

¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
 Ese apoyo que siempre quiso tener...

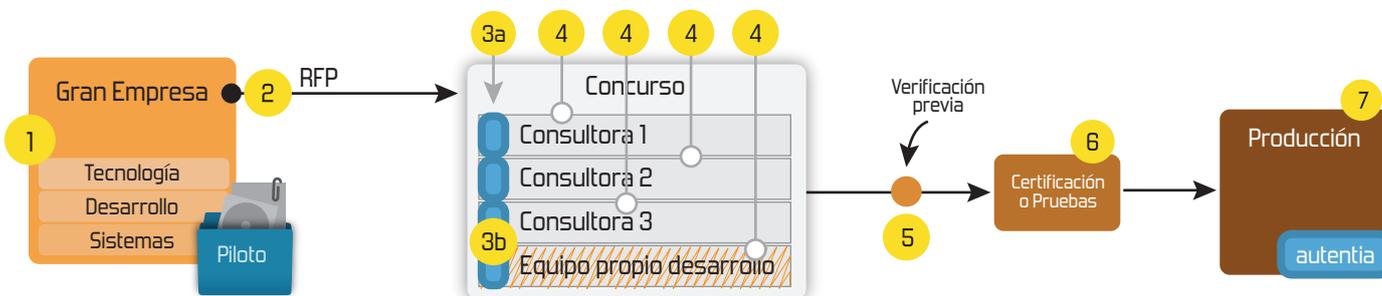
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
 Gestor de contenidos (Alfresco)
 Aplicaciones híbridas

Tareas programadas (Quartz)
 Gestor documental (Alfresco)
 Inversión de control (Spring)

Control de autenticación y
 acceso (Spring Security)
 UDDI
 Web Services
 Rest Services
 Social SSO
 SSO (Cas)

JPA-Hibernate, MyBatis
 Motor de búsqueda empresarial (Solr)
 ETL (Talend)

Dirección de Proyectos Informáