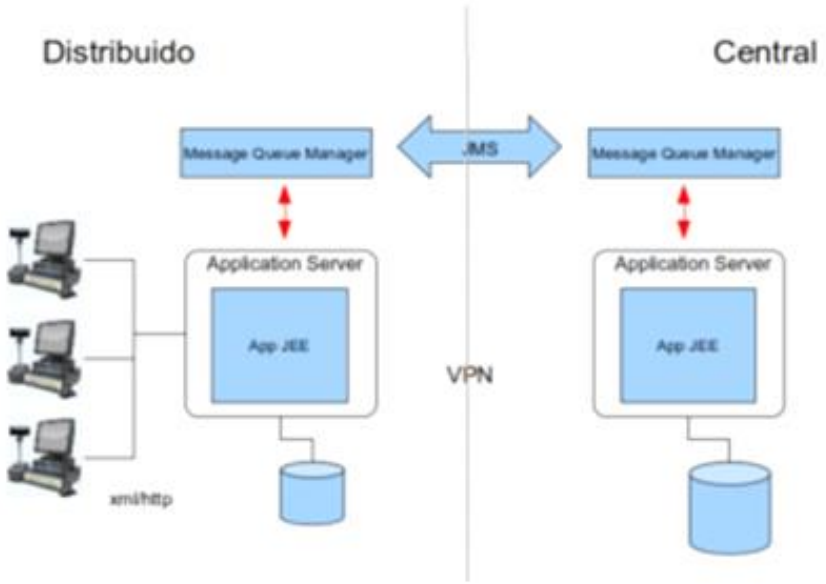
MTIS

Message Oriented Middleware

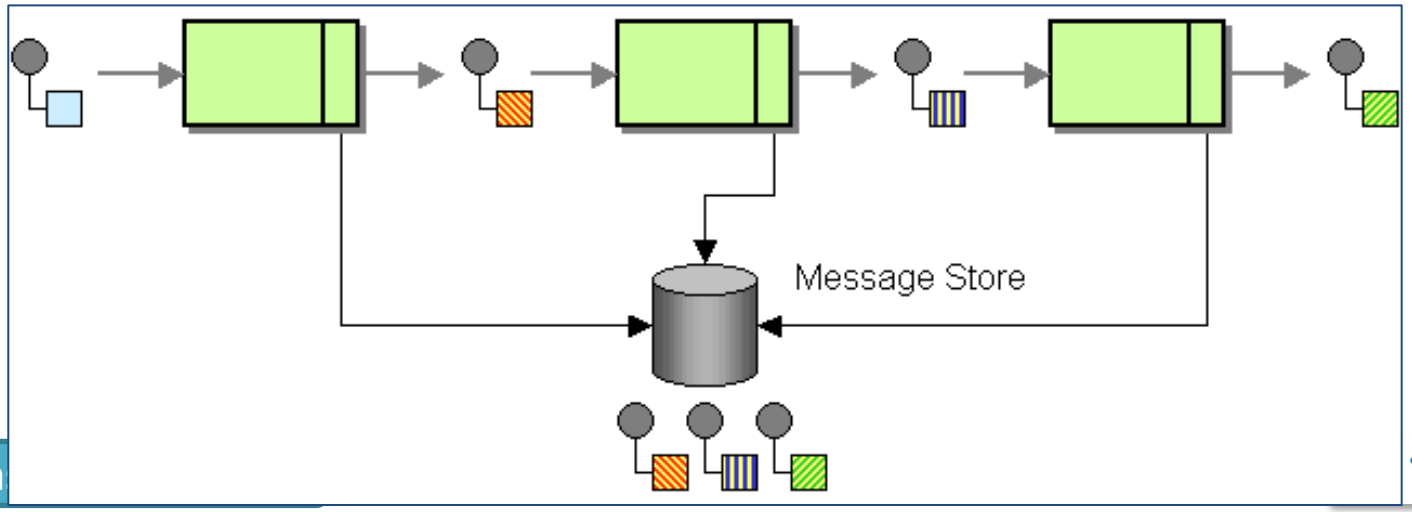
Message Oriented Middleware

- Paradigma de computación distribuida
- Basado en el paso de mensajes
- Permite a aplicaciones distribuidas comunicar e intercambiar información a través del envío y recepción de mensajes.
- Los Propósitos principales:
  - **Desacoplar**: En tiempo, en espacio y a nivel de sincronización.
  - **Confiabilidad**: A nivel de transacción y de persistencia.
- Comunicación
  - Asíncrona
  - Desacoplada
  - B2B
- Implementaciones
  - JMS, MSMQ, MQSeries,…
- Tipos
  - Punto a punto → 1:1 (queue)
  - Publicación-Subscripción → 1:M (topic)

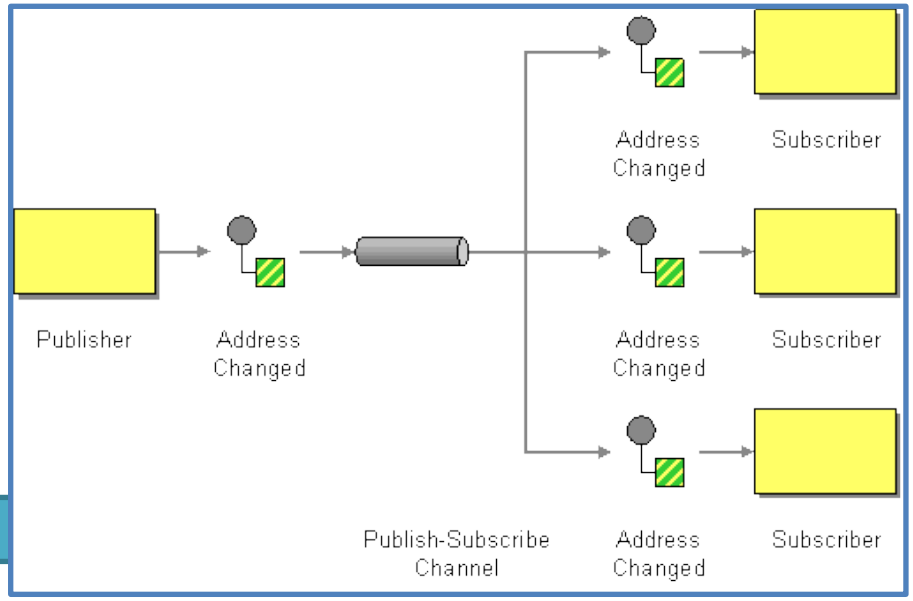


- **Beneficios**
  - **Comunicación Síncrona/Asíncrona/Colas**:
    - Hacer una solicitud (enviar un mensaje) y que te respondan después de cierto tiempo.
  - **Persistencia de mensajes**:
    - Hacer una solicitud y que la petición siga viva a pesar de que los sistemas *no hayan estado operativos* después de haber realizado la solicitud,
    - Almacenar de manera persistente los mensajes de *las solicitudes excedentes* efectuadas a un servicio o simplemente, almacenar con gran rapidez y garantía los mensajes de las solicitudes realizadas.
  - **Publicación/Suscripción**:
    - Publicar un mensaje de cierto tópico y que sólo ciertos clientes reciban el mensaje.

- **Casos de uso**
  - Los más representativos están definidos en los Enterprise Integration Patterns.
    - **Message Store EIP**
      - http://www.eaipatterns.com/MessageStore.html
      - Nos permitirá dar persistencia a los mensajes (Reliability), Colas (Queues), etc.



- **Casos de uso**
  - Los más representativos están definidos en los Enterprise Integration Patterns.
    - **Publish-Subscribe Channel**
      - http://www.eaipatterns.com/PublishSubscribeChannel.html
      - Para realizar Monitoring, Tracking, Broadcasting, etc.



- Dependiendo del tipo de mensaje y del tipo de clientes existen muchos tipos de Message Brokers, algunos por ejemplo:
  - Sensores (clientes MB)
  - Monitorización (cliente suscriptor MB)
  - Vigilancia (cliente suscriptor MB)
  - Business Activity Monitoring (BAM)
  - Scheduling Systems
  - Redes Sociales
- Actualmente su uso es muy frecuente en sectores inimaginables, sobre todo en los ***Sistemas de Información Críticos***,
  - "*donde fallar no es una opción*".
- Los escenarios no son los tradicionales, son escenarios relacionados con ***Internet of Things*** y comunicación ***Machine-to-Machine*** .

- **ActiveMQ** (http://activemq.apache.org): proyecto de Apache, implementa muchos Enterprise Integration Patterns (EIP) a través de Apache Camel (http://camel.apache.org). Implementa una variedad de protocolos como JMS y MQTT (machine-to-machine / "Internet of Things" connectivity binary protocol – http://mqtt.org).
- **WSO2 Message Broker**: basado en Apache Qpid (http://qpid.apache.org) e implementa el "Advanced Message Queueing Protocol" (AMQP – http://www.amqp.org) y otros estándares relacionados como JMS y WS-Eventing. Es rápido, potente y muy ligero, la persistencia de mensaje se logra con el uso de Apache Cassandra y en combinación con Apache Zookeper nos permite coordinar colas distribuidas.
- **MSMQ** : sistema MOM implementado por Microsoft, https://msdn.microsoft.com/en-us/library/ms711472(v=vs.85).aspx.

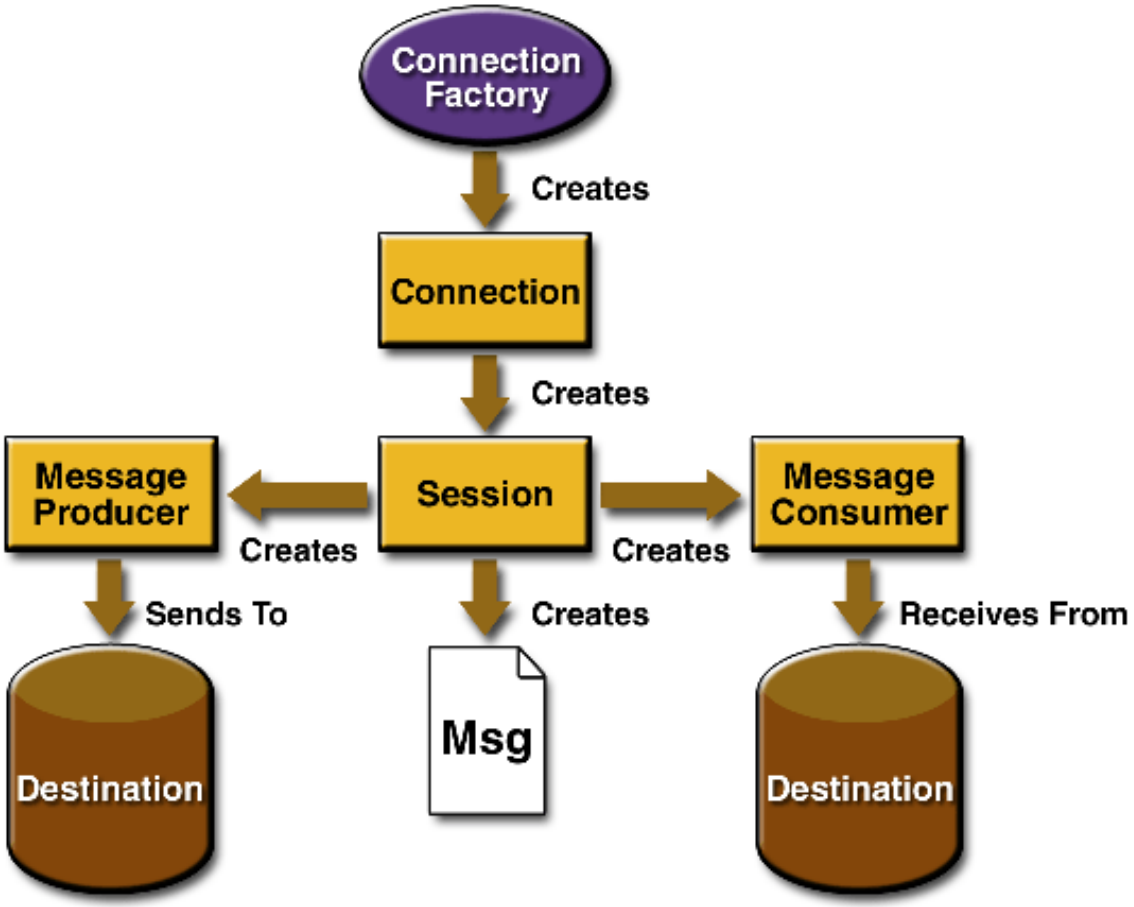
- **Rabbit MQ** (http://www.rabbitmq.com): a diferencia de los anteriores implementa muchos protocolos a nivel de transporte y mensaje, entre ellos AMQP, STOMP (text-based messaging protocol emphasising simplicity – http://stomp.github.io), MQTT y HTTP.
- **Mosquitto** (http://mosquitto.org): Message Broker para el protocolo MQTT muy potente y veloz, muy usado en IoT / M2M.
- **Paho** (http://eclipse.org/paho): altamente escalable, optimizado para equipos de reducida potencia y para redes muy restringidas, junto a Mosquitto son los brokers más usados en IoT / M2M, ambos poseen muchos clientes desde C++ a JavaScript, como tal implementa también el protocolo MQTT.
- **ActiveMQ Apollo** (http://activemq.apache.org/apollo): es la nueva generación de Apache ActiveMQ que soporte protocolos como STOMP, AMQP, MQTT, Openwire, SSL y WebSockets. Muy bien valorado y con unas prestaciones muy interesantes

- Implementación del paradigma MOM de código abierto de la fundación Apache
  - http://activemq.apache.org
- Orientado a la interoperabilidad de plataformas y aplicaciones
- Implementa la especificación JMS
- Modos de despliegue
  - Standalone
  - Dentro de servidores de aplicaciones
  - Dentro de otros procesos
- Buena integración con otros productos
- Arquitectura básica fundamentada sobre JMS

- Java Message Service
  - Especificación creada por SUN
  - Es el API de Java orientado a MOM
  - Incluido en JEE y disponible de forma independiente
  - JMS = JDBC

- Conceptos JMS
  - Clientes JMS
  - Clientes no JMS
  - Productor JMS
  - Consumidor JMS
  - Proveedor JMS
  - Mensaje JMS
  - Objetos administrados
    - ConnectionFactory
    - Destinos JMS
      - Queue
      - Topic
  - Dominio JMS
    - Punto a Punto
    - Publicación/Subscripción

- Modelo de programación del API JMS



- Modelo de programación del API JMS

Generica	Point-to-Point	Publish-Suscribe
Destination	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Session	QueueSession	TopicSession
MessageConsumer	QueueReceiver	TopicSuscriber
MessageProducer	QueueSender	TopicPublisher

- Clientes JMS
  - Utiliza el API JMS para conectar a los proveedores JMS
  - Interfaces
    - MessageProducer
      - Productor de mensajes
        - » MessageProducer.send()
    - MessageConsumer
      - Consumidor de mensajes
        - » MessageConsumer
          - receive()
          - MessageListener.onMessage()

- Clientes no JMS
  - Utiliza un API de cliente nativo ofrecido por el proveedor
    - CORBA-IIOP
    - Java RMI
    - Servicios Web
      - Rest
      - SOAP

- Mensajes JMS
  - Principal elemento en JMS
    - Cómo se transmiten datos de negocio y eventos
    - Diferentes tipos de mensajes
    - Diferentes formatos
      - Texto, binario, etc.
  - Dos partes
    - Cabecera (header)
      - El API provee métodos para manipular las cabeceras
      - Algunas cabeceras son asignadas automáticamente
    - Datos transmitidos o contenido (payload)
      - 6 tipos de mensajes
        - » Message
        - » TextMessage
        - » MapMessage
        - » BytesMessage
        - » StreamMessage
        - » ObjectMessage

- Mensajes JMS
  - Cabeceras
    - JMSDestination
      - Indica el destino al cual el mensaje será enviado
      - Importante para clientes que consumen de más de un destino
    - JMSDeliveryMode
      - Dos modos de mensajes
        - » El mensaje debe ser entregado una única vez
        - » Afecta al rendimiento
        - » **Persistente**
          - Mantiene el mensaje si el servidor MOM falla
        - » **No persistente**
          - Si el servidor falla se pierde el mensaje
        - » El modo se indica en el productor para todos los mensajes que envíe
        - » Puede ser personalizado para cada mensaje de forma individual

- Mensajes JMS
  - Cabeceras
    - JMSExpiration
      - Tiempo en el que un mensaje tiene validez
      - Por defecto no expira (valor = 0)
      - MessageProducer.setTimeToalive(…)
        - » Establece el tiempo de expiración para todos los mensajes del productor
      - MessageProducer.send(…)
        - » Puede establecer el tiempo para mensajes concretos
    - JMSMessageID
      - Cadena para identificar mensajes
      - Uso
        - » Identificar mensajes para ver cómo se procesa
        - » Historial de mensajes y estadísticas
        - » Para evitar sobrecarga desde el productor se puede indicar que se deshabilite si el proveedor lo soporta
          - MessageProducer.setDisableMessageID()

- Mensajes JMS
  - Cabeceras
    - JMSPriority
      - Establece la prioridad de los mensajes
      - Establecida en el productor para todos los mensajes
      - Puede ser sobrescrita para mensajes individuales
      - Rango de 0 a 9 → menor a mayor prioridad
    - JMSTimestamp
      - Marca de tiempo que refleja cuándo el mensaje fue enviado desde el productor al proveedor MOM
      - Se puede deshabilitar para evitar sobrecarga si el proveedor lo soporta
        - » MessageProducer.setDisableMessageTimestamp()
    - Opcionales por el cliente
      - JMSCorrelationID: Permite establecer relaciones entre mensajes
      - JMSReplyTo: Usado para especificar un destino para la respuesta a un mensaje.
        - » El consumidor determinará si lo envía o no.
      - JMSType: Permite identificar el tipo de mensaje a nivel semántico

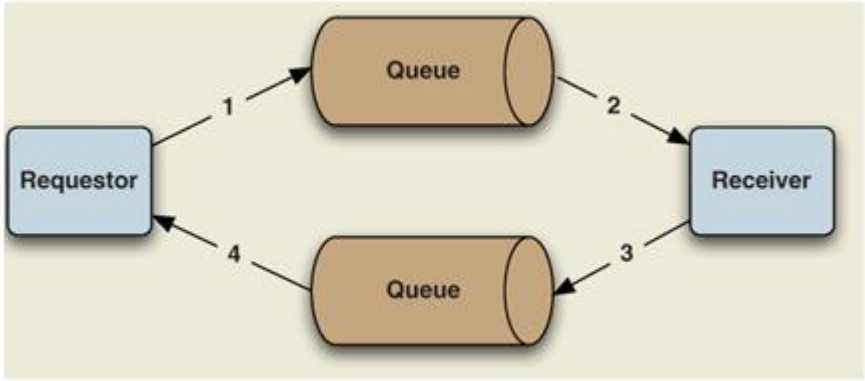
- Mensajes JMS
  - Cabeceras
    - Opcionales por el proveedor
      - JMSRedelivered: Para indicar la consistencia de que un mensaje ha sido entregado a un consumidor pero no confirmado por éste
        - » Cuando un consumidor falla al confirmar la entrega o el proveedor no ha recibido confirmación
    - Propiedades
      - Cabeceras propietarias opcionales
      - Creadas por el propio desarrollador
        - » textMessage.setStringProperty("Organizacion", "UA");
  - Filtros (Message Selectors)
    - Permite usar propiedades de cabecera para discriminar mensajes
    - Expresiones basadas en un subconjunto de SQL92
      - Condiciones de selección de mensajes

```
...
String selector = "SYMBOL = 'AAPL' AND PRICE > "
    + getPreviousPrice();

MessageConsumer consumer =
    session.createConsumer(destination, selector);
...
```

