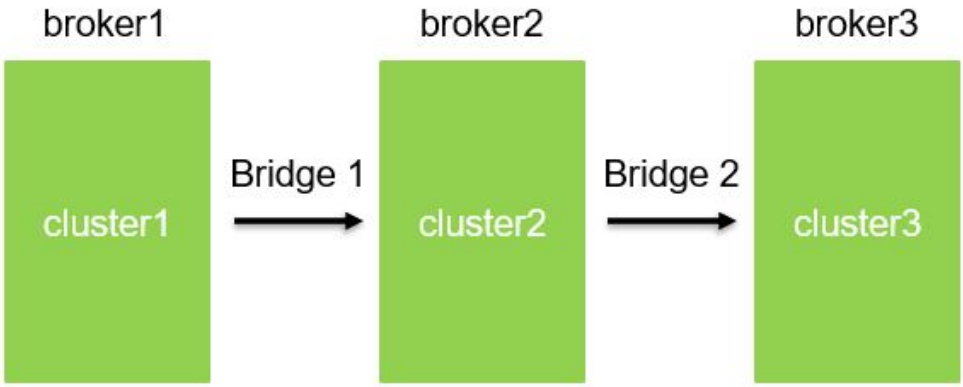

Figura 9: Conexión entre los brokers del cluster

Para la conexión al cluster los clientes deben utilizar la IP del nodo 4 (192.168.2.104) que es el master y el puerto 1883 que es el puerto escogido para la comunicación MQTT.

Gracias a la funcionalidad de HAProxy también se puede visualizar las estadísticas de cada uno de los nodos del cluster, al igual que el estado en el que se encuentra cada nodo, lo que nos es de gran ayuda para poder comprobar que los nodos del cluster estén activos, y en el caso de fallo en alguno de los nodos nos permite detectar rápidamente el nodo afectado. En la figura (10) podemos visualizar lo mencionado. Para el acceso a las estadísticas del cluster se configuró el puerto 7000, y se accede por la misma IP de acceso al cluster que es la IP del nodo 4 (192.168.2.104)

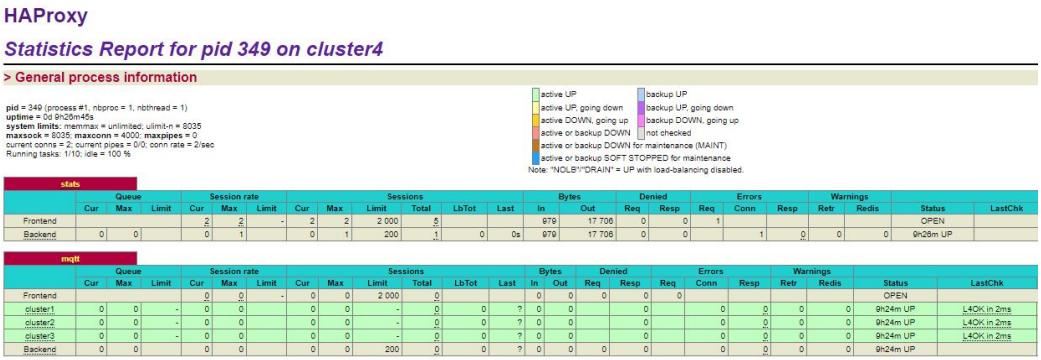

Figura 10: Estadísticas del Cluster con HAProxy

Esta funcionalidad de HAProxy también nos permite poder visualizar la cantidad de conexiones de clientes entre suscriptores y publicadores simultáneas que se tienen en total, y cuantas está gestionando cada nodo del cluster como se visualiza en la figura (11). Asimismo podemos ver el máximo de conexiones que se han tenido desde que el cluster se encuentra activo, el máximo de sesiones por segundo, el total de conexiones desde que se encendió el cluster, la cantidad de bytes de entrada y salida, entre otros datos.

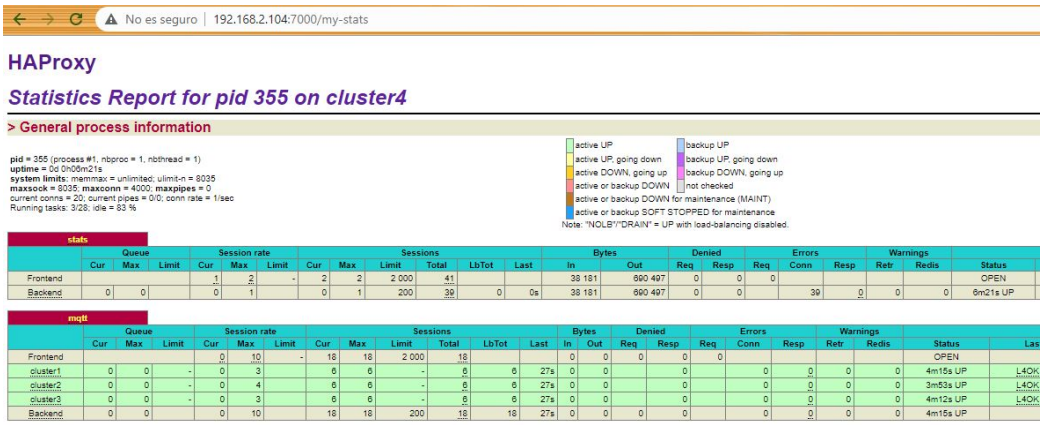

Figura 11: Estadísticas del Cluster con HAProxy

Cuando se inicia cada broker, lo primero que podemos visualizar es que se arma el puente de conexión entre ellos para que puedan tener comunicación y no funcionar como brokers individuales si no como uno solo en el cluster, en las figuras (12), (13), (14) podemos apreciar la conexión del bridging formando el esquema de la figura (9).

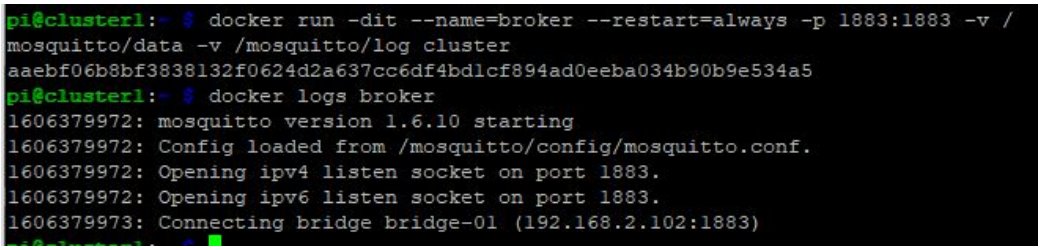

Figura 12: Inicio conexión nodo 1 del cluster

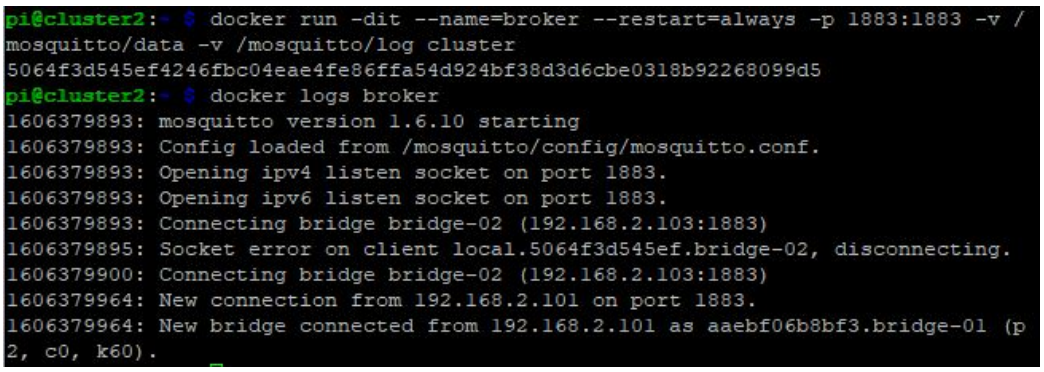

Figura 13: Inicio conexión nodo 2 del cluster

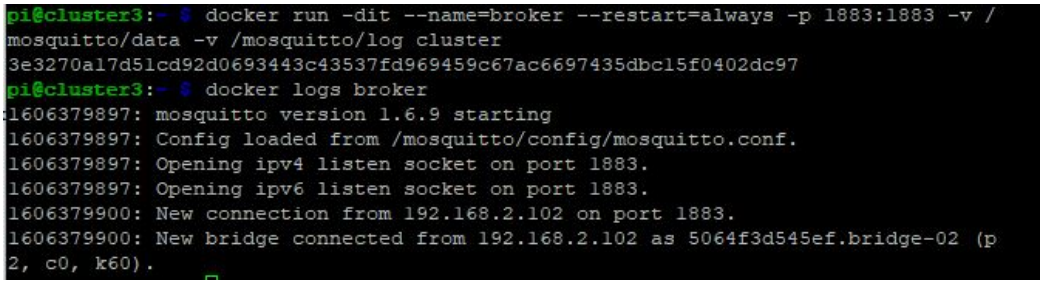

Figura 14: Inicio conexión nodo 3 del cluster

Con el cluster ya activo podemos ver en las figuras (15), (16), (17) que cuando se realiza la conexión de clientes hacia el nodo master (192.168.2.104:1883) estos se distribuyen entre los diferentes brokers del cluster. En este ejemplo se realizo la conexión de 8 suscriptores y 10 publicadores que se distribuyeron para la conexión entre los 3 brokers.

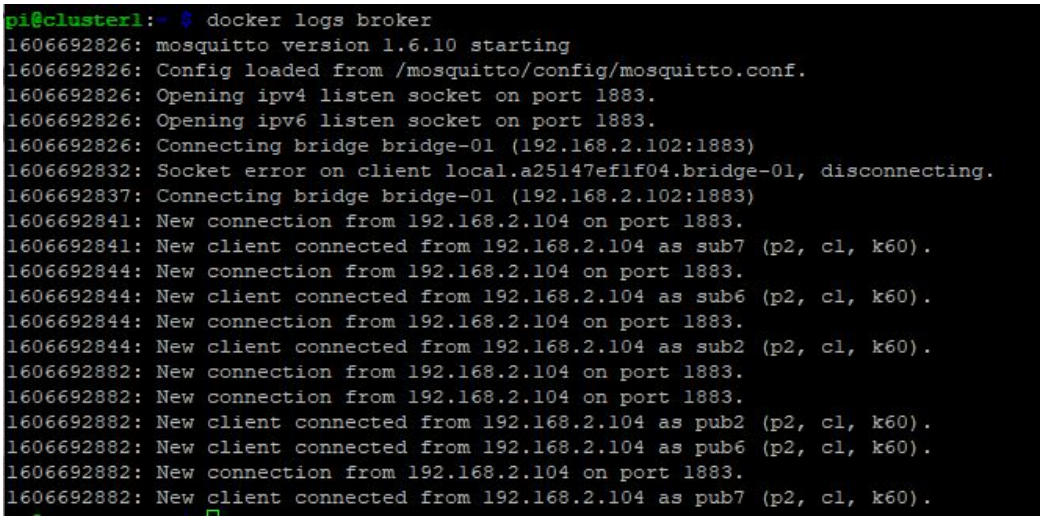

Figura 15: Logs de conexiones al nodo 1 del cluster

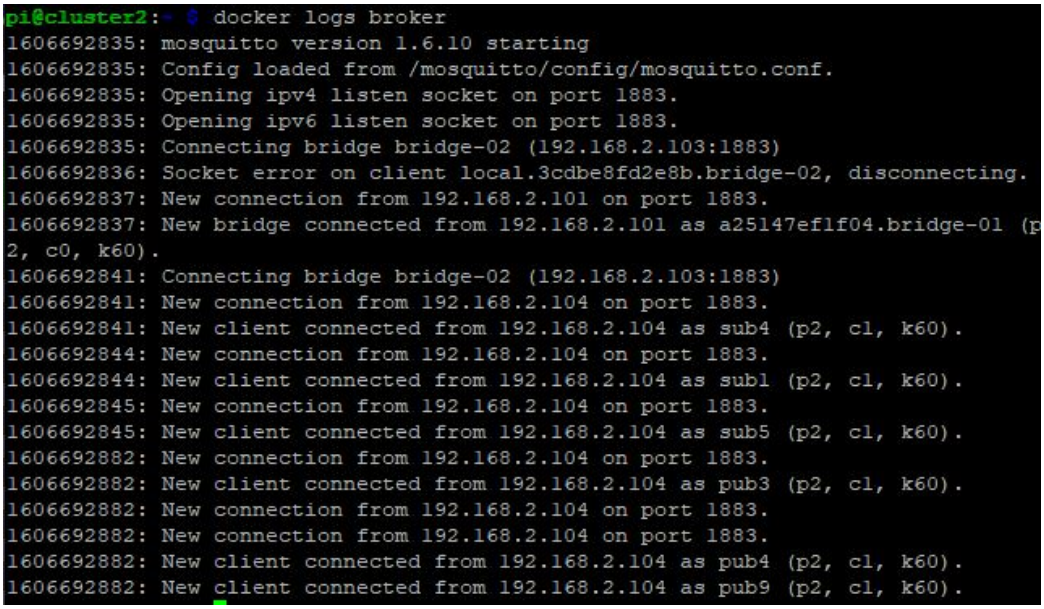

Figura 16: Logs de conexiones al nodo 2 del cluster

Figura 17: Logs de conexiones al nodo 3 del cluster

El desarrollo de este trabajo fin de master se realizó dentro de una red interna sin acceso a internet, si se desea acceder desde otra red se puede conectar el router a internet y realizar en la configuración del mismo una redirección de puertos utilizando la IP pública de salida a internet para la conexión externa y re-direccionando los accesos del puerto 1883 y 7000 hacia el nodo 4 del cluster dentro de la red interna. Se debe tener en cuenta que es necesario que la IP pública sea fija, ya que de ser dinámica en el momento que cambie todos los clientes que tengan registrada la IP anterior no van a poder conectarse al cluster hasta que se configure la nueva IP lo que podría generar perdida de información o inconvenientes a nivel de gestión ya que es probable que esta configuración se deba realizar de manera manual en algunos dispositivos.

Capítulo 4

Resultados

Con el cluster ya activo se realizaron varias pruebas para poder comparar la eficiencia de utilizar el cluster en relación a utilizar un solo broker.

Para las pruebas se generaron gran cantidad de clientes publicando mensajes a diferentes frecuencias, para poder comparar los tiempos de transmisión que se obtiene con el cluster y los tiempos de transmisión en las mismas condiciones con un solo broker.

Los clientes tanto los publicadores como los suscriptores se generaron con un programa escrito en python que utiliza hilos para la activación y gestión simultánea de los clientes. Esto se realizó desde una laptop Lenovo con sistema operativo Windows 10, que tiene las características detalladas en la tabla (4.1).

CPU	
Tipo	Core i5 6200U
Núcleos	2
Velocidad Núcleo	2400 MHz

RAM	
Tipo	DDR3
Tamaño	8034 MBytes
Velocidad	1600 MHz

Tabla 4.1: Características PC utilizada para generar los clientes

En las pruebas se utilizó una base de 8 topics, los que se encuentran detallados

en la tabla (4.2). Los clientes publicadores al momento de conectarse al cluster seleccionan de manera aleatoria un topic de los que se encuentran en esta base.

	Topics
1	/Casa/Interior/Cocina/Temperatura
2	/Casa/Interior/Sala/Temperatura
3	/Casa/Interior/Cocina/Luces
4	/Casa/Interior/Cuarto/Luces
5	/Casa/Interior/Sala/Luces
6	/Casa/Interior/Cuarto/Temperatura
7	/Casa/Exterior/Patio/Temperatura
8	/Casa/Exterior/Patio/Luces

Tabla 4.2: Base de topics usados en la prueba

También se utilizaron 8 clientes conectados como suscriptores, los que, con la ayuda de un comodín, se los suscribió a todos los topics con el fin de generar mayor carga de trabajo hacia el broker, ya que cada mensaje publicado es reenviado a los 8 suscriptores.

En todas las pruebas se utilizó QoS 0, tanto para los clientes publicadores como suscriptores.

4.1 Escenarios

Se realizaron varias pruebas, generando diferentes escenarios, en las que se varió el número de clientes simultáneos conectados al cluster; Para esto se trabajo cambiando el número de clientes publicadores entre 1, 5, 10, 50, 100, 500 y 1000, conectados de manera simultánea. En todos los casos se mantuvieron los 8 clientes suscriptores detallados anteriormente.

Los clientes publicadores se encontraban realizando publicaciones de manera constante, en las cuales se realizaron variaciones en las frecuencias de envíos de mensajes. El tiempo entre cada publicación medido en segundos se varió entre 0.5, 1, 3, 5 y 10.

4.2 Estructura del mensaje

El mensaje que se publica se puede estructurar de diferentes maneras dependiendo de los requerimientos de la aplicación utilizada y del usuario, colocando la información que se requiera enviar a los suscriptores. Para las pruebas se generó un mensaje en el que el nodo publicador envía el número del publicador que se asigna cuando se crea, este se mantiene mientras el publicador se encuentre activo, y es usado también como identificador del cliente, el número del mensaje enviado, la fecha y hora en la que se envió el mensaje, el Topic y el QoS. Esta información también es almacenada en una base de datos para análisis de las publicaciones, para mayor ilustración se puede observar la figura (18).

```
pub1, 1 fecha y hora de envio: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Cuarto/Luces
pub2, 1 fecha y hora de envio: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Luces
pub3, 1 fecha y hora de envio: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Cocina/Luces
pub4, 1 fecha y hora de envio: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Temperatura
pub5, 1 fecha y hora de envio: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Luces
pub1, 2 fecha y hora de envio: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Cuarto/Luces
pub2, 2 fecha y hora de envio: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Sala/Luces
pub4, 2 fecha y hora de envio: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Sala/Temperatura
pub3, 2 fecha y hora de envio: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Cocina/Luces
pub5, 2 fecha y hora de envio: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Sala/Luces
```

Figura 18: Información almacenada de los publicadores

En el cliente suscriptor se toma el mensaje recibido y se obtiene el identificador del cliente que publica, el número del mensaje recibido, la hora en la que el publicador envía el mensaje, con la cual por medio de una resta entre la hora en que el suscriptor recibe el mensaje y la hora detallada en el mensaje recibido (hora de publicación), se obtiene el tiempo de transmisión del mensaje con unidad de tiempo de segundos. De igual manera que en el publicador esta información se almacena en una base de datos para análisis, para mayor ilustración se puede observar la figura (19).

```
pub1, 1 fecha y hora de recepción: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Cuarto/Luces - tiempo de envio del mensaje: 0.0878145694732666
pub2, 1 fecha y hora de recepción: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Luces - tiempo de envio del mensaje: 0.08881354331970215900039
pub5, 1 fecha y hora de recepción: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Luces - tiempo de envio del mensaje: 0.08881306648254395
pub4, 1 fecha y hora de recepción: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Temperatura - tiempo de envio del mensaje: 0.0898122787475586
pub2, 1 fecha y hora de recepción: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Luces - tiempo de envio del mensaje: 0.09479618072509766
pub4, 1 fecha y hora de recepción: Fri Nov 27 10:44:57 2020 Topic= /Casa/Interior/Sala/Temperatura - tiempo de envio del mensaje: 0.09380054473876953
pub1, 2 fecha y hora de recepción: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Cuarto/Luces - tiempo de envio del mensaje: 0.0067861080169677734
pub1, 2 fecha y hora de recepción: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Cuarto/Luces - tiempo de envio del mensaje: 0.008904218673706055
pub1, 2 fecha y hora de recepción: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Cuarto/Luces - tiempo de envio del mensaje: 0.008904218673706055
pub1, 2 fecha y hora de recepción: Fri Nov 27 10:45:07 2020 Topic= /Casa/Interior/Cuarto/Luces - tiempo de envio del mensaje: 0.009785652160644531
```

Figura 19: Información almacenada de los suscriptores

4.3 Pruebas

Con lo detallado anteriormente se realizaron las pruebas, aplicando las mismas condiciones con un solo broker y con el cluster para poder realizar la comparación y visualizar la eficiencia del cluster.

En la figura (20) se puede visualizar la comparación del tiempo promedio de transmisión de los mensajes con el cluster activo y con el cluster inactivo, variando la cantidad de los publicadores conectados simultáneamente y realizando publicaciones con una frecuencia de 0.5 segundos. De igual manera, pero con una frecuencia de publicaciones de 1 segundo se encuentra la comparación en la figura (21). También se realizó la comparación con una frecuencia de 3 segundos en las publicaciones y se encuentra en la figura (22), con la frecuencia de 5 segundos se encuentra graficado en la figura (23), y en la figura (24) se puede observar la comparación de los tiempos promedios de transmisión con una frecuencia de publicación de 10 segundos.

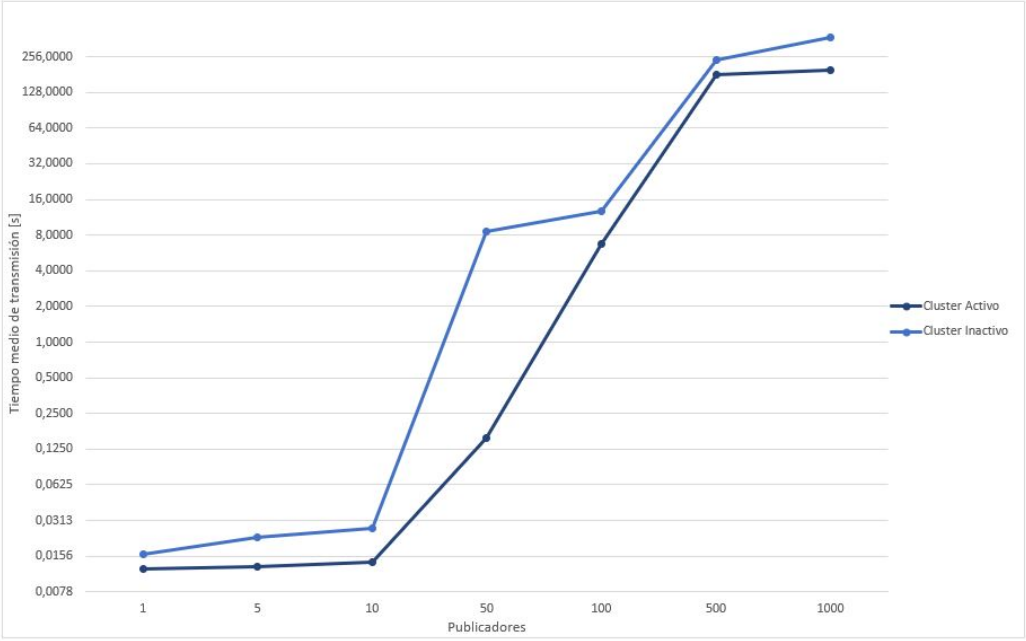

Figura 20: Tiempo promedio de transmisión variando la cantidad de publicadores con frecuencia 0.5 segundos

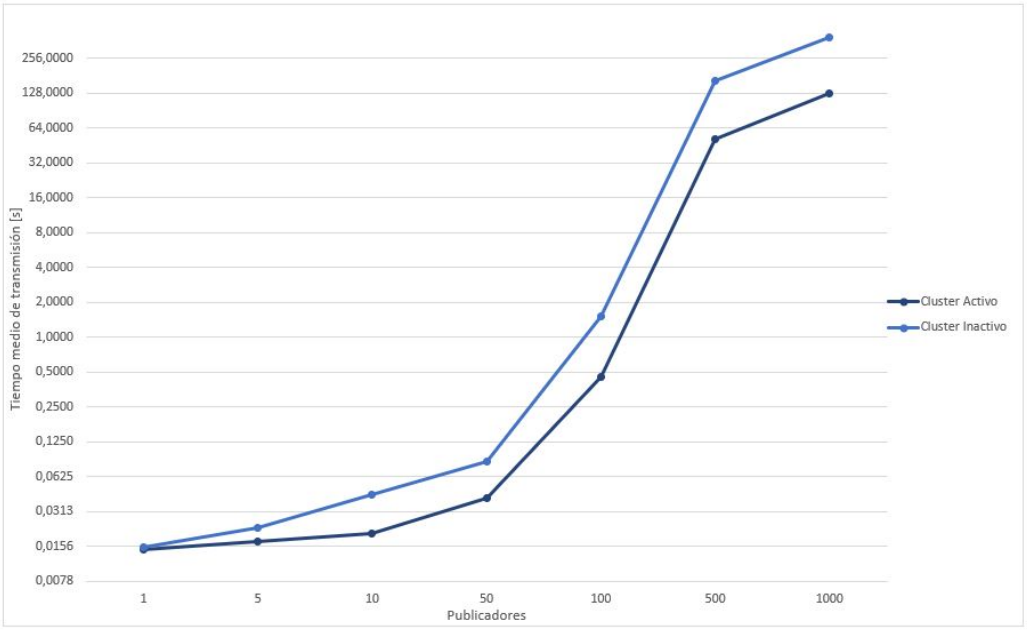

Figura 21: Tiempo promedio de transmisión variando la cantidad de publicadores con frecuencia 1 segundo

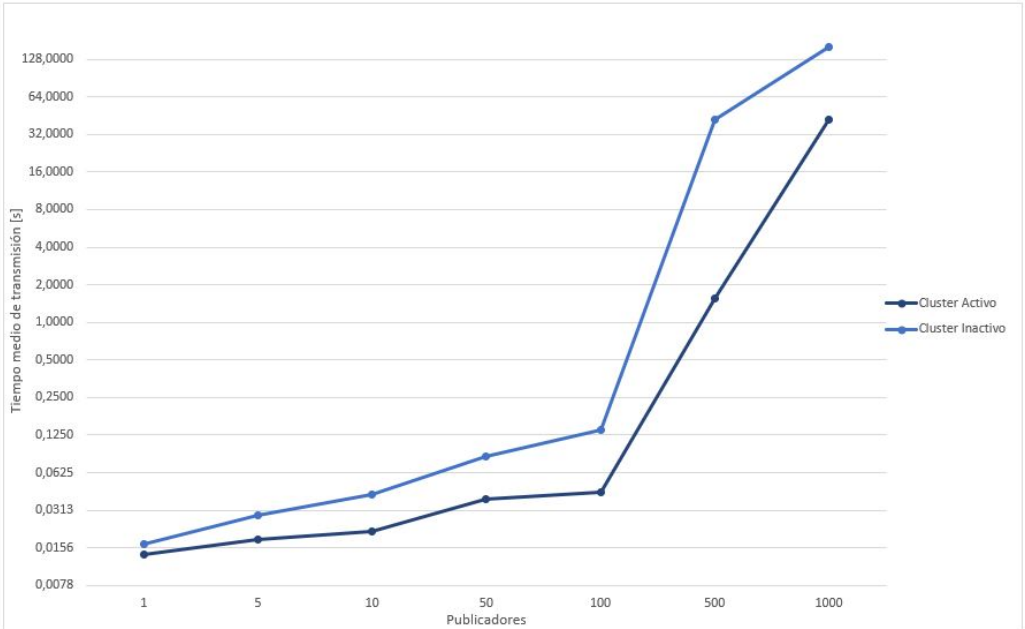

Figura 22: Tiempo promedio de transmisión variando la cantidad de publicadores con frecuencia 3 segundos

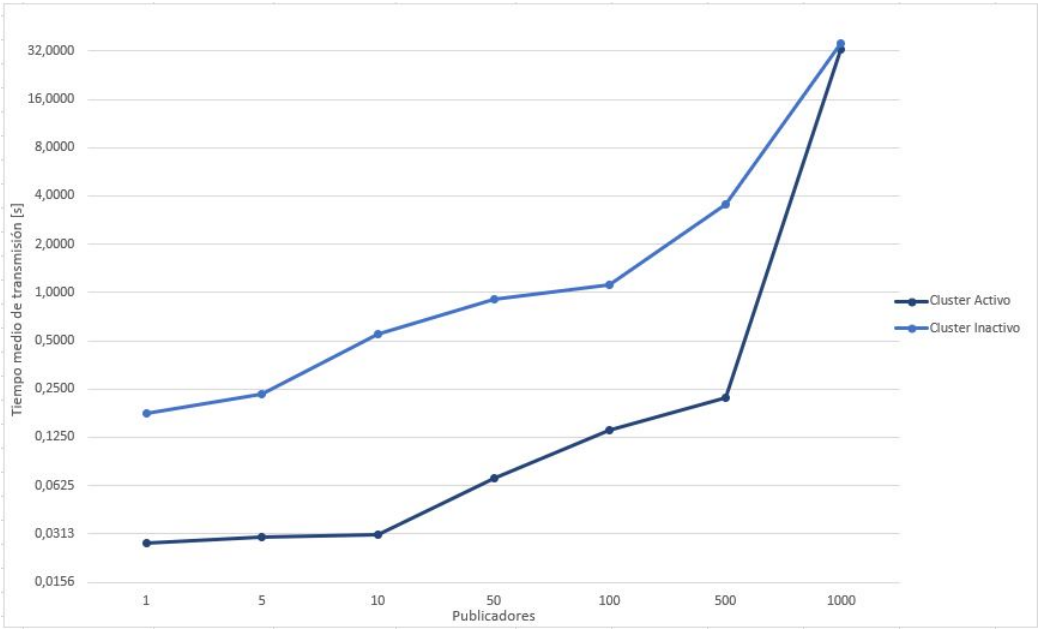

Figura 23: Tiempo promedio de transmisión variando la cantidad de publicadores con frecuencia 5 segundos

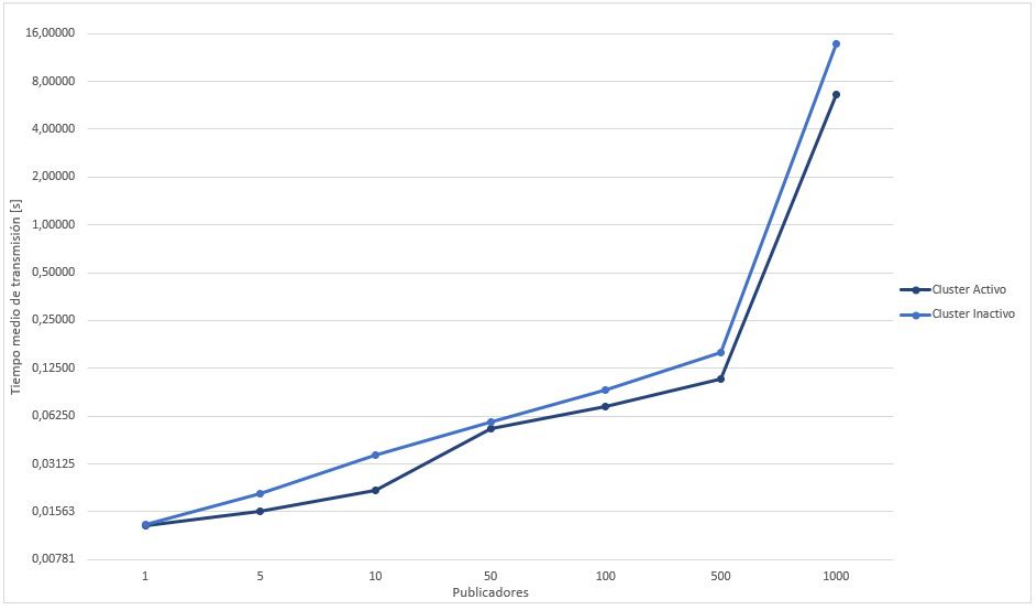

Figura 24: Tiempo promedio de transmisión variando la cantidad de publicadores con frecuencia 10 segundos

Al momento de analizar las gráficas se debe tener en cuenta que la escala utilizada para representar el tiempo de transmisión (eje y) es escala logarítmica. Con las gráficas podemos apreciar que el cluster proporciona menores tiempos promedios de transmisión en todos los casos, también podemos observar que se obtienen mayores diferencias en los tiempos, siendo mejor los del cluster, cuando se tienen mayor cantidad de clientes realizando publicaciones de manera simultánea. Esto también se puede visualizar de mejor manera con las pruebas en las que se utilizó menor frecuencia de transmisión para las publicaciones, esto es debido a que estos son los momentos de mayor saturación para el cluster en relación a cuando la cantidad de publicadores no es mucho, o a cuando la frecuencia de los mensajes es menor, ya que en estos casos se genera una menor carga hacia el broker.

En los escenarios en los que no se genera mucha carga hacia el broker o hacia el cluster se puede comprobar que los tiempos promedio de mensaje son más aproximados ya que no se esta saturando los nodos por lo que se da tiempo al broker de realizar un buen procesamiento, al igual que el cluster.

Mientras se realizaron las pruebas se validó que al tener muchos clientes ejecutándose en el mismo dispositivo se presentaron ocasiones en las que se tuvo que rehacer las pruebas ya que se tenia elevado el rendimiento de la PC. La comunicación de la PC con el cluster y el broker individual se realizó mediante la red wifi, con lo que también se verifico que cuando se realizaron las pruebas de muchos clientes y al mismo tiempo se observaba que otros dispositivos externos a la red pero cercanos utilizaban en gran medida el canal de 2.4 GHz que es la banda de wifi se generaba interferencias que influían en los tiempos de transmisión tanto para el cluster activo como con el broker individual. A causa de esto se busco realizar las pruebas en momentos en los que no se mantenía esta afectación. En base a los registros de los mensajes publicados y los mensajes recibidos por los suscriptores se realizó el calculo del porcentaje de pérdida de mensajes y se lo detalló en la tabla (4.3). Se calculó el porcentaje medio total de todas las pruebas y se las agrupo en base a la frecuencia de transmisión con lo que se refleja nuevamente que cuando las publicaciones se realizan con menor frecuencia se obtiene menor cantidad de mensajes perdidos, ya que la carga hacia el broker o el nodo no es tanta como cuando se publica con mayor frecuencia.

Frecuencia de publicación	Cluster activo	Cluster Inactivo
0.5[s]	13.01	31.60
1[s]	9.26	22.35
3[s]	7.82	14.29
5[s]	5.71	13.28
10[s]	0.22	3.05

Tabla 4.3: Comparación porcentaje de pérdida de mensajes

Capítulo 5

Conclusiones

Se implementó un cluster de brokers utilizando dispositivos de bajo coste como 3 nodos broker, y se comprobó la efectividad del mismo con diferentes escenarios en los que se busco sobrecargar el cluster y un broker individual con el fin de apreciar la conveniencia del cluster analizando los tiempos de transmisión y la perdida de los mensajes.

Se concluye que la implementación del cluster en las SBCs de bajo coste si es conveniente ya que se obtienen menores tiempos de transmisión que con un broker individual, lo que permite que el canal de transmisión este ocupado por menor tiempo y se pueda generar una mayor transmisión de mensajes por unidad de tiempo. Adicional que al tener una menor carga sobre el broker también se logra una menor pérdida de mensajes.

De igual manera la utilización de los contenedores Docker facilita la ejecución de los brokers gracias a todas las ventajas que proporcionan, y aporta portabilidad por tanto, de presentarse algún fallo en uno de los nodos del cluster el mismo puede ser reemplazado de manera más sencilla ya que únicamente se debe colocar el contenedor.

Lista de Acrónimos

AMQP Advanced Message Queuing Protocol

CoAP Constrained Application Protocol

HAProxy High Availability Proxy

HTTP Hypertext Transfer Protocol

IoT Internet of Things

MQTT Message Queue Telemetry Transport

SBCs Single Board Computers

Bibliografía

[1] B. M. Biswajeeban Mishra, "Evaluating and analyzing mqtt brokers with stress-testing," *Article*, 2020.

[2] R. Pi. (2019) Raspberry pi blog. Last visited: 2020-09-11. [Online]. Available: https://www.raspberrypi.org/blog/

[3] C. Digest. (2019) Installing and testing mosquitto mqtt broker on raspberry pi for iot communication. Last visited: 2020-11-11. [Online]. Available: https://circuitdigest.com/microcontroller-projects/installing-and-testing-mosquitto-mqtt-broker-on-raspberry-pi

[4] M. Serozhenko. (2017) Mqtt vs. http: which one is the best for iot? Last visited: 2020-10-30. [Online]. Available: https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105#:~:text=MQTT%20is%20data%20centric%20whereas,always%20optimized%20for%20mobile%20devices.&text=Besides%2C%20publish%2Fsubscribe%20model%20provides,reliability%20of%20the%20whole%20system.

[5] S. M. Casco, "Raspberry pi, arduino y beaglebone black comparación y aplicaciones," *Article*, 2014.

[6] F. M. Cerdà, "Demostrador arquitectura publish/subscribe con mqtt," *Tesis final de grado-UPC*, 2018.

[7] L. Llamas. (2019) ¿qué es mqtt? su importancia como protocolo iot. Last visited: 2020-11-1. [Online]. Available: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/#

[8] J. Crespo. (2018) Mqtt. Last visited: 2020-08-25. [Online]. Available: https://aprendiendoarduino.wordpress.com/2018/11/19/mqtt/

[9] B. Wire. (2020) Eclipse mosquitto™ an open source mqtt broker. Last visited: 2020-11-11. [Online]. Available: https://mosquitto.org/

[10] J. Garzas. (2015) Entendiendo docker. conceptos básicos: Imágenes, contenedores, links.... Last visited: 2020-08-20. [Online]. Available: https://www.javiergarzas.com/2015/07/entendiendo-docker.html

[11] AWS. (2020) ¿qué es docker? Last visited: 2020-08-20. [Online]. Available: https://aws.amazon.com/es/docker/

[12] D. Inc. (2020) Docker. Last visited: 2020-11-19. [Online]. Available: https://hub.docker.com/

[13] E. R. D. Luis. (2018) De cero a maker: todo lo necesario para empezar con raspberry pi. Last visited: 2020-08-15. [Online]. Available: https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi