UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TEMA:

Estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica.

AUTOR:

Pérez Leones, Kamila

Trabajo de Titulación previo a la obtención del título de

INGENIERO EN TELECOMUNICACIONES

TUTOR:

M. Sc. Alvarado Bustamante, Jimmy Salvador

Guayaquil, Ecuador

12 de Septiembre del 2019

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

CERTIFICACIÓN

Certificamos que el presente trabajo fue realizado en su totalidad por el Srta. **Pérez Leones, Kamila** como requerimiento para la obtención del título de **INGENIERO EN TELECOMUNICACIONES**.

TUTOR

_____

M. Sc. Alvarado Bustamante, Jimmy Salvador

DIRECTOR DE CARRERA

_____

M. Sc. Heras Sánchez, Miguel Armando

Guayaquil, a los 12 días del mes de septiembre del año 2019

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

Yo, **Pérez Leones, Kamila**

DECLARÓ QUE:

El Trabajo de Titulación**: Estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica,** previo a la obtención del Título de **Ingeniero en Telecomunicaciones**, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Titulación referido.

Guayaquil, a los 12 días del mes de septiembre del año 2019

EL AUTOR

_____

PÉREZ LEONES, KAMILA

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO
CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

AUTORIZACIÓN

Yo, **Pérez Leones, Kamila**

Autorizó a la Universidad Católica de Santiago de Guayaquil, la publicación, en la biblioteca de la institución del Trabajo de Titulación: **Estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica**, cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y total autoría.

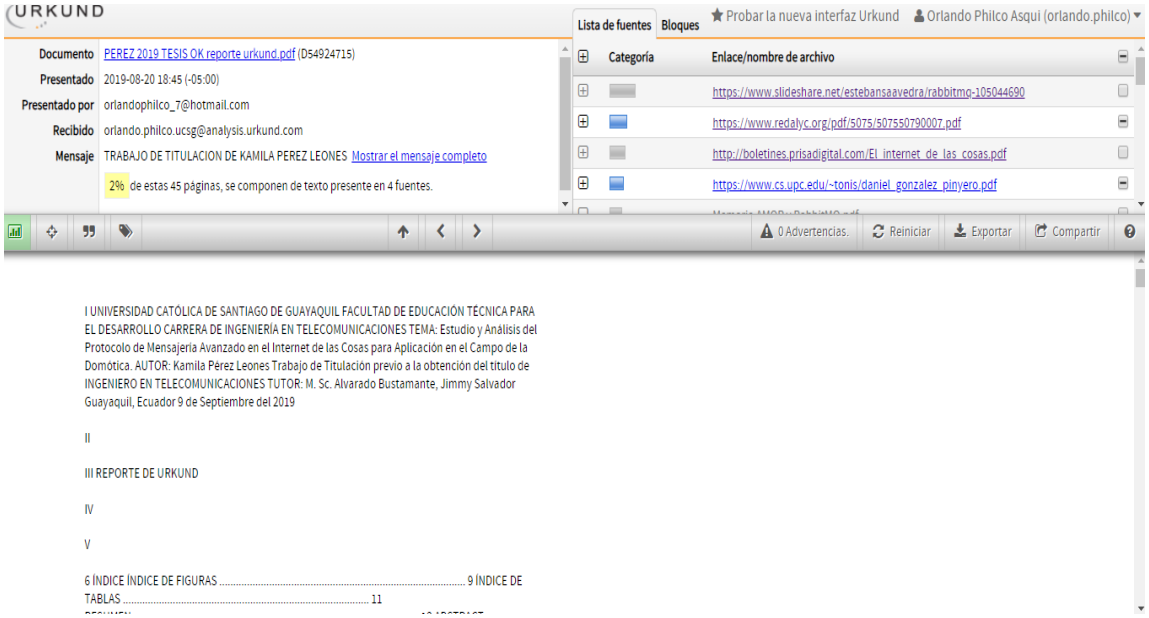
Guayaquil, a los 12 días del mes de septiembre del año 2019

EL AUTOR

_____

PÉREZ LEONES, KAMILA

REPORTE DE URKUND



Reporte Urkund del trabajo de titulación en Ingeniería en Telecomunicaciones titulado: **“Estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica”** de la estudiante: Kamila Pérez Leones, el análisis de coincidencia indica el 2% de coincidencias.

Atentamente

Ing. Orlando Philco A.

Revisor

DEDICATORIA

Dedicado a Dios por guiarme en este camino de la vida.

A mi familia por siempre estar para mí, sobre todos a mis padres que, gracias a su esfuerzo y dedicación, podré obtener mi carrera universitaria y darme aliento para seguir adelante.

EL AUTOR

_____

PÉREZ LEONES, KAMILA

AGRADECIMIENTO

Agradecida con Dios,
Mi familia, y
mis compañeros de trabajo que me ayudaron en este trayecto de mi trabajo de titulación con sus conocimientos desde el punto de vista profesional.

EL AUTOR

_____

PÉREZ LEONES, KAMILA

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TRIBUNAL DE SUSTENTACIÓN

f. _____

M. Sc. ROMERO PAZ, MANUEL DE JESUS

DECANO

f. _____

M. SC. PALACIOS MELÉNDEZ, EDWIN FERNANDO

COORDINADOR DEL ÁREA

f. _____

M. Sc. RUILOVA AGUIRRE, MARÍA LUZMILA

OPONENTE

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 2: FUNDAMENTACIÓN TEÓRICA

CAPÍTULO 3: RESULTADOS DEL ESTUDIO DEL PROTOCOLO AMQP UTILIZANDO RABBITMQ

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo de titulación está orientado en el estudio y análisis del protocolo de mensajería avanzado del internet de las cosas para aplicación en el campo de la domótica, evaluando su actuación en la capa de aplicación dentro del modelo de referencia OSI, en donde intervienen diferentes tipos de programas y la razón de ser uno de los protocolos más conocido en la IoT. Existe el bróker de mensajería, donde se gestiona las colas de mensajes de código abierto que intervienen en el protocolo estudiado. El presente trabajo de titulación se compone por los siguientes capítulos: el capítulo 1 consta de la introducción, antecedentes, planteamiento del problema, justificación, objetivos, metodología de la investigación, el capítulo 2 se define los conceptos esenciales del protocolo AMQP y su diferencia ante los demás protocolos de la IoT, el capítulo 3 contiene una visualización y resultados del estudio del protocolo utilizando una herramienta de software de distribución libre, tal como, RabbitMQ es el que ha alcanzado una popularidad especial.

Palabras claves: PROTOCOLO, IoT, OSI, BRÓKERS, MENSAJERÍA, RABBITMQ.

ABSTRACT

The present graduation work is oriented towards the study and analysis of the advanced message protocol of the internet of things to be applied in the field of home automation; by evaluating its performance in the application layer within the OSI reference model, reviewing the different types of programs that are involved, as well the reasons for which AMQP is one of the best known protocols in the IoT. we will review a fundamental aspect of this protocol, the messenger broker, that provides flexibility and reliability on message transmission. This graduation work consists of the following chapters: Chapter 1 consists on introduction, background, problem statement, justification, objectives and research methodology. Chapter 2 defines the essential concepts of the AMQP protocol and its difference among other protocols within IoT. Chapter 3 contains an implementation example as well as the results of the protocol study using a free distribution software tool such as RabbitMQ, an application that has reached special popularity.

Keywords: PROTOCOL, IoT, OSI, BROKER, MESSAGE, RABBITMQ.

CAPÍTULO 1: INTRODUCCIÓN

1.1. Introducción.

En los últimos años, se han desarrollado protocolos del internet de las cosas con el propósito de solucionar problemas para la automatización de procesos y acciones con el fin de minimizar la interacción humana aumentando la productividad, y crear normas para que varios dispositivos conectados a la red mantengan comunicación continua y privada durante el intercambio de información, como la cantidad dispositivos que se va a interactuar ya sea estos de computación, máquinas digitales y mecánicas entre otros. Por otro lado, estos protocolos son capaces de recopilar, procesar y enviar datos a otras aplicaciones, objetos o servidores sin perder información.

Entre los protocolos IoT, el más popular en el sistema de mensajes que se usa en middleware corporativo es el distinguido AMQP (Advanced Message Queing Protocol). En un medio de mayor rendimiento, latencia de red y capacidad de cálculo no llega a presentarse como un problema. AMQP fue creado para el sector industrial con el fin de asegurar la confiabilidad y la interoperabilidad en aplicaciones corporativas.

El Internet de las cosas, ha buscado a lo largo del tiempo una vida moderna mediante la conexión entre dispositivos inteligentes, aplicaciones y tecnologías. IoT es una red gigante con muchos dispositivos conectados, en la que estos dispositivos se reúnen y comporten información relacionado al uso y en el medio que operan, todo esto se hace utilizando sensores o diferentes equipos. Hoy en día, IoT ha evolucionado de una forma inimaginable, ya que ahora las industrias están dispuestos a adoptar esta nueva tecnología de comunicación.

En este documento se presenta el estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica evaluando su factibilidad considerando la reducción tanto de

costo y su alta disponibilidad que lo diferencian de los demás protocolos de la internet, haciéndolo confiable, seguro e interoperable para el intercambio de mensajes entre diferentes plataformas.

1.2. Antecedentes.

Los protocolos del internet de las cosas tienen un largo camino por recorrer, ya que la transmisión de información es fundamental para el medio en el que actualmente vivimos. El protocolo AMQP, en inglés “Advanced Message Queuing Protocol”, surgió desde el 2003, concebida en JPMorgan Chase, la cual es una empresa líder en inversiones bancarias que brinda gestión de activos, servicios e inversiones financieras, entre otros, su sede se encuentra en Estados Unidos. John O’Hara inventó AMQP (ISO 19464), tiempo después lo adoptaron Amazon, Microsoft y otras empresas, en la cual O’Hara fue el director de esta propuesta y tomó la decisión de desarrollarlo junto con OASIS, organización sin fines de lucro, basándose en estándares de código abierto como TCP/IP y que sea gratuito. Surge este protocolo al momento de intercambiar volúmenes de miles de mensajes por segundo, y resultaba peligroso que no lleguen o que lleguen tarde para una institución financiera con grandes cantidades de transacciones. Ningún protocolo era capaz de satisfacer los requerimientos antes indicados.

AMQP se creó y tuvo su lanzamiento a mediados del año 2006. El proyecto, sirvió para 2.000 usuarios y llegó a procesar 3000 millones de mensajes por día. Fue diseñado para ser usado desde diferentes entornos de programación, sistemas operativos, y dispositivos de hardware, así como facilitar implementaciones de alto rendimiento en diversos transportes de red.

La IoT es resultado de la evolución del Internet por su interconexión mucho más extensa y completos servicios inteligentes. También, actualmente existen dispositivos inteligentes que se comunican entre sí y con el sistema de control, conocido como M2M (comunicaciones de máquina a máquina), lo cual se ha ido desarrollando al pasar de los años.

1.3. Definición del Problema.

En la actualidad no es considerado en el sector industrial el uso de dispositivos para el internet de las cosas o “IoT” de bajo costo y de alta disponibilidad, los cuales reducirían los tiempos de implementación y costos para diferentes industrias.

1.4. Justificación del Problema.

El comportamiento del protocolo de mensajería avanzado en cola juega un papel de suma importancia para las industrias por su bajo costos de aplicación para que pueda abrir puertas en el área de la domótica en la que permite conectar y controlar cualquier “objeto o cosa”. El AMQP se muestra como una solución, ya que es un protocolo que transmite información a través de un bróker. Es conveniente el uso del protocolo AMQP ya que resuelve diferentes problemas al mismo tiempo por una parte es el encargado de trasmitir sólidamente datos y por otra, permite almacenar mensajes en una cola o “queu”.

Por lo tanto, este trabajo de titulación tiene como principal objetivo, estudiar y analizar al protocolo de mensajería avanzado del internet de las cosas para aplicación en el campo de la domótica, de manera que se evalúa a través de una herramienta de software de distribución libre para obtener la factibilidad del bajo costo de aplicación y la alta disponibilidad del protocolo AMQP.

1.5. Objetivos del Problema de Investigación.

1.5.1. Objetivo General.

Desarrollar el estudio y análisis de factibilidad del protocolo AMQP del internet de las cosas para aplicación en el campo de la domótica.

1.5.2. Objetivos Específicos.

- Demostrar con bases teóricas las diferentes aplicaciones de AMQP en la domótica.

- Evaluar y estudiar una herramienta de software de distribución libre para visualizar la información requerida a través del protocolo AMQP.
- Analizar los distintos escenarios que se pueden aplicar en el protocolo de mensajería avanzado en cola.

1.6. Hipótesis.

Partiendo del análisis de factibilidad a realizar, se logra obtener información que permite el desarrollo de proyectos de domótica orientados a distintos sectores que al momento no consideran el uso de otras herramientas informáticas para mejorar sus procesos.

1.7. Metodología de Investigación.

Para este trabajo de investigación se emplea tres tipos de metodologías de investigación, las cuales se describen a continuación:

Metodología Descriptiva: Se emplea este tipo de metodología porque se describe de manera teórica las características del protocolo AMQP permitiendo mejorar la comunicación entre las “cosas” de forma mucho más confiable e interoperable haciendo que se diferencie entre los demás protocolos del IoT.

Metodología Analítica: Se ha optado este tipo de metodología porque se requiere de un enfoque profundo, donde abarca desde el funcionamiento a la observación del protocolo AMQP del Internet de las Cosas, el cual al usarse resulta de bajo costo y de alta disponibilidad, es decir se reúne información fundamental que nos permite aclarar ciertas definiciones y nuestros conocimientos adquiridos.

CAPÍTULO 2: FUNDAMENTACIÓN TEÓRICA

2.1. Protocolo de comunicación y el modelo de arquitectura.

2.1.1. Definición de protocolo de comunicación.

En las telecomunicaciones, se lo define al protocolo de comunicaciones como al sistema de reglas que consiste en que dos o más entidades de un sistema de comunicación, se comuniquen entre sí con la finalidad de transmitir información a través de cualquier variación de una magnitud o medida física fundamental, se describe también que son métodos de recuperación de errores basados en reglas o normas.

A continuación, se muestra la estructura básica de un sistema de comunicación:

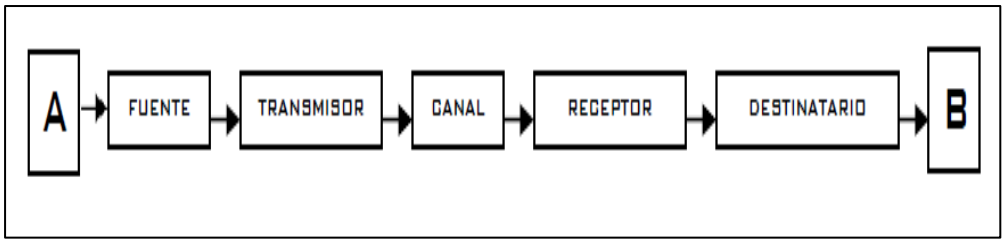


Figura 2.1 Estructura básica de un sistema de comunicación.
Fuente: (Alfaro Malatesta, 2006)

El modelo de arquitectura OSI posee 7 capas que progresivamente fue sustituido por el modelo TCP/IP, éste modelo solo considera 4 capas o niveles. Por medio de este modelo, el protocolo de mensajería avanzado en cola juega un papel muy importante. Se presenta en la tabla 2.1 la arquitectura del protocolo TCP/IP con respecto al modelo OSI.

2.1.2. Modelo de Arquitectura TCP/IP.

TCP/IP es un conjunto de protocolos de comunicación, sus siglas significan "Protocolo de control de transmisión/Protocolo de Internet". Proviene de los nombres de dos protocolos importantes del conjunto de protocolos, es decir, del protocolo TCP y del protocolo IP. El modelo TCP/IP, similar al modelo OSI, fue desarrollado por la Organización Internacional para la Normalización (ISO) para estandarizar las comunicaciones entre

equipos.(Amaya Prieto, 2017). A continuación, la tabla de los modelos de arquitectura OSI y TCP/IP.

Tabla 2.1 Modelo de arquitectura TCP/IP.

N° de Referencia Capas o Niveles OSI	MODELO	
	OSI	TCP/IP
7	Aplicacación	Aplicación
6	Presentación	
5	Sesión	
4	Transporte	Transporte
3	Red	Internet
2	Enlade de Datos	Acceso a la Red
1	Física	

Nota: Tabla de la arquitectura del modelo TCP/IP frente al modelo OSI.

Elaborado por: Autor.

En la tabla 2.1 se observa el Modelo TCP/IP con respecto al modelo OSI, cumpliendo las mismas funciones, como en la transmisión y en la recepción.

2.1.3. Capas del modelo TCP/IP.

A continuación, la descripción de cada una de las capas del modelo:

1. **Capa de acceso a la red:** Determina la forma en la que los datos se deben enrutar, sin importar el tipo de red utilizado (LAN, WAN) (Amaya Prieto, 2017).
2. **Capa de Internet:** Controla la comunicación entre los equipos, es responsable de proporcionar el paquete de datos y enrutarlo para llegar a su destino, mediante el protocolo de internet conocido como en sus siglas IP. (Amaya Prieto, 2017).
3. **Capa de transporte:** Brinda los datos de enrutamiento, junto con los mecanismos que permiten conocer el estado de la transmisión por medio de un programa de aplicación y facilita la comunicación punto a punto. Se utilizan los protocolos TCP (Transmission Control Protocol) y UDP (User Datagram Protocol) (Amaya Prieto, 2017).

4. **Capa de aplicación:** Incorpora aplicaciones de red estándar (Telnet, SMTP, FTP, etc.). (Amaya Prieto, 2017). En la capa de aplicación se encuentra el protocolo de mensajería avanzado en cola (AMQP), permitiendo que las aplicaciones "cliente" sean capaces de comunicarse con el bróker e intercambien información, sus elementos deben conectarse a cadenas de procesamiento que se encuentra en el servidor, intercambiar para luego recibir mensajes de las aplicaciones en los publicadores y enrutar el mensaje, estableciendo la relación entre la cola del mensaje y el intercambiador.

2.2. Protocolos de mensajería para sistemas IoT: MQTT, AMQP, CoAP y HTTP.

En esta parte del documento se presenta los cuatro protocolos de mensajería aceptados y de suma importancia para sistemas IoT como son MQTT, AMQP, CoAP y HTTP, lo cual se encuentran en la parte superior de la pila de familia de protocolos como se puede observar en la figura 2.2.

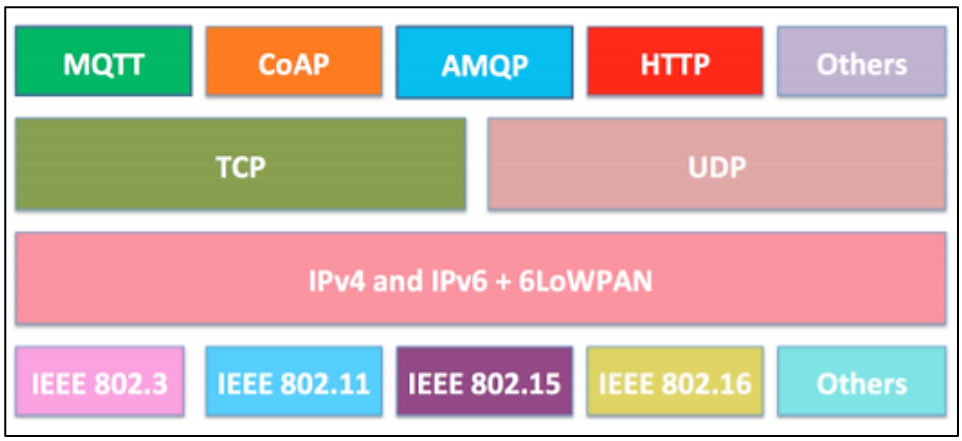


Figura 2.2 Pila de familia de protocolos para sistemas IoT.
Fuente: (Naik, 2017).

Se conoce a la pila de protocolos como a una colección ordenada de éstos que se encuentran organizados en capas, lo cual se colocan una encima de otra como se puede observar en la figura 2.2. y en donde los protocolos de la capa inferior prestan sus servicios a los demás protocolos de la capa superior pero los de la capa superior al absorber los servicios inferiores, también realizan su propia funcionalidad.

2.2.1. MQTT (Message Queuing Telemetry Transport o Protocolo Cola de Mensaje de Telemetría y Transporte).

MQTT es uno de los protocolos de comunicación M2M (machine to machine) más antiguo, introducido en 1999. Fue desarrollado por Andy Stanford-Clark de IBM y Arlen Nipper de Arcom Control Systems Ltd (Eurotech). Es un protocolo de mensaje de publicación/suscripción diseñado para comunicaciones M2M livianas en redes restringidas.

El cliente MQTT publica mensajes a un intermediario o bróker MQTT, que están suscritos por otros clientes o puede ser considerado para la futura suscripción. Cada mensaje se publica en una dirección, conocida como un “topic” o tema. Los clientes pueden suscribirse a múltiples temas y recibe todos los mensajes publicado a cada tema. (Naik, 2017).

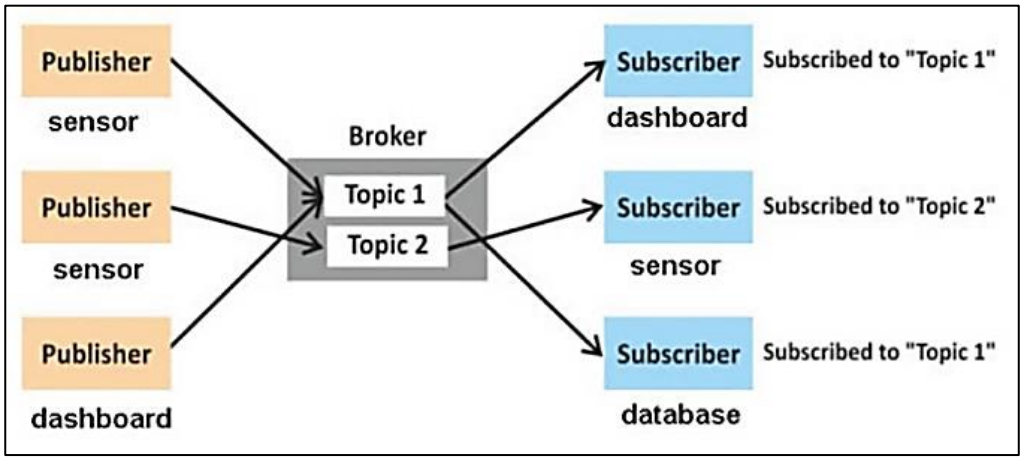


Figura 2.3 Ejemplo de arquitectura básica de MQTT

Fuente: (Molinero, 2018)

2.2.2. CoAP (Constrained Application Protocol o protocolo de aplicación restringida).

CoAP es un protocolo M2M liviano del Grupo de Trabajo IETF CoRE (Entornos RESTful Restringidos). CoAP admite la arquitectura de solicitud/respuesta (una variante de publicación/suscripción).

CoAP está desarrollado principalmente para interoperar con HTTP y RESTful Web a través de simples proxies. A diferencia de MQTT, CoAP usa el Identificador Universal de Recursos (URI) en lugar de los temas. El

publicador, publica datos en el URI y el suscriptor se suscribe a un recurso en particular indicado por el URI. Cuando un editor publica nuevos datos en el URI, a todos los suscriptores se les notifica sobre el nuevo valor como lo indica el URI. (Naik, 2017).

REST representa una forma simple de intercambiar datos entre clientes y servidores sobre HTTP, puede verse como un protocolo de conexión que depende de la arquitectura cliente/servidor. Se usa en aplicaciones móviles y de redes sociales (Molinero, 2018).

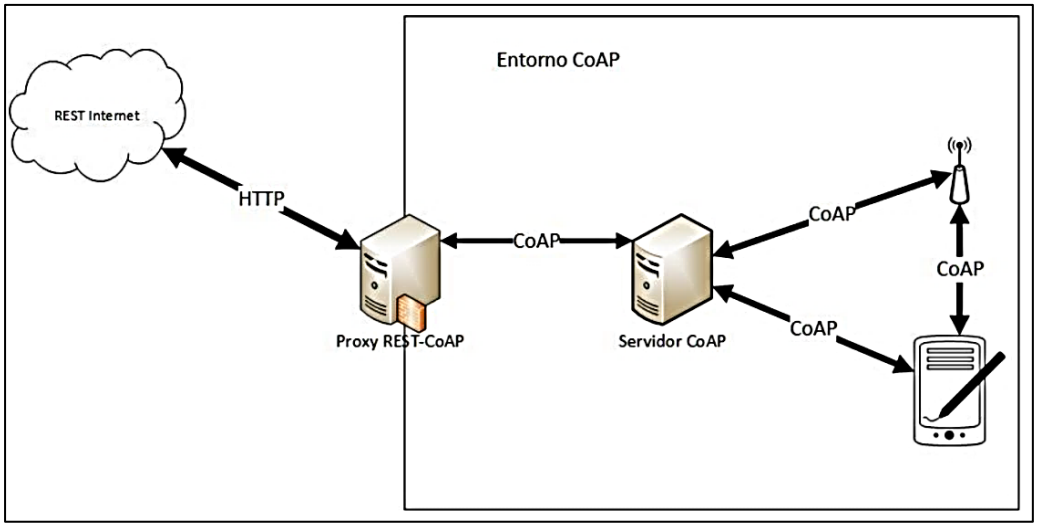


Figura 2.4 Entorno CoAP
Fuente: (García González, 2017)

2.2.3. HTTP (Hyper Text Transport Protocol o protocolo de transferencia de hipertexto).

HTTP es un protocolo de mensajería web, que fue desarrollado originalmente por Tim Berners-Lee. El protocolo HTTP soporta solicitud/respuesta y la arquitectura web RESTful. A semejanza con CoAP, HTTP utiliza el Identificador Universal de Recursos (URI) en lugar de topics (temas).

El servidor envía datos a través del URI y el cliente recibe datos a través de un URI particular. HTTP es un protocolo basado en texto y no define el tamaño de la carga útil del encabezado y el mensaje, sino que depende del servidor web o de la tecnología de programación.

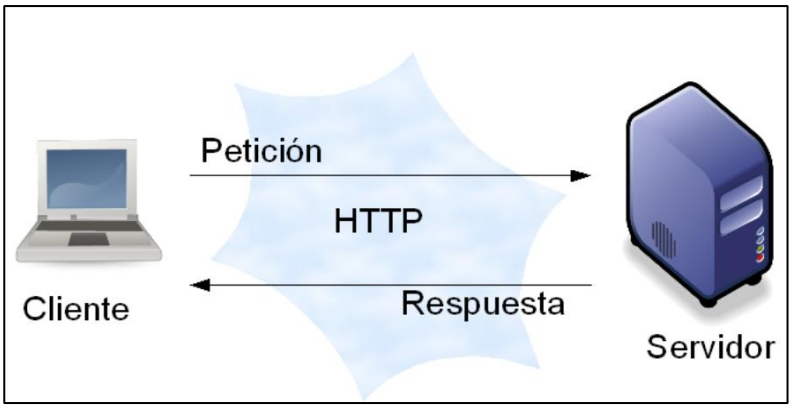


Figura 2.5 Arquitectura Básica HTTP

Fuente: (Rochin, 2016)

2.2.4. AMQP (Advanced Message Queuing Protocolo o Protocolo de Mensajería Avanzado en cola).

El protocolo que se estudia en esta investigación es el protocolo AMQP de la IoT. Es otro protocolo que publica y suscribe el cual proviene del sector de servicios financieros. Tiene su presencia en TIC (Tecnología de la información y comunicación). El mayor beneficio de AMQP es su modelo robusto de comunicaciones que soporta transacciones. A diferencia de MQTT, AMQP puede garantizar transacciones completas, lo cual es útil. AMQP se agrupa a menudo con protocolos IoT, pero es bastante limitada en la industria (Semle & eFalcom, 2016).

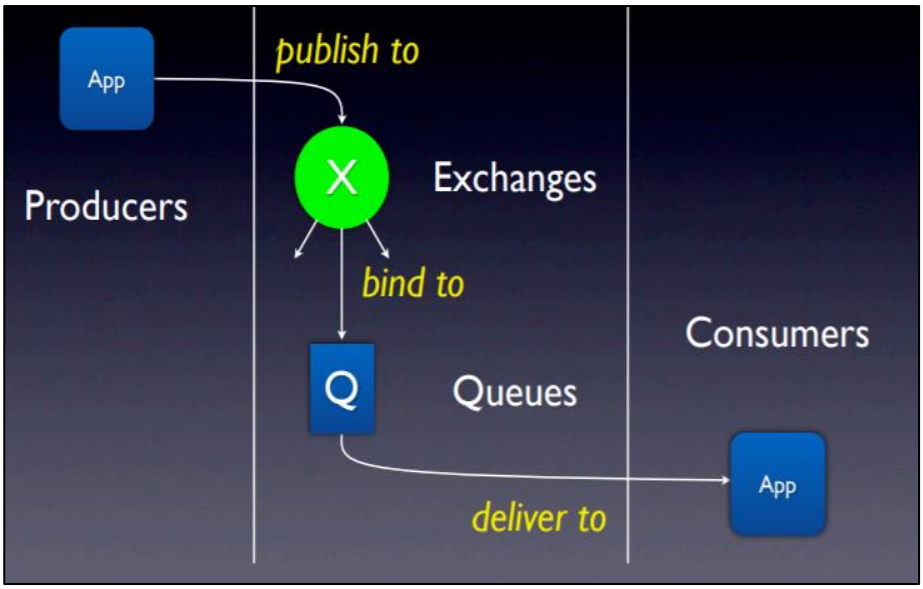


Figura 2.6 Modelo del protocolo AMQP

Fuente: (Garnock-Jones, 2010)

2.3. Cuadro de comparación a gran escala de protocolos de mensajería para sistemas IoT.

En esta sección se muestra un cuadro comparativo a gran escala de los protocolos de mensajerías para sistemas IoT: MQTT, CoAP, AMQP y HTTP.

Tabla 2.2 Análisis comparativo a gran escala de protocolos de mensajería para sistemas IoT: MQTT, CoAP, AMQP y HTTP.

Criterios	MQTT	CoAP	AMQP	HTTP
Año	1999	2010	2003	1997
Arquitectura	Cliente/Bróker	Cliente/Servidor o Cliente/Bróker	Cliente/Bróker o Cliente/Servidor	Cliente/Servidor
Abstracción	Publicar/Subscribir	Solicitud/Respuesta o Publicar/Subscribir	Publicar/Subscribir o Solicitud/Respuesta	Solicitud/ Respuesta
Tamaño del encabezado	2 Byte	4 Byte	8 Byte	Indefinido
Tamaño del mensaje	Pequeño e indefinido (hasta 256 MB de tamaño máximo)	Pequeño e indefinido (normalmente pequeño para caber en un solo datagrama de IP)	Negociable e indefinido	Largo e indefinido (depende del servidor web o de la tecnología de programación)
Semántica/ Métodos	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Consume, Deliver, Publish, Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close	Get, Post, Head, Put, Patch, Options, Connect, Delete
Soporte de Cache y Proxy	Parcial	Sí	Sí	Sí
Calidad de Servicio (QoS)/ Confiabilidad	Sí	Sí	Sí	Limitado (vía del protocolo de Transporte - TCP)
Normas	OASIS, Eclipse Foundation	IETF, Eclipse Foundation	OASIS, ISO/IEC	IETF y W3C
Protocolo de Transporte	TCP (MQTT-SN puede usar UDP)	UDP, SCTP	TCP, SCTP	TCP
Seguridad	TLS/SSL	DTLS, IPSec	TLS/SSL, IPSec, SASL	TLS/SSL
Puerto Predeterminado	1883/ 8883 (TLS/SSL)	5683 (UDP Port)/ 5684(DLTS)	5671 (TLS/SSL), 5672	80/ 443 (TLS/SSL)
Formato de Codificación	Binario	Binario	Binario	Texto
Modelo de Licencia	Open Source - Software de código abierto	Open Source - Software de código abierto	Open Source - Software de código abierto	Gratis
Apoyo Organizacional	IBM, Facebook, Eurotech, Cisco, Red Hat, Software AG, Amazon Web Services	Grandes Comunidades de Soporte Web, Cisco, Contiki, Erika, IoTivity	Microsoft , JP Morgan, Bank of America, Barclays, Goldman Sachs, Credit Suisse	Global Web Protocol Standard

Nota: En esta sección se presenta un cuadro comparativo de los protocolos de IoT según los criterios mostrados en la tabla

Fuente: (Naik, 2017)

2.4. Domótica y sus principales aplicaciones.

2.4.1. Definición de domótica

La palabra Domótica proviene de la unión de las palabras domus (casa en latín) y tica (de automática), vocablo griego que indica “funciona por sí sola”, en fin, la domótica no tiene que ver únicamente con el confort, como puede pensarse a primera vista. Entre sus fines también figuran el ahorro energético, la protección de bienes (seguridad y vigilancia) y el uso de energías renovables (nuevas fuentes de energía y calor) (G. Morales, 2011).

El inicio de la domótica se justifica por el ritmo de vida que es cada vez más acelerado y la rápida evolución tecnológica de la electrónica e informática, que han inundado nuestro entorno con televisiones, teléfonos, equipos de fax, modem, redes y sistemas informáticos, por medio de sistemas tecnológicos integrados. De esa necesidad, surge el concepto de “Domótica”, implantándose en los hogares como medio para satisfacer las necesidades básicas de seguridad, comunicación, gestión energética y confort del usuario (G. Morales, 2011).

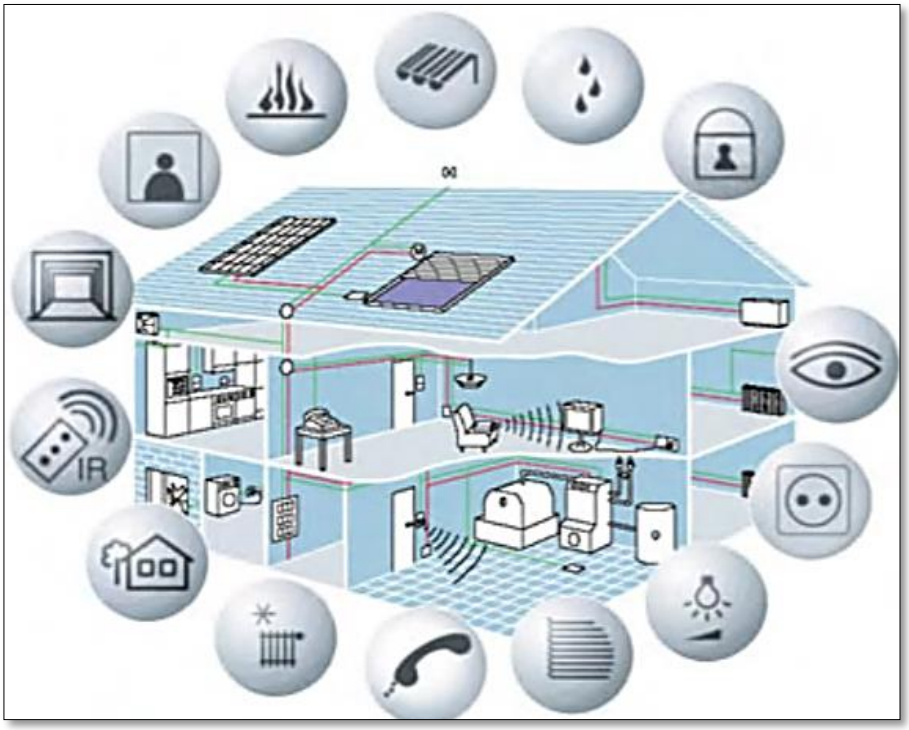


Figura 2.7 Automatización y control del hogar digital

Fuente: (APIEM, 2007)

2.4.2. Diferentes dispositivos del IoT en la domótica para AMQP.

El internet de las cosas es un nuevo concepto de la domótica que está evolucionando cada día, ya sea, por su bajo costo, por plug & play (enchufar, conectar y usar, es una novedosa tecnología que permite a un dispositivo informático conectarse a una computadora sin hacer configuraciones) y proporciona soluciones sencillas sin tener que involucrarse en el mundo profesional de los sistemas. El mundo del IoT se está expandiendo en las diferentes aplicaciones como: la medicina, aplicaciones de consumo, transporte, industrias, etc.

En IoT existen una gran cantidad de dispositivo, de las cuales algunos serán de pequeños recursos, como los sensores o actuadores, pero en otro caso serán más grandes, como un servidor (ejemplo el bróker AMQP, ver figura 2.8) que es capaz de recoger información, almacenar datos, y procesar estadísticas. Respecto a la gran variedad y la cantidad de dispositivos, hoy en día, se requiere interoperabilidad y seguridad, es decir que funcione correctamente a pesar de la variedad de dispositivos, sistemas operativos, y lenguajes de programación, lo cual lo brinda el protocolo AMQP.

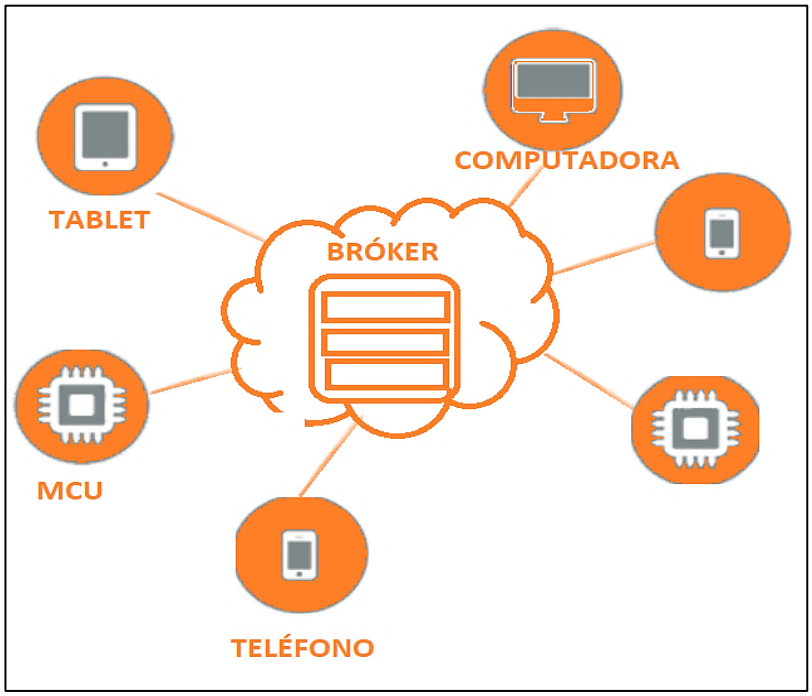


Figura 2.8 Servidor Central para la comunicación de dispositivos.

Elaborado por: Autor

Una aplicación de un bróker de mensajería AMQP, es en una cola de mensaje o “Message Queue”, la cual se define como una aplicación de mensajería, tales como, Whastapp o Telegram, donde sin importar estar conectado, el usuario recibirá los mensajes.

Otra aplicación muy común, pero poco usado en la actualidad, es el buzón de correo del hogar, si alguien está de viaje, todos los mensajes o sobres estarán esperando en el buzón.

En esta parte abarca, los dispositivos físicos (CPU, sensores, actuadores) se conocen como Hardware, pero a la vez el Software es el que envía instrucciones al Hardware haciendo posible su funcionamiento.

Posibilitan que objetos de la vida cotidiana interactúen entre ellos y los seres humanos a través de Internet o redes dedicadas, recopilando información del entorno o interactuado con él. Estos dispositivos son cada vez de menor tamaño, facilitando su integración en cualquier objeto. (García González, 2017). Sensores y actuadores son piezas fundamentales para IoT.

Sensores: La información recogida por los sensores es convertida al mundo digital para poder así conocerla, almacenarla y enviarla a otros dispositivos (García González, 2017). Los sensores pueden clasificarse de acuerdo a la magnitud:

- **Físicos:** Transforman una magnitud física en información. Se encuentran dentro de esta clasificación los sensores de:
  - temperatura,
  - presión,
  - acelerómetros,
  - galgas extensiométricas,
  - sensores de lux giróscopos,
  - inclinómetros, entre otros.

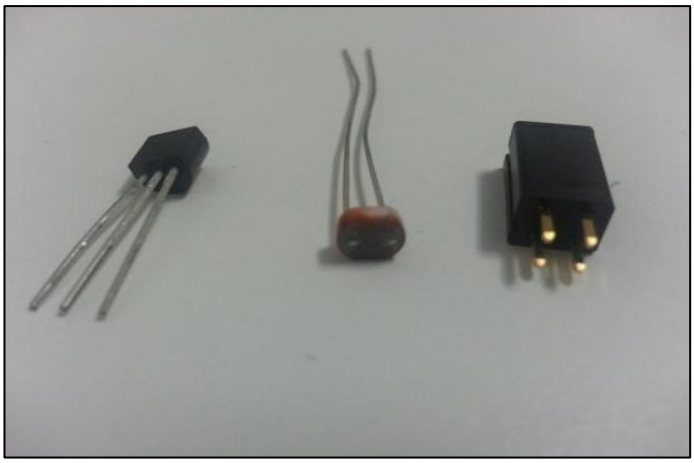


Figura 2.9 Diferentes tipos de sensores.

Fuente: (García González, 2017).

- **Químicos y bioquímicos:** Sensores químicos miden concentraciones de distintos elementos o moléculas, proporcionando lecturas de la concentración de los elementos o moléculas medidas respecto del entorno. Los sensores bioquímicos pueden medir la concentración concreta de una proteica, ácido nucleico, entre otros (García González, 2017).

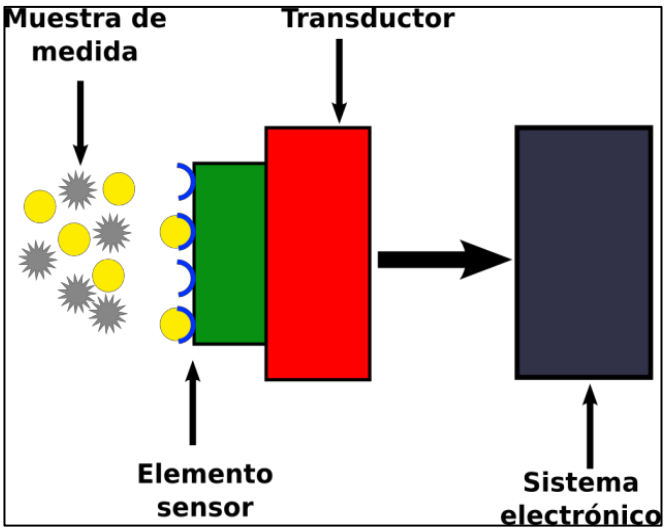


Figura 2.10 Esquema de sensor bioquímico.

Fuente: (García González, 2017).

Actuadores: No están tan implantados como los sensores y a diferencia de éstos, los actuadores, a partir de una información digital, actúan en el mundo real, como: motores y teléfonos móviles.

- **Motores:** Dentro de los actuadores, los de uso más común, son los motores, existen múltiples tipos y formas diferentes de controlarlos. Habitualmente estos se controlan utilizando modulación por ancho de pulso (PWM). Se envían variables por anchos de pulsos para que el motor gire de forma proporcional a la anchura del pulso. Ejemplo, los motores paso a paso (Steppers) que son motores que pueden avanzar un determinado número de grados o pasos (steps) respecto de su eje (García González, 2017).

Figura 2.11 Motor paso a paso

Fuente: (García González, 2017).

- **Teléfonos móviles (smartphones):** Pueden ser sensores y actuadores. Además de permitir recibir llamadas y realizarlas, entre otras muchas más funcionalidades, disponen de múltiples sensores y actuadores (García González, 2017):

➢ Acelerómetro: Permite medir movimientos y conocer la posición del móvil.
➢ Magnetómetro: Mide el campo magnético de la tierra.
➢ Giróscopo: Mide movimientos, el ángulo y la velocidad de giro en las tres coordenadas espaciales.
➢ Sensores de iluminación: Registra la cantidad de luz ambiental.
➢ Sensores de temperatura: Temperatura ambiental limitada a un rango de valores habituales (-20º a 50º).
➢ Sensores acústicos: Están equipados con micrófonos que permiten registrar el sonido.
➢ Barómetro: Mide la presión atmosférica.

- Sensor táctil: Recibe entradas múltiples simultáneamente.
- GPS: Proporciona la posición en coordenadas espaciales y la velocidad a la que se mueve el dispositivo, con precisiones de pocos metros y Km/h.

2.4.3. Servicios según sus aplicaciones en el campo de la domótica.

Los servicios que brinda la domótica se agrupan según sus aplicaciones o ámbitos principales como se describe a continuación:

- Ahorro energético se define como la acción de gestionar las energías que usan en una propiedad, esta gestión contiene tres pilares fundamentales:

- El ahorro energético: Busca reducir el consumo evitando el despilfarro de energía, un ejemplo sería un aviso de puertas y ventanas abiertas cuando la calefacción se encuentre encendida.
- La generación de energía: Busca transformar los tipos de energía como química, térmica, entre otros, en energía eléctrica.
- La eficiencia energética: No disminuye el consumo de energía, pero logra que éste sea aprovechado a su máxima potencia. En esta aplicación, la domótica cuenta con suficiente inteligencia.

- Confort: Conlleva todas las acciones que puedan llevar a cabo para mejorar el confort de la vivienda como: iluminación, apagado general de todas las luces de la vivienda, automatización del apagado/ encendido en cada punto de luz, regulación de la iluminación según el nivel de luminosidad ambiente, entre otros. (G. Morales, 2011)

- Seguridad: Consiste en una red de seguridad encargada de proteger tanto los bienes patrimoniales y la seguridad personal, por ejemplo, simulación de presencia, alarmas de detección de incendio, fugas de gas, escapes de agua, acceso a cámaras IP. (G. Morales, 2011)

- Comunicaciones: Son los sistemas o infraestructuras de comunicaciones que posee el hogar. Ubicuidad en el control tanto externo como interno, control y acceso remoto desde Internet, transmisión de alarmas, entre otros. (G. Morales, 2011)

- Telegestión y Accesibilidad: Este enfoque constituye un reto ético y creativo. Donde las personas con discapacidad puedan acceder a estas tecnologías sin temor a un obstáculo del tipo de tecnología o arquitectura, por ejemplo, las casas domotizadas. (G. Morales, 2011)

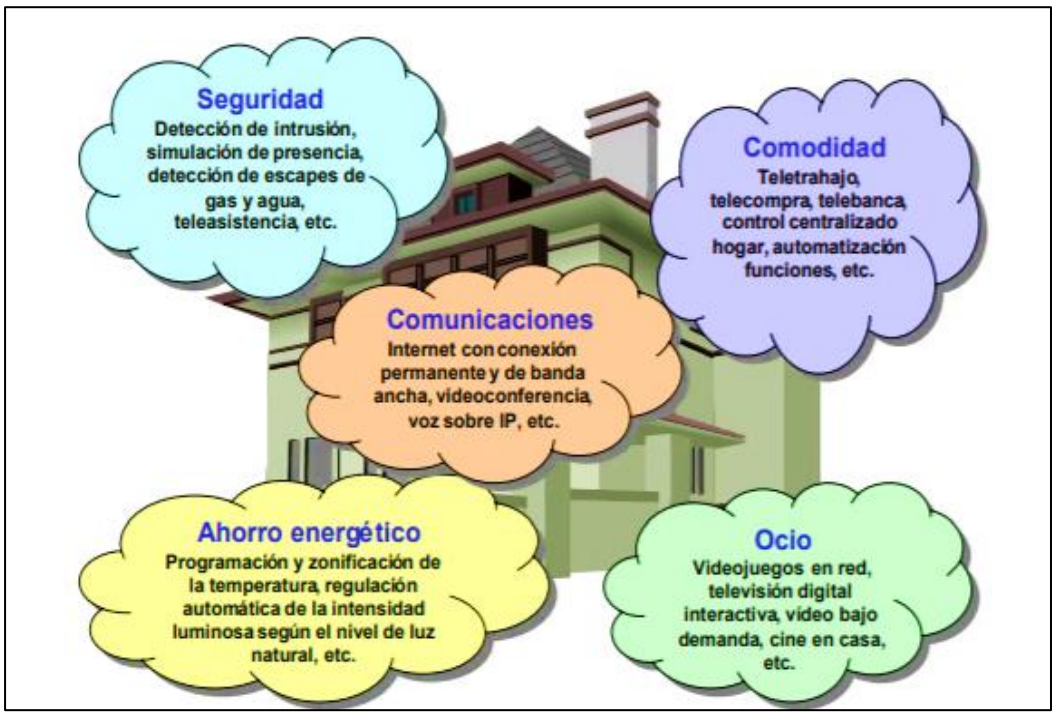


Figura 2.12 Esquema de aplicaciones de la domótica.
Fuente: (APIEM, 2007)

En definitiva, la domótica de hoy, contribuye a aumentar la calidad de vida, hace más versátil la distribución de la casa, cambia las condiciones ambientales creando diferentes escenas predefinidas, ayuda al ahorro energético, seguridad y consigue que la vivienda sea más funcional al permitir desarrollar facetas domésticas, profesionales, y de ocio bajo un mismo techo. Todo ello hace que se aproveche más el tiempo que pasamos en casa, que hoy día tiende a ser cada vez menor. (G. Morales, 2011)

2.5. El uso del estándar ISO y el Middleware de mensajería.

Las empresas tuvieron la necesidad de crear su propio middleware de mensajería, de las cuales en el camino algunas empresas han salido del mercado o se han separado para convertirse en soluciones comerciales patentadas. El middleware de mensajería es un software complejo que intercambia información entre aplicaciones, representada por una capa de software ubicada entre la capa más alta (usuarios y aplicaciones), y la más baja (sistemas operativos y mecanismos de comunicación básicos) (Talaminos Barroso, 2011).

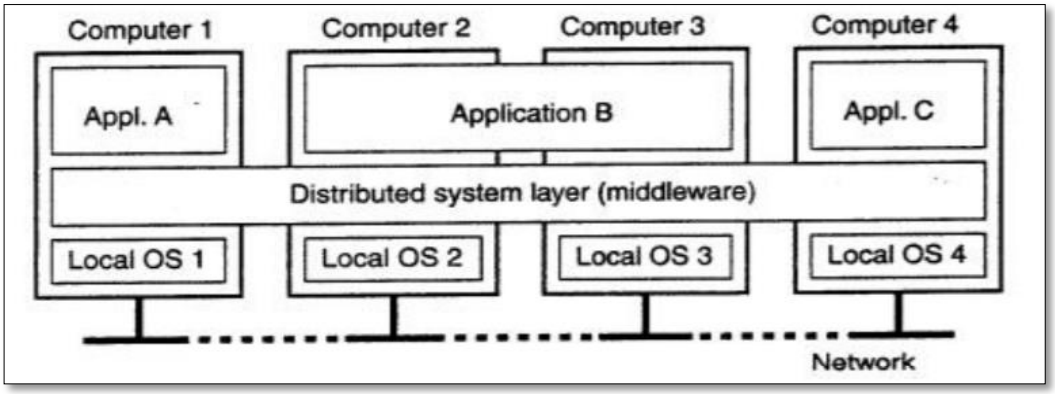


Figura 2.13 Concepto de Middleware
Fuente: (Talaminos Barroso, 2011)

Están diseñados para el intercambio de mensajes entre procesos de forma asíncrona. De esta forma, las aplicaciones únicamente se encargan de "colocar" y "sacar" mensajes de las colas, no se conectan directamente entre ellas. No existen los términos de cliente y servidor, por lo que se pierde el concepto de petición de servicio, y cualquier entidad puede participar en el intercambio de información en cualquier momento, por lo que no necesariamente se requiere respuesta (Talaminos Barroso, 2011).

Construyen su propio middleware porque la industria de servicios tiene necesidades exigentes de mensajería tanto en la entrega garantizada como en la publicación/suscripción. Las demandas a menudo superan las capacidades del software actualmente disponible. Construir nuestro propio middleware es, por lo tanto, un enfoque creíble (O'Hara, 2007).

La estandarización de la mensajería a un protocolo común permite que cualquier sistema de software que implementa el protocolo se comunique con cualquier otro sistema utilizando el mismo protocolo. Como un estándar abierto, AMQP es un estándar internacional que cuenta con la certificación ISO e IEC como ISO / IEC 19464: 2014. (Pietschmann, 2016). Se designa el formato ISO usando ISO (/IEC) nnnnn: (yyyy), en la cual nnnnn es el número del estándar, yyyy es el año de publicación. IEC contiene si el estándar es el resultado del trabajo de ISO/IEC JT1 (El Comité Conjunto Técnico) (ISO, 2006).

ESTÁNDAR	EDICIÓN	Título	Comité
ISO/ IEC 19464: 2014	1st	Información de tecnología -- Advanced Message Queuing Protocol (AMQP) v1.0 specification	JTC1

Figura 2.14 Estandarización ISO de la mensajería del protocolo AMQP.
Elaborado por: Autor

ISO (la Organización Internacional de Normalización) e IEC (la Comisión Electrotécnica Internacional) conforman el sistema especializado para la estandarización mundial.

Los organismos nacionales que son miembros de ISO o IEC participan en el desarrollo de normas internacionales a través de comités técnicos establecidos por la organización respectiva para tratar campos específicos de actividad técnica. Los comités técnicos de ISO y IEC colaboran en campos de interés mutuo. (ISO, 2014).

ISO / IEC 19464 fue preparado por el Comité Técnico del Protocolo Avanzado de Message Queue Server (como OASIS Advanced Message Queuing Protocol (AMQP)) y fue adoptado, bajo el procedimiento PAS, por el Comité Técnico Conjunto ISO / IEC JTC 1, Tecnología de la información. paralelamente a su aprobación por los organismos nacionales de ISO e IEC. PAS fue creado para estar al tanto de cómo y cuándo actuar en caso de algún accidente, son las iniciales de las palabras: Proteger, Avisar y Socorrer. (ISO, 2014)

2.6. Protocolo de Mensajería Avanzado en Cola (AMQP).

2.6.1. Definición del protocolo AMQP.

El protocolo AMQP, en inglés es Advanced Message Queuing Protocol, se define como la IP de los sistemas de negocios que permiten que las transmisiones pasen entre aplicaciones y un objetivo importante del protocolo AMQP fue lograr la creación de pilas de protocolo de estándar abierto para la mensajería de empresas, dentro de una igual organización como entre diferentes organizaciones. Otra definición del protocolo AMQP, es un protocolo de comunicación o estándar abierto con un conjunto bien definido de comportamientos para transmitir mensajes de aplicación entre sistemas que utilizan una combinación de almacenamiento y envío, publicación y suscripción, y otras técnicas (O'Hara, 2007).

Se usa el término "mensajes de aplicación" para distinguir AMQP de mensajería instantánea u otras formas de mensajería del usuario final. AMQP aborda el escenario en el que es probable que exista algún impacto económico si se pierde un mensaje, no llega de manera oportuna o se procesa incorrectamente. El protocolo está diseñado para ser utilizado desde diferentes entornos de programación, sistemas operativos y dispositivos de hardware (O'Hara, 2007).

Para intercambiar información y para transmitir datos, los servicios tienen que ser capaces de comunicarse entre sí. Para evitar el caos por el tipo de conversaciones entre diversas aplicaciones, es de suma importancia mantener un protocolo para superar varias dificultades, como en informática que hay barreras lingüísticas, es decir varios lenguajes de programación, por lo que es necesario un protocolo.

El protocolo AMQP, Advanced Message Queuing Protocol se muestra como una solución, es decir un protocolo común que consiste en transportar información por medio de un intermediario. El mencionado protocolo a estudiar, solventa distintos problemas en un mismo periodo de tiempo:

- Por una parte, el protocolo AMQP con ayuda de un bróker de mensajería, es el encargado de una transmisión sólida de datos.
- Por otra parte, el protocolo AMQP almacena mensajes en una cola. La cual va a permitir una comunicación asíncrona, es decir que el transmisor y receptor no se comunican al mismo tiempo. No necesariamente el receptor o consumidor del mensaje debe aceptar, procesar la información y dar confirmación de la recepción al emisor o productor. Al contrario, se recupera el mensaje de la cola cuando tenga la capacidad disponible para ello. Lo cual brinda al productor, la posibilidad de continuar trabajando y prevenir tiempos de inactividad.

El diseño del modelo AMQ fue impulsado por estos requisitos principales:

- Ser flexible y extensible, pero simple.
- Garantizar la interoperabilidad entre implementaciones conformes.
- Para proporcionar un control explícito sobre la calidad del servicio.

La popularidad del relativamente novedoso protocolo AMQP, se relaciona con la interoperabilidad, lo cual es una de sus importantes características. El protocolo de mensajería avanzado en cola establece un principio, permitiendo que diferentes aplicaciones puedan interactuar en distintos lenguajes de programación.

De este modo, los programas son capaces de comunicarse entre sí sin complicación alguna, a pesar de que no pertenezcan a las mismas empresas proveedoras de software.

También, AMQP es un protocolo que se encuentra disponible de manera gratuita, por lo tanto, cualquier empresa puede recurrir al uso del protocolo sin disponer de costos adicionales.

2.6.2. Características del AMQP.

Las capacidades de middleware de mensajería, como uno de los objetivos de AMQP, es que puedan llegar a la red, y que, a través de la disponibilidad generalizada de middleware de mensajería, se puedan desarrollar nuevos tipos de aplicaciones útiles (Vinoski, 2006). Adicional de ser un protocolo de mensajería común entre los sistemas, el protocolo AMQP se diseñó con las siguientes características:

Tabla 2.3 Características del protocolo AMQP.

CARACTERISTICAS DEL PROTOCOLO AMQP	
Protocolo	Seguro y exacto.
	Confiable.
	Interoperable y estandarizado.
	Abierto.
	De transporte TCP y de red IP.
	Localizado en capa 7 (capa de aplicación) del modelo OSI.
	Orientado a mensajes.
	Encolamiento de mensaje (queuing).
	De enrutamiento (publicación/suscripción).

Nota: Se presenta las características más distinguidas del protocolo de mensajería avanzado.
Elaborado por: Autor

2.6.3. Ventajas y desventajas del protocolo AMQP

Tabla 2.4 Ventajas y desventajas del protocolo AMQP.

TABLA DE VENTAJAS Y DESVENTAS DEL PROTOCOLO DE MENSAJERÍA AVANZADO EN COLA
VENTAJAS
Confiable: AMQP trabaja sobre TCP y de esta manera garantiza que los mensajes sean entregados al destino sin errores y en el mismo orden que se transmitieron.
Asíncrono: Los mensajes publicados en el bróker son enviados a los suscriptores bajo una previa solicitud de éstos. De este modo, la información puede ser tratada de forma asíncrona y los publicadores se pueden abstraer del estado de los suscriptores.

Fiable: AMQP garantiza la entrega de los mensajes mediante el intercambio de mensajes de control.
Contempla el uso de transacciones: AMQP permite el intercambio de mensajes a través de transacciones.
Soporta control de acceso: AMQP soporta el uso de credenciales para restringir el acceso a los recursos del bróker a aquellos clientes que estén autorizados
Soporta encriptación TLS (Transport Layer Security): Permite la implementación del protocolo TLS para el cifrado de las comunicaciones. TLS permite establecer una conexión segura por medio de un canal entre el cliente y el servidor.
DESVENTAJAS
Alto uso de recursos: Los clientes AMQP requieren de una cierta capacidad de procesamiento. No está pensado para su implementación en pequeños dispositivos con pocos recursos computacionales. Además, debido a la gran cantidad de tráfico de control que se introduce para ofrecer fiabilidad no está pensado para su implementación en redes con bajo ancho de banda.
Centralizado: El paradigma publicación/suscripción se basa en un equipo central (bróker) a través del cual los clientes intercambian toda la información. Es posible la implementación de clústers. Sin embargo, esto supone un aumento de la complejidad. Esta característica supone una limitación a la hora de ofrecer disponibilidad.

Nota: Tabla de ventajas y desventajas del protocolo AMQP según sus características.

Fuente: (Gil, 2018).

2.7. Arquitectura y funciones del protocolo AMQP.

El protocolo AMQP opera en la capa de aplicación del modelo de referencia OSI, en la que tiene contacto directo con los diferentes programas como IMAP “Internet Message Access Protocol”, sirve para correo electrónico; FTP “File Transfer Protocol”, sirve para la transferencia de archivos; e IRC “Internet Relay Chat” se encuentran activos en la capa de aplicación y sirve para mensajería instantánea. Para la transmisión de los mensajes, el protocolo apuesta por los intermediarios: los llamados brókers de mensajería.(O’Hara, 2007)

Desde el principio, el objetivo de diseño de AMQP fue definir MOM (Message Oriented Middleware) para satisfacer las necesidades de la mayoría

de los sistemas informáticos comerciales y hacerlo de una manera eficiente y luego en futuro integrarse en la infraestructura de la red. Incorpora patrones comunes para facilitar el paso de los firewalls mientras conserva la seguridad y permitir la calidad de servicio de la red. Una característica de ejemplo de AMQP es cuando un cliente puede usar AMQP para solicitar servicios de un grupo de servidores conectados a las colas del bróker AMQP.

Entre los servicios suscritos a una cola de solicitudes, y el número de procesos que proporcionan un servicio en la que puede, mediante el bróker AMQP, equilibrar solicitudes al crecer o reducir dinámicamente sin afectar a los clientes. Sin embargo, si el grupo de servicios se reduce a cero, AMQP puede informar al cliente mediante el Modo de Entrega Obligatoria porque puede ser un error operacional de la aplicación.

Un componente del modelo de AMQP es el Host virtual, se define como una colección de intercambios, colas de mensajes y objetos asociados. Los hosts virtuales son dominios de servidores independientes que comparten un entorno de autenticación y cifrado común (Vinoski, 2006).

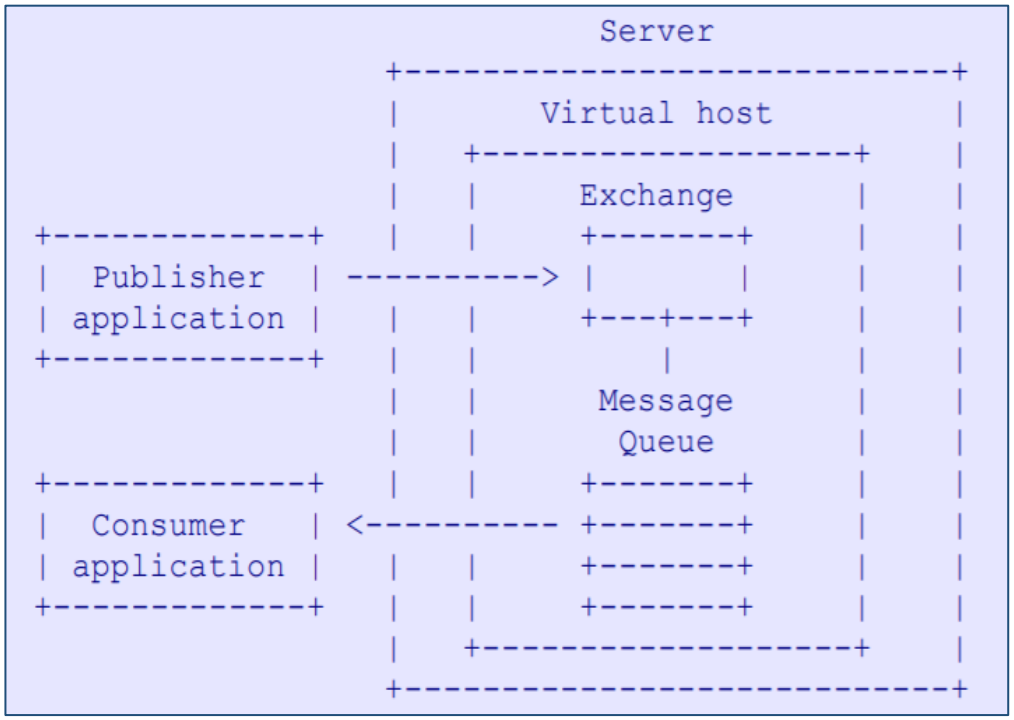


Figura 2.15 Diagrama del modelo general de AMQ

Fuente: (Vinoski, 2006)

Un servidor de middleware es un servidor de datos que acepta mensajes y realiza dos procesos principales con ellos, los dirige a diferentes consumidores según criterios arbitrarios y los almacena en la memoria o en el disco cuando los consumidores no pueden aceptarlos lo suficientemente rápido (Vinoski, 2006).

AMQP tiene un modelo en capas definido de la siguiente manera desde una perspectiva ascendente (Patierno, 2016):

- **Mensajería:** Proporciona capacidades de mensajería a nivel de aplicación en la parte superior de la capa anterior (ver figura 2.17), definiendo la entidad de mensaje como la construcción de uno o más paquetes (Patierno, 2016).

- **Transporte/paquete**: Define el comportamiento de conexión y la capa de seguridad entre pares, encima de un protocolo de transporte de red (por ejemplo, TCP).

Las entidades principales que crean pares en una red AMQP son (Patierno, 2016):

Nodos: Entidades con nombre responsables de almacenar y/o entregar mensajes. En el espacio de mensajería, un nodo podría ser, por ejemplo, un productor/consumidor o una cola. Un nodo es direccionable y se puede organizar de forma plana, jerárquica o gráfica (Patierno, 2016).

Contenedor: En términos generales, un contenedor es una aplicación. Los nodos definidos previamente viven en un contenedor que podría ser un cliente con sus productores y/o consumidores, o un intercambiador con sus entidades de almacenamiento (por ejemplo, las colas) (Patierno, 2016).

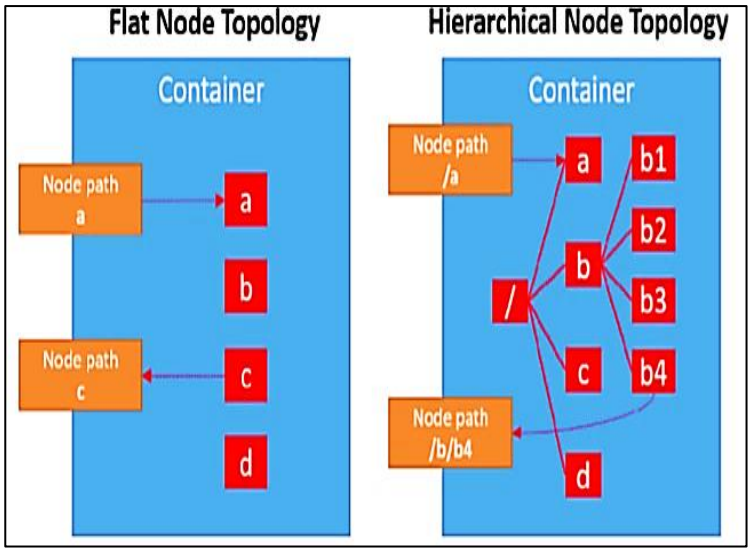


Figura 2.16 Nodos y contenedores.

Fuente: (Patierno, 2016)

- **Transporte de Red:** En cuanto a la capa de transporte de red, AMQP no está fuertemente vinculado a TCP y, como tal, se puede utilizar con cualquier protocolo de transporte de secuencia confiable; por ejemplo, SCTP (Stream Control Transmission Protocol) (Patierno, 2016).

En AMQP, las aplicaciones tienen una relación muy íntima con su middleware, esto debe estar bien definido o de lo contrario la interoperabilidad no se puede lograr. Se espera que TCP / IP sea la opción predeterminada para la mayoría de los usuarios finales para una mejor interoperabilidad (O'Hara, 2007). La capa de transporte se encarga de trasladar los mensajes desde la aplicación al servidor y viceversa, y de manejar la multiplexación de canales, la paquetización, la codificación de contenido, la representación de datos y el manejo de errores (Vinoski, 2006).

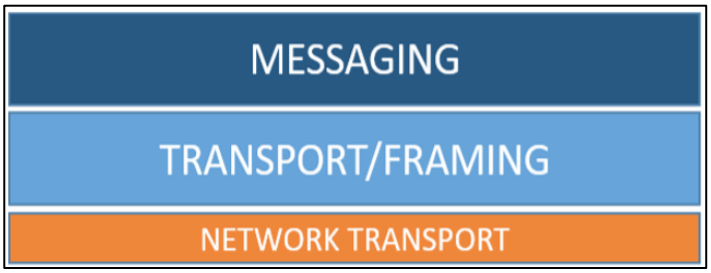


Figura 2.17 Modelo en capas de AMQP.

Fuente: (Patierno, 2016)

Por lo tanto, en el nivel de transporte, se comunican entre sí, los contenedores y los nodos y son capaces de intercambiar mensajes. En primer lugar, se establece una conexión TCP entre los contenedores con o sin una capa de seguridad mediante el protocolo SSL/TLS. Después de eso, una conexión AMQP se crea encima de la conexión de red, debido al intercambio de algunos paquetes del preámbulo de la conexión con la información de la versión del protocolo; la misma seguridad de nivel de transporte (por ejemplo, SSL/TLS) se puede negociar en línea usando el protocolo SASL (Simple Authentication and Security Layer).

Tal conexión proporciona una comunicación dúplex completa con una secuencia ordenada de paquetes cuyo tamaño máximo se negocia para proporcionar un primer nivel de control de flujo. La misma conexión se divide en canales multiplexados y unidireccionales, y todos los paquetes fluyen a través de ellos con un identificador de canal asignado. Después de la conexión, una sesión AMQP se establece entre dos pares; enlaza dos canales unidireccionales para formar una conversación bidireccional con un mecanismo de control de flujo basado en el número de tramas intercambiadas. Por supuesto, una conexión admite varias sesiones. Por último, para intercambiar mensajes entre nodos, se crea un enlace (link) AMQP entre ellos. La cual es una ruta unidireccional asociada a cada nodo que podría ser el origen o el destino, y es responsable de realizar un seguimiento del estado de intercambio de un mensaje. El enlace proporciona el tercer nivel de control de flujo. (Patierno, 2016).

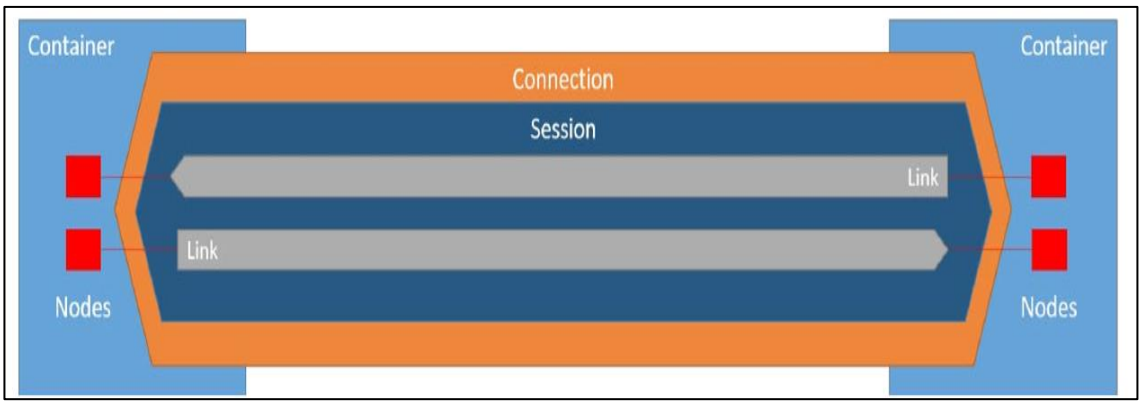


Figura 2.18 Contenedores, sesiones y enlaces.

Fuente: (Patierno, 2016)

Las conexiones y las sesiones son efímeras, por lo que no conservan ningún estado si se pierde la conexión de red; en la reconexión, el par tiene que crear una nueva conexión y una nueva sesión no relacionada con las anteriores. Los enlace (binding), sin embargo, son recuperables, y si la red no está disponible durante la transferencia de mensajes, cada enlace se recupera con el estado de entrega del mensaje anterior (relacionado con el QoS solicitado) (Patierno, 2016).

Conexión se define como el contacto que se establece entre dos o más dispositivos para intercambiar información.

Paquetes de AMQP.

Se define paquete o “frame” como la unidad básica en AMQP, por lo tanto, es una conexión compuesta por una secuencia de paquetes.

Se refiere al término de orden a la referencia de que el último paquete no tiene que llegar al receptor antes de que los demás alcancen su objetivo de llegar el mensaje completo y confiable al consumidor. Los paquetes se dividen en tres partes:

- **Encabezado del paquete:** Es un encabezado obligatorio que posee un tamaño de 8 bytes. Se encuentra la información que fija el enrutamiento del mensaje.
- **Encabezado extendido:** Es un encabezado opcional que sirve para extender la información del encabezado si se requiere en un futuro.
- **Cuerpo del paquete:** Se encuentra los datos que se van a trasmitir y el tamaño se selecciona de manera arbitraria. En este campo puede quedar en vacío, ya que el paquete solo mantiene la conexión. Es también una secuencia de bytes que tiene un formato que depende del tipo de trama.

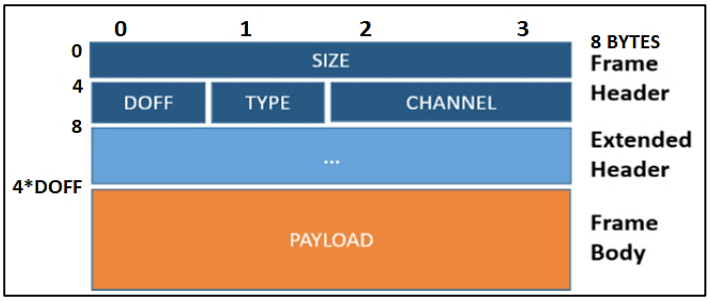


Figura 2.19 Formato de paquetización.

Fuente: (Patierno, 2016).

El cuerpo del paquete es interesante porque se define como payload (carga útil) a la aplicación con datos para transmitir, ya que están relacionados con la apertura/cierre de la conexión, el inicio/finalización de una sesión, la fijación/desvinculación de un enlace, la transferencia de contenido y el control de flujo (Patierno, 2016). Payload es el conjunto de datos a transmitirse incluido el mensaje a enviar. El cuerpo de un paquete puede adoptar nueve formas diferentes:

1. **Open:** Negocia los campos de conexión entre el bróker y el cliente.
2. **Begin:** Muestra que una conexión se ha iniciado.
3. **Attach:** Es necesario para la transferencia de datos y es el que adjunta un enlace al mensaje.
4. **Flow:** Realiza un cambio al estado de un enlace.
5. **Transfer:** Notifica modificaciones en la entrega de la información.
6. **Detach:** Sirve para eliminar el enlace.
7. **End:** Muestra que se ha terminado la conexión.
8. **Close:** Finaliza la conexión e informa que no se enviará más paquetes.

COMUNICACIÓN: OPEN

Para abrir la comunicación con un par de contenedores (ver figura 2.21), primero hay un handshake (apretón de manos) entre AMQP/SASL en la conexión TCP sin procesar, luego AMQP "open" se intercambia para definir el tamaño máximo de paquetes (control de flujo), el número máximo de canales, y así sucesivamente. Dentro de la conexión, se inicia una sesión utilizando

"begin" especificando el tamaño de la ventana (número de paquetes para el control de flujo). Por último, un paquete llamado "attach" se utiliza para adjuntar un vínculo (Patierno, 2016).

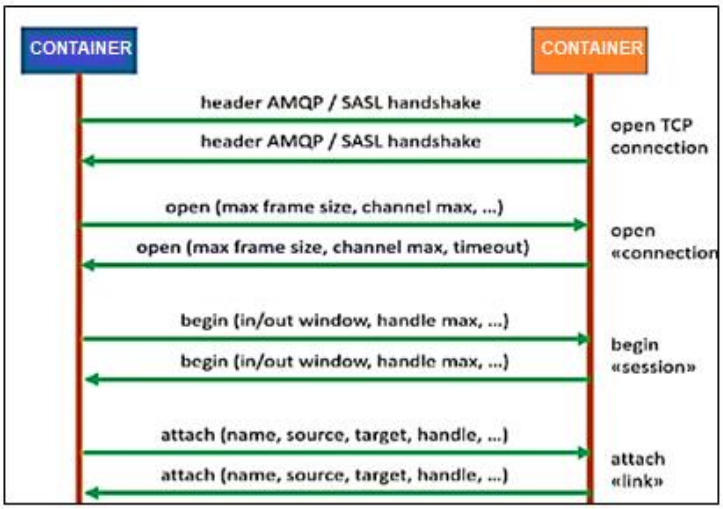


Figura 2.20 Comunicación: Open.

Fuente: (Patierno, 2016)

COMUNICACIÓN: SEND

Después de que se adjunte el enlace, el receptor puede enviar un "flow" (flujo) al remitente especificando el número de crédito para limitar el número de mensajes que puede recibir (control de flujo). El productor envía datos utilizando "transfer", el cual el receptor responde con "disposition" solo si el nivel de QoS está en el nivel uno (Patierno, 2016).

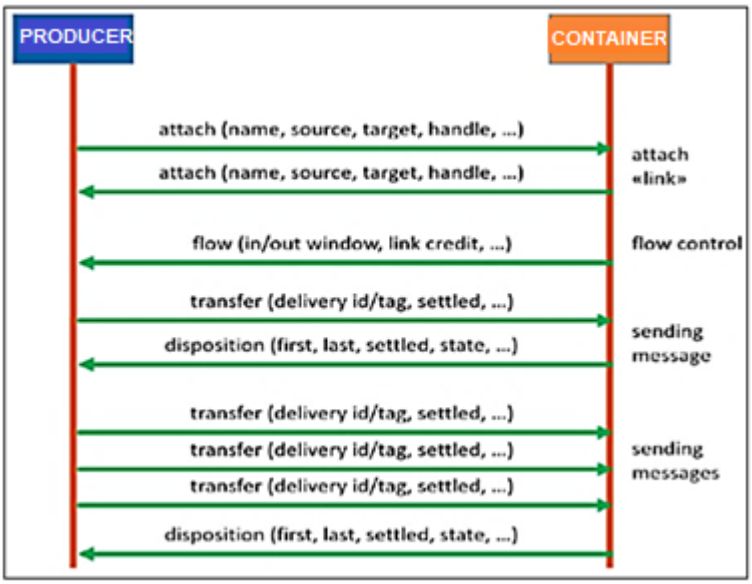


Figura 2.21 Comunicación: Send.

Fuente: (Patierno, 2016).

COMUNICACIÓN: RECEIVE.

Una comunicación de recepción es el flujo opuesto de un envío. El receptor envía “flow” para establecer el control de flujo basado en créditos y cuántos mensajes puede recibir antes de procesarlos. Para uno o más “transfer”, responde con “disposition” si el nivel de QoS es mayor que cero.

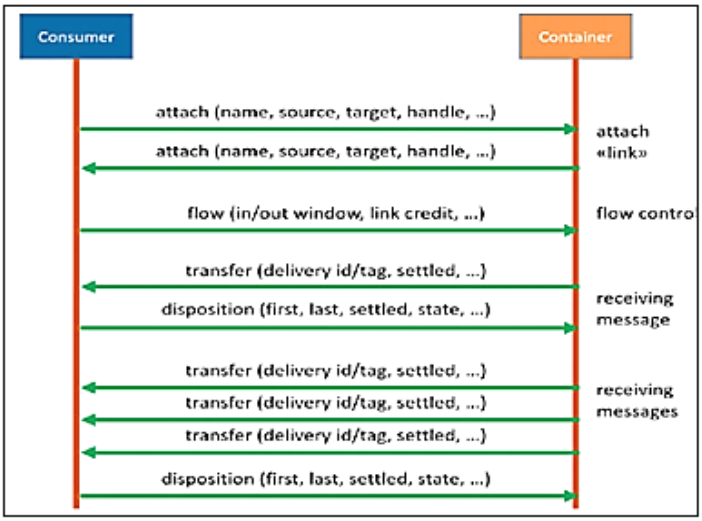


Figura 2.22 Comunicación: Receive.

Fuente: (Patierno, 2016).

COMUNICACIÓN: CLOSE

El cierre de una comunicación significa que se desconecta de todos los enlaces activos utilizando "Detach". Después, "end" se usa para finalizar la sesión, y "close" se usa para cerrar la conexión. Por supuesto, el último paso es cerrar la conexión de red a nivel de socket (Patierno, 2016).

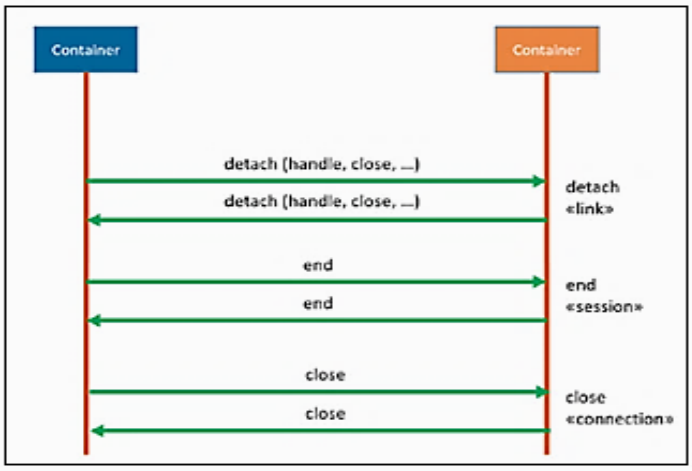


Figura 2.23 Comunicación: Close

Fuente: (Patierno, 2016)

2.7.1. Productor

El productor es la entidad que envía o publica el mensaje, es decir, una aplicación cliente que publica mensajes en un intercambiador o exchange (ver figura 2.24).

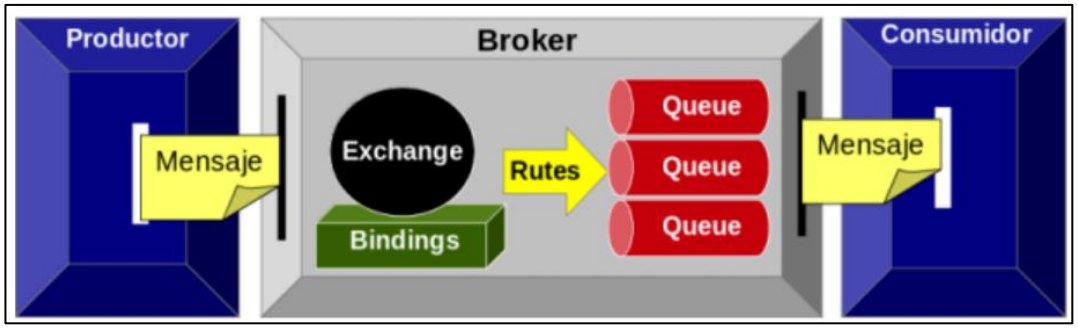


Figura 2.24 Arquitectura de AMQP.

Fuente: (A. Morales, 2014).

2.7.2. Bróker de mensajería.

El Bróker es la entidad que interviene como intermediario de los mensajes entre el consumidor y el productor, es decir el bróker distribuye el mensaje con reglas determinadas en diferentes colas (ver figura 2.24). En el bróker de mensajería se realizan procesos donde el intercambiador “exchange” recibe los mensajes y transporta la información a la cola correcta, dependiendo a qué topic o tema están suscritos los clientes. Los enlaces “bindings” notifica al intercambiador sobre la cola a la que corresponde el mensaje. Existen cuatros formas en las que un intercambiador transmite mensajes, la cual será descrito en este documento.

2.7.3. Intercambiador (Exchange).

Los intercambiadores son el servicio de entrega de mensajes. En la analogía postal, los intercambios proporcionan servicios de clasificación y entrega (O’Hara, 2007).

En el modelo AMQP, seleccionar un proveedor diferente es cómo se seleccionan las diferentes formas de entregar el mensaje. El intercambio utilizado por una operación de publicación determina si la entrega será directa o de publicación/suscripción. El concepto de intercambio es cómo AMQP

reúne y abstrae diferentes modelos de entrega de middleware. También es el punto de extensión principal en el protocolo (O'Hara, 2007).

Un cliente elige el intercambio utilizado para entregar cada mensaje a medida que se publica. El intercambio examina la información en los encabezados de un mensaje y selecciona a dónde deben transferirse. Así es como AMQP reúne los diversos lenguajes de mensajería, los clientes pueden seleccionar qué intercambio desean utilizar para enrutar sus mensajes (O'Hara, 2007). Existen varios tipos de intercambiadores, siendo los siguientes:

- **El intercambiador directo:** Pondrá en cola un mensaje directamente en una sola cola, seleccionando la cola en base al encabezado de "clave de enrutamiento" en el mensaje y haciendo coincidir por nombre. Así es como un cartero envía un mensaje a una dirección postal (O'Hara, 2007).
- **El intercambiador con un tema definido (topic):** Copiará y pondrá en cola el mensaje a todos los clientes que hayan expresado su interés en una coincidencia de patrón rápida con el encabezado de la clave de enrutamiento. Puede pensar en la clave de enrutamiento como una dirección, pero es un concepto más abstracto y útil para varios tipos de enrutamiento (O'Hara, 2007).
- **El intercambiador "fanout"**: Un intercambio de fanout copia el mensaje y lo enruta a todas las colas que están vinculadas a él, se ignora la clave de enrutamiento. (O'Hara, 2007).
- **El intercambio de encabezados:** Examinará todos los encabezados en un mensaje, evaluándolos contra los "headers" de consulta proporcionados por los clientes interesados que usan esos encabezados para seleccionar las colas finales, copiando el mensaje según sea necesario (O'Hara, 2007).

A lo largo de este proceso, los intercambios nunca almacenan mensajes, pero sí conservan los parámetros de enlaces o bindings proporcionado a ellos, por los clientes que los utilizan. Estos enlaces son los argumentos a las

funciones de enrutamiento de intercambio que permiten la selección de una o más colas (O'Hara, 2007).

2.7.4. Enlace (Bindings).

Los argumentos proporcionados a los intercambios para permitir el enrutamiento de mensajes se conocen como enlaces (ver la figura 2.24). Los enlaces varían dependiendo de la naturaleza del intercambio; el intercambio directo requiere menos información vinculante que el intercambio de encabezados (O'Hara, 2007).

En particular, no siempre está claro qué entidad debe proporcionar la información de enlace para una interacción de mensajería en particular. En el intercambio directo, el remitente asocia una clave de enrutamiento y la cola de destino deseada (O'Hara, 2007).

En el intercambio de temas, es el cliente receptor el que proporciona la información de enlace, especificando que cuando el intercambio de temas vea un mensaje que coincida con cualquier(s) enlace(s) cliente(s), el mensaje debe enviarse a todos ellos (O'Hara, 2007).

Por lo tanto, AMQP le permite enrutar sus mensajes entre intercambios entrantes y colas salientes. Un enlace es una unión entre una cola y un intercambio. Un enlace es hacia qué determinadas colas debe dirigirse un mensaje que llega a un intercambio. La forma en que se enruta un mensaje depende del tipo de intercambio al que estaba destinado el mensaje y la "clave de enrutamiento".

2.7.5. Canal y clave de enrutamiento.

El canal es una conexión lógica que se enlaza a otra conexión. Cuando los clientes realizan transacciones simultáneas por medio de una misma conexión se debe conservar un canal diferente para cada una de las transacciones.

El intercambiador examina las propiedades de un mensaje, los campos de encabezado y el contenido de su cuerpo, y al usarlo los datos de otras fuentes, decide cómo enrutar el mensaje. En la mayoría de los casos simples, el intercambio examina un solo campo de clave, que llamamos "clave de enrutamiento". La clave de enrutamiento es una dirección virtual que el intercambio puede usar para decidir cómo enrutar el mensaje. Para el enrutamiento punto a punto, la clave de enrutamiento suele ser el nombre de una cola de mensajes. En casos más complejos, la clave de enrutamiento se puede combinar con el enrutamiento en los campos del encabezado del mensaje y/o su contenido (Vinoski, 2006).

2.7.6. Cola de Mensajes (Message Queu).

Las colas son el concepto central en AMQP. Cada mensaje siempre termina en una cola, incluso si se trata de una cola privada almacenada en una memoria que alimenta a un cliente directamente. Para extender la analogía postal, las colas son buzones de correo en el destino final o áreas de espera en la oficina de clasificación. Las colas pueden almacenar mensajes en la memoria o en el disco. Pueden buscar y reordenar mensajes, y pueden participar en transacciones. El administrador puede configurar los niveles de servicios que esperan de las colas con respecto a la latencia, la durabilidad, la disponibilidad, etc. (O'Hara, 2007)

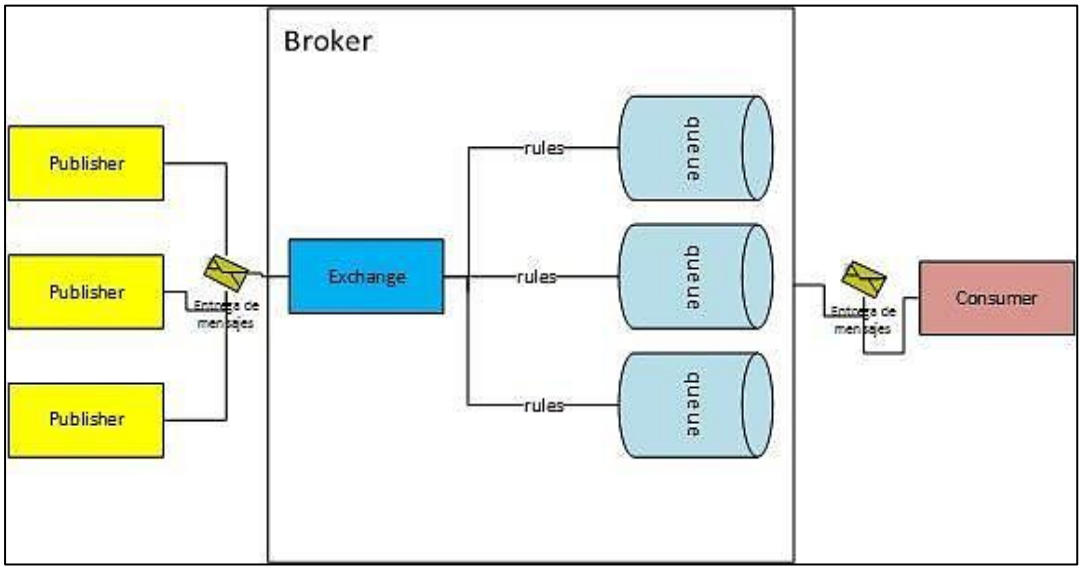


Figura 2.25 Otra alternativa de arquitectura del protocolo AMQP.

Fuente: (García González, 2017).

Los mensajes en AMQP es un componente importante de la comunicación. Una cola de mensajes almacena los mensajes en la memoria o en el disco, y los entrega en secuencia a una o más aplicaciones del consumidor. Las colas de mensajes son entidades de almacenamiento y distribución de mensajes.

Cada cola de mensajes es completamente independiente y es un objeto razonablemente inteligente. Tiene varias propiedades, tales como, privada o compartida, duradera o temporal, con nombre de cliente o nombre de servidor, etc.

Cuando una aplicación “cliente” crea una cola de mensaje, puede seleccionar algunas importantes propiedades (Vinoski, 2006):

- **Nombre:** Si no se especifica, el servidor elige un nombre y se lo proporciona al cliente. Generalmente, cuando las aplicaciones comparten una cola de mensajes, acuerdan un nombre de cola de mensajes, y cuando una aplicación necesita una cola de mensajes para sus propios fines, permite que el servidor proporcione un nombre.
- **Exclusivo:** Si se establece, la cola pertenece sólo a la conexión actual y se elimina cuando se cierra la conexión.
- **Duraderos:** Si se establece, la cola de mensajes permanece presente y activa cuando el servidor se reinicia. Es un problema perder mensajes transitorios si el servidor se reinicia.

Para seleccionar las propiedades que se anotan a continuación, podemos usar una cola de mensajes para implementar entidades de middleware como:

- **Una cola de almacenamiento y reenvío compartido:** Las colas de almacenamiento y envío suelen ser duraderas y se comparten entre varios consumidores.

- **Una cola de respuesta privada:** Contiene mensajes y los reenvía a un solo consumidor. Las colas de respuesta suelen ser temporales con nombre de servidor y privadas para un consumidor.
- **Una cola de suscripción privada:** Contiene los mensajes recopilados de varias fuentes "suscritas" y las reenvía a un solo consumidor.

Las colas de suscripción suelen ser temporales y son privadas para un consumidor. Las categorías antes mencionadas no se definen formalmente en AMQP, ya que son aplicaciones de cómo se pueden usar las colas de mensajes. Es común crear nuevas entidades, como colas de suscripción duraderas y compartidas.

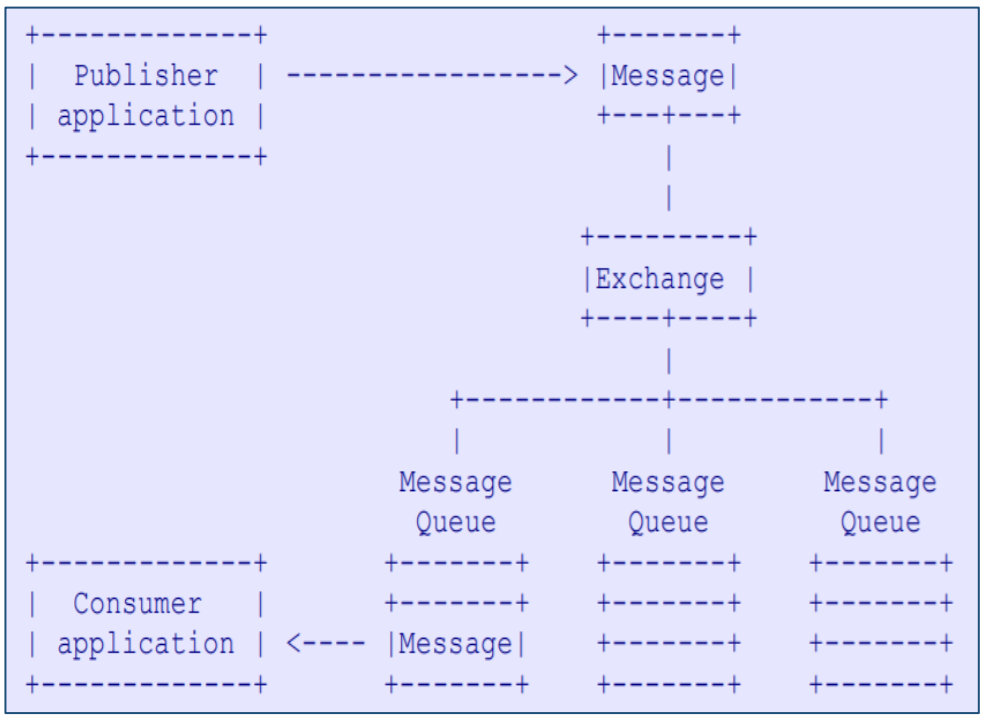


Figura 2.26 El flujo de mensajes a través del modelo AMQP.

Fuente: (Vinoski, 2006).

2.7.7. Consumidor.

El consumidor es la entidad que recibe o consume el mensaje, es decir, una aplicación cliente que solicita mensajes de una cola de mensajes. A continuación, se observa el modelo de arquitectura de AMQP, pero en un diagrama diferente (ver figura 2.27).

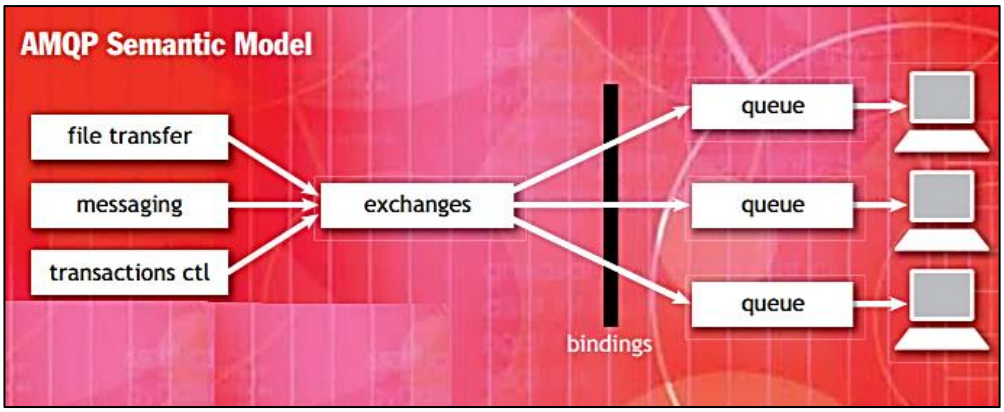


Figura 2.27 Modelo Semántico de AMQP.

Fuente: (O'Hara, 2007).

2.8. Internet de las cosas.

La mayoría de las conexiones a Internet en todo el mundo son dispositivos que utilizan directamente los seres humanos, como computadoras y teléfonos móviles. La principal forma de comunicación es humano/humano. Internet cambia y evoluciona continuamente, IoT proporciona conectividad para todos y todo. El IoT incorpora cierta inteligencia en objetos conectados a Internet para comunicarse, intercambiar información, tomar decisiones, solicitar acciones y proporcionar servicios sorprendentes.

El número de dispositivos conectados a Internet está aumentando a la velocidad rápida. Estos dispositivos incluyen computadoras personales, laptops, tabletas, teléfonos inteligentes y otros dispositivos portátiles dispositivos integrados. La mayoría de los dispositivos móviles incorporan diferentes sensores y actuadores que pueden detectar, realizar cálculos, tomar decisiones inteligentes y transmitir información recopilada útil a través de Internet (Khan, Khan, Zaheer, & Khan, 2012).

El IoT consta de objetos, sensores, infraestructura de comunicación, unidad computacional y de procesamiento que puede colocarse en la nube y tomar decisiones. Los objetos tienen ciertas características únicas y son únicamente identificable y accesible a Internet (Khan et al., 2012).

Las etiquetas RFID (radio frequency identification, en español, identificación por radiofrecuencia) son pequeños dispositivos, similares a una adhesivo o pegatina, que pueden ser adheridos a un producto, persona o animal para almacenar información relevante y dinámica.

Mediante radiofrecuencia, la información viaja a un ordenador o dispositivo móvil con acceso a Internet. Dicha información puede ser recibida por un usuario para su interpretación. También existe la posibilidad de que el extremo final sea otra máquina que interprete los datos y actúe según parámetros preestablecidos, como se puede observar en la figura 2.28 (Fundación de Innovación Bankinster, 2011).

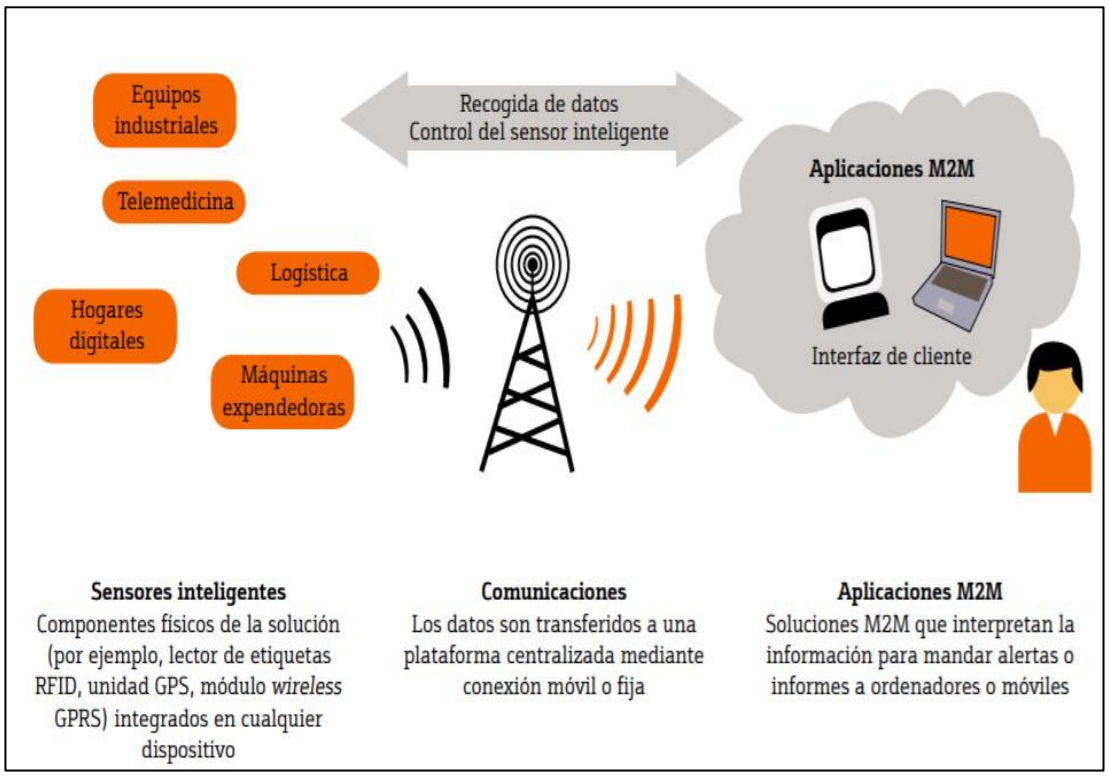


Figura 2.28 M2M: Cuando las cosas se vuelven inteligentes.
Fuente: (Fundación de Innovación Bankinster, 2011)

2.8.1. Concepto de Internet de las Cosas.

El Internet de las Cosas (IoT) consiste en que las cosas tengan conexión a Internet en cualquier momento y lugar. En un sentido más técnico, consiste en la integración de sensores y dispositivos en objetos cotidianos que quedan

conectados a Internet a través de redes fijas e inalámbricas (Fundación de Innovación Bankinster, 2011).

El hecho de que Internet esté presente al mismo tiempo en todas partes permite que la adopción masiva de esta tecnología sea más factible. Dado su tamaño y costo, los sensores son fácilmente integrables en hogares, entornos de trabajo y lugares públicos (Fundación de Innovación Bankinster, 2011).

De esta manera, cualquier objeto es susceptible de ser conectado y manifestarse en la Red. Además, el IoT implica que todo objeto puede ser una fuente de datos. Esto está empezando a transformar la forma de hacer negocios, la organización del sector público y el día a día de millones de personas (Fundación de Innovación Bankinster, 2011).

2.8.2. Componentes de IoT:

Se describen dos modos básicos de comunicación en Internet de las cosas: objeto-persona y objeto-objeto (Agudelo & Valbuena, 2016).

Objeto - Persona: Las comunicaciones de este tipo una serie tecnologías y aplicaciones en las cuales las personas interactúan con cosas y viceversa. Las más comunes son el acceso a distancia, control remoto y por monitoreo. También existen objetos que informan a las personas el estado de la temperatura y humedad alrededor de su recinto, detección de fuga de gas, etc. (Agudelo & Valbuena, 2016).

Objeto - Objeto: Abarca tecnologías y aplicaciones en donde objetos interactúan sin que ningún humano haya iniciado la interacción ni sea receptor o intermediario. Los objetos pueden controlar otros objetos, tomar medidas correctivas y realizar notificaciones a las personas según sea necesario (suelen denominarse aplicaciones M2M, Machine to Machine).

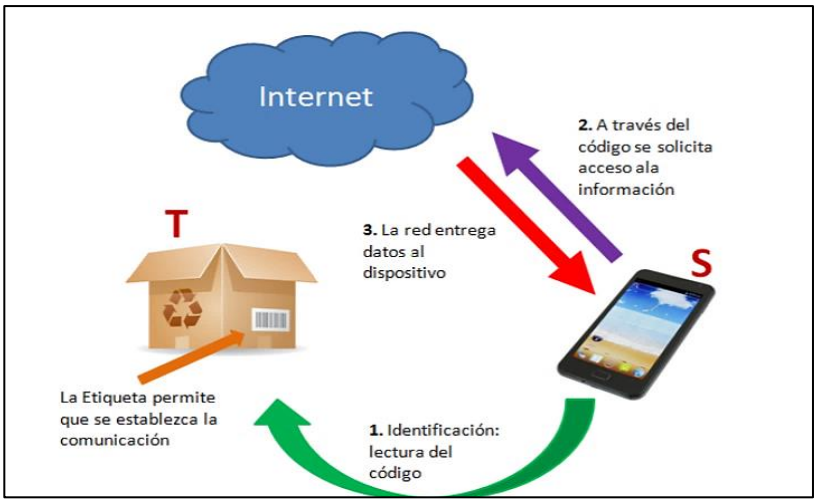


Figura 2.29 Interacción básica entre un objeto etiquetado (T) y un dispositivo inteligente (S) este es el tipo de esquema utilizado en la Internet de información de productos.

Fuente: (Agudelo & Valbuena, 2016)

En este contexto, un objeto o cosa se considera “Inteligente” si es capaz de entregar algún tipo de información a otra. siendo la interacción más elemental aquella entre un objeto T, cuyo recurso inteligente es una etiqueta que puede ser leída por la cosa S, quien a su vez es capaz de usar los datos contenidos en la etiqueta para obtener una información adicional de contexto, la cual se hace disponible a través de una red. Como se observa en la figura 2.29, S tiene una inteligencia “superior” a T ya que este es capaz de leer códigos e interactúa con una red (Agudelo & Valbuena, 2016).

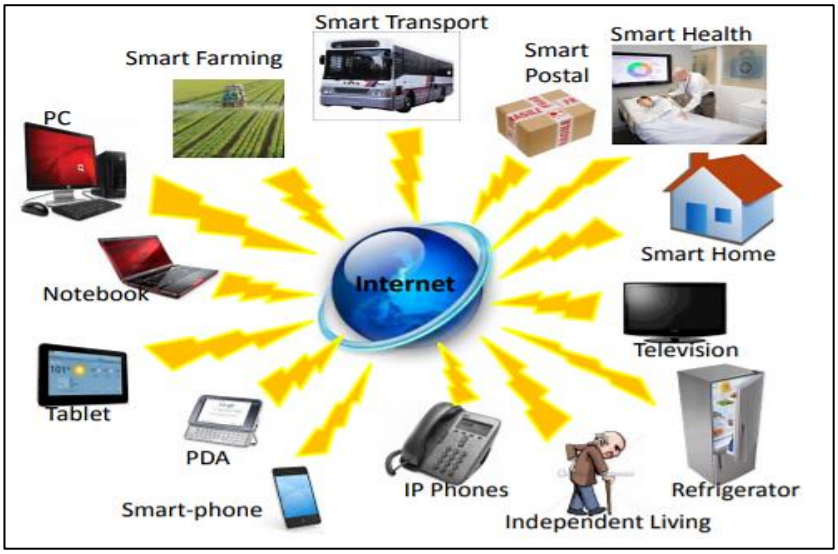


Figura 2.30 El escenario genérico de IoT

Fuente: (Khan et al., 2012).

2.8.3. Arquitectura de IoT

El IoT debe ser capaz de interconectar miles de millones o billones de objetos heterogéneos a través de Internet, por lo que hay una necesidad crítica de una arquitectura en capas flexibles. Hay algunos proyectos como la IoT, que tratan de diseñar una arquitectura común basada en el análisis de las necesidades de los investigadores y la industria (Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015). La información encontrada en internet da la posibilidad hoy en día, interconectar servicios y bienes entre dispositivos, objetos conectados en la red.

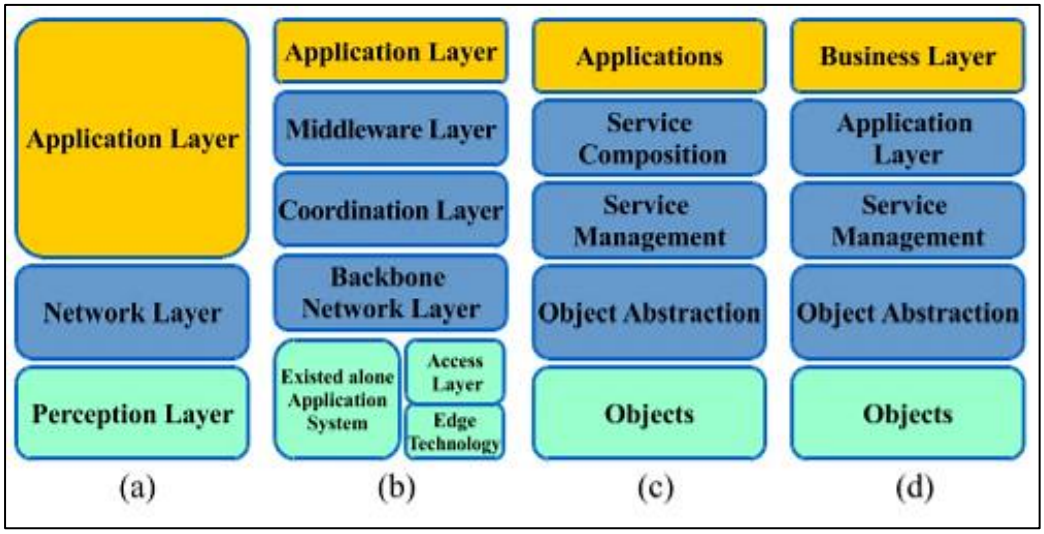


Figura 2.31 Arquitectura de IoT(a)Three-layer(b)Middle-ware based.(c)SOA based.(d)Five-layer.

Fuente: (Al-Fuqaha et al., 2015)

La figura 2.31, presenta algunas arquitecturas comunes, entre ellas se encuentra el modelo de 5 capas (no debe confundir con las capas TCP/IP). Otros autores consideran que la arquitectura de IoT, generalmente se dividida en cinco capas, como se muestra en la figura 2.32, esta propuesta es una mezcla entre las arquitecturas propuestas en la figura 2.31, esto es debido a que los modelos de tres capas no se ajustan a entornos reales del mundo de IoT, se debe tener en cuenta que las capas se ejecutan en dispositivos con recursos limitados, según los autores consultados, la arquitectura de cinco capas es el modelo más aplicable para aplicaciones IoT (Valiente Cristancho, 2017) .

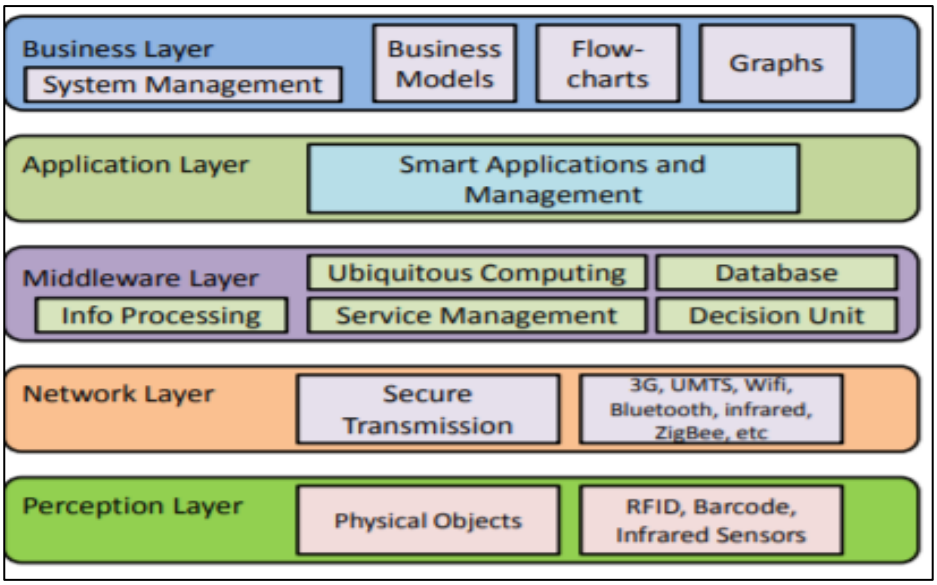


Figura 2.32 Arquitectura de IoT.

Fuente: (Khan et al., 2012).

A continuación, se describen brevemente las capas de la arquitectura de IoT de cinco capas:

Capa de Percepción: Se conoce como "Capa de Dispositivos u objetos", representa los sensores físicos del IoT, el objetivo de esta capa es recopilar y procesar información. Esta capa incluye sensores y actuadores para realizar diferentes funcionalidades tales como localización, temperatura, peso, movimiento, vibración, aceleración, humedad, etc. (Valiente Cristancho, 2017).

Capa de red: Se lo conoce también como “Capa de transmisión”. Esta capa se responsabiliza de conectar de forma segura la información de los dispositivos (sensores) y servidores para transmitir y procesar datos almacenados por los sensores. El medio de transmisión puede ser cableado o inalámbrico y la tecnología puede ser 3G, WiFi, Bluetooth, etc, dependiendo de los dispositivos sensores. Por lo tanto, la capa de red transfiere la información de la capa de percepción a la capa de Middleware. (Khan et al., 2012).

Capa de Gestión de Servicios o Middleware: Los dispositivos en IoT implementan diferentes tipos de servicios. Cada dispositivo se conecta y se comunica con sólo aquellos otros dispositivos que implementan el mismo tipo de servicio. Esta capa es responsable de la gestión de servicios y tiene un enlace a la base de datos. Recibe la información de la capa de red y la almacena en la base de datos. Realiza el procesamiento de la información y computación ubicua y toma la decisión automática basada en los resultado (Khan et al., 2012).

Capa de aplicación: Esta capa proporciona una gestión global de la aplicación basada en la información de objetos procesada en la capa de middleware. La importancia de esta capa para el IoT, es que tiene la capacidad de proporcionar servicios inteligentes de alta calidad para satisfacer las necesidades de los clientes. Las aplicaciones implementadas por IoT abarca numerosos mercados verticales como salud inteligente, agricultura inteligente, casa inteligente, ciudad inteligente, transporte inteligente, etc. (Khan et al., 2012).

Capa de Negocios: Esta capa es responsable de la gestión global del sistema IoT, incluyendo las aplicaciones y servicios. Construye modelos de negocio, gráficos, diagramas de flujo, etc, basados en los datos recibidos de la capa de aplicación, diseña, analiza, implementa, evalúa, monitorea y desarrolla elementos relacionados con el sistema IoT. El verdadero éxito de la tecnología IoT también depende de los buenos modelos de negocio, basados en el análisis de resultados; esta capa es la encargada de determinar las acciones futuras y estrategias de negocio. (Khan et al., 2012).

2.9. Retos del Internet de las Cosas (IoT).

Aunque el internet de las cosas puede ser visto fácilmente como una red de dispositivos conectados a internet, este no se limita a solo a esto. Tiene características propias, como la complejidad tecnología y el estilo de sus aplicaciones, dotando a estas últimas con funciones de procesamiento, trasmisión y decisión, que implican el uso conjunto de tecnologías que

envuelven complicadas redes de sensores, sistemas de comunicaciones en red, complejos sistemas de procesamiento de datos, entre otros.

2.9.1. Desarrollo del IoT según los desafíos

En el desarrollo del IoT, es posible encontrar desafíos que requieren especial atención, de acuerdo al siguiente detalle: (Valiente Cristancho, 2017):

- **Interoperabilidad de datos:** En la actualidad existe una gran cantidad de dispositivos que se comunican en lenguajes totalmente distintos, por lo que el principal reto en este caso está en lograr que todos estos dispositivos puedan comunicarse usando un lenguaje común y estandarizado (Valiente Cristancho, 2017).
- **Dispositivos de bajo consumo:** Con la intención de permitir que todos los dispositivos en un entorno puedan estar conectados es necesario que estos sean de bajo consumo y se comuniquen inalámbricamente, lo que supone un problema debido a que las redes actuales suponen un funcionamiento continuo de cada dispositivo (Valiente Cristancho, 2017).
- **Seguridad y privacidad:** Es importante que todos los dispositivos de la red, estén protegidos frente a un agente malicioso que pueda infectar o destruir la infraestructura global de comunicación (Valiente Cristancho, 2017).
- **Análisis de datos:** La cantidad de datos que pueden ser enviados sin saturar las redes y la capacidad de procesamiento de los centros de red es limitada, por lo que es necesario optimizar la transmisión de los dispositivos enlazados a la red (Valiente Cristancho, 2017).
- **Software:** Es necesario proporcionar un lenguaje, infraestructura y patrones de programación que incluyan a los desarrolladores de software que se ven envueltos dentro del IoT (Valiente Cristancho, 2017).

2.9.2. Aplicaciones del Internet de las Cosas.

IoT, supone un avance para estrechar el espacio existente entre el usuario y la máquina, al impulsar la comunicación entre máquinas y sistemas

que cooperan entre sí, realimentando y evolucionando proactivamente los procesos y permitiendo un control sobre nuestro entorno cada vez más efectivo (García González, 2017).

Todo se materializa en una nueva ola tecnológica, con nuevos dispositivos servicios y aplicaciones, que marcará el crecimiento socioeconómico durante los próximos años. IoT va a cambiar nuestro día a día, mediante dispositivos que se conectan entre sí y a la red. La información sobre nuestro entorno estará disponible de forma accesible y en tiempo real, de forma que pueda enriquecer nuestra experiencia y facilitar la actividad cotidiana, mediante aplicaciones y servicios que pueden ir desde la optimización del uso energético hasta aplicaciones de seguridad, salud, ciudades inteligentes, entre otros. Mediante dispositivos conectados, las ciudades crearán ecosistemas que podrán ser empleados para el desarrollo de aplicaciones que mejoren los procesos típicos de la ciudad y permitirán ofrecer a los ciudadanos información en tiempo real (García González, 2017).

2.9.3. Campos de aplicación del IoT.

Las posibilidades de aplicación de IoT son muy variadas, abarcando diversos campos de actividad. IoT ha evolucionado tanto, que prácticamente abarca cualquier campo de la vida que podamos imaginar. A continuación, se describen los que mayor protagonismo y evolución están alcanzando (García González, 2017).

- **Smart Home (Casas Inteligentes):** Uno de los campos de aplicación en el que se han desarrollado más proyectos es el de las Smart Home o casas inteligentes. Sensores de temperatura, actuadores para apertura y cierre de persianas, electrodomésticos inteligentes que avisan de la caducidad de los alimentos y son capaces de realizar pedidos cuando alguna existencia alcanza un determinado nivel (García González, 2017).

Figura 2.33 Smart Home

Fuente: (SORTILEGIO LTDA, 2019)

En un Smart Home, los sensores son los encargados de dar información de la casa. Cada habitación necesita aportar una determinada información que consiste en: temperatura, presencia, estado de ventanas, persianas y puertas y cualquier otra necesaria (García González, 2017).

En este tipo de entornos es donde el uso de distintas tecnologías de comunicación, de corto alcance que sirve para recoger datos de los dispositivos y de largo alcance para el transporte de estos datos a la nube, donde el desarrollo de estándares debe de permitir que diferentes tecnologías puedan interactuar entre sí (García González, 2017).

- **Smart City (Ciudad Inteligente):** Las Smart Citys, o ciudades inteligentes, son aquellas ciudades que aplican las tecnologías de la información y de la comunicación (TIC) para proveerlas de infraestructuras que garanticen (García González, 2017):

- El desarrollo sostenible.
- Aumento de la calidad de vida.
- La mayor eficacia de los recursos disponibles.
- Fomentar la participación ciudadana activa.

Figura 2.34 Smart City

Fuente: (García González, 2017)

Smart City es otro campo en el que IoT está muy presente, a través de dispositivos que recopilan la mayor cantidad de datos para facilitar la vida al ciudadano (García González, 2017).

Las aplicaciones pueden ir desde el Smart Parking, indicando donde existe aparcamiento libre, o indicar que una plaza de minusválido está siendo ocupada por un vehículo no autorizado, el Smart Traffic para informar, tiempo real, del tráfico y proponer rutas alternativas, gestión eficiente del alumbrado, etc. (García González, 2017).

Estos son solo unos ejemplo de lo que se pretende implementar con esta tecnología, su futuro es muy prometedor sobre todo viendo como muchas comunidades y empresas están apostando claramente por convertir las ciudades en un ecosistema más inteligente y conectado (García González, 2017).

- **Wearables (Artículos vestibles):** Un tipo de dispositivos que están tomando gran popularidad son los wearables. Se trata, entre otros, de dispositivos que nos monitorizan durante todo el día determinados

parámetros de nuestro cuerpo y entorno más cercano, mostrando esa información en nuestro Smartphone (García González, 2017).

Figura 2.35 Wearable Techonology
Fuente: (García González, 2017)

Todos estos dispositivos se agrupan en la denominada tecnología vestible (wearable technology), tecnología corporal, ropa tecnológica, ropa inteligente, tecnología llevable o complementos inteligentes. En la actualidad los dispositivos vestibles pueden categorizarse de la siguiente manera (García González, 2017):

- Salud.
- Deporte y bienestar.
- Entretenimiento.
- Industrial.
- Militar.

2.10. Herramientas de distribución libre para análisis de protocolos.

Hoy en día el manejo de software y herramientas informáticas representa un pilar fundamental en el sector industrial y en otros sectores, sin embargo, hay herramientas cuyo costo, las vuelve prácticamente inaccesibles.

Por lo que se ha desarrollado a lo largo del tiempo, el manejo de software libre para eliminar los problemas que representa adquirir la licencia para un sistema operativo o herramienta de software. El Software representa el esfuerzo de varios desarrolladores por crear y producir herramientas técnicas y de acceso gratuito, la cual están orientadas a solucionar problemas de ciencia e ingeniería o aplicaciones de investigación.

Se usa el término software libre cuando se refiere a la libertad que tiene un usuario para modificar, copiar y distribuir un software sin que ninguna compañía o individual pueda emprender acciones legales contra él (González Piñero, 2004).

Para que un software pueda ser considerado libre tiene que cumplir con reglas establecidas que aseguren que sigue la filosofía del software libre y se les llama las cuatro libertades, y son (González Piñero, 2004):

- Ejecutar el programa para cualquier propósito.
- Estudiar el funcionamiento del programa y adaptarlo a sus necesidades.
- Redistribuir copias.
- Mejorar el programa, y poner sus mejoras a disposición del público, para beneficio de todo tipo de sector.

De esta manera, un usuario es completamente libre de modificar el código fuente del software para mejorarlo o adaptarlo a las necesidades que tenga, sin tener que pagar a nadie por ello. De la misma manera, puede distribuir copias del software libremente, no sólo del software tal y como lo consiguió, sino que puede redistribuirlo con las modificaciones que haya llevado a cabo en él. Pero si distribuye un software modificado se tiene que seguir cumpliendo las cuatro libertades y proporcionar la fuente. Esto está protegido por el copyleft (González Piñero, 2004).

El copyleft (juego de palabras proveniente de copyright) es una regla que obliga a que todas las redistribuciones de un software cumplan las cuatro

libertades, aunque este alguien haya modificado en alguna manera el software (González Piñero, 2004).

Es decir, protege los derechos del autor y protege las libertades de todos los usuarios para ejecutar, modificar y distribuir ese software sin restricciones (González Piñero, 2004).

Open Source (Código abierto)

En los años 90 surge un grupo llamado OSI (Open Source Initiative), el cual llama open source (código abierto) al software libre. De tal modo que el software libre es considerado un sinónimo de código abierto y por lo tanto ambas trabajan con las cuatro libertades que se deben cumplir.

Una de las primeras implementaciones proviene del mundo de código abierto creado por la Fundación Apache con el proyecto Apache Qpid que proporciona una implementación del protocolo AMQP en Java, C++ y otros lenguajes de programación. Del mismo modo, existe la biblioteca AMQP.Net Lite que es una implementación de código abierto .Net y C # de Microsoft.

Active MQ y RabbitMQ son uno de las más populares y potentes servidores de patrones de integración y mensajería de código abierto que admite el protocolo AMQP. Para otros protocolos se puede usar las herramientas Node Red, Grafana entre otras.

CAPÍTULO 3: RESULTADOS DEL ESTUDIO DEL PROTOCOLO AMQP UTILIZANDO RABBITMQ.

De lo estudiado y analizado en los capítulos anteriores, esta parte del documento se demuestra de factibilidad en su bajo costo y alta confiabilidad basado en los conceptos principales del protocolo AMQP del Internet de las Cosas.

3.1. RabbtiMQ, herramienta de distribución libre.

En este presente trabajo de titulación, se escogió el bróker de mensajería RabbitMQ, la cual es una herramienta de software libre de fácil uso que soporta diversas plataformas de desarrollo, tales como, Ruby, Node.Js, Java, Python, Erlang entre otros, y colecciona o recopila diversos servidores en un bróker lógico. También es una aplicación dirigida a la distribución de mensajes en aplicaciones.

Se usa RabbitMQ como el middleware de mensajería o un servicio del mismo, de tal modo que esta herramienta implementa el protocolo estándar llamado Advance Message Queuing Procolo (AMQP), concentrado en el comportamiento del publicador o servidor que brinda los mensajes, como en el comportamiento del cliente de la mensajería (PacktLib, 2013).

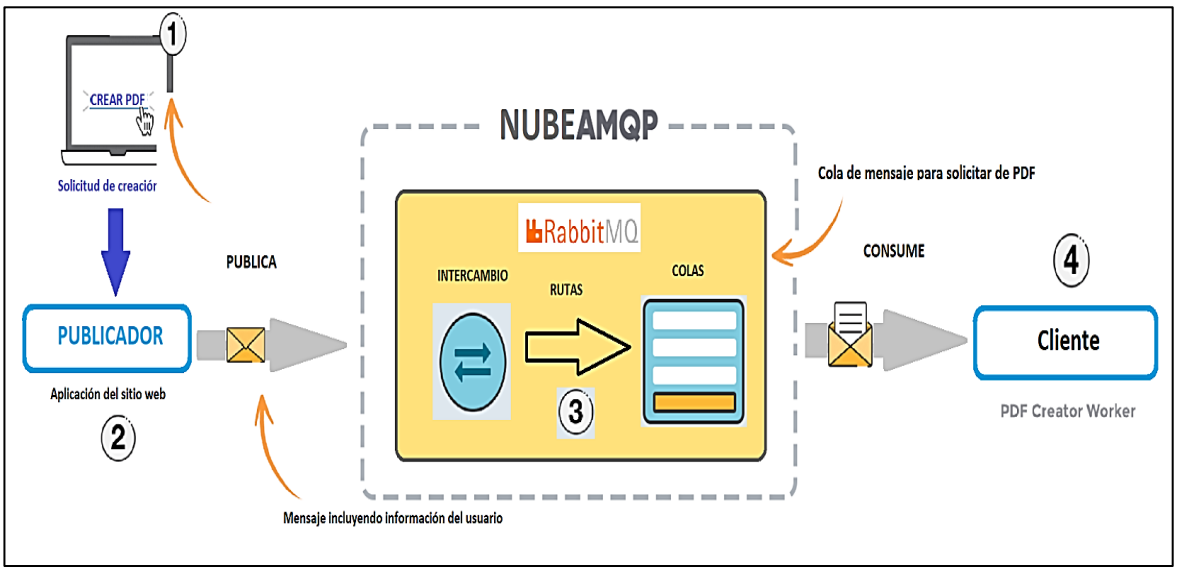


Figura 3.36 Nube AMQP con el bróker RabbitMQ

Elaborado por: Autor.

Según la fuente, el protocolo avanzado de mensajes en cola (AMQP) se ha desarrollado debido a la necesidad de interoperabilidad entre las diferentes soluciones de mensajería, que fueron desarrolladas hace unos años por muchos proveedores diferentes, como IBM MQ-Series, TIBCO o Microsoft MSMQ (PacktLib, 2013). RabbitMQ es una implementación de bróker AMQP gratuita y completa, incluye:

- El bróker, es decir, el servicio que manejará los mensajes que las aplicaciones enviarán y recibirán.
- Es muy típico que los consumidores utilicen diferentes tecnologías y programación, como Java, Python y Ruby lo que muestra la interoperabilidad con AMQP.

RabbitMQ se puede instalar en Windows, Linux, sistemas MacOS y también existe la nube AMQP que ofrecen diferentes soportes técnicos. Para sistemas pequeños, se puede utilizar hardware de gama baja como Raspberry Pi (PacktLib, 2013). Para descargar e instalar RabbitMQ, se puede visitar la página que se menciona a continuación: https://www.rabbitmq.com/download.html.

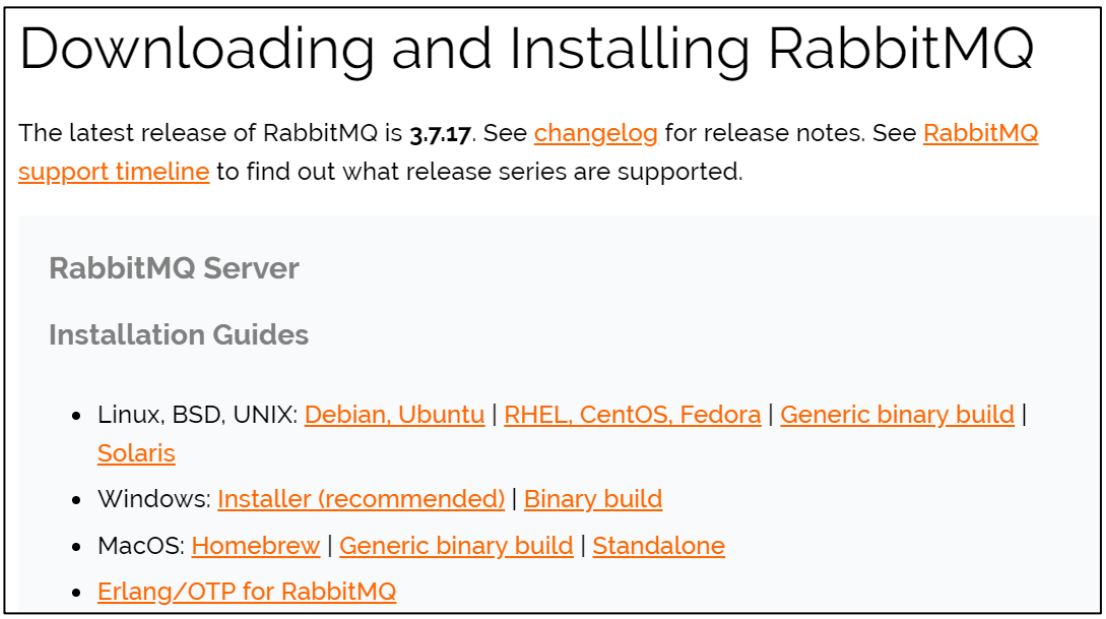


Figura 3.37 Descarga e instalación RabbitMQ gratis.

Fuente: (RabbitMQ, 2017)

3.2. Conexión al Bróker de mensajería (RabbitMQ).

En esta parte del documento, se utiliza Java, ya que este lenguaje se usa en el desarrollo de software empresarial, integración y distribución. RabbitMQ es ideal en este entorno.

Con el fin de ejecutar la siguiente aplicación, se sigue los siguientes pasos (PacktLib, 2013):

- Instalar Java JDK 1.6+.
- Instalar la biblioteca cliente Java RabbitMQ.
- Configurar correctamente **CLASSPATH** y su entorno de desarrollo preferido, es decir las palabras de código en el texto, nombres de carpeta, nombres de archivo, extensiones de archivo, nombres de ruta. Entorno como: Eclipse, NetBeans, y así sucesivamente.
- Instalar el servidor RabbitMQ en una máquina (esto puede ser la misma máquina local).

Cada aplicación que usa AMQP necesita establecer una conexión con el bróker del mismo. Por defecto, RabbitMQ funciona a través de TCP como protocolo de transporte confiable en el puerto 5672, es decir, el puerto asignado por IANA. Los puertos asignados, no deben utilizarse sin un registro de IANA, Internet Assigned Numbers Authority, es la entidad que controla la asignación global de direcciones IP, servidores con nombres de dominio DNS y otros recursos relacionados a los protocolos de Internet (PacktLib, 2013).

Se refiere a la conexión y al canal como los resultados de las operaciones presentadas a continuación. Se necesita configurar el entorno de desarrollo de Java para cualquier entrada o salida de línea de comandos se escribe (PacktLib, 2013):

```
java-cp./binrmqexample.Publish[RabbitMQ-host]
```

Para hacer que un cliente Java se conecte al bróker RabbitMQ, se deberá realizar los siguientes pasos (PacktLib, 2013):

1. Importar las categorías necesarias de la biblioteca del cliente Java RabbitMQ en espacio de nombres (namespace) del programa (PacktLib, 2013):

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import
com.rabbitmq.client.ConnectionFactory;
```

2. Crear una instancia del cliente (`ConnectionFactory`) (PacktLib, 2013):

```
ConnectionFactory factory = new ConnectionFactory();
```

3. Configurar la opción (`ConnectionFactory`) (PacktLib, 2013):

```
factory.setHost(rabbitMQhostname);
```

4. Conectarse al bróker RabbitMQ (PacktLib, 2013):

```
Connection connection = factory.newConnection();
```

5. Crear un canal a partir de la conexión recién creada (PacktLib, 2013):

```
Channel channel = connection.createChannel();
```

6. Cuando termine la conexión con RabbitMQ, liberar el canal y la conexión (PacktLib, 2013):

```
channel.close();
connection.close();
```

El uso de la API de cliente Java, la aplicación debe crear una instancia de (`ConnectionFactory`) y establecer el host donde RabbitMQ debe estar en ejecución con el método `setHost()` (PacktLib, 2013).

Después de que las importaciones de Java (paso 1), se ha instanciado el objeto `factory` (paso 2). En esta aplicación se establece, el nombre del host que se ha elegido para llegar a la línea de comandos (paso 3). En el paso 4 se ha establecido la conexión TCP con el bróker RabbitMQ. En este procedimiento se ha utilizado los parámetros de conexión del usuario: `guest`, contraseña: `guest` y host virtual: /. Sin embargo, todavía no está para comunicarse con el bróker; se tiene que establecer un canal de comunicación

(paso 5). Este es un concepto avanzado de AMQP; es posible hacer que muchas diferentes sesiones de mensajería utilicen la misma conexión lógica (PacktLib, 2013).

Hosts virtuales son contenedores administrativos que permiten configurar muchos hosts de bróker independientes dentro de una instancia única RabbitMQ, para que muchas diferentes aplicaciones compartan el mismo servidor RabbitMQ. Cada máquina virtual puede ser configurado con su conjunto independiente de permisos, intercambios, y las colas y trabajará en un entorno lógicamente separados (PacktLib, 2013).

3.2. Producción de mensajes

En esta parte, se aprende cómo enviar un mensaje a una cola de AMQP. Se estudia los bloques de la mensajería AMQP: mensajes, las colas y los intercambios. Puede encontrar la fuente en: `Chapter01/Recipe02/src/rmqexample.`

Se necesita configurar el entorno de desarrollo de Java cómo se indicó anteriormente. Después de conectarse con el bróker, como se mencionó anteriormente, se puede empezar a enviar mensajes realizando los siguientes pasos (PacktLib, 2013):

1. Declarar la cola, llamando al método `queueDeclare()` (PacktLib, 2013):

```
com.rabbitmq.client.Channel:
String myQueue = "myFirstQueue";
channel.queueDeclare(myQueue, true, false, false, null);
```

2. Enviar el primer mensaje al bróker RabbitMQ (PacktLib, 2013):

```
String message = "My message to myFirstQueue";
channel.basicPublish("",myQueue,
null,message.getBytes());
```

3. Enviar el segundo mensaje con diferentes opciones (PacktLib, 2013):

```
channel.basicPublish("",myQueue,MessageProperties.PERSISTENT_TEXT_PLAIN,message.getBytes());
```

Los nombres de las colas son sensibles a mayúsculas: `MYFIRSTQUEUE` es diferente de `myFirstQueue`. En esta aplicación básica que se ha enviado un mensaje a RabbitMQ. Después de establecer el canal de comunicación, el paso 1, es asegurar que existe la cola de destino. Esto se consigue declarando la cola (paso 1) llamando `queueDeclare()`.Todas las operaciones que necesitan la interacción con el bróker se llevan a cabo a través de canales. Respecto al método de `queueDeclare()`, la plantilla (figura 3.38) se puede encontrar en la documentación de Java con referencia de cliente ubicado en (PacktLib, 2013):
http://www.rabbitmq.com/releases/RabbitMQ-java-client/current/javadoc/.

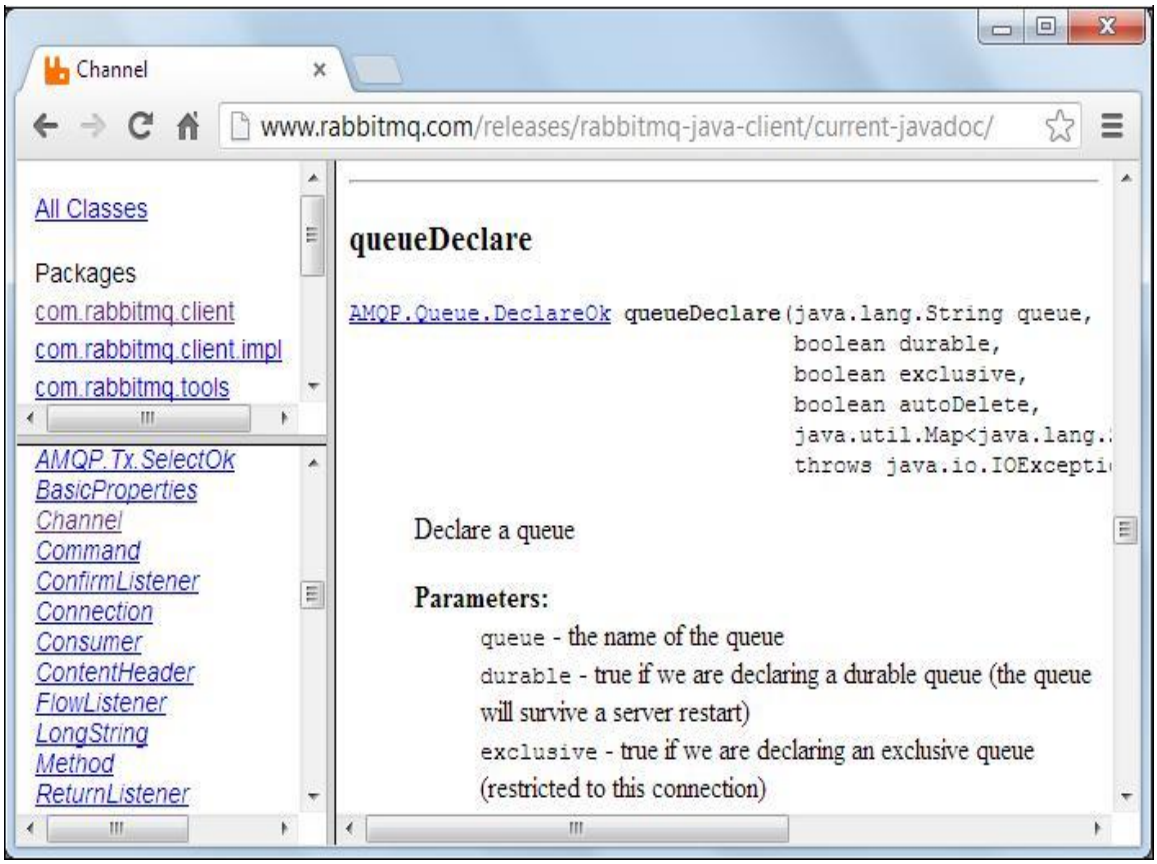


Figura 3.38 queueDeclare, documentación de Java.
Fuente: (PacktLib, 2013)

En particular, se ha utilizado el siguiente método (PacktLib, 2013):

```
AMQP.Queue.DeclareOk queueDeclare(java.lang.String queue, Boolean
  durable, boolean exclusive, booleanautoDelete,
            java.util.Map<java.lang.String,java.lang.Object> arguments)
                  throws java.io.IOException
```

Resumen de la terminología empleada en AMQP (PacktLib, 2013):

- **Cola:** Esto es sólo el nombre de la cola donde se va a almacenar los mensajes.
- **Durable:** Declara si la cola sobrevivirá al reinicio del servidor. Tener en cuenta que se requiere para una cola, ser declarado como duradero si se desea que los mensajes sobrevivan a un reinicio del servidor.
- **Exclusivo:** Declara si la cola se limita sólo a esta conexión.
- **AutoDelete:** Declara si la cola se borrará automáticamente por el bróker RabbitMQ, ya que no está en uso.
- **Argumentos:** Son parámetros opcionales para la construcción de colas.

En el paso 2 se ha enviado un mensaje al bróker RabbitMQ. El cuerpo del mensaje no será abierto por RabbitMQ. Los mensajes son entidades no visibles para el bróker AMQP. A menudo se usa JSON, XML entre otros estándares, pero todas son alternativas válidas. Lo importante es que las aplicaciones cliente deben saber cómo interpretar los datos. Luego, se examina el comando `basicPublish()`, lo cual es el método del canal de la interfaz para el "overload" (PacktLib, 2013):

```
void basicPublish(java.lang.String exchange,
java.lang.String routingKey, AMQP.BasicProperties props, byte[]
body) throws java.io.IOException
```

En este aplicativo, el intercambio "argumento" se ha establecido en la cadena o string vacío "", es decir, el intercambio predeterminado, y el argumento `routingKey` para el nombre de la cola. En este caso, el mensaje se envía directamente a la cola especificada como `routingKey`. El `body` del argumento se establece en `byte` en nuestra cadena de comandos, es decir, es sólo el mensaje que enviamos. Los `props` del argumento se establecen en `null`, éstas son las propiedades del mensaje (PacktLib, 2013).

Según el paso 3, se ha enviado un mensaje idéntico, pero con `props` establecidos en (PacktLib, 2013).

`MessageProperties.PERSISTENT_TEXT_PLAIN;`

De esta manera se solicita a RabbitMQ marcar este mensaje como un mensaje persistente. Ambos mensajes han sido enviados al bróker RabbitMQ, en cola en `myFirstQueue`. Los mensajes permanecerán almacenados hasta que lo reciba un cliente (normalmente, un cliente diferente). Si la cola se ha declarado con el `durable` indicador establecido a `true` y el mensaje se ha marcado como persistente, es almacenada en la memoria por el bróker. En este último caso, los mensajes almacenados temporalmente no sobrevivirán a un reinicio RabbitMQ, pero la entrega de mensajes y la recuperación será mucho más rápido (PacktLib, 2013).

3.3. Comprobación del estado RabbitMQ

Con el fin de comprobar el estado RabbitMQ, se puede utilizar la herramienta de control de línea de comandos `rabbitmqctl`. Debería estar en el `PATH` en la configuración de Linux. En Windows se puede encontrar ejecutando el comando RabbitMQ navegando de la siguiente manera: **Menu de inicio | Todos los programas | RabbitMQ servidor | Preguntar RabbitMQ Comando (sbin dir).** Se puede ejecutar `rabbitmqctl.bat` de esta línea de comandos. Comprobar el estado de la cola con el comando `rabbitmqclt list_queues`. En la siguiente figura 3.39, se lleva a cabo justo antes y después de llevar a cabo la aplicación. (PacktLib, 2013).

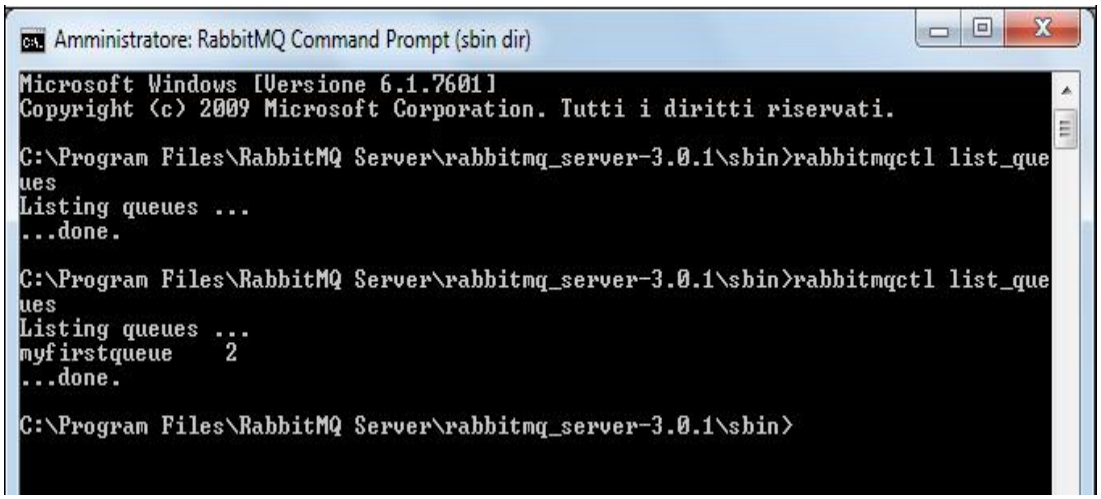


Figura 3.39 Cómo comprobar el estado de RabbitMQ.

Fuente: (PacktLib, 2013)

Se puede observar **myfirstqueue** cola (ver figura 3.39), seguido del número de 2, lo cual es el número de los mensajes almacenados temporalmente en la cola (PacktLib, 2013).

En caso de existir problemas en el servidor principal y se reinicie la máquina, por consecuencia se reiniciará también RabbitMQ, el reinicio de RabbitMQ para que sea de forma exitosa, dependerá del sistema operativo utilizado (PacktLib, 2013):

- En Linux, RedHat, CentOS, Fedora, Raspbian, y así sucesivamente (PacktLib, 2013):

  ```
  service rabbitmq-server restart
  ```

- En Linux, Ubuntu, Debian, y así sucesivamente (PacktLib, 2013):

  ```
  /etc/init.d/rabbitmq restart
  ```

- En Windows (PacktLib, 2013):

  ```
  sc stop rabbitmq / sc start rabbitmq
  ```

3.4. El consumo del mensaje.

En esta parte del documento, se finaliza el bucle. Se estudió como enviar mensajes a RabbitMQ a cualquier bróker AMQP y a continuación se aprenderá a recuperarlos. Puede encontrar el código de origen en (PacktLib, 2013):

```
Chapter01/Recipe03/src/rmqexample/ nonblocking.
```

Y también se necesita configurar el entorno de desarrollo de Java, como se indica al principio. Con el fin de consumir los mensajes enviados, realice los siguientes pasos (PacktLib, 2013):

1. Declarar la cola donde se desea consumir los mensajes de (PacktLib, 2013):

   ```
   StringmyQueue="myFirstQueue";
   channel.queueDeclare(myQueue, true, false, false, null);
   ```

2. Definir una clase de consumidores especializada en `DefaultConsumer` (PacktLib, 2013):

```
public class ActualConsumer extends DefaultConsumer
{ public ActualConsumer(Channel channel)
  {
super(channel);
}
@Override
public void handleDelivery(
String consumerTag, Envelope
envelope,    BasicProperties
properties,
byte[] body) throws java.io.IOException { String message = new
String(body); System.out.println("Received: " + message); }}
```

3. Crear un objeto `consumer`, que es una instancia con destino al canal (PacktLib, 2013):

```
ActualConsumer consumer = new ActualConsumer(channel);
```

4. Empezar a consumir mensajes (PacktLib, 2013):

```
String consumerTag = channel.basicConsume(myQueue, true, consumer);
```

5. Una vez hecho esto, detener el consumidor (PacktLib, 2013):

```
channel.basicCancel(consumerTag);
```

Después de haber establecido la conexión y el canal para el bróker AMQP, se debe garantizar que la cola de la que se va a consumir los mensajes existe (paso 1) (PacktLib, 2013).

De hecho, es posible que se inicie el consumidor antes de que cualquier productor haya enviado un mensaje a la cola y la cola en sí en realidad puede no existir en absoluto. Para evitar el fracaso de las operaciones posteriores en la cola, lo que se necesita para declarar la cola está definido en el paso 2 (PacktLib, 2013).

En el paso 2 se ha definido el consumidor que anula el manejo de las entregas de los mensajes, `handleDelivery()`, y está instanciado en el

paso 3. En el cliente Java, las devoluciones de llamada de consumo se definen por la interfaz `com.rabbitmq.client.Consumer`. (PacktLib, 2013).

En el paso 3, llamando `channel.basicConsume()`, se empieza a consumir los mensajes. Ahora que se ha activado un consumidor para `myQueue`, la biblioteca cliente Java empieza a recibir mensajes de esa cola en el bróker de RabbitMQ, e invoca `handleDelivery ()` para cada uno. Sólo después de que pulse **ENTER**, la ejecución continúa en la etapa 5, cancelando el consumidor (PacktLib, 2013).

3.5. Configuración de colas altamente disponibles.

Por fortuna, RabbitMQ admite colas altamente disponibles (básicamente respondiendo a las colas a través de nodos en un clúster) para garantizar que no necesite ser bloqueado por un solo punto de falla en su infraestructura de mensajería y aún puede ser rentable y escalable. Se configura un intercambio y una cola de alta disponibilidad para el envío de mensajes y asegurar de que se respondan o repliquen en todos los nodos. Para esta aplicación se usó Pyton y Pika. Se instancia en este prueba (Grzeszczak, 2012):

`pika.adapter import BlockingConnection.`

```
#!/usr/bin/env python

from pika.adapters import BlockingConnection
from pika import BasicProperties

connection = BlockingConnection()

channel = connection.channel()

client_params = {"x-ha-policy": "all"}

exchange_name = 'public'
queue_name = 'test_queue'
routing_key = 'test_routing_key'

channel.exchange_declare(exchange=exchange_name, type='topic')

channel.queue_declare(queue=queue_name, durable=True, arguments=client_
params )

channel.queue_bind(exchange=exchange_name, queue=queue_name, routing_ke
y=routing_key)

connection.close()
```

Figura 3.40 Configurar colas altamente disponibles.

Fuente: (Grzeszczak, 2012).

Analizando la figura 3.40, inicialmente se declara el intercambio. Se observa que el método: `queue_declare`, tiene el siguiente argumento: `arguments=client_params. "x-ha-policy": "all"` informa a **Rabbitmq** que esta cola debe estar altamente disponible y se replique en los nodos agrupados. También se crea un enlace, y luego se publica mensajes y rabbitmq se encargará de toda la replicación a través de los nodos del clúster (Grzeszczak, 2012). A continuación, se muestra un caso de uso realizado por Pete Metcalfe con el bróker de mensajería RabbitMQ.

3.6. Aplicación utilizando RabbitMQ para el Internet de las Cosas.

Conectando varios protocolos y servidores en aplicaciones de IoT (Internet de las Cosas), se realiza un caso de uso con el servidor RabbitMQ y se analiza algunas diferencias entre los protocolos de mensajería MQTT y AMQP. Finalmente, una aplicación del protocolo MQTT usando arduino, y se presentará elementos de MQTT y AMQP en un panel Node-Red (Metcalfe, 2018).

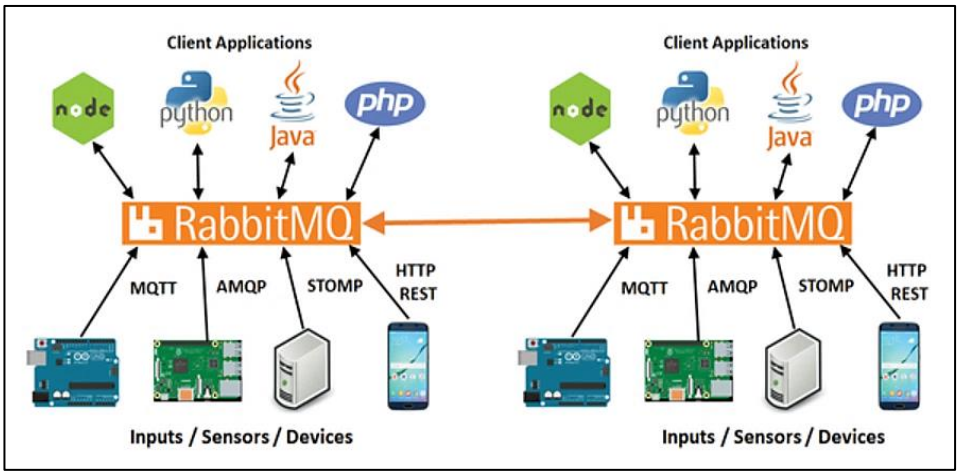


Figura 3.41 Descripción general de RabbitMQ.

Fuente: (Metcalfe, 2018)

Ingresar el siguiente comando para instalar y ejecutar RabbitMQ en un sistema Ubuntu (Metcalfe, 2018):

```
sudo apt-get update

sudo apt-get install rabbitmq-server

sudo service rabbitmq-server start
```

El próximo paso es agregar “puglins” como se ve a continuación, para esto, se carga los “puglins” MQTT y los de administración web (Metcalfe, 2018):

```
sudo rabbitmq-plugins enable rabbitmq_mqtt

sudo rabbitmq-plugins enable rabbitmq-management
```

`Rabbitmqctl` es la herramienta de línea de comandos que permite configurar y examinar el servidor RabbitMQ (Metcalfe, 2018). Se debe ingresar los siguientes comandos para añadir un usuario llamado `admin1`, con una contraseña `admin1`, cual debe tener derechos de configuración, lectura y escritura, para acceder como usuario de administración y administrado (Metcalfe, 2018):

```
sudo rabbitmqctl add_user admin1 admin1

sudo rabbitmqctl set_permissions -p / admin1 ".*" ".*" ".*"

sudo rabbitmqctl set_user_tags admin1 management administrator
```

Una vez definido el usuario administrativo, es posible acceder al “puglin” de administración web RabbitMQ en: http://ip_address:15672.

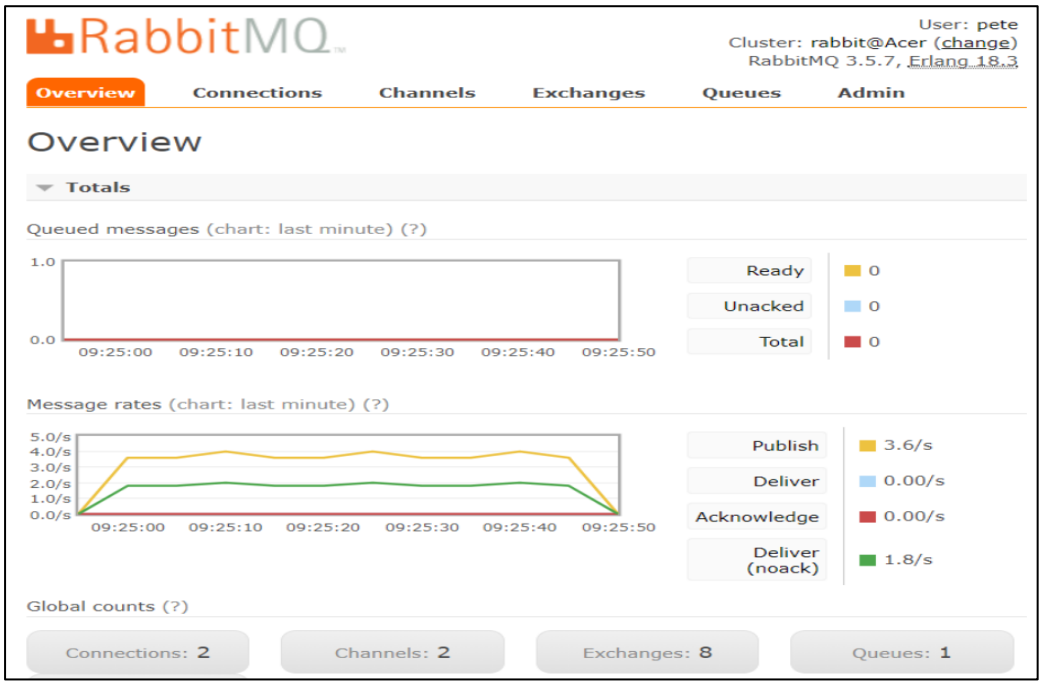


Figura 3.42 Administración Web RabbitMQ.

Fuente: (Metcalfe, 2018)

La herramienta de administración web RabbitMQ ofrece una visión general de la carga actual del sistema, intercambios, conexiones y las colas, ya antes descritas anteriormente (Metcalfe, 2018).

RabbitMQ, la herramienta de administración web, es ideal para mínimos cambios manuales, pero si busca realizar una mayor cantidad de cambios o agregar comandos, se puede utilizar `rabbitmqadmin`, la cual es una herramienta de administración de líneas de comandos. Esta herramienta es instalada de la siguiente manera (Metcalfe, 2018):

```
# Get the cli and make it available to use.

wget http://127.0.0.1:15672/cli/rabbitmqadmin

sudo chmod +x rabbitmqadmin
```

3.6.1. Uso de los Protocolos MQTT y AMQP.

La diferencia entre el protocolo MQTT y AMQP es que, MQTT es un protocolo ligero de mensajería basado en publicación/suscripción que trabaja correctamente, sin complicaciones, con hardware de baja calidad y con un ancho de banda limitado siendo MQTT una excelente opción para las aplicaciones de tipo arduino donde solamente es necesario filtrarse por ciertos datos del sensor. Pero, AMQP es un protocolo avanzado que incluye orientación de mensajes, colas, enrutamiento, confiabilidad y seguridad, lo cual este protocolo es diferente de los demás por sus características antes mencionadas (Metcalfe, 2018).

Actualmente no hay bibliotecas principales de AMQP para Arduino, pero hay una gran variedad de numerosas opciones de programación como los sistemas Linux, Windows, MacOs y Raspberry Pi (Metcalfe, 2018).

Una breve aplicación del protocolo AMQP del Internet de las cosas es el envío de fallas de sensor, alarmas para mantenimiento y colas de alarmas (Metcalfe, 2018).

Diferencia entre las colas (Queu) de los protocolos MQTT y AMQP.

La mayor diferencia es que, las colas de MQTT solo están diseñadas para mostrar el último mensaje disponible (ver figura 3.44), a la vez se puede fusionar con AMQP, donde AMQP está diseñada para el almacenamiento de múltiples mensajes en una cola. Quiere decir que, el mensaje "MQTT publicado" incluye un cuerpo de mensaje, una bandera de retención (retain flag) y un valor de calidad de servicio (QoS). Es posible la publicación de un mensaje AMQP con algunas propiedades adicionales, tales como: registro de tiempo, tipo del mensaje y caducidad de la información. En estos mensajes se pueden agregar valores de encabezado de manera personalizada (Metcalfe, 2018). Se muestra a continuación una aplicación de Python que define el tipo de mensaje como "Pi Sensor", e incluye encabezados personalizados para el estado de alarma (Metcalfe, 2018).

Lista 1: Aplicación de publicación de Python con AMQP:

```
01 #!/usr/bin/env python
02 import pika
03 node = "192.168.0.121"
04 user = "pete"
05 pwd = "pete"
06
07 # Connect to a remote AMQP server with a username/password
08 credentials = pika.PlainCredentials(user, pwd)
09 connection = pika.BlockingConnection(pika.ConnectionParameters(node,
10         5672, '/', credentials))
11 channel = connection.channel()
12 # Create a queue if it doesn't already exist
13 channel.queue_declare(queue='Rasp_1',durable=True)
14 # Define the properties and publish a message
15 props = pika.BasicProperties(
16     headers= {'status': 'Good Quality',"alarm":"HI"},
17     type ="Pi Sensor")
18 channel.basic_publish(exchange='',
19     routing_key='Rasp_1',body='99.5', properties = props)
20 connection.close()
```

Los resultados del aplicativo, se pueden examinar utilizando la opción "*Cola→ Get Message"* en la interfaz de administración web (Metcalfe, 2018).

Figura 3.43 Cola AMQP con propiedades personalizadas.

Fuente: (Metcalfe, 2018)

Mensajería RabbitMQ

Existe una gran variedad de formas en que los mensajes AMQP son capaces de: publicar y suscribir. La manera más simple es la creación de una cola para que después los mensajes puedan publicarse y suscribirse desde esa cola.

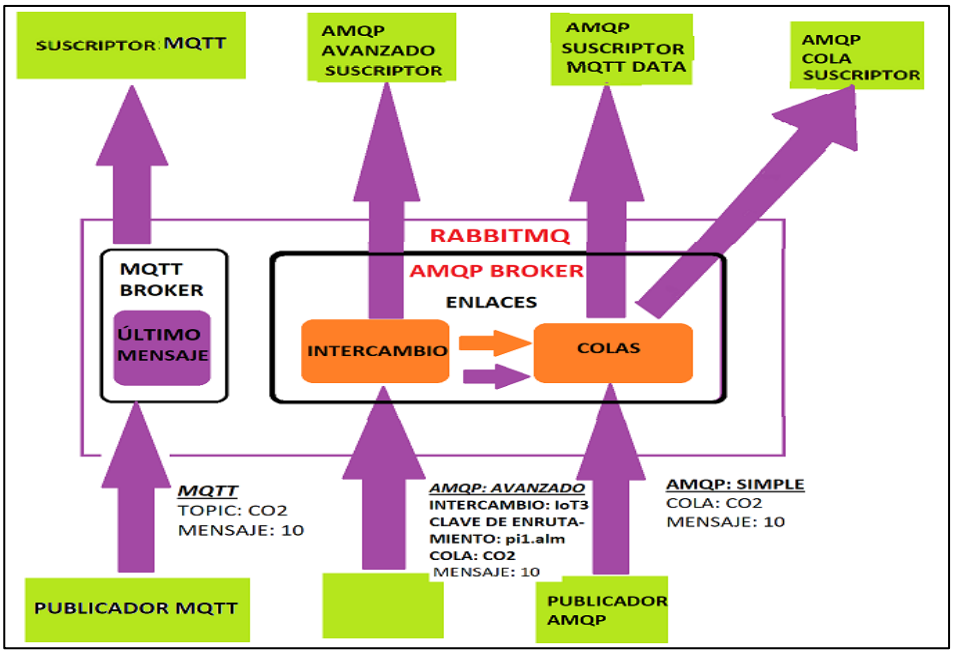


Figura 3.44 Descripción general del bróker RabbitMQ.

Fuente: (Metcalfe, 2018)

Con la ayuda de la distribución y filtrado de mensajes, AMQP acepta varios tipos de intercambiadores diferentes. En un intercambio, los mensajes, utilizan enlaces basados en una clave de enrutamiento para vincularlos en una cola (Metcalfe, 2018).

Los principales tipos de intercambios son: directo, fanout, encabezados (headers) y tema (topic (Metcalfe, 2018)).

Una aplicación de IoT de un intercambio directo, ver figura 3.45, sería en el caso de que un grupo de valores del sensor Raspberry Pi ingresara en un intercambio “Sensor Rasp Pi”. Cuando Rasp Pi publica un resultado del sensor para el intercambio, a su vez, el mensaje también contiene una clave de enrutamiento para vincular o enlazar el mensaje a la cola correcta (Metcalfe, 2018).

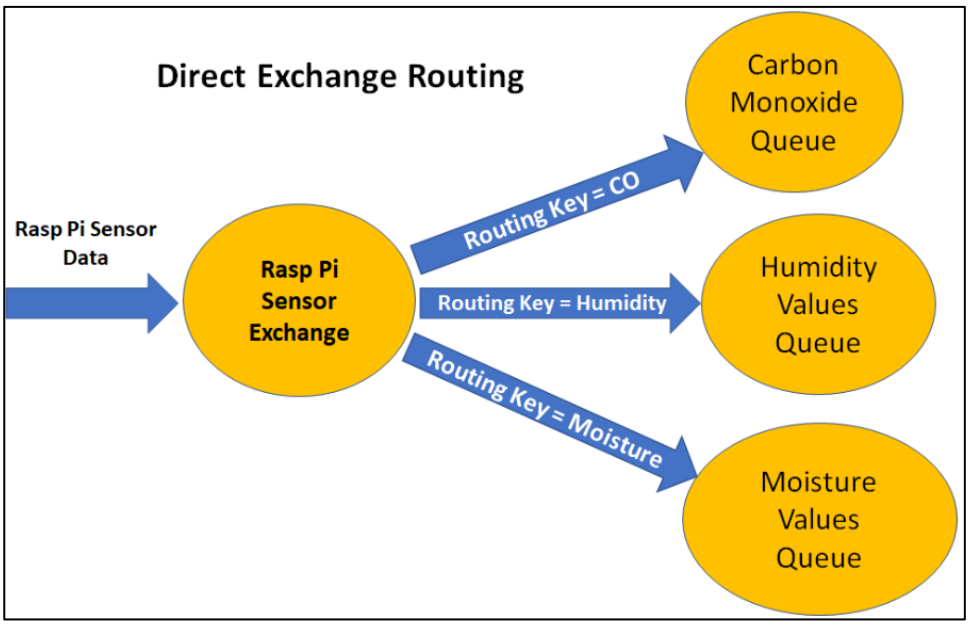


Figura 3.45 Enrutamiento de intercambio directo RabbitMQ.
Fuente: (Metcalfe, 2018)

Una aplicación de IoT de un intercambio de despliegue (fanout), ver figura 3.46. sería con problemas críticos del sensor. Es decir que, el mensaje de problema crítico del sensor se envía a la “cola de tarea de Bill y Sam”, y también a la “cola de todos los puntos de mantenimiento” al mismo tiempo (Metcalfe, 2018).

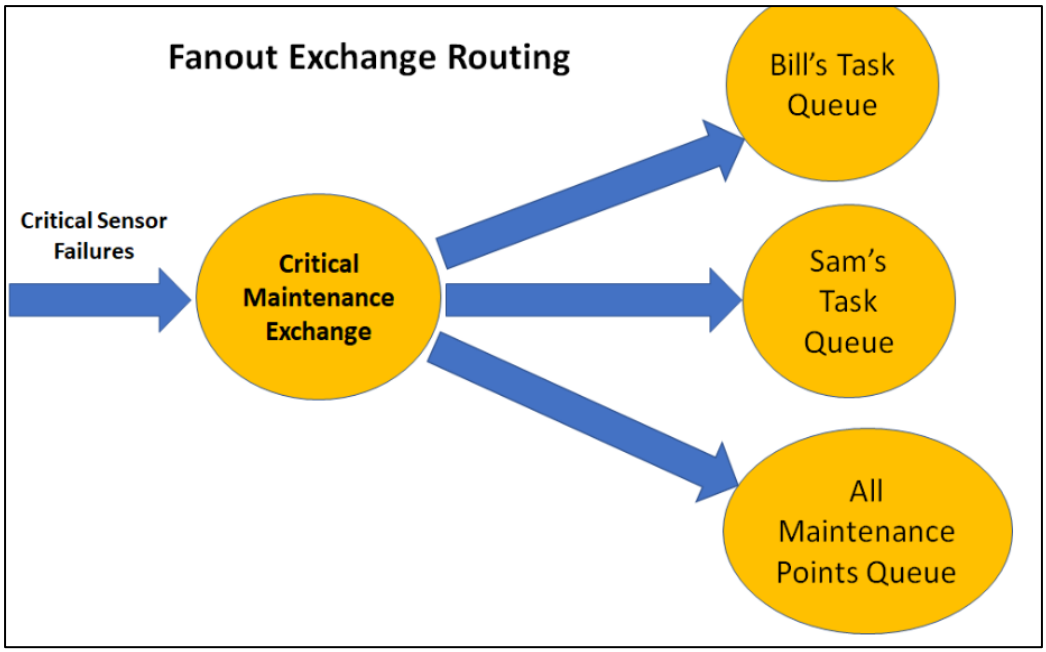


Figura 3.46 Enrutamiento de intercambio de fanout RabbitMQ.

Fuente: (Metcalfe, 2018)

Conexión con MQTT

Según la fuente Mecalfe, después de instalar el "puglin" MQTT, RabbitMQ puede actuar como un bróker MQTT independiente. Pueden estar disponibles, los datos MQTT, a través de una suscripción AMQP vinculando el intercambio MQTT a una cola RabbitMQ.

Para una aplicación con MQTT, se puede utilizar cualquier hardware Arduino compatible con ESP8266, que dispositivos con conectividad, pequeños, baratos y de bajo consumo como el ESP8266 o el ESP32. Para esta demostración, se utiliza `PubSubClient` que se puede instalar con Arduino Library Manager (Metcalfe, 2018).

Como prueba, se usa el sensor de gas combustible MQ-2 de bajo costo ($3) que mide una combinación de GLP, alcohol, propano, hidrógeno e incluso metano. Tener en cuenta que para utilizar completamente este sensor se requiere calibración. En el sensor MQ2, la señal analógica se conecta al pin Arduino A0 y la función analogRead (thePin) se usa para leer el valor del sensor (Metcalfe, 2018).

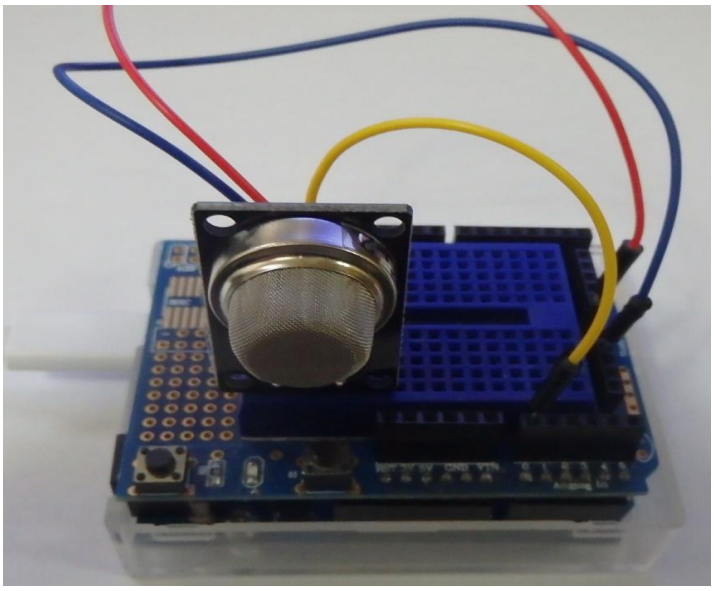


Figura 3.47 Aplicación usando Arduino.

Fuente: (Metcalfe, 2018)

Se muestra un código arduino para leer el sensor de gas MQ2 y publicarlo en el intercambio RabbitMQ de MQTT con un nombre de tema o topic de: `mq2_mqtt` (Metcalfe, 2018).

Lista 2: arduino_2_mqtt.ino

```
01 #include <ESP8266WiFi.h>
02 #include <PubSubClient.h>
03
04 // Update these with values suitable for your network.
05 const char* ssid = "your_ssid";
06 const char* password = "your_password";
07 const char* mqtt_server = "192.168.0.121";
08 const char* mqtt_user = "admin1";
09 const char* mqtt_pass= "admin1";
10
11 const int mq2pin = A0; //the MQ2 analog input pin
12
13 WiFiClient espClient;
14 PubSubClient client(espClient);
15
16 void setup_wifi() {
17   // Connecting to a WiFi network
18   WiFi.begin(ssid, password);
19   while (WiFi.status() != WL_CONNECTED) {
20     delay(500);
```

```
    Serial.print(".");
  }
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  // Loop until we're reconnected
  Serial.println("In reconnect...");
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("Arduino_Gas", mqtt_user, mqtt_pass)) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(9600);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  char msg[8];
  if (!client.connected()) {
    reconnect();
  }

  sprintf(msg,"%i",analogRead(mq2pin));
  client.publish("mq2_mqtt", msg);
  Serial.print("MQ2 gas value:");
  Serial.println(msg);

  delay(5000);
}
```

Cuando se publica el valor MQTT, cualquier cliente MQTT es capaz de suscribirse a él. A continuación, se muestra una suscripción a Python MQTT (Metcalfe, 2018).

Lista 3: mqtt_sub.py

```
01 import paho.mqtt.client as mqtt
02
03 def on_message(client, userdata, message):
04     print ("Message received: "  + message.payload)
05
06 client = mqtt.Client()
07 client.username_pw_set("admin1", password='admin1')
08 client.connect("192.168.0.121", 1883)
09
10 client.on_message = on_message       #attach function to callback
11
12 client.subscribe("mq2_mqtt")
13 client.loop_forever()                 #start the loop
```

3.6.2. Lectura de los mensajes MQTT usando AMQP

Los clientes MQTT pueden suscribirse a los mensajes MQTT directamente o es posible configurar RabbitMQ para tener los datos MQTT accesibles como AMQP. El enrutamiento de mensajes MQTT a colas AMQP se realiza utilizando el método de intercambio directo (Metcalfe, 2018).

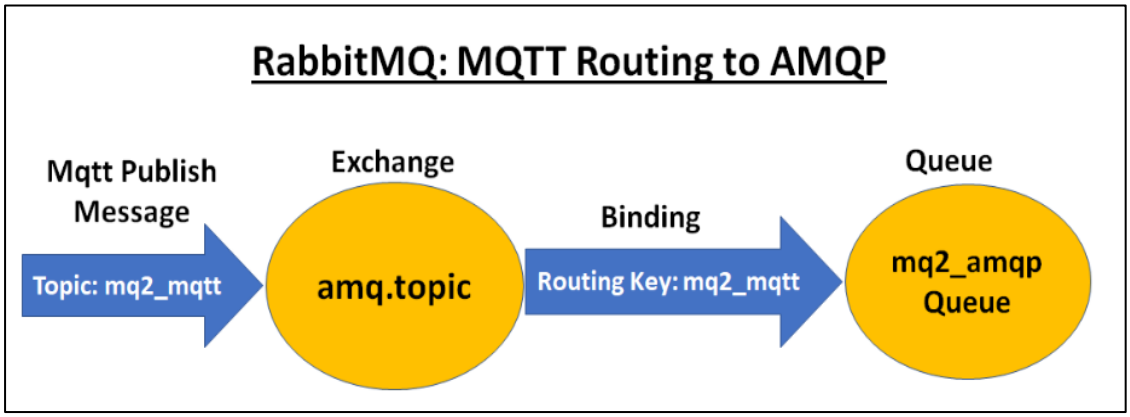


Figura 3.48 Enrutamiento de mensajes MQTT a colas AMQP.

Fuente: (Metcalfe, 2018)

Configurar RabbitMQ para el reenvío del protocolo MQTT, se realizan los siguientes pasos (Metcalfe, 2018):

1. Crear una nueva cola RabbitMQ. (Metcalfe, 2018).
2. Crear un enlace entre el intercambio MQTT y la cola**:** Todo el topic o tema MQTT va al intercambio `amq.topic`. Para los elementos MQTT en este intercambio, se utiliza un enlace, generalmente el nombre del tema MQTT, como la clave de enrutamiento a la cola AMQP (Metcalfe, 2018).

Estos pasos se pueden realizar de varias maneras: manualmente, en el archivo de configuración RabbitMQ, utilizando la herramienta de línea de comandos `rabbitmqadmin` o mediante un programa. Como se realiza esto para múltiples señales, se usa la herramienta **rabbitmqadmin**, y la cadena es (Metcalfe, 2018):

```
./rabbitmqadmin declare queue name=mq2_amqp durable=true

./rabbitmqadmin declare binding source=amq.topic \

destination_type=queue destination=mq2_amqp routing_key=mq2_mqtt
```

El administrador web RabbitMQ se utiliza para verificar el intercambio con el enlace de la cola (Metcalfe, 2018).

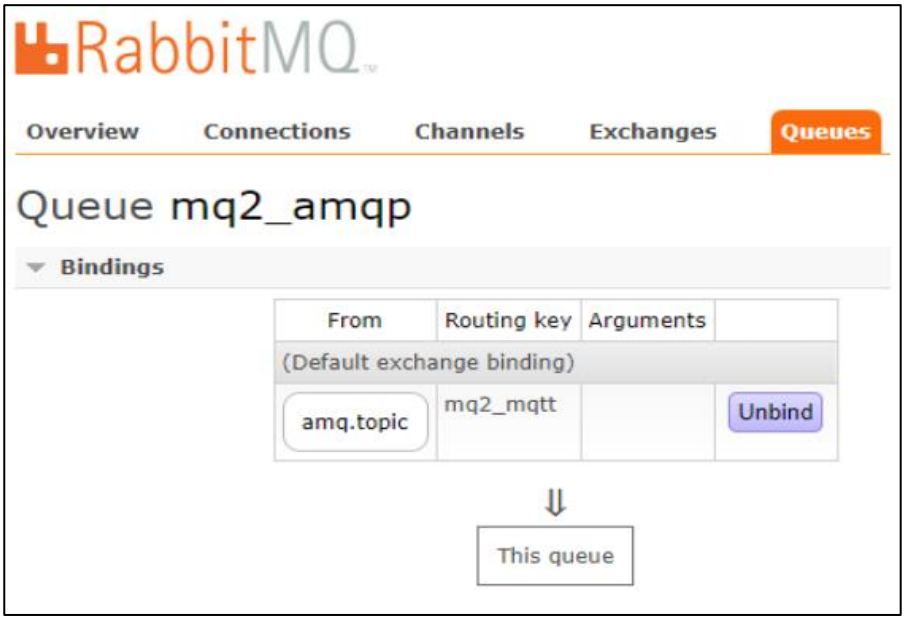


Figura 3.49 Verificación del intercambio con el enlace de la cola.

Fuente: (Metcalfe, 2018).

En ./rabbitmqadmin get queue=mq2_amqp, también se puede usar para ver si hay valores en la cola:

Lista 4: Revisar las colas por mensajes.

```
./rabbitmqadmin get queue=mq2_amqp
+-------------+----------+--------------+---------+
| routing_key | exchange | message_count | payload |
+-------------+----------+--------------+---------+
| mq2_mqtt    | amq.topic | 77           | 157     |
-------------+----------+--------------+---------+
```

3.6.3. Lógica Node Red

Node Red es un entorno de programación visual que permite a los usuarios crear aplicaciones arrastrando y soltando nodos en la pantalla. Los flujos lógicos se crean conectando los diferentes nodos. Es factible instalar Node Red en Raspbian Jesse pero también se puede instalar en Windows, Linux y OSX (Metcalfe, 2018). Para instalar y ejecutar Node Red en un sistema específico, consultar: https://nodered.org/docs/getting-started/installation.

Para instalar los componentes AMQP, seleccionar la opción "*Manage palette*" en el lado derecho de la barra de menú. Luego buscar "AMQP" e instalar **node-red-contrib-amqp**. Si la instalación de Node Red no tiene paneles instalados, buscar: **node-red-dashboard** (Metcalfe, 2018)**.**

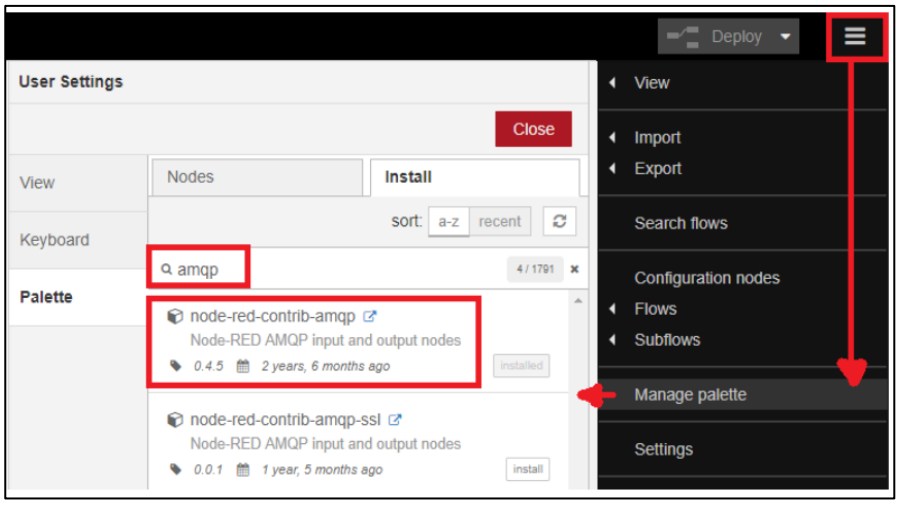


Figura 3.50 Instalando AMQP en Node-RED.

Fuente: (Metcalfe, 2018)

Para este caso de uso, Node Red MQTT y AMQP, usa un nodo MQTT y un nodo AMQP del grupo de paleta de entrada, y dos nodos de medidor del grupo de panel. La lógica Node Red para esto, se realiza en solo 4 nodos (Metcalfe, 2018).

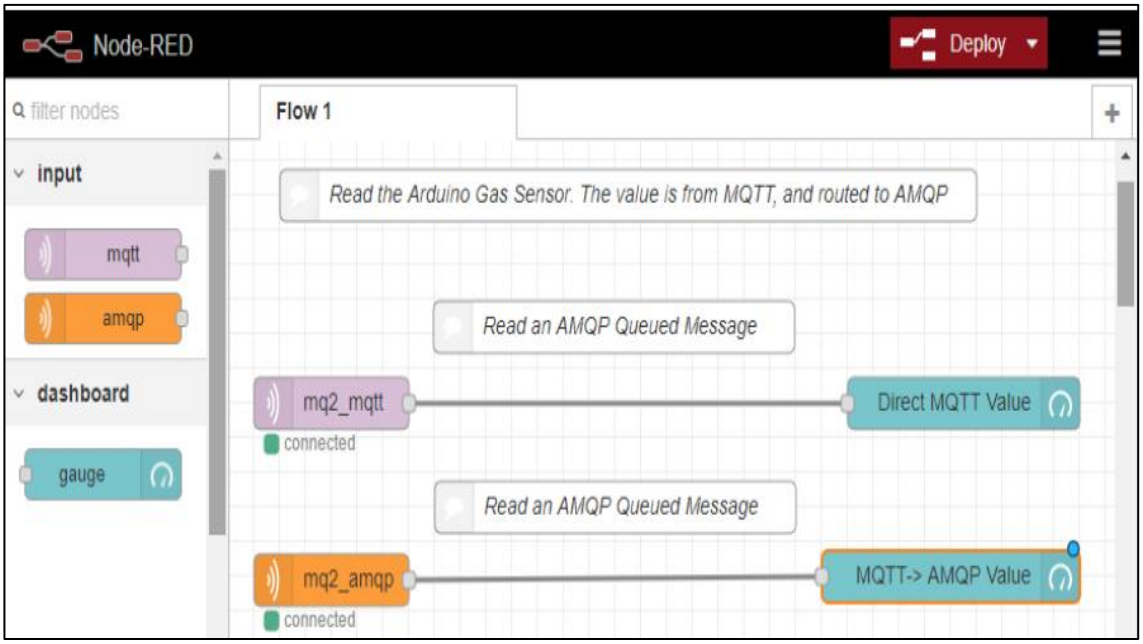


Figura 3.51 Lógica del panel Node-RED.

Fuente: (Metcalfe, 2018).

Los nodos se agregan seleccionándolos, arrastrándolos y soltándolos en la hoja de flujo central (flow sheet) (Metcalfe, 2018).

La lógica se crea haciendo cables de conexión entre las entradas y salidas de un nodo. Después de presentar la lógica, hacer doble clic en cada uno de los nodos para configurar sus propiedades específicas (Metcalfe, 2018).

Se debe especificar las definiciones MQTT y AMQP de su dirección IP RabbitMQ, los derechos de usuario, el tema MQTT y el nombre de la cola AMQP (Metcalfe, 2018).

También se debe hacer doble clic en los nodos de medidor para configurar el aspecto del panel web. Una vez completada la lógica, seleccionar

y presionar el botón “Deploy” (implementar) en el lado derecho de la barra de menú para ejecutar la lógica (Metcalfe, 2018).

Se accede a la interfaz de usuario del panel Node Red en: **http://ipaddress:1880/ui.** Para esta demostración, se utilizó varios sensores y entradas MQ (message queu) diferentes (Metcalfe, 2018).

A continuación, se muestra una imagen del panel web Node-Red que se creó con el mismo valor MQTT, que se muestra como la de origen y como un valor en cola AMQP (Metcalfe, 2018).

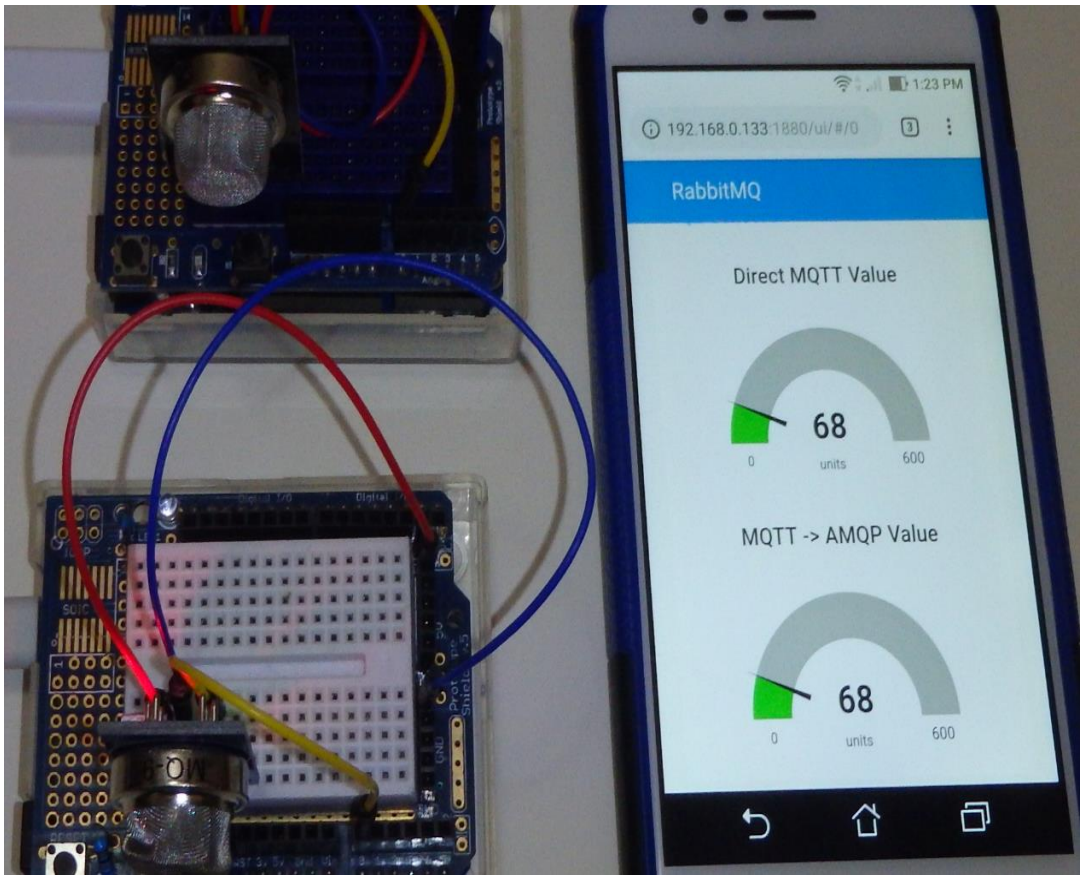


Figura 3.52 Panel de instrumentos Node-RED con datos RabbitMQ.
Fuente: (Metcalfe, 2018)

3.7. Cuadro de costo detallado de la aplicación.

En esta sección se muestra un cuadro de costo detallado de la aplicación utilizando herramientas según hardware y software.

Tabla 3.5 Cuadro de costos de las herramientas informáticas y componentes físicos para la aplicación utilizando AMQP.

	NOMBRE	DEFINICIÓN	COSTO
HARDWARE	RabbitMQ	Es un software que maneja colas de mensajes llamado bróker de mensajería.	GRATUITO
	Node Red	Node Red es un entorno de programación visual que permite a los usuarios crear aplicaciones arrastrando y soltando nodos en la pantalla.	GRATUITO
	Otros como: Grafana	Es un software libre la cual permite la visualización y el formato de datos métricos como un sistema de monitoreo.	GRATUITO
SOFTWARE	Raspberri Pi	Es una placa computadora de mínimo costo, también se la conoce como un ordenador de tamaño reducido	$50 - $60
	Arduino	Se la conoce como plataforma de código abierto para prototipos electrónicos.	$10
	Jumpers	Es un cable con un conector en cada punta terminal que es usado para interconectar entre sí los elementos en una placa de pruebas o protoboard.	20x3: 1.25$
	ESP8266	Es una placa de Wi-Fi de muy bajo costo con pila TCP/IP y con capacidad de MCU (Micro Controller Unit)	$5 - $8
	ESP32	Tiene la misma definición que el dispositivo ESP8266, pero es más complejo porque contiene muchos pines de conexión.	$13 - $14
	Sensor de gas combustible MQ-2	Es un sensor útil en la domótica, ya que usa para detectar fugas de gas de equipos en los mercados de consumo y en las industrias. Contiene un voltaje de entrada 5V y una corriente máxima 150mA.	$3
	Adafruit Proto Shield for Arduino Unassembled Kit - Stackable - Version R3	Este prototipo es el mejor que existe y es mejor con Versión R3. Contiene mayor compatibilidad con la mayoría de los Arduinos.	$9.95
	Mini protoboard	Es una tabla con orificios que están conectados eléctricamente entre sí internamente en la que siguen patrones de líneas, también es posible insertar componentes electrónicos y cables para el armado de circuitos electrónicos.	$1

Nota: En esta tabla se muestra los posibles costos que va entre $93.20 y $107.20 para una aplicación usando herramientas del protocolo AMQP, tener en consideración que el Software es totalmente gratuito.

Elaborado por: Autor

CAPITULO 4: CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones.

- Analizando la información anteriormente mencionada en este trabajo de titulación, se detectó la alta disponibilidad que se logra con la implementación de AMQP como protocolo de mensajería, especialmente para el sector industrial como las aplicaciones de sensores de gas, debido a la prioridad e importancia que se da a la eficiencia de los procesos y a la disponibilidad de los recursos requeridos por los mismos.

- Como se demostró en el estudio, la factibilidad por bajos costos está dada por las herramientas de distribución libre usadas en el protocolo AMQP, las cuales agregan un costo mínimo a los proyectos brindando los resultados esperados, tal como se detalla en la tabla de costeo.

- Las bases de este proyecto que es de actualidad, presente y futuro de IoT a nivel de inteligencia en la domótica, pueden ser utilizadas para cualquier tipo de industria que requiera entrega de mensajería de telemetría de manera confiable orientada a cualquier tipo de dispositivo a ser medido para aplicación en las áreas de automotriz, robótica, aeroespacial, medicina e instituciones educativas, por lo tanto, concluyo que el protocolo AMQP es altamente factible para el uso en los distintos tipos de industria sin comprometer su disponibilidad por costo.

- Con respecto a la aplicación con la herramienta RabbitMQ, se concluye que es fácil de instalar y que el “puglin” de administración web con la herramienta de comandos rabbitmqadmin, facilitó que el sistema fuera fácil de mantener por su alta disponibilidad. RabbitMQ es una opción interesante debido a los intercambios y colas del protocolo AMQP.

4.2. Recomendaciones.

- Se requiere adquisición de equipos de recursos de gama media y alta para todo tipo de industrias que requiere de cierta capacidad de procesamiento para la instalación de RabbitMQ debido que habría una mayor cantidad de mensajes que demanda un mayor rendimiento.

- Se sugiere el uso de lenguaje Python para el uso del protocolo AMQP, por ser este un lenguaje altamente utilizado en la actualidad por su compatibilidad con distintas plataformas y lenguajes de programación en las industrias debido a sus múltiples usos para desarrollar aplicaciones.

- A nivel de hardware en caso de uso del protocolo AMQP, se sugiere un disco de estado sólido de alta velocidad de procesamiento y capacidad para el almacenamiento de la cola de paquetes para un mayor número de transacciones, conforme a la cantidad de paquetes a ser entregadas y a su vez guardados en la cola del protocolo de mensajería.

Bibliografía

Agudelo, D. A. P., & Valbuena, G. G. (2016). Diseño e Implementación de un Sistema de Medición de Consumo de Energía Eléctrica y Agua Potable Remoto con Interacción al Usuario Basado en el Concepto “Internet de las Cosas”.

Alfaro Malatesta, S. A. (2006, noviembre). Análisis del proceso de autoconstrucción de la vivienda en Chile. Bases para la ayuda informática para los procesos comunicativos de soporte. Capítulo 6: Modelo de Comunicación. Recuperado de https://www.tesisenred.net/bitstream/handle/10803/6843/07SAam07de18.pdf?sequence=7&isAllowed=y

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys Tutorials, 17(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Amaya Prieto, C. J. (2017). Aplicación de mensajería instantánea modelo cliente/servidor por protocolo tcp para el departamento de operaciones de la empresa Orange Business Services Colombia S.A. 2017, 40.

APIEM. (2007). La Domótica como Solución de Futuro. Recuperado de https://www.fenercom.com/pdf/publicaciones/la-domotica-como-solucion-de-futuro-fenercom.pdf

Fundación de Innovación Bankinster. (2011). El Internet de las Cosas. En un mundo conectado de objetos inteligentes. Recuperado de http://boletines.prisadigital.com/El_internet_de_las_cosas.pdf

García González, A. J. (2017). IoT: Dispositivos, tecnologías de transporte y aplicaciones. 73.

Garnock-Jones, T. (2010, marzo). AMQP and Beyond Messaging by Extending RabbitMQ. Recuperado de http://tech.labs.oliverwyman.com/downloads/dev.lshift.net/tonyg/SF%20EF%202010%20-%20AMQP%20and%20Beyond%20(slides%20and%20notes).pdf

Gil, G. (2018). Comparativa teórica y práctica de middlewares MQTT. 79. Recuperado de Trabajo fin de máster.

González Piñero, D. (2004, de Enero de). Software libre en los institutos. Recuperado de: https://www.cs.upc.edu/~tonis/daniel_gonzalez_pinyero.pdf

Grzeszczak, K. (2012, octubre 24). RabbitMQ Colas y clustering altamente disponibles con Amazon EC2 · karlgrz. Recuperado 12 de agosto de 2019, de https://karlgrz.com/rabbitmq-highly-available-queues-and-clustering-using-amazon-ec2/

ISO. (2006). Publicly Available Standards. Recuperado 20 de julio de 2019, de ISO the International Organization for Standardization website: https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

ISO. (2014, mayo 1). ISO/IEC 19464:2014(en), Information technology—Advanced Message Queuing Protocol (AMQP) v1.0 specification [ORGANIZATION CERTIFIED]. Recuperado 20 de julio de 2019, de ISO the International Organization for Standardization website: https://www.iso.org/obp/ui/#iso:std:iso-iec:19464:ed-1:v1:en

Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012). Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. 2012 10th International Conference on Frontiers of Information Technology, 257–260. https://doi.org/10.1109/FIT.2012.53

Metcalfe, P. (2018, diciembre 7). RabbitMQ » Linux Magazine. Linux Magazine. Recuperado de: http://www.linux-magazine.com/Issues/2019/221/IoT-with-RabbitMQ

Molinero, J. L. (2018). Implementación del protocolo MQTT-S sobre IEEE 802.15.4e en plataformas OpenMOTE. 73.

Morales, A. (2014). Pi rational Consultoría, Capacitación, Investigación y Desarrollo de Software. Recuperado 22 de julio de 2019, de http://pirational.260mb.net/es/tutoriales/00amqp.html

Morales, G. (2011, abril 1). La domótica como herramienta para un mejor confort, seguridad y ahorro energético. Recuperado 22 de julio de 2019, de https://www.redalyc.org/pdf/5075/507550790007.pdf

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. 2017 IEEE International Systems Engineering Symposium (ISSE), 1–7. https://doi.org/10.1109/SysEng.2017.8088251

O'Hara, J. (2007). ACM Association for Computing Machinery QUEUE Vol. 5 No. 4. Cap. Toward a Commodity Enterprise Middleware AMQP. Recuperado de: https://web.archive.org/web/20120211141750/http://www.acm.org/acmqueue/digital/Queuevol5no4_May2007.pdf

PacktLib. (2013). RabbitMQ Cookbook. Recuperado de: http://naneport.arg.in.th/books/ComputerIT/RabbitMQ%20Cookbook.pdf

Patierno, P. (2016). AMQP Essentials. En DZone REFCARDZ: Vol. VERSION 1.0 (p. 225).

Recuperado de https://dzone.com/storage/assets/1000606-dzone-rc-amqpessentials.pdf

Pietschmann, C. (2016, abril 15). Messaging Architecture: What Is AMQP? Recuperado 20 de julio de 2019, de Build Azure website: https://buildazure.com/messaging-architecture-what-is-amqp/

RabbitMQ. (2017). RabbitMQ by Pivotal. Recuperado 6 de agosto de 2019, de https://www.rabbitmq.com/

Rochin, D. (2016, agosto 24). 1.4 Protocolos de comunicación—Redes de Computadoras. Recuperado 14 de julio de 2019, de https://sites.google.com/site/redescomputadorasjdrc/unidad-1/protocolos-de-comunicación

Saavedra, E. (2018). RabbitMQ Infraestructura. Recuperado de https://www.slideshare.net/estebansaavedra/rabbitmq-105044690

Semle, A., & eFalcom, K. (2016, julio). Protocolos IIot para considerar. www.efalcom.com, 32–35.

SORTILEGIO LTDA. (2019). Ventajas y beneficios de nuestra domótica. Recuperado 18 de agosto de 2019, de website: https://www.smarthomecolombia.com/domotica/en-que-consiste-nuestra-domotica-y-automatizacion/ventajas-y-beneficios-de-nuestra-domotica

Talaminos Barroso, A. (2011, diciembre). Nuevo sistema de información basado en el conocimiento: una aplicación al control de glucosa en pacientes diabéticos, middleware.
Recuperado de
http://bibing.us.es/proyectos/abreproy/70308/fichero/3.+ESTADO+DEL+ARTE+2.pdf

Valiente Cristancho, A. (2017, mayo). Integración de internet de las cosas en sistema embebido system on chip, con aplicación a domótica.

Vinoski, S. (2006). Advanced Message Queuing Protocol. IEEE Internet Computing, 10(6), 87–89. https://doi.org/10.1109/MIC.2006.116

Glosario

AMQP: Advanced Message Queuing Protocol.

MQTT: Message Queue Telemetry Transport

CoAP: Constrained Application Protocol.

HTTP: Hyper Text Transport Protocol.

IoT: Internet of Things.

TCP/IP: Protocolo de control de transmisión/Protocolo de Internet.

CÓDIGO ABIERTO: Software libre o gratuito

APLICACIÓN: Tipo de software de computadora que sirve para realizar un grupo de tarea o funciones coordinadas siempre pensando en el usuario.

OBJETO: Quiere decir que varios dispositivos/objetos/cosas conectados a la red mantienen comunicación continua y privada durante el intercambio de información, ya sea estos de computación, máquinas digitales y mecánicas entre otros, sin necesidad de interacción humano/humano.

MIDDLEWARE: Software que facilita la comunicación entre el cliente y servidor, asistiendo a una aplicación para comunicarse con otras aplicaciones, sistemas operativos, entre otros.

OASIS: Organización sin fines de lucro.

M2M: Machine to Machine.

LAN/WAN: Local Area Network/Wide Area Network.

QUEU: La cola es el concepto central en AMQP.

OSI: Open Systems Interconnect

ISO: International Organization for Standardization.

UDP: User Datagram Protocol.

NUBE: Significa trabajar con aplicaciones y programas que no se encuentran instalados en dispositivos o que almacenan datos en servidores externos.

URI: Identificador Universal de Recursos.

RESTFUL WEB: REpresentational State Transfer.

TIC: Tecnología de información y comunicación.

TRANSACCIÓN: Interacción que contiene una estructura de información bastante compleja, establecida por diversos procesos que se han de emplear uno después del otro.

SCTP: Stream Control Transmission Protocol.

TLS/SSL: Transport Layer Security/ Secure Sockets Layer.

IPSec: Internet Protocol Security.

SASL: Simple Authentication and Security Layer.

MESSAGE QUEU: Cola de mensajes.

PWM: Pulse Width Modulation/Modulación por Ancho de Pulso.

SOFTWARE: Es un conjunto de dispositivos físicos por el cual está diseñado un equipo.

HARDWARE: Conjunto de aplicaciones, programas, normas, entre otras que permite el funcionamiento adecuado del equipo.

ASÍNCRONO: No tiene coincidencia temporal, es decir están separadas por un rango de tiempo.

IEC: Comisión Electrotécnica Internacional.

JTC: Joint Technical Committee on Information Technology (ISO/IEC).

PAS: Proteger, Avisar y Socorrer.

SISTEMAS: Permite almacenar y procesar información, también es el conjunto de partes interconectadas: software, hardware y personal informático

SERVICIOS: Se lo conoce cuando un proveedor de servicios, establece a través de un sistema de telecomunicación, satisfacer necesidades del cliente.

CLÚSTER: Conjunto de ordenadores que se interconectan entre sí y se comportan como si fueran un solo ordenador o computadora.

BRÓKER: Es un intermediario de mensajería que realiza transacciones entre un productor y un cliente.

MOM: Message Oriented Middleware.

HOST VIRTUAL: Definido como una colección de intercambios, colas de mensajes y demás objetos asociados.

PAQUETE: Son bloques que dividen la información para enviar al destino en el nivel de red.

CONEXIÓN: Contacto que se establece entre dos o más dispositivos para intercambiar información.

INFOIMPLEMENTOS: Utensilio informático con capacidad de procesamiento.

QoS: Quality of Service.

RFID: Radio Frecuency Identification.

OVERLOAD: La sobrecarga del consumidor, es decir que la cola de mensajes es procesada y guardada en RAM por el intermediario. Por lo tanto, los consumidores pueden verse abrumados por la tasa de entregas, lo que puede acumular un retraso en la memoria y quedarse sin memoria o hacer que el sistema operativo termine su proceso.

INTERFAZ: Conexión entre sistemas, dispositivos, programas, entre otros, que facilita una comunicación permitiendo el intercambio de información.

PAYLOAD: Conjunto de datos a transmitirse incluyendo el mensaje a enviar.

SIZE: Tamaño del paquete, primero 4 bytes muestran el tamaño del paquete.

DOFF: Data Offtset. Da la posición del cuerpo dentro del paquete.

TYPE: Tipo de paquete que incluye el formato y el propósito del paquete.

ROUTING KEY: Clave de enrutamiento, es la parte del encabezado de cada mensaje para así enrutarlo a las colas de mensajería.

CHANNEL: Abre un canal de comunicación para conectarse a las colas de mensajería.

CLOUDAMQP: Servidor RabbitMQ administrados en la nube. Ofrece varias herramientas de monitoreo.

SERVIDOR: Es una máquina o tipo de equipo infórmatico encargado de proporcionar información a clientes.

PUGLIN: Es un programa o código que se adjunta a otro programa para darle más opciones y funciones.

OBJETO (PROGRAMACIÓN): Es el que permite separar los diferentes componentes de un programa, ayudando a simplificar su elaboración, depuración y futuras renovaciones.

MÉTODOS: En programación, son las funciones que permite efectuar el objeto y que nos rinden algún tipo de servicios durante el transcurso del programa.

INSTANCIA: En programación, se lo conoce a la instancia a todo objeto que deriva de algún otro, entonces todos los objetos son instancias de algún otro.

CLASE (PROGRAMACIÓN): Es la descripción de un objeto.

INTERFAZ: Es un acceso o salida de información.

API: Application Programming Interface. Programa que permite la interacción entre la aplicación y el usuario final.

HOST: Dispositivo conectado en la red con capacidad de recibir, procesar y enviar información.

SESIÓN DE MENSAJERÍA: Se define como un intercambio de información o como diálogo y conversación entre muchos dispositivos de comunicación, usuario o un ordenador. En programación, es el componente de programación de las tecnologías de web scripting que almacenan información sobre un usuario al pasar de una página a otra.

ESTADO: Es el modo de acción en la que se encuentra un dispositivo, en función de su tarea en el sistema.

IDLE: (libre o disponible) es decir que en el momento no realiza una acción sin embargo el sistema lo reconoce como recurso disponible para usarse.

RUNING O ACTIVO: realiza una acción para mantener un sistema en operaciones constantes.

INACTIVO: No realiza ninguna acción y el sistema lo mantiene como un dispositivo pasivo.

RETAIN FLAG: En programación, es parte del mensaje enviado por el Publicador que solicita al cliente mantener o descartar (true or false) el mensaje recibido.

RASPBIAN: Se lo conoce como el sistema operativo para Raspberry Pi y está actualizada a Debian 8 “Jessie”, también es conocida como una mini computadora lo cual es la más vendida y utilizada.

Anexos.

Anexo 1. Ejemplo de una Cosa/Objeto/Dispositivo:

Una cosa, en la internet de las cosas, puede ser una persona con un implante de monitor de corazón, un animal de granja con un transpondedor de biochip, un automóvil que tiene sensores incorporados para alertar al conductor cuando la presión de los neumáticos es baja, o cualquier otro objeto natural o artificial al que se puede asignar una dirección IP y darle la capacidad de transferir datos a través de una red.

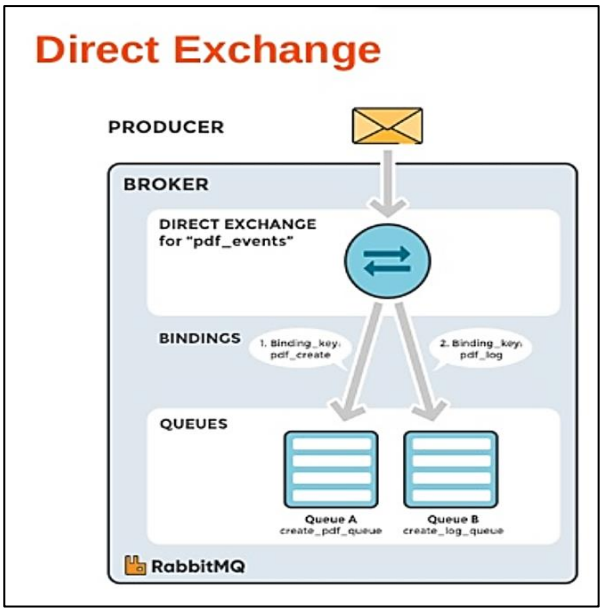
Anexo 2. Funcionamiento estándar de los “Exchange”

El funcionamiento básico o estándar que posee RabbitMQ es el siguiente(Saavedra, 2018):

1. El productor publica un mensaje para el “exchange” (Saavedra, 2018).
2. El exchange recibe el mensaje y es responsable del enrutamiento del mensaje (Saavedra, 2018).
3. Se debe establecer un enlace entre la cola y el “exchange”. En este caso, tenemos enlaces a dos colas diferentes del intercambio. El intercambio enruta el mensaje a las colas (Saavedra, 2018).
4. Los mensajes permanecen en la cola hasta que sean manejados por un consumidor (Saavedra, 2018).
5. El consumidor maneja el mensaje (Saavedra, 2018).

Anexo 2.1. Ejemplo y gráfica del Intercambio Directo

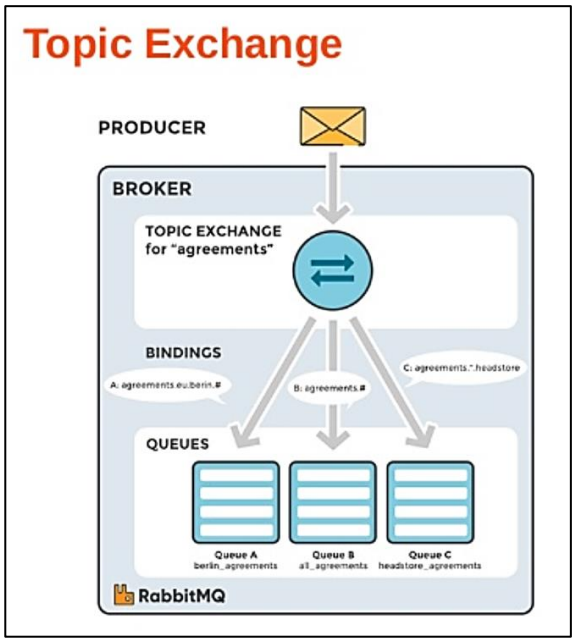
Ejemplo: Se envía un mensaje con la clave de enrutamiento pdf_log al intercmabio pdf_events. Los mensajes se enrutan a pdf_log_queue porque la clave de enrutamiento (pdf_log) coincide con la clave del enlace (pdf_log). Si la clave de enrutamiento de mensajes no coincide con ninguna clave de enlace, el mensaje será descartado (Saavedra, 2018).



Anexo 1. Funcionamiento del intercambio Directo
Fuente: (Saavedra, 2018)

Anexo 2.2. Ejemplo y gráfica del Intercambio por topic o tema.

Ejemplo: Se envía un mensaje con la clave de enrutamiento agreements.eu.berlin a los acuerdos de intercambio (Saavedra, 2018). Los mensajes se enrutan a la cola berlin_agreements porque el patrón de enrutamiento de "agreements.eu.berlin.#" coincide, asi también se enruta a la cola all_agreements porque la clave de enrutamiento (agreements.eu.berlin) coincide con el patrón de enrutamiento (agreements. #) (Saavedra, 2018).

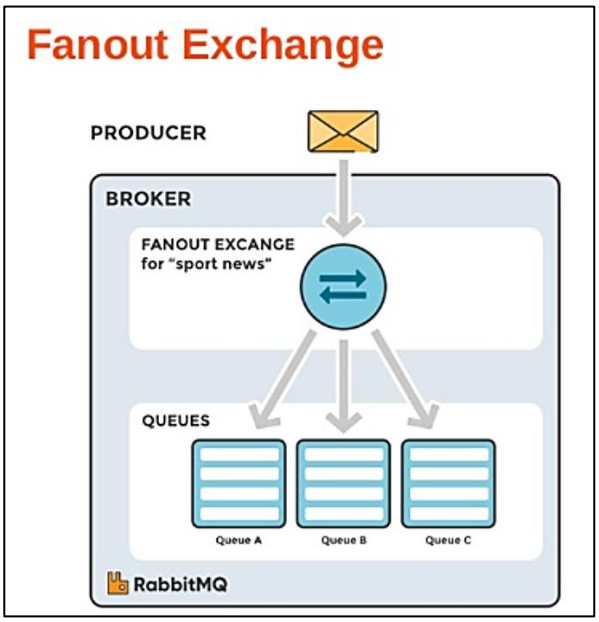


Anexo 2. Funcionamiento del intercambio por tema
Fuente: (Saavedra, 2018)

Anexo 2.3. Ejemplo y gráfica del Intercambio Fanout

El intercambio Fanout copia y enruta un mensaje recibido a todas las colas que están vinculadas a él, independientemente de las claves de enrutamiento o la coincidencia de patrones como con los intercambios directos y de temas. Las claves proporcionadas simplemente serán ignoradas. Los intercambios de Fanout pueden ser útiles cuando el mismo mensaje debe enviarse a una o más colas con consumidores que pueden procesar el mismo mensaje de diferentes maneras. La gráfica muestra un ejemplo en el que un mensaje recibido por el intercambio se copia y enruta a las tres colas que están vinculadas al intercambio (Saavedra, 2018).

Ejemplo Se envía un mensaje al intercambio sport_news. El mensaje se enruta a todas las colas (Cola A, Cola B, Cola C) porque todas las colas están vinculadas al intercambio. Siempre que se ignoren las claves de enrutamiento (Saavedra, 2018).



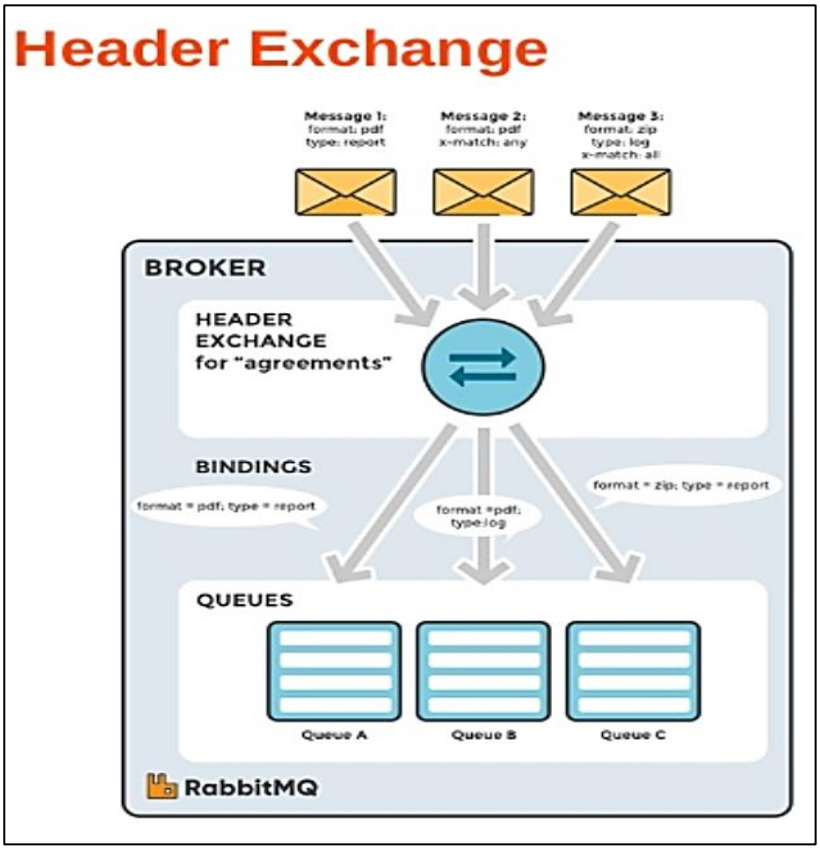
Anexo 3. Funcionamiento del intercambio "fanout"
Fuente: (Saavedra, 2018)

Anexo 2.4. Ejemplo y gráfica del Intercambio de encabezado.

El header exchange intercambia la ruta en función de los argumentos que contienen encabezados y valores opcionales. Son muy similares a los Topic Exchange, pero se enruta en función de los valores de encabezado en lugar de las claves de enrutamiento. Un mensaje se considera coincidente si

el valor del encabezado es igual al valor especificado al vincularse (Saavedra, 2018).

La propiedad "x-match" puede tener dos valores diferentes: "any" o "all", donde "all" es el valor predeterminado. Un valor de "all" significa que todos los pares de encabezado (clave, valor) deben coincidir y un valor de "any" significa que al menos uno de los pares de encabezado debe coincidir. El argumento "any" es útil para dirigir mensajes que pueden contener un subconjunto de criterios conocidos o desordenados (Saavedra, 2018).



Anexo 4. Funcionamiento del intercambio de encabezado
Fuente: (Saavedra, 2018)

DECLARACIÓN Y AUTORIZACIÓN

Yo, **Pérez Leones, Kamila** con C.C: # 0919904730 autor del Trabajo de Titulación: **ESTUDIO Y ANÁLISIS DEL PROTOCOLO DE MENSAJERÍA AVANZADO EN EL INTERNET DE LAS COSAS PARA APLICACIÓN EN EL CAMPO DE LA DOMÓTICA,** previo a la obtención del título de **INGENIERO EN TELECOMUNICACIONES** en la Universidad Católica de Santiago de Guayaquil.

1.- Declaro tener pleno conocimiento de la obligación que tienen las instituciones de educación superior, de conformidad con el Artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de titulación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la SENESCYT a tener una copia del referido trabajo de titulación, con el propósito de generar un repositorio que democratice la información, respetando las políticas de propiedad intelectual vigentes.

Guayaquil, 12 de Septiembre del 2019

f. _____

Nombre: Pérez Leones, Kamila

C.C: 0919904730

REPOSITORIO NACIONAL EN CIENCIA Y TECNOLOGÍA

FICHA DE REGISTRO DE TESIS/TRABAJO DE TITULACIÓN

TÍTULO Y SUBTÍTULO:	Estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica.		
AUTOR(ES)	PÉREZ LEONES, KAMILA		
REVISOR(ES)/TUTOR(ES)	M. Sc. Alvarado Bustamante, Jimmy Salvador		
INSTITUCIÓN:	Universidad Católica de Santiago de Guayaquil		
FACULTAD:	Facultad de Educación Técnica para el Desarrollo		
CARRERA:	Ingeniería en Telecomunicaciones		
TITULO OBTENIDO:	Ingeniero en Telecomunicaciones		
FECHA DE PUBLICACIÓN:	12 de Septiembre del 2019	**No. DE PÁGINAS:**	110
ÁREAS TEMÁTICAS:	Comunicaciones Inalámbricas.		
PALABRAS CLAVES/ KEYWORDS:	PROTOCOLO, IoT, OSI, BRÓKERS, MENSAJERÍA, RABBITMQ.		

RESUMEN/ABSTRACT (150-250 palabras): El presente trabajo de titulación está orientado en el estudio y análisis del protocolo de mensajería avanzado del internet de las cosas para aplicación en el campo de la domótica, evaluando su actuación en la capa de aplicación dentro del modelo de referencia OSI, en donde intervienen diferentes tipos de programas y la razón de ser uno de los protocolos más conocido en la IoT. Existe el bróker de mensajería, donde se gestiona las colas de mensajes de código abierto que intervienen en el protocolo estudiado. El presente trabajo de titulación se compone por los siguientes capítulos: el capítulo 1 consta de la introducción, antecedentes, planteamiento del problema, justificación, objetivos, metodología de la investigación, el capítulo 2 se define los conceptos esenciales del protocolo AMQP y su diferencia ante los demás protocolos de la IoT, el capítulo 3 contiene una visualización y resultados del estudio del protocolo utilizando una herramienta de software de distribución libre, tal como, RabbitMQ es el que ha alcanzado una popularidad especial.

ADJUNTO PDF:	☒ SI	☐ NO
CONTACTO CON AUTOR/ES:	**Teléfono:** +5939676608298	E-mail: kamilaperezleones@gmail.com
CONTACTO CON LA INSTITUCIÓN: COORDINADOR DEL PROCESO DE UTE	**Nombre:** Palacios Meléndez Edwin Fernando	
	Teléfono: +593-9-68366762	
	E-mail: edwin.palacios@cu.ucsg.edu.ec	
SECCIÓN PARA USO DE BIBLIOTECA		
Nº. DE REGISTRO (en base a datos):		
Nº. DE CLASIFICACIÓN:		
DIRECCIÓN URL (tesis en la web):		