Patrones de Arquitectura - ESB

Juan Nicolás Nontoa Caballero, Escuela Colombiana de Ingeniería, Patrones de arquitectura-ESB

***Abstract*— This workshop has as main objective the management and study of an ESB, for the understanding of this ServiceMix will be used. Apache ServiceMix is a flexible, open-source integration container that unifies the features and functionality of Apache ActiveMQ, Camel, CXF, and Karaf into a powerful runtime platform you can use to build your own integrations solutions. It provides a complete, enterprise ready ESB exclusively powered by OSGi.**

I. INTRODUCCION

Este documento se toma como referencia ayudando a entender el manejo de las arquitecturas ESB realizado en el entorno ServiceMix. Generalmente una arquitectura ESB proporciona una capa de abstracción construida sobre una implementación de un sistema de mensajes de empresa que permita a los expertos en integración explotar el valor del envío de mensajes sin tener que escribir código. Un ESB no implementa en sí mismo una arquitectura orientada a servicios (SOA), sino que proporciona las características mediante las cuales sí se puede implementar, además trata de aislar el acoplamiento entre el servicio solicitado y el medio de transporte.

II. SERVICE MIX

Es un contenedor flexible de integración de código abierto que unifica las funciones y la funcionalidad de Apache AtiveMQ, Camel, CXF y Karaf para proporcionar un ESB completo y listo para la empresa que funciona con OSGi. ServiceMix es un componente liviano en comparación con otros ESB.

Funcionalidades:

- Federación, clustering y failover proporcionado por el contenedor.
- Implementación en caliente y administración del ciclo de vida de los objetos de negocio.
- Independencia del vendedor de productos con licencia del vendedor.
- Cumplimiento de la especificación JBI JSR 208.
- Cumplimiento de la especificación OSGi 4.2 a través de Apache Felix.

A. *ActiveMQ:*

Se Es un proyecto de Apache, es el intermediario de mensajes que usa para intercambiar mensajes entre componentes. Además de esto, ActiveMQ también se puede usar para crear un ESB completamente distribuido.

B. *Camel:*

Es el motor de enrutamiento avanzado que permite la configuración del enrutamiento.

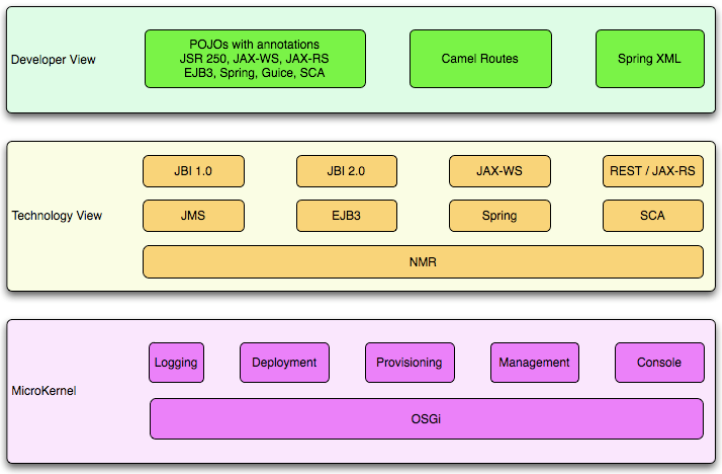
C. *CXF:*

Permite exponer y enrutar POJOs de Java anotados con anotaciones JAX-WS y generar con ellos Web Services.

D. *Karaf:*

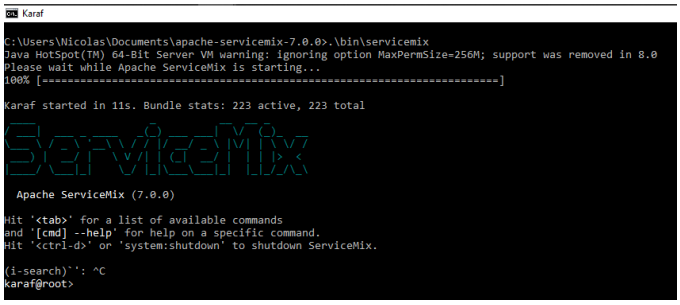
Runtime Container OSGi que proporciona capacidades adicionales. Apache Karaf proporciona un contenedor liviano para implementar varios componentes y aplicaciones.

III. DIAGRAMA DE ARQUITECTURA SERVICEMIX

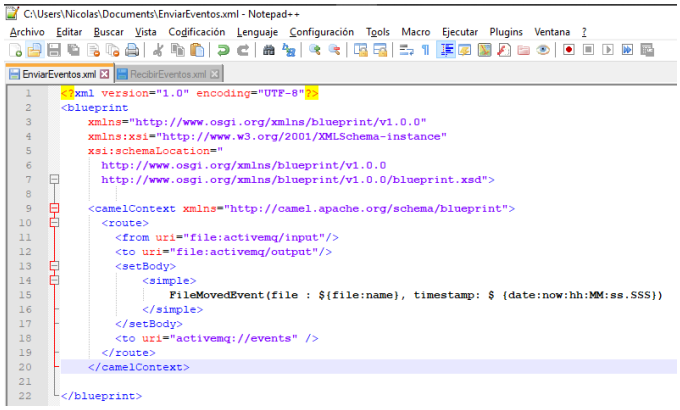


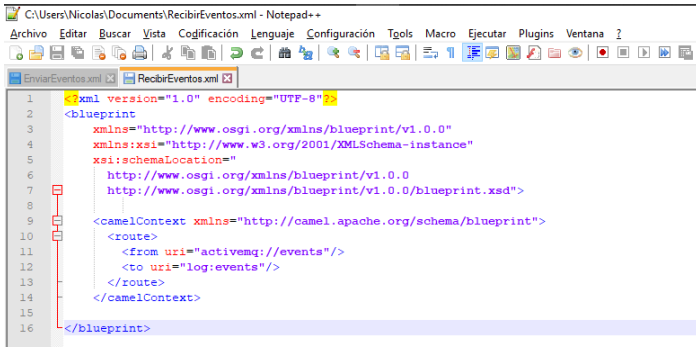
IV. ARQUITECTURA ESB

El primer paso será instalar ServiceMix, solo tendremos que ir a la página oficial y descargar el .zip o .tar dependiendo de nuestro sistema operativo, descomprimirlo y accederlo desde la terminal.

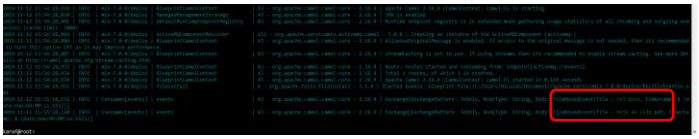


Enseguida se configura ServiceMix para que envíe los mensajes de la cola de eventos a la cola de salida. Para esto crearemos dos archivos .xml con la siguiente estructura. Uno para que configure que lo archivos ingresados a la carpeta input pasen o sean trasladados a la carpeta output.

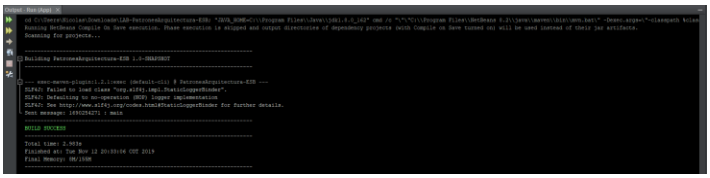


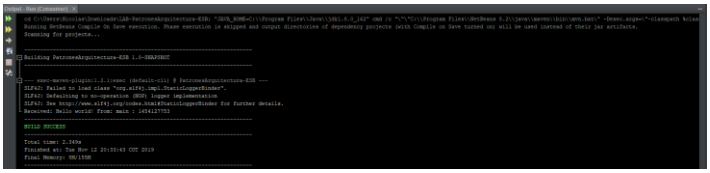


Al momento de crearlos se deben guardar en la carpeta deploy y probaremos mirando los logs en la consola.

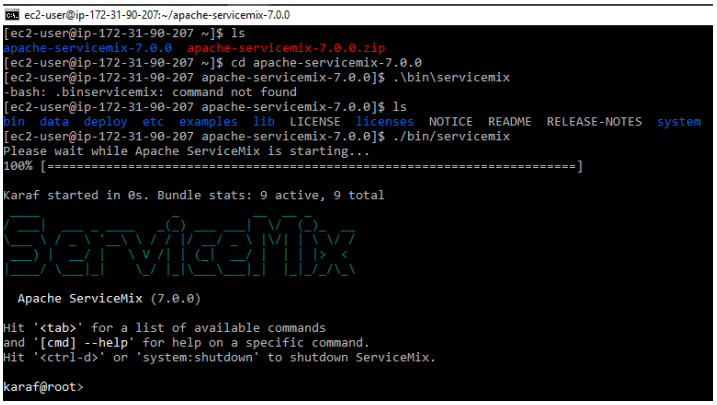


A continuación, enviamos el mensaje con el cliente y luego lo recibimos.





Ahora que vemos que todo funciona en localhost, procederemos a montar el servicio de ServiceMix en una maquina virtual EC2 en AWS. Para esto debemos instalar java 8 en la máquina virtual para evitar inconvenientes con ServiceMix. Una vez realizado esto corremos el servicio en la máquina virtual.

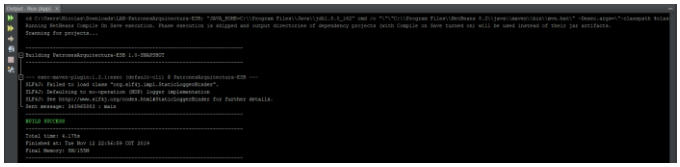


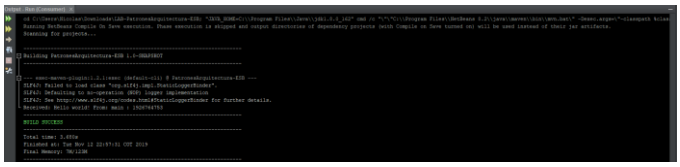
Enseguida vamos a modificar nuestro cliente y nuestro servidor. Pondremos la IP correspondiente a nuestra máquina virtual la cual es escalable y esta totalmente configurada y corriendo ServiceMix como en el paso anterior. No hay que olvidar habilitar el puerto 61616 en las reglas de entrada y salida del Security Group de la instancia.

```
public class App
{
    public static void main( String[] args )
    {
        App aplicacion = new App();
        aplicacion.run();
    }

    public void run(){
        CamelContext context = new DefaultCamelContext();
        //String brokerURL = args[0];
        ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://54.166.174.196:61616?jms.useAsyncSend=true");
        try {
            // Create a Connection
            Connection connection = connectionFactory.createConnection("smx","smx");
            connection.start();

            // Create a Session
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

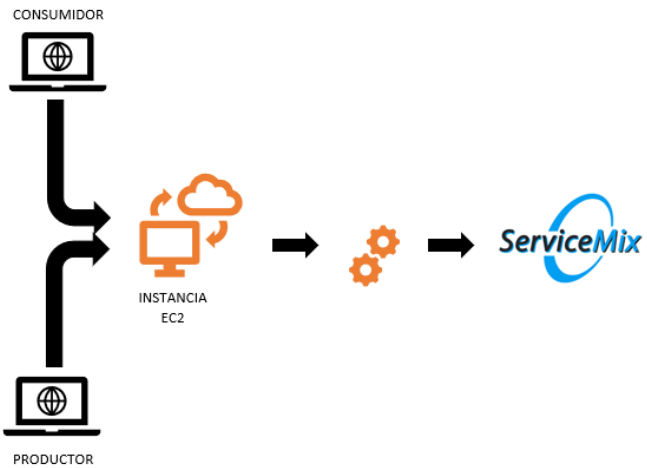
```
public class Consumer implements Runnable, ExceptionListener {

    public static void main( String[] args )
    {
        Consumer hello = new Consumer();
        hello.run();
    }

    public void run() {
        try {

            // Create a ConnectionFactory
            ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://54.166.174.196:61616?jms.useAsyncSend=true");

            // Create a Connection
            Connection connection = connectionFactory.createConnection("smx","smx");
            connection.start();
```

Por último probaremos que cuando se envie y se reciba el mensaje por medio de la IP de la máquina virtual desplegada en AWS funcione correctamente.





V. DIAGRAMA DE DESPLIEGUE



VI. CONCLUSION

Gracias al Desarrollo de este este laboratorio se pudo apreciar de un modo más claro la importancia de usar un bus de servicios. Se conocieron nuevas herramientas como ServiceMix la cual tiene múltiples funcionalidades y nos es muy útil. Además evidenciamos que se puede correr en una instancia de EC2 en AWS para mayor escalabilidad.

Juan Nicolás Nontoa Caballero nació en Tunja, Boyacá, Colombia en 1999. Graduado del Colegio San Jerónimo Emiliani en el 2015, inició sus estudios de Ingeniería de Sistemas en 2016 en la Escuela Colombiana de Ingeniería Julio Garavito.