6

# Features of the framework for building applications integrating with messaging systems

Java Backend Developer I

# Objetivos

- Introducción AMQP
- AMQP vs JMS
- JMS Implementations
- amqp-template
- AMQP vendors:
  1. Rabbit MQ Broker
  2. Azure Service Bus
  3. Kafka

# Agenda

Revisión de los siguientes conceptos:

- Introducción AMQP
- AMQP vs JMS
- JMS Implementations
- amqp-template
- AMQP vendors:
  1. Rabbit MQ Broker
  2. Azure Service Bus
  3. Kafka

# Advanced Message Queuing Protocol AMQP

## Introducción

AMQP es un protocolo de comunicación abierto orientado y diseñado como un servicio para el intercambio de mensajes entre diferentes plataformas. Trabaja a nivel de encolamiento (queuing).

Para ello debemos de tener claro 3 conceptos:

- Broker
- Publishers
- Subscribers

# Advanced Message Queuing Protocol AMQP

**Conceptos importantes**

- Broker es el puente o intermediario que une y comunica a los publishers con los subscribers.
- Publishers es aquel que publicará o enviará un mensaje al broker. El broker se encargará de recibir el mensaje y distribuirlo a todos los subscribers que esten conectados con el broker. La entrega será solo ha aquellos subscribers interesados en dicho mensaje para ello se utiliza el concepto de topics para enrutar los mensajes.
- Subscribers es aquel que recibe el mensaje.

# Advanced Message Queuing Protocol AMQP

## Definición

El estándar **AMQP** (**Advanced Message Queuing Protocol**) es un protocolo de estándar abierto en la capa de aplicaciones de un sistema de comunicación. Las características que definen al protocolo AMQP son la orientación a mensajes, encolamiento ("queuing"), enrutamiento (tanto punto-a-punto como publicación-subscripción), exactitud y seguridad

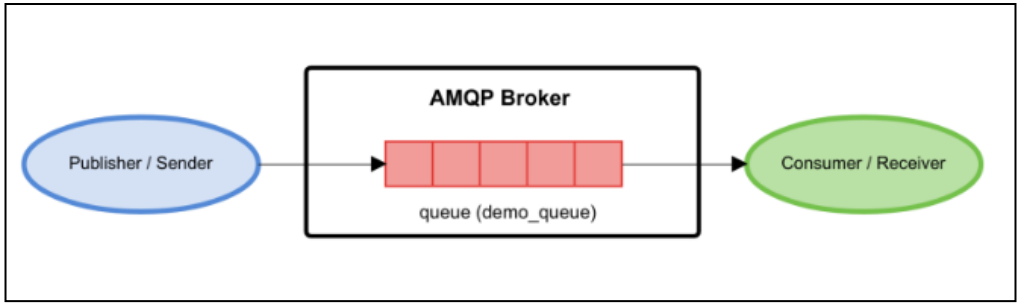
# Advanced Message Queuing Protocol AMQP

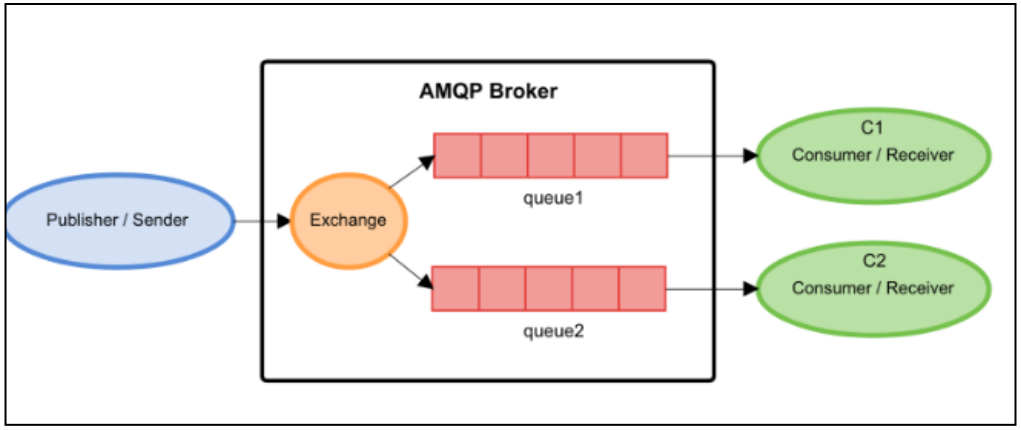
## Definición

Proporciona el comportamiento tanto del servidor que provee los mensajes como del cliente de la mensajería hasta el punto de que las implementaciones de diferentes proveedores son verdaderamente interoperables, de la misma manera que los protocolos SMTP, HTTP, FTP y análogos han creado sistemas interoperables

# Advanced Message Queuing Protocol AMQP

## Diagrama



# Java Message Service (JMS)

## Concepto

La API Java Message Service (JMS) es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma Java2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera asíncrona.

El servicio de mensajería instantánea también es conocido como Middleware Orientado a Mensajes (MOM por sus siglas en inglés) y es una herramienta universalmente reconocida para la construcción de aplicaciones empresariales.

# AMQP vs JMS

**Comparativa**

JMS es una API Java estándar para comunicarse con Middleware Orientado a Mensajes (MOM).

JMS es parte de Java J2EE, y está definido por JSR 914. Se considera robusto y maduro.

JMS permite que dos aplicaciones Java diferentes se comuniquen, esas aplicaciones podrían estar utilizando clientes JMS y aún así ser capaces de comunicarse, pero seguir desacopladas.

# AMQP vs JMS

**Comparativa**

Si se requiere un broker de mensajería JMS, probablemente se seleccione como cliente ActiveMQ dentro del código por ser más popular.

Por lo tanto, JMS permite que los componentes de Java envíen y reciban mensajes de manera confiable, con solo unas pocas líneas de código, que aún no se han acoplado libremente.

Uno puede reemplazar cualquier broker JMS con otro con cambios mínimos o incluso sin cambios en su código.

# AMQP vs JMS

**Comparativa**

AMQP es una especie de protocolo de capa de aplicación estándar abierto para entregar mensajes que fue desarrollado inicialmente por O'Hara de JPMorgan.

AMQP puede poner en cola y enrutar mensajes. Puede ir P-2-P (One-2-One), publicar / suscribirse, y algo más, de manera confiable y segura.

Si desea AMQP, probablemente elija RabbitMQ, Qpid o StormMQ.

AMQP proporciona una descripción sobre cómo se debe construir un mensaje. No proporciona una API sobre cómo se debe enviar el mensaje.

# AMQP vs JMS

**Comparativa**

JMS es una API y AMQP es un protocolo.

JMS, cuando se definió, en realidad no forzó un protocolo entre el cliente JMS y el servidor de mensajería JMS.

El cliente JMS, que implementa la API JMS, puede usar cualquier protocolo para comunicarse con el servidor JMS. Sin embargo, el cliente JMS debe asegurarse de cumplir con la API JMS.

AMQP, por otro lado, no es más que un protocolo entre un cliente de mensajería y el servidor de mensajería. Por lo tanto, un cliente JMS puede usar AMQP como protocolo para comunicarse con el servidor de mensajería.

# AMQP vs JMS

**Comparativa**

De hecho, por ejemplo, ActiveMQ ofrece soporte para AMQP.

Si bien hay implementaciones como ActiveMQ, y SonicMQ, que ofrecen cierto nivel de interoperabilidad de productos cruzados, esto se hace a través de protocolos propietarios, API y bibliotecas de clientes. No puede simplemente reemplazar un broker con otra sin ir a su código, como fue el caso de las aplicaciones Java.

Estamos profundamente acoplados aquí a un broker específico que no puede reemplazarse fácilmente, si eso es posible.

AMQP se basa en un protocolo de cable binario que fue diseñado para la interoperabilidad entre diferentes proveedores y plataformas. (Vendors).

# AMQP vs JMS

## Comparativa

Avancemos un paso más y consideremos el caso en el que un cliente Java desea enviar un mensaje a un cliente Ruby. O viceversa. El cliente Ruby no puede hablar JMS, por lo que necesitamos un intermediario de mensajes que pueda unir las dos plataformas, Java y Ruby. Es posible que queramos usar STOMP para eso a través de ActiveMQ.

Eso parece resolver el problema, pero debido a ello estamos acoplados a una solución de proveedor específica.

Pero, ¿qué sucede si se trata de un cliente C # que habla de la API NMS que intenta conectarse a un cliente Ruby que está configurado para usar STOMP? No va a funcionar En el escenario de la vida real, podemos dar muchos más ejemplos utilizando combinaciones de plataformas mixtas y requisitos.

# AMQP vs JMS

## Comparativa

Aquí es donde AMQP se está volviendo útil. AMQP proporciona un protocolo de mensajería estándar que se encuentra en todas las plataformas. No importa qué cliente AMQP se use, siempre y cuando sea AMQP, se mantendrá. El cliente que usa AMQP es completamente independiente en cuanto a qué API de cliente AMQP o broker AMQP se usa.

Finalmente se puede concluir en que AMQP trata de definir el 'qué' y JMS trata de definir el 'cómo'.

# JMS Implementations

- Amazon SQS's Java Messaging Library
- Apache ActiveMQ
- Apache Qpid, usando AMQP
- IBM MQ (formalmente MQSeries, y WebSphere MQ)
- IBM WebSphere Application Server's Service Integration Bus (SIBus)
- JBoss Messaging and HornetQ from JBoss
- JORAM from the OW2 Consortium
- Open Message Queue from Oracle
- OpenJMS from the OpenJMS Group
- Oracle WebLogic Server and Oracle AQ
- RabbitMQ from Pivotal Software

# AMQP TEMPLATE

## Concepto

Al igual que con muchas otras abstracciones de alto nivel proporcionadas por Spring Framework y proyectos relacionados, Spring AMQP proporciona una "plantilla" que desempeña un papel central. La interfaz que define las operaciones principales se llama AmqpTemplate. Esas operaciones cubren el comportamiento general para enviar y recibir mensajes. En otras palabras, no son exclusivos de ninguna implementación, de ahí el "AMQP" en el nombre. Por otro lado, hay implementaciones de esa interfaz que están vinculadas a implementaciones del protocolo AMQP.

# AMQP TEMPLATE

## Concepto

A diferencia de JMS, que es una API de nivel de interfaz en sí, AMQP es un protocolo de nivel de cable. Las implementaciones de ese protocolo proporcionan sus propias bibliotecas de cliente, por lo que cada implementación de la interfaz de plantilla dependerá de una biblioteca de cliente particular. Actualmente, solo hay una implementación única: RabbitTemplate. En los ejemplos que siguen, a menudo verá el uso de un "AmqpTemplate", pero cuando mira los ejemplos de configuración, o cualquier extracto de código donde se instancia la plantilla y / o se invocan los configuradores, verá el tipo de implementación (p. Ej. "RabbitTemplate").

# RabbitMQ

## Característica

RabbitMQ se implementa en Erlang.

Dado que Erlang requiere su propio tiempo de ejecución, antes que nada tenemos que instalar el tiempo de ejecución de Erlang / OTP (Open Telecom Platform).

Elija la versión R14B02 para la plataforma Windows en la página de descarga.

Elegimos la siguiente carpeta de instalación C: \ erl5.8.3 y definimos una variable de entorno que apunta a esa carpeta

```
ERLANG_HOME=C:\erl5.8.3
```

# RabbitMQ

## Característica

Instalación de RabbitMQ

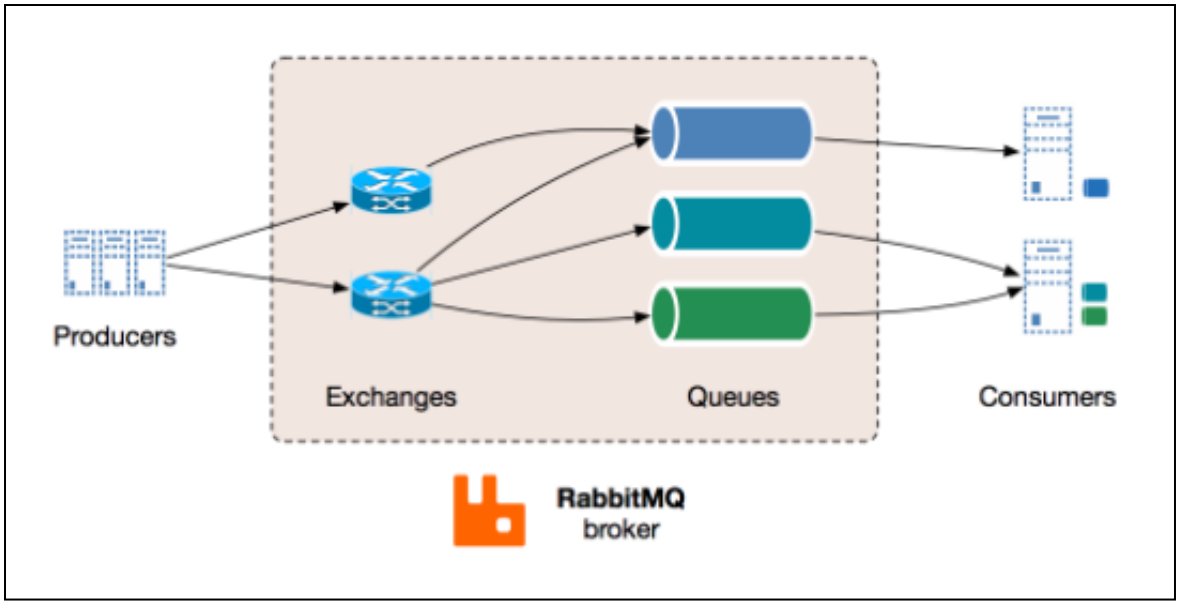
Después de descargar RabbitMQ extraemos el ZIP a C: \ rabbitmq_server-2.4.1. RabbitMQ se inicia con el siguiente script:

```
C:\rabbitmq_server-2.4.1\sbin\rabbitmq-server.bat
```

# RabbitMQ

## Característica

RabbitMQ presenta una pequeña initial memory footprint y tiene un tiempo de aceleración corto, dos ventajas para entornos de nube elástica. Las API de cliente se ofrecen para varios idiomas, incluidos Java y .NET.



# AMQP Vendor RabbitMQ

## Concepto

RabbitMQ es el agente de mensajes de vFabric Cloud Application Platform. Su compatibilidad con el estándar de protocolo de mensajería AMQP hace que RabbitMQ sea una combinación perfecta para escenarios de alta disponibilidad. RabbitMQ es de código abierto y se puede usar fuera de la plataforma vFabric. El soporte comercial está disponible bajo demanda.

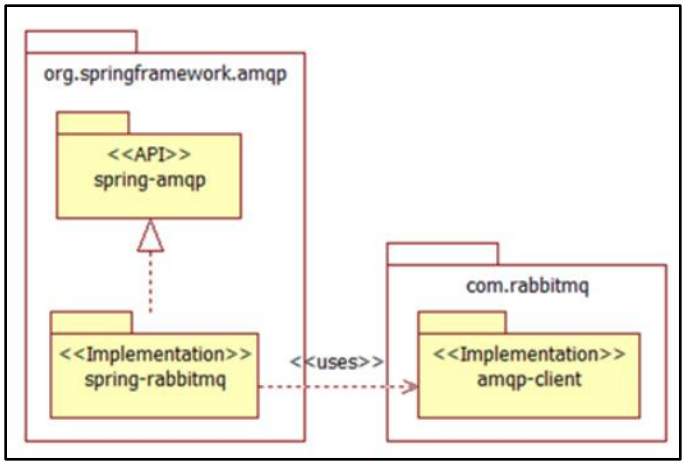
Vamos a conocer cómo puede usar Spring AMQP para integrar un broker RabbitMQ con su aplicación Java.

# AMQP Vendor RabbitMQ

## Spring AMQP

Spring AMQP ofrece una API para un fácil acceso a los brokers de mensajes AMQP. La plantilla para AMQP, es AmqpTemplate.

Las dependencias de los proyectos Spring involucrados se muestran en la siguiente figura



# AMQP Vendor RabbitMQ

**Spring AMQP**

El proyecto spring-amqp contiene todas las interfaces generales esenciales (por ejemplo, la plantilla AmqpTemplate) y las clases API, mientras que la implementación específica del intermediario entra en spring-rabbitmq, que a su vez se basa en la API Java general para RabbitMQ amqp-client.

Su aplicación cliente solo depende de spring-amqp para lograr un acoplamiento débil.

Esto le permite cambiar de un broker AMQP a otro sin ningún cambio importante en el código.

# AMQP Vendor RabbitMQ

## AMQP Template

El contexto de la aplicación contiene una fábrica de conexiones y AmqpTemplate. Para fines administrativos, agregamos otro bean.

```
<!-- Connection Factory -->
<bean id="rabbitConnFactory"
        class="org.springframework.amqp.rabbit.connection.SingleConnectionFactory">
        <constructor-arg><value>localhost</value></constructor-arg>
        <property name="username" value="guest" />
        <property name="password" value="guest" />
        <property name="virtualHost" value="/" />
        <property name="port" value="5672" />
</bean>

<!-- Spring AMQP Template -->
<bean id="template"
        class="org.springframework.amqp.rabbit.core.RabbitTemplate">
        <property name="connectionFactory" ref="rabbitConnFactory" />
        <property name="routingKey" value="test.queue"/>
        <property name="queue" value="test.queue"/>
</bean>

<!-- Spring AMQP Admin -->
<bean id="admin" class="org.springframework.amqp.rabbit.core.RabbitAdmin">
        <constructor-arg ref="rabbitConnFactory" />
</bean>
```

# AMQP Vendor RabbitMQ

## AMQP Template

La fábrica de conexiones básicamente debe configurarse con los parámetros de conexión TCP / IP para ubicar el intermediario RabbitMQ. Usamos el puerto predeterminado 5672 y las credenciales guest / guest.

La plantilla está configurada para usar una cola llamada test.queue.

Nuestro ejemplo utiliza el autowiring ya que configuramos exactamente una implementación.

# AMQP Vendor RabbitMQ

## AMQP Template

AmqpAdmin y AmqpTemplate se inyectan así:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class RabbitMQClientTest {

        @Autowired private AmqpAdmin admin;
        @Autowired private AmqpTemplate template;

        @Test public void simpleProducerConsumerTest() {
                try {
                        String sent = "Catch the rabbit! " + new Date();
                        admin.declareQueue( new Queue("test.queue") );

                        // write message
                        template.convertAndSend( sent );
                        // read message
                        String received = (String)template.receiveAndConvert(

                        System.out.println( "Msg: " + received );
                        Assert.assertEquals( sent, received );

                } catch (AmqpException e) {
                        Assert.fail( "Test failed: " + e.getLocalizedMessage(
                }
        }
}
```

# AMQP Vendor RabbitMQ

## AMQP Template

Primero usamos AmqpAdmin para declarar la cola test.queue. Esta operación es idempotente, es decir, la cola se crea solo si no existe.

Después de eso, ***convertAndSend*** *(...)* se puede usar para enviar fácilmente cualquier objeto a través del cable. Dado que la carga útil de mensajes de AMQP es básicamente una matriz de bytes, AmqpTemplate realiza una conversión oculta, siempre y cuando no configure su MessageConverter personalizado. Para nuestros propósitos, la conversión estándar es suficiente, porque tanto el productor como el consumidor de mensajes son Java puro.

# AMQP Vendor RabbitMQ

## AMQP Template

Finalmente, utilizamos ***receiveAndConvert(...)*** para realizar una lectura sincrónica en la cola e imprimir la representación de cadena del mensaje.

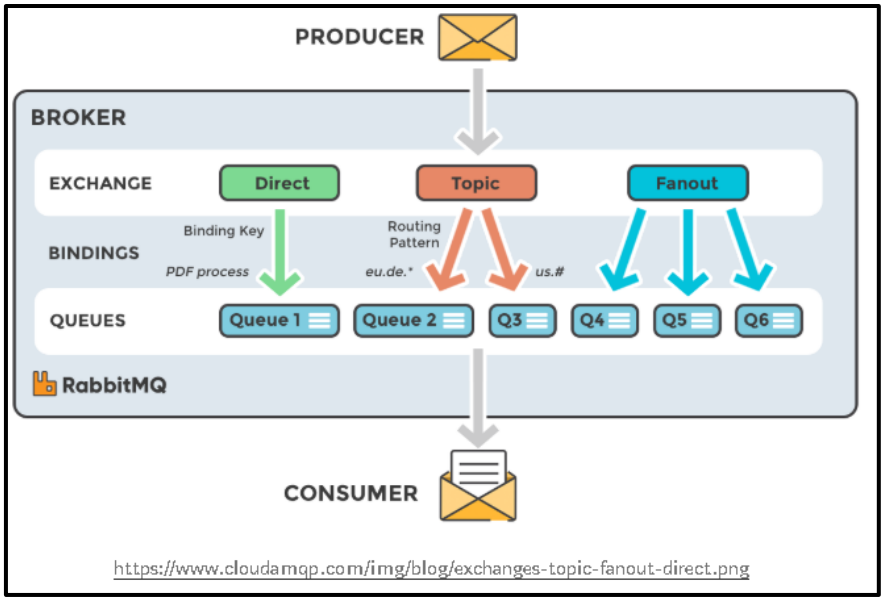
La AmqpException es una RuntimeException, por lo que no sería necesario hacerle catch.

Por lo tanto hemos usado AmqpTemplate para actuar como productor y consumidor de mensajes.

# AMQP Vendor RabbitMQ

## Rabbit MQ Broker

RabbitMQ es una parte de Message Broker que implementó el Protocolo Avanzado de Cola de Mensajes (AMQP), que ayuda a que su aplicación se comunique entre sí, cuando extiende su escala de aplicación.



# AMQP Vendor RabbitMQ

**Rabbit MQ Broker**

Existen dos formas de comunicarse entre microservicios. Son Punto a Punto (P2P) / Comunicación Sincrónica, y Publicar-Suscribir (Pub-Sub) / Comunicación Asincrónica. ¿Cuáles son las diferencias entre esos dos?

P2P, ofrece comunicación sincrónica, se sabe que una aplicación se comunicará directamente con otra aplicación, utilizando el protocolo HTTP, y es la aplicación que requiere una respuesta inmediata directamente del servidor.

Pub-Sub, ofrece comunicación asincrónica, se sabe que no requiere una respuesta inmediata del servidor, y el mensaje enviado se colocará en una cola de mensajes (o conocida como Cola de eventos en el sistema de mensajería empresarial).

# AMQP Vendor RabbitMQ

**Rabbit MQ Broker**

RabbitMQ también es llamado un middleware basado en Erlang, ya que puede ser tanto micro-servicios como una aplicación. RabbitMQ admite múltiples protocolos.

Los protocolos que admite RabbitMQ:

- AMQP
- HTTP
- STOMP
- MQTT

# AMQP Vendor RabbitMQ

**Como funciona**

Exchange es un algoritmo que decide qué cola almacenará el mensaje. (obtener mensaje del productor, incluir en la cola del consumidor). Cada consumidor obtiene su propia Cola basada en la lógica que usa, hay 4 tipos de lógica que puede usar en Exchange:

- **Intercambio directo**: será directo a la cola en función de una clave de enrutamiento de mensajes

- **Fanout Exchange**: publicará en todas las colas que tengan la misma clave de enrutamiento

# AMQP Vendor RabbitMQ

## Como funciona

- **Intercambio de temas**: se publicará en todas las colas que tengan la misma clave de enrutamiento y el mismo patrón de enrutamiento especificado en el enlace
- **Intercambio de encabezados**: encabezado significa encabezado al enviar un archivo http, como cuando envía Imagen, el encabezado es "imagen / *"

# AMQP Vendor RabbitMQ

## Porque debemos usarlo

## Desacoplamiento

Desacoplar es separar los componentes principales de la aplicación. Esto es lo que quería cualquier aplicación que implementara microservicios. Porque su aplicación será mantenible y mejorará la calidad del Principio de responsabilidad única.

## Flexibilidad

Debido a que la aplicación se ha desacoplado, la aplicación será lo suficientemente flexible para desarrollarse en la siguiente fase. Pero no solo es tan flexible, porque si está utilizando RabbitMQ, podrá conectar 2 aplicaciones / servicios diferentes que están escritos por diferentes aplicaciones, estas aplicaciones se hablarán entre sí con la ayuda de un "traductor" que es MOM.

# AMQP Vendor RabbitMQ

**Porque debemos usarlo**

Otro beneficio de usar RabbitMQ:

- Cola altamente disponible
- Protocolo múltiple
- Muchos clientes
- Agrupación
- IU de gestión
- Rastreo (el uso del tablero puede rastrear el soporte)
- Sistema de complementos (Extienda la funcionalidad del intermediario principal en una variedad de formas)

# AMQP Vendor Azure Service Bus

**Concepto**

Azure proporciona una plataforma de mensajería asincrónica denominada Azure Service Bus ("Service Bus") que se basa en el estándar Advanced Message Queueing Protocol 1.0 (AMQP 1.0).

Service Bus puede usarse en el intervalo de plataformas de Azure admitidas.

Azure Service Bus es un servicio de mensajería en la nube que se usa para conectar aplicaciones, dispositivos y servicios que se ejecutan en la nube a otras aplicaciones o servicios. Como resultado, actúa como una red troncal de mensajería para aplicaciones disponibles en la nube o en cualquier dispositivo.

# AMQP Vendor Azure Service Bus

## Concepto

Un servicio confiable de mensajería en la nube (MaaS) *. Es un sistema de mensajería en la nube para conectar aplicaciones y dispositivos a nubes públicas y privadas. Si uno o más están fuera de línea, puede contar con ellos si necesita un servicio confiable de mensajería en la nube entre la aplicación y el servicio.

Microsoft Azure Service Bus es un agente de mensajes de integración empresarial completamente administrado. Service Bus puede desacoplar aplicaciones y servicios. Service Bus ofrece una plataforma confiable y segura para la transferencia asincrónica de datos y estado.

# AMQP Vendor Azure Service Bus

## Concepto

Los datos se transfieren entre distintas aplicaciones y servicios mediante *mensajes*. Un mensaje está en formato binario y puede contener JSON, XML o simplemente texto.

Algunos escenarios de mensajería comunes son:

*Mensajería*. Transferir datos empresariales, como pedidos de ventas o compras, diarios o movimientos de inventario.

*Desacoplar aplicaciones*. Mejore la confiabilidad y la escalabilidad de las aplicaciones y los servicios. El cliente y el servicio no tienen que estar en línea al mismo tiempo.

*Temas y suscripciones*. Habilite 1:*n* relaciones entre publicadores y suscriptores.

*Sesiones de mensajes*. Implemente flujos de trabajo que requieran la ordenación de mensajes o el aplazamiento de mensajes.

# AMQP Vendor Azure Service Bus

## Concepto

Los datos se transfieren entre distintas aplicaciones y servicios mediante *mensajes*. Un mensaje está en formato binario y puede contener JSON, XML o simplemente texto.

Tanto el servicio en la nube de Azure Service Bus como el Service Bus para Windows Server (Service Bus 1.1) local, admiten el protocolo AMQP (Advanced Message Queueing Protocol) 1.0. AMQP le permite construir aplicaciones híbridas, entre plataformas, utilizando un protocolo estándar abierto. Puede construir aplicaciones mediante componentes creados con distintos lenguajes y marcos, y que se ejecutan en diferentes sistemas operativos. Todos estos componentes se pueden conectar a Service Bus e intercambiar directamente mensajes empresariales estructurados de manera eficaz y con total fidelidad.

# AMQP Vendor Azure Service Bus

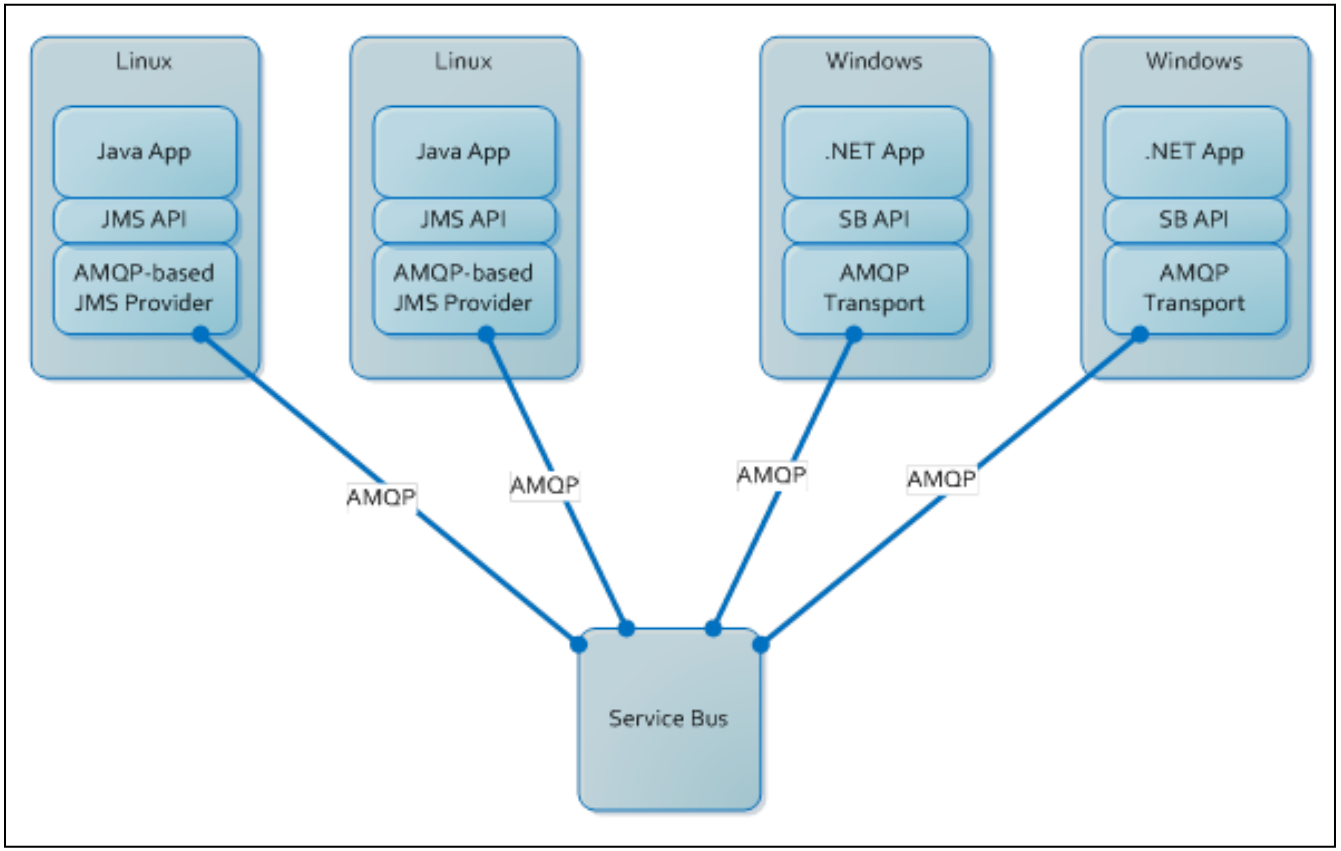
## Concepto

La compatibilidad con AMQP 1.0 en Azure Service Bus implica que ahora puede aprovechar sus características de encolamiento de Service Bus y de la publicación/suscripción de mensajería asíncrona desde una amplia variedad de plataformas mediante un eficaz protocolo binario. Además, puede desarrollar aplicaciones formadas por componentes creados con una mezcla de lenguajes, marcos y sistemas operativos.

En la siguiente ilustración se muestra una implementación de ejemplo en el que clientes de Java que se ejecutan en Linux, escritos usando la API estándar Java Message Service (JMS), y clientes .NET que se ejecutan en Windows, intercambian mensajes a través de Service Bus mediante AMQP 1.0.

# AMQP Vendor Azure Service Bus

Implementación de mensajes entre Plataformas mediante Service Bus y AMQP 1.0



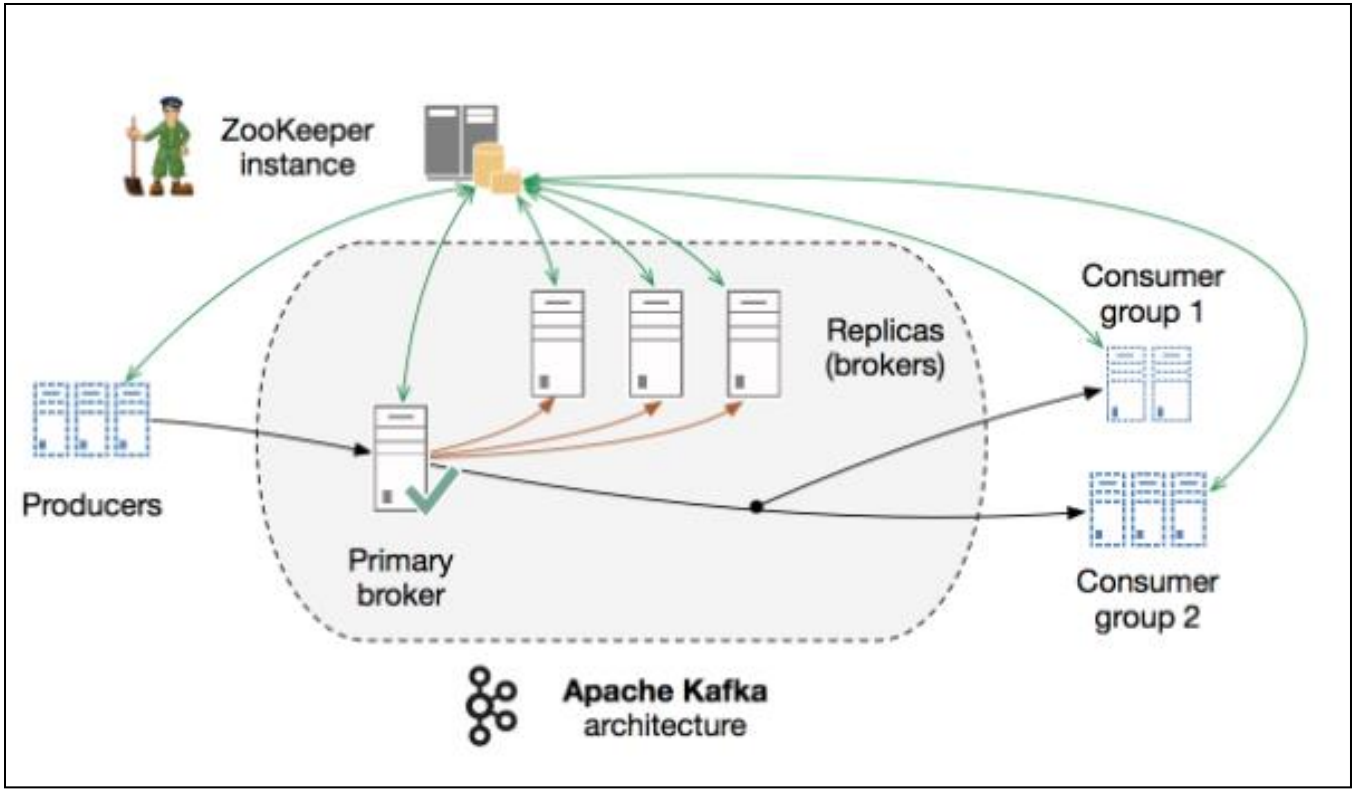
# Apache Kafka

## Concepto

Apache Kafka es un proyecto de intermediación de mensajes de código abierto desarrollado por LinkedIn y donado a la Apache Software Foundation escrito en Java y Scala. El proyecto tiene como objetivo proporcionar una plataforma unificada, de alto rendimiento y de baja latencia para la manipulación en tiempo real de fuentes de datos. Puede verse como una cola de mensajes, bajo el patrón publicación-suscripción, masivamente escalable concebida como un registro de transacciones distribuidas, lo que la vuelve atractiva para las infraestructuras de aplicaciones empresariales.

El diseño tiene gran influencia de los registros de transacción

# Apache Kafka Arquitectura



# Apache Kafka

## Concepto

Kafka no es solo un broker, es una plataforma de transmisión y hay muchas herramientas disponibles que son fáciles de integrar con Kafka fuera de la distribución principal. El ecosistema de Kafka consta de Kafka Core, Kafka Streams, Kafka Connect, Kafka REST Proxy y el Registro de esquemas. Tenga en cuenta que la mayoría de las herramientas adicionales del ecosistema Kafka provienen de Confluent y no son parte de Apache.

Lo bueno de todas estas herramientas es que puede configurar un sistema enorme antes de que necesite escribir una sola línea de código.

# Apache Kafka

## Concepto

Kafka Connect le permite integrar otros sistemas con Kafka. Puede agregar una fuente de datos que le permita consumir datos de esa fuente y almacenarlos en Kafka, o al revés, y tener todos los datos de un tema enviados a otro sistema para su procesamiento o almacenamiento. Hay muchas posibilidades con el uso de Kafka Connect, y es fácil comenzar ya que ya hay muchos conectores disponibles.

El Proxy REST de Kafka le brinda la oportunidad de recibir metadatos de un clúster y producir y consumir mensajes a través de una API REST simple. Esta característica se puede habilitar fácilmente desde el Panel de control para su clúster.

# AMQP Vendor Apache Kafka

**Concepto**

AMQP es un protocolo, mientras que Kafka es un sistema de mensajería con su propio protocolo. La forma en que funcionan ambos protocolos es fundamentalmente diferente. AMQP se centra en la entrega de mensajes discretos (publicación y entrega transaccional, enrutamiento, seguridad, etc.), donde Kafka enfatiza el procesamiento por lotes y tiene un estilo de redundancia completamente diferente (es efectivamente un registro distribuido).

Podrías unir las dos tecnologías usando middleware personalizado (imagino que sería trivial con algo como Apache Camel), pero desarrollaron Kafka porque ningún otro agente de mensajes cumplió con su uso casos.

# Lecturas adicionales

Para obtener información adicional, puede consultar los siguientes enlaces:

https://docs.spring.io/spring-amqp/docs/1.4.5.RELEASE/reference/html/amqp.html
https://docs.spring.io/spring-amqp/docs/2.3.0-SNAPSHOT/reference/html/#resources
https://docs.microsoft.com/es-es/azure/service-bus-messaging/service-bus-amqp-overview
https://docs.microsoft.com/es-es/azure/developer/java/spring-framework/configure-spring-cloud-stream-binder-java-app-with-service-bus

# Resumen

En este capítulo, usted aprendió:

- AMQP
- AMQP vs JMS
- JMS Implementations
- amqp-template
- AMQP vendors:
  1. Rabbit MQ Broker
  2. Azure Service Bus
  3. Kafka