PRACTICA 4 CSD –APUNTES—

ARTEMIS

- Apache ActiveMQ Artemis es un **proveedor JMS**.

- Artemis lo componen un conjunto de librerías y aplicaciones desarrolladas en Java que se distribuye como un archivo comprimido. Para esta práctica, ofrecemos a los alumnos un script de instalación (install artemis-csd.sh) que facilita dicho proceso.

- Está disponible para su descarga en la página web de Apache ActiveMQ.

- Para evitar problemas de cuotas de espacio en disco, la distribución de Artemis utilizada está disponible en la carpeta asigDSIC, común a todos los equipos remotos de los usuarios.

- Se puede acceder a la consola de gestión de Artemis (Management Console), disponible en http://localhost:12501/console desde cualquier navegador. Le solicitará un usuario y contraseña. Deberá indicar admin (para Username) y admin (para Password).

INTALACIÓN ARTEMIS

1º CREAMPOS CARPETA EN HOME

```
cd
mkdir csdJMS
```

2º EJECUTAMOS SCRIPT DE INSTALACIÓN INDICANDO EL PUERTO Y EL DIRECTORIO EN EL QUE QUEREMOS QUE SE INSTALE EL BROKER DE ARTEMIS.

El puerto que le indiquemos será el que use el bróker, el siguiente disponible será para la consola Web de Artemis.

```
cd
cd asigDSIC/ETSINF/csd/jms/artemis/bin
bash install-artemis-csd.sh 12500 ~/csdJMS/bróker
```

ChatServerJMS

- Corresponde a la **aplicación servidor** (solo debe haber 1)

- No sería necesario disponer de un servidor ChatServerJMS, si los clientes fueran quienes solicitaran al broker Artemis la creación de sus propias colas y los temas de los canales ya estuvieran inicialmente creados.

- Sin embargo se incluye el servidor ChatServerJMS para que **se encargue de gestionar los usuarios conectados y los canales disponibles**.

    - De esta manera se evita que usuarios diferentes se conecten con nicks duplicados y podemos llevar el seguimiento de cuántos usuarios se conectan a cada canal.

    - Independencia del proveedor en la gestión de usuarios.

- **Se encarga de**:

    - Los **usuarios conectados**

    - Los **canales disponibles**

    - Podría servir para controlar la autenticación de los usuarios, aunque esto no se ha tenido en cuenta.

- Cuando se lanza por primera vez:

    - Crea 3 canales: Linux, Spain, Friends

    - **Solicita a Artemis la creación de una cola de tipo multicast** (en Artemis se considera como un tema, es decir, puede tener múltiples consumidores) **por cada canal creado**, a la que se suscribirá, para así poder recibir los mensajes LEAVE y poder mantener actualizada la lista de suscriptores al canal. Cada una de estas colas se denomina **channel-**NAME (siendo NAME el nombre del canal correspondiente).

    - **Solicita a Artemis la creación de la cola ChatServer, de tipo anycast** (equivalente al destino de tipo "queue" de JMS), a través de la cual **recibirá los mensajes de los usuarios.**

ChatClientJMS

- Corresponde a la **aplicación cliente** (uno por cada usuario conectado al sistema: cuantos se quiera tener)

- Utiliza una interfaz gráfica

- **Permite a los usuarios**:

    - **Conectarse a la aplicación de Chat**

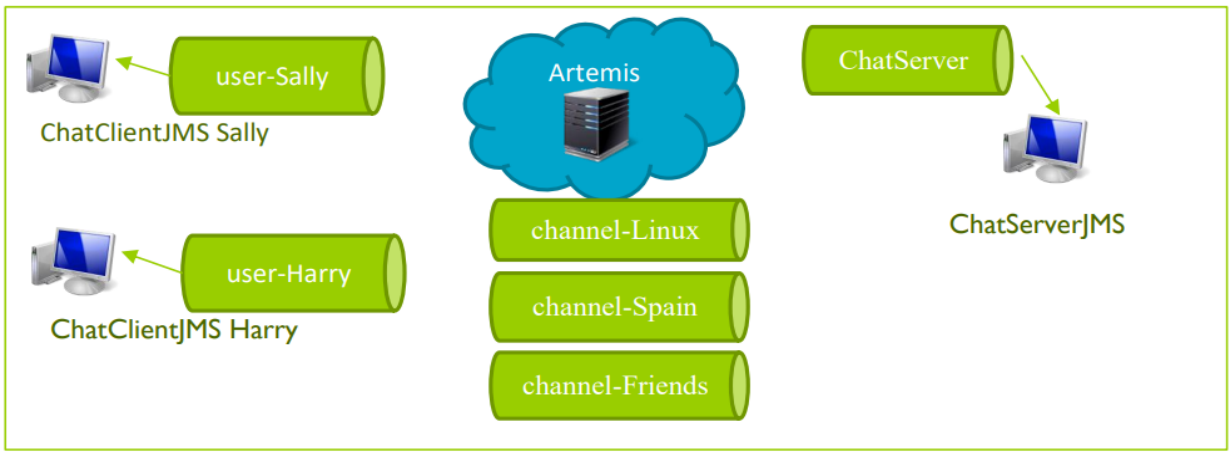
    - **Unirse a canales de chat**

    - **Enviar mensajes a los usuarios de un canal** (tanto mensajes públicos al canal como mensajes privados a otros usuarios)

- Cuando se conecte por primera vez:

    - **Nuestro servidor ChatServerJMS ordena a "artemis" que cree una cola para el usuario (de tipo anycast**, un único consumidor), con el nombre **user-USERNAME** (siendo USERNAME el nombre concreto del usuario). Dicha cola existirá de forma permanente, hasta que se reinstale Artemis.

- Es posible enviar mensajes a un usuario, aunque el usuario no esté conectado en ese momento y que los mensajes se entregan cuando el usuario se conecta posteriormente. **Esto es gracias a que los mensajes se guardan en las colas del proveedor JMS. La aplicación es persistente**



LANZAMIENTO A EJECUCIÓN

El orden de lanzamiento a ejecución se debe hacer en el orden que se muestra: Artemis -> ChatServerJMS -> ChatClientJMS.

Cada componente se debe lanzar de un terminal distinto.

Se pueden lanzar desde la misma máquina o desde distintas, ya que se trata de una aplicación distribuida.

ARTEMIS

cd

cd csdJMS/broker/bin

./artemis run (para lanzar el bróker en primer plano)

./artemis-service start (para lanzarlo en background)

se recomienda iniciarlo con el mandato "run", pues en caso de que se haya configurado de forma errónea el broker, nos mostrará mensajes sobre aquellos puntos de la configuración donde existen problemas. Y podremos así tratar de solucionarlos.

Además, se ha definido el usuario administrador (user "admin", password "admin") y se ha creado una cola llamada "csd". En nuestra aplicación de Chat no se ha hecho uso de ella. Pero nos permite mostrar cómo se pueden crear colas en el proceso de instalación de Artemis.

Para detenerlo : ./artemis-service stop

ChatServerJMS

cd

cd W/DistributedChatJMS

javac -cp lib/*:. *.java

java -cp lib/*:. ChatServerJMS url=localhost:12500

Se indica como parámetro de entrada la dirección donde está escuchando el broker Artermis.

ChatClientJMS

cd

cd W/DistributedChatJMS

javac -cp lib/*:. *.java

java -cp lib/*:. ChatClientJMS url=localhost:12500

Se indica como parámetro de entrada la dirección donde está escuchando el broker Artermis.

CHAT ROBOT

cd

cd W/DistributedChatJMS

javac -cp lib/*:. *.java

java -cp lib/*:. ChatRobot url=ldsic-vdi13:12500

Se indica como parámetro de entrada la dirección donde está escuchando el broker Artermis.

COMPONENTES DE JMS

PROOVEDOR JMS

- Sistema de mensajería encargado de implementar las interfaces JMS. Se encarga por tanto de la recepción y entrega de los mensajes a las aplicaciones cliente.

- Proporciona herramientas administrativas (para crear destinos de colas y temas, y factorías) y de control.

- Incluido en los servidores de la plataforma JAVA Enterprise Edition (JEE) y también como servidores independientes (solo JMS).

CLIENTE JMS

- Programa escrito en JAVA que hace uso de la API JMS.

- Produce o consume mensajes, relacionándose así con el proveedor.

MENSAJES

- Objetos que comunican información entre clientes.

- Estructura:

    - Cabecera: Campos fijos determinados por JMS.
    - Propiedades: Extensión de la cabecera. Campos definidos por el programa.
    - Cuerpo: Tipos: vacío, texto, bytes, objeto serializado…

FACTORÍAS DE CONEXIÓN

- Creadas mediante las herramientas administrativas del proveedor JMS

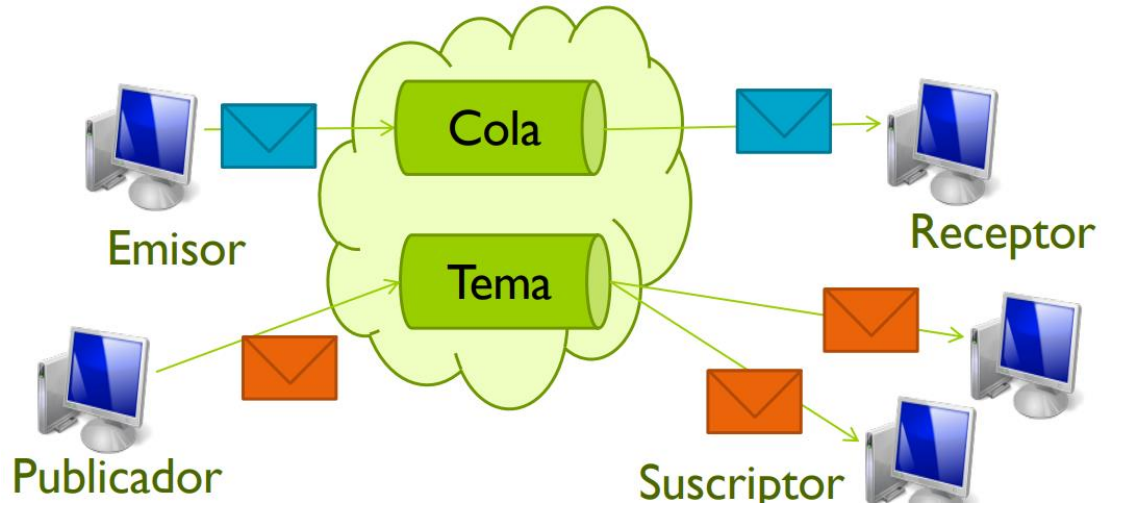
- Se utilizan para crear las conexiones de los clientes al sistema de mensajería.

DESTINOS

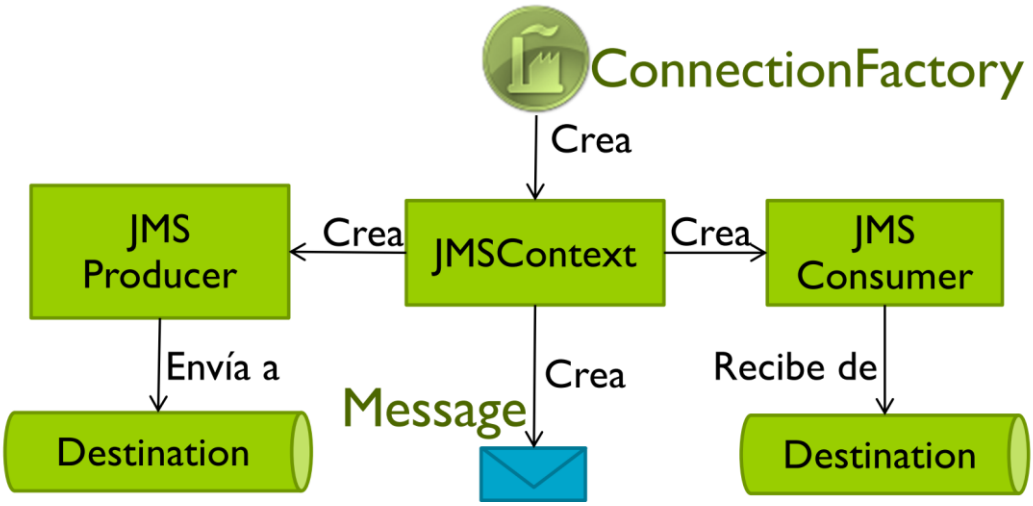
- Creados también por las herramientas administrativas del proveedor JMS

- Pueden ser de dos tipos:

    - Colas: Entrega a un solo cliente.
    - Temas: Entrega a múltiples clientes.



INTERFACES DEL MODELO DE PROGRAMACIÓN

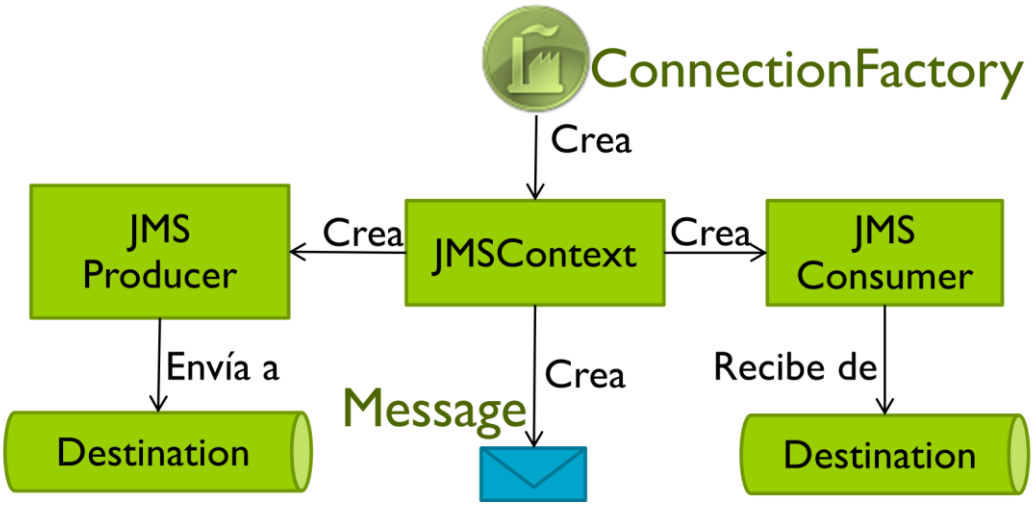


Los objetos que la implementa:

- **ConnectionFactory**
    - Vinculan la aplicación con un objeto administrado
    - Crean conexiones con el proveedor JMS

- **JMSContext**
    - Mantienen una conexión con el proveedor JMS
    - Son utilizables por un solo hilo de ejecución
    - Procesan envíos y recepciones de forma secuencial
    - Una aplicación puede usar varios objetos JMSContext si requiere procesamiento concurrente

- **JMSProducer**
    - Permiten enviar mensajes a colas y temas



- **JMSConsumer**
    - Permiten recibir mensajes de colas y temas

- **Destination**
    - Vinculan la aplicación con un objeto administrado
    - Encapsulan una dirección específica del proveedor JMS
    - Subinterfaces:
        - Queue
        - Topic

- **Message**
    - Son los mensajes que se envían y reciben en JMS
    - Subinterfaces:
        - TextMessage
        - BytesMessage
        - ObjectMessage
        - MapMessage } etc.

IMPLEMENTACIÓN CHAT ROBOT

1. Realizar la conexión a Artemis y obtener la cola del servidor.
    a) Conectar a Artemis, utilizando la factoría de conexión denominada "ConnectionFactory".
    b) Crear un contexto JMS por defecto.
    c) Obtener la cola del servidor de nuestra aplicación (llamada “ChatServer”). Este nombre está guardado en la constante SERVER_QUEUE de la clase Destinations (en paquete utils_jms).

```
    InitialContextLoader.setDefaultArtemisURL (serverUrl);
    ic = InitialContextLoader.getInitialContext(); // Create the JMS initial context

//STEP 1: Connection to Artemis and getting the server queue.
    //1a. COMPLETE: Connect to Artemis, using the connection factory named "ConnectionFactory"

    cfac = (ConnectionFactory) ic.lookup("ConnectionFactory");



    //1b. COMPLETE: Create a JMS context

        try {
           ctx = cfac.createContext();
        } catch (Exception e) {
           throw new Exception ("Cannot contact Artemis at url: " + serverUrl);
        }


        //1c. COMPLETE: Obtain the queue of our ChatServer application. Its name is stored in the SERVER_QUEUE constant.
        //This constant is declared at Destinations class (in utils_jms package)

        serverQueue = (Queue) ic.lookup (SERVER_QUEUE);
```

2. Envíe un mensaje CONNECT a la cola del servidor (i.e. cola ChatServer) y espere una respuesta. Dicha respuesta incluye la lista de usuarios y los canales registrados en el servidor.
    a) Crear un productor.
    b) Construir un mensaje de tipo "ConnectMessage", para indicar al servidor ChatServerJMS que queremos conectarnos al chat. Utilice la clase MessageFactory para construir el mensaje del tipo ConnectMessage.
    c) Crear una cola temporal en la que recibir la respuesta del servidor.
    d) Asignar esta cola temporal en la propiedad "replyTo" del mensaje, para así permitir que el servidor, al recibir nuestro mensaje, sepa a qué cola debe responder.
    e) Utilizando el productor creado anteriormente, enviar el mensaje a la cola de ChatServer.
    f) Crear un consumidor para nuestra cola temporal.
    g) El consumidor espera la recepción de un mensaje. Si todo ha ido bien, el servidor habrá creado, en su caso, la cola asociada al usuario y nos habrá enviado un mensaje de tipo ConnectOKMessage. Opcionalmente, el alumno puede implementar el código necesario para leer el mensaje recibido y obtener de dicho mensaje la lista de usuarios y canales.

```
    // STEP 2: Now we will send a "connect" message to the server

    //2.a COMPLETE: Create a producer

    producer = ctx.createProducer();


    //2.b COMPLETE: Create an object message, using the message factory to construct a "connectMessage"

   Message msg = ctx.createObjectMessage (MessageFactory.connectMessage(nick));

    //2.c COMPLETE: Create a temporary queue for receiving server's answer

    Queue tempQueue = (Queue) ctx.createTemporaryQueue();


    //2.d COMPLETE: Assign the "replyTo" property to this temporary queue

    msg.setJMSReplyTo (tempQueue);



    //2.e COMPLETE: Send the message to the queue, using the producer.

    try {
    producer.send (serverQueue, msg);
} catch (Exception e) {
    // The most likely reason is that queue does not exist. Meaning, it ChatServerJMS never
    // run after installing Artemis.
    ctx.close();
    throw new Exception ("Cannot send message to ChatServerJMS queue");
}


//2.f COMPLETE: Create a consumer for our temporary queue

consumer = ctx.createConsumer (tempQueue);


//2.g COMPLETE: Consumer waits the reception of a message.

ObjectMessage m = (ObjectMessage) consumer.receive(1000);
```

3. Ahora haremos que ChatRobot se conecte al canal indicado y envíe un mensaje de bienvenida. Para ello, debe:
    a) Construir un mensaje de tipo "Join", para indicar al servidor ChatServerJMS que queremos unirnos al canal. Asigne la cola temporal creada en el paso 2 a la propiedad "replyTo" del mensaje, para así permitir que el servidor, al recibir nuestro mensaje, sepa a qué cola debe responder.
    b) Enviar este mensaje creado, utilizando el productor que hemos creado en el paso 2, a la cola de ChatServer.
    c) Obtener la referencia al Topic JMS asociado al canal. (Nota: este paso ya se proporciona en el código).
    d) Crear un mensaje (de tipo ChatMessage) para enviar un mensaje de bienvenida al canal. Enviarlo usando el productor.

```
    // STEP 3: Join a channel and send a welcome message
    //3.a COMPLETE: create a Join message. Set the temporary queue as the replying queue

    Message msgJoin = ctx.createObjectMessage( MessageFactory.joinMessage (myChannelName, nick)) ;
    msgJoin.setJMSReplyTo (tempQueue);



    //3.b COMPLETE: send the join message to the ChatServer queue

    try {
    producer.send (serverQueue, msgJoin);
} catch (Exception e) {
    // The most likely reason is that queue does not exist. Meaning, it ChatServerJMS never
    // run after installing Artemis.
    ctx.close();
    throw new Exception ("Cannot send message to ChatServerJMS queue");
}


    //3.c Obtain the channel TOPIC
    channelTopic = (Topic) ic.lookup (CHANNEL_TOPIC + myChannelName);

//3.d ChatRobot sends a welcome message to the channel
   try {
    producer.send(channelTopic, ctx.createObjectMessage ( MessageFactory.chatMessage(nick, "BIENVENIDOS")));
} catch (Exception e) {
    // The most likely reason is that queue does not exist. Meaning, it ChatServerJMS never
    // run after installing Artemis.
    ctx.close();
    throw new Exception ("Cannot send message to ChatServerJMS queue");
}
```

4. Finalmente, haremos que ChatRobot itere de forma indefinida, esperando mensajes en el canal, los procese y, en caso de que sean mensajes de tipo JOIN, enviará un mensaje de saludo al canal. Como guía, puede utilizar el código de ChatServerJMS (revise el método runServerLoop de dicha clase). En nuestro caso, debe realizar:
    a) Crear un consumidor para el Topic asociado al canal.
    b) El consumidor espera la recepción de un mensaje, con espera bloqueante.
    c) En caso de recibir un mensaje de tipo USER_JOINS, enviamos un mensaje al canal, saludando al usuario concreto que se ha unido al canal.

```
        // STEP 4: Finally run the ChatRobot "forever", processing messages as they arrive in the channel
        //4.a Create a consumer for the channel queue

            JMSConsumer cons = null;
    try {
        cons = ctx.createConsumer (channelTopic);
    } catch (Exception e) {
        // Main queue should be configured with "maxCounsumers" set to 1.
        // We can confirm this point using Artemis console.
        //
        System.out.println ("ERROR ==> Cannot create server consumer. Some other ChatServerJMS is running for same Artemis.");
        return; // we end here.
    }

    // This server is a classic "while(true)" server.
    // Receive a client request, and serve it.

    while (true) {

        // Wait for a new client request.
        //
        System.out.println("Waiting for client messages...");
        ObjectMessage msga = (ObjectMessage) cons.receive();

        //
        // Get the message content
        AMessage bmsg = (AMessage) msga.getObject();
        System.out.println ("OK ==> We received a message of type: " + bmsg.getType());

        switch (bmsg.getType()) {

            case CONNECT: {
                ConnectMessage cm = (ConnectMessage) bmsg;
                //onConnect (cm, msg.getJMSReplyTo());
                break;
            }
            case USER_JOINS: {
                UserJoinsMessage jm = (UserJoinsMessage) bmsg;
                //onJoin (jm, msg.getJMSReplyTo());
                producer.send(channelTopic, ctx.createObjectMessage ( MessageFactory.chatMessage(nick, "BIENVENIDO " + jm.getNick())));
                break;
            }

            default: {
                System.out.println("Unknown message type: " + bmsg.getType());
                break;
            }
        }
    }


                //4.c In case of receiving a USER_JOINS message, we send a greeting message to the channel, saying "hello" to the user that has just joined


} catch (Exception e) {
    System.err.println ("Error in ChatRobot: " + e);
    e.printStackTrace (System.err);
    System.exit(-1);
}
```