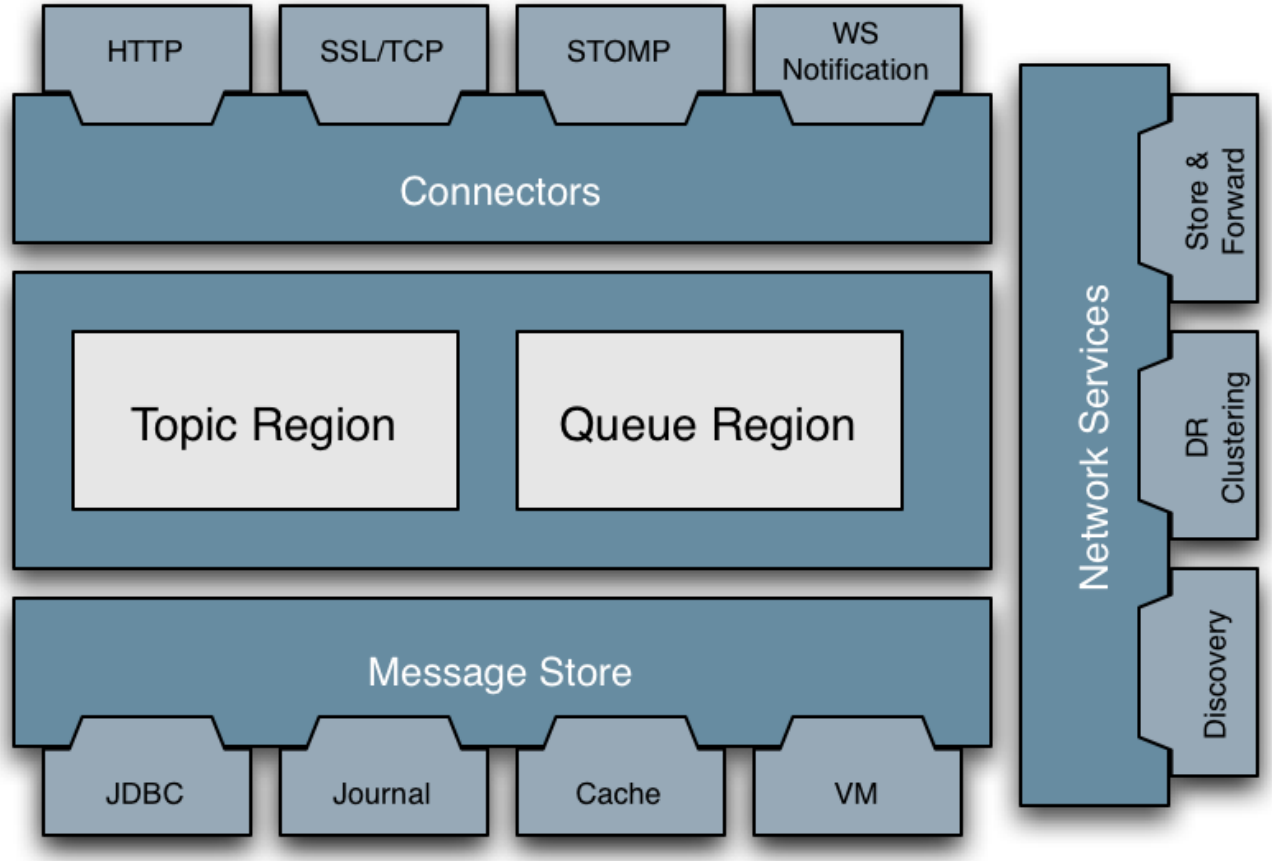
Item	Values
Literals	Booleans TRUE/FALSE; numbers such as 5, -10, +34; numbers with decimal or scientific notation such as 43.3E7, +10.5239
Identifiers	A header or property field
Operators	AND, OR, LIKE, BETWEEN, =, <>, <, >, <=, =>, +, -, *, /, IS NULL, IS NOT NULL

- Dominios JMS
  - Punto-a-Punto
    - Queues
  - Publicación/Subscripción
    - Topics
    - Subscripción duradera
      - Mantienen el estado de subscripción (subscripción infinita)
      - El consumidor puede desconectarse pero cuando vuelva a conectarse recibirá todos los mensajes enviados el proveedor durante el tiempo de inactividad
    - Subscripción no duradera
      - tanto el subscriptor como la subscripción comienzan y finalizan al mismo tiempo
      - Su vida es idéntica.
      - Al cerrar un subscriptor, la subscripción finaliza.
- Modos de consumo de mensajes
  - Modo síncrono
    - MessageConsumer.receive()
  - Modo asíncrono
    - Registrando un MessageListener
    - MessageConsumer.setMessageListener(…)

- Comunicación síncrona básica
  - TopicRequestor
  - QueueRequestor
  - Patrón básico
    - Método request()
    - Espera a la respuesta de su petición
    - Usa la cabecera JMSReplyTo
    - Crea intermediarios temporales



- Arquitectura



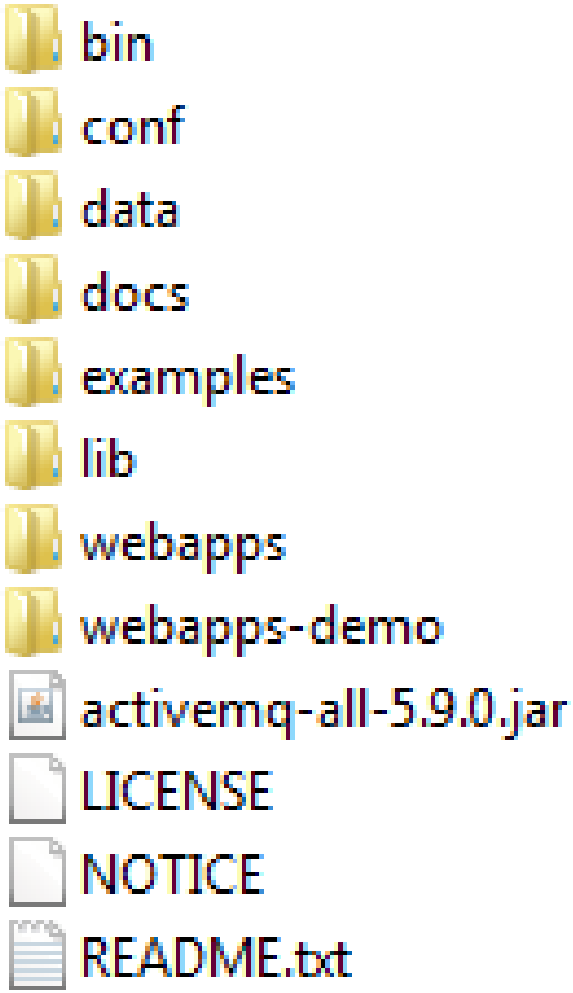
- Para que un manejador de mensajes pueda ser proveedor JMS debe de soportar los modelos de mensajería **publicación/suscripción** y **punto a punto**.
- ActiveMQ para soportar estos modelos se define los conceptos de Queue y Topic:
  - **Queue:**
    - Es el tipo de destino o canal virtual por el que tenemos que enviar un mensaje para seguir el modelo punto a punto.
    - Solamente un receptor de los suscritos a ese canal recibirá el mensaje.
  - **Topic:**
    - Es el tipo de destino o canal virtual por el que tenemos que enviar el mensaje para seguir el modelo publicación/suscripción.
    - Todos los clientes suscritos a ese canal recibirán el mensaje.

- Conectores
  - Mecanismos de conectividad entre los participantes
    - Client-Broker
      - <transportConnectors>
    - Broker-Broker
      - <networkConnectors>
  - Requerimientos de conectividad
    - Rendimiento, seguridad, interoperabilidad,…
  - URI
    - Base para direccionar conexiones a brokers
    - Ejemplo, tcp://localhost:61616

```
<!-- The transport connectors ActiveMQ will listen to -->
<transportConnectors>
   <transportConnector name="openwire" uri="tcp://localhost:61616"
     discoveryUri="multicast://default"/>
   <transportConnector name="ssl"     uri="ssl://localhost:61617"/>
   <transportConnector name="stomp"   uri="stomp://localhost:61613"/>
   <transportConnector name="xmpp"    uri="xmpp://localhost:61222"/>
</transportConnectors>
```

- Conectores de transporte
  - Client-Broker
  - Protocolos de red
    - TCP, NIO, UDP, SSL, HTTP/HTTPS, VM
  - Protocolos de aplicación soportados
    - Wire format
    - OpenWire, Rest, Stomp, WS-Notification, AMQP, …
- Conectores de red
  - Permite establecer redes de brokers
  - Propiedades
    - Balanceo de carga
    - Alta disponibilidad
    - Tolerancia a fallos
  - Static (IP), failover, multicast, zeroconf, peer, fanout, discovery

- Primeros pasos
  - http://activemq.apache.org/getting-started.html
  - Descargar ActiveMQ
    - http://activemq.apache.org/download.html
  - Instalación
  - Arranque y parada
    - …ActiveMQ\bin\winXX\activemq.bat
    - CTRL+C
  - Verificar la instalación
    - netstat -an|find "61616"
  - Monitorización
  - Configuración



- Scripts de arranque y parada
- Archivos de configuración
- Archivos de log y persistencia
- Consola Web para ActiveMQ

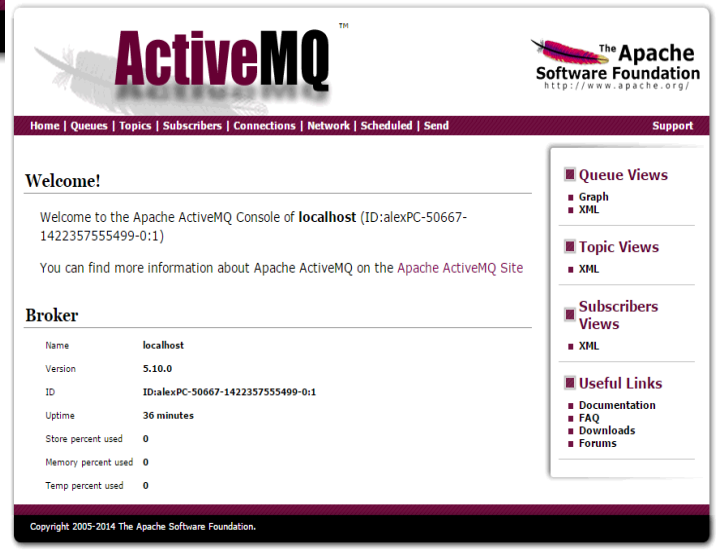
- Archivo principal
  - Activemq.xml
  - Al arrancar se puede indicar el archivo de configuración
- Elementos a configurar
  - Transport connectors
  - Network connectors
  - Discovery agents
  - Persistence providers and locations
  - Custom messages

- Creación de destinos
  - Por regla general son creados bajo demanda
    - Cuando los clientes comienzan a usarlas
  - Se pueden crear cuando arranque el servidor ActiveMQ
    - En el archivo de configuración Activemq.xml

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframewo
  http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-c
  http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring

  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer" />

  <broker xmlns="http://activemq.apache.org/schema/core">
    <destinations>
      <queue physicalName="FOO.BAR" />
      <topic physicalName="SOME.TOPIC" />
    </destinations>

  </broker>

</beans>
```

- Creación de conector de transporte
  - Nos permiten acceder al servidor ActiveMQ

```
<!-- The transport connectors ActiveMQ will listen to -->
<transportConnectors>
   <transportConnector name="openwire" uri="tcp://localhost:61616"
     discoveryUri="multicast://default"/>
   <transportConnector name="ssl"     uri="ssl://localhost:61617"/>
   <transportConnector name="stomp"   uri="stomp://localhost:61613"/>
   <transportConnector name="xmpp"    uri="xmpp://localhost:61222"/>
</transportConnectors>
```



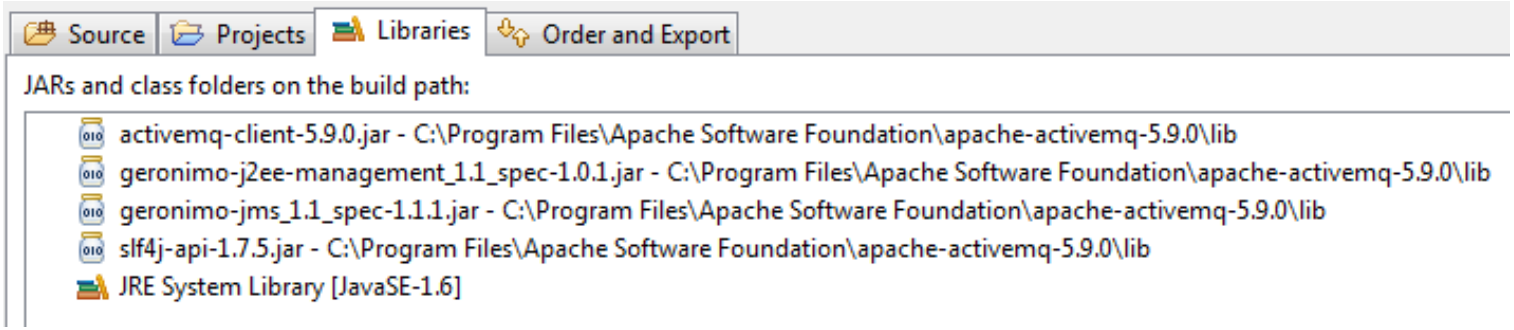
Consola Principal
http://localhost:8161/
Consola de administración
http://localhost:8161/admin/
User: admin
Password: admin

- **Almacenamiento de los mensajes por ActiveMQ**
  - El almacén de mensajes recomienda el uso de mensajes de uso general. Desde la versión 5.3 de ActiveMQ es KahaDB.
  - Se trata de un almacén de mensajes basado en archivos, que combina un diario de transacciones para el almacenamiento de mensajes muy confiable y una recuperación con un buen nivel de escalabilidad.
  - El almacén de mensaje de KahaDB utiliza un registro transaccional para sus índices y sólo utiliza un archivo de índice para todos sus destinos.
    - Se ha utilizado en entornos de producción con 10.000 conexiones activos

- **Seguridad en ActiveMQ**
  - ActiveMQ proporciona seguridad a través de diferentes proveedores.
  - Los proveedores más comunes son:
    - Autenticación JAAS.
    - Mediante mecanismo de autorización predeterminado que utiliza un simple archivo de configuración XML.
  - http://activemq.apache.org/security.html

- Lenguajes de cliente soportados
  - Java (JMS)
  - ActionScript3
  - Ajax
  - C y C++
  - C#
  - Delphi y Delphi/FreePascal
  - Erlang
  - Flash / ActionScript
  - Haskell
  - JavaScript (ajax o websocket)
  - Perl
  - PHP
  - Pike
  - Python
  - Ruby and Rails (ActiveMessaging)
  - Smalltalk
  - WebSocket
- http://activemq.apache.org/cross-language-clients.html

- Ejemplo:
  - Historiales clínicos
  - Modelo Publicación/Subscripción
    - Topic “historial”



```
// URL of the JMS server.
String url = "tcp://localhost:61616";
// Name of the topic we will be sending messages to
String subject = "historiales";

// Getting JMS connection from the server and starting it
ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(url);
Connection connection = connectionFactory.createConnection();
connection.start();

// JMS messages are sent and received using a Session. We will
// create here a non-transactional session object. If you want
// to use transactions you should set the first parameter to 'true'
Session session = connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);

// Destination represents here our topic 'historiales' on the
// JMS server. You don't have to do anything special on the
// server to create it, it will be created automatically.
Destination destination = session.createTopic(subject);
```

Message Oriented Middleware

```
// MessageProducer is used for sending messages (as opposed
// to MessageConsumer which is used for receiving them)
MessageProducer producer = session.createProducer(destination);

// We will send a small text message in XML format with the history of patient
TextMessage message = session.createTextMessage("<history><patient><name>Manolo Gar

// Here we are sending the message!
producer.send(message);
System.out.println("Sent message '" + message.getText() + "'");

connection.close();
```

```
// URL of the JMS server.
String url = "tcp://localhost:61616";
// Name of the topic we will receive messages from
String subject = "historiales";

try{
    // Getting JMS connection from the server
    ConnectionFactory connectionFactory
        = new ActiveMQConnectionFactory(url);
    Connection connection = connectionFactory.createConnection();
    connection.start();

    // Creating session for seding messages
    Session session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);

    // Getting the topic
    Destination destination = session.createTopic(subject);
```

```
// MessageConsumer is used for receiving (consuming) messages
MessageConsumer consumer = session.createConsumer(destination);

// Here we receive the message.
// By default this call is blocking, which means it will wait
// for a message to arrive on the topic.
Message message = consumer.receive();

// There are many types of Message and TextMessage
// is just one of them. Producer sent us a TextMessage
// so we must cast to it to get access to its .getText()
// method.
if (message instanceof TextMessage) {
    TextMessage textMessage = (TextMessage) message;
    System.out.println("Received message '"
        + textMessage.getText() + "'");
}
connection.close();
```

```
// URL of the JMS server.
String url = "tcp://localhost:61616";
// Name of the topic we will receive messages from
String subject = "historiales";
CountDownLatch latch = new CountDownLatch(1);
try{
    // Getting JMS connection from the server
    ConnectionFactory connectionFactory
        = new ActiveMQConnectionFactory(url);
    Connection connection = connectionFactory.createConnection();
    connection.start();

    // Creating session for seding messages
    Session session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);

    // Getting the topic
    Destination destination = session.createTopic(subject);

    // MessageConsumer is used for receiving (consuming) messages
    MessageConsumer consumer = session.createConsumer(destination);

    // Here we receive the message.
    // By default this call is blocking, which means it will wait
    // for a message to arrive on the topic.
    consumer.setMessageListener(new MTISSubscribeAsync());
    latch.await();
    consumer.close();
    connection.close();
```

```
@Override
public void onMessage(Message message) {
    try {
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            System.out.println("Received message '"
                + textMessage.getText() + "'");
        }
    } catch (JMSException e) {
        System.out.println("Got a JMS Exception!");
    }
}
```

- Bibliografía:
  - https://holisticsecurity.wordpress.com/2014/03/07/message-brokering-y-recoleccion-datos-big-data-wso2/
  - http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=ActiveMQ
  - http://activemq.apache.org/
  - https://mesmerismo.wordpress.com/2014/02/05/activemq-for-dummies/
  - ActiveMQ in Action: Bruce Snyder, Dejan Bosanac, and Rob Davies
    - MEAP Release: August 2008
    - Softbound print: December 2010 (est.) | 375 pages
    - ISBN: 1933988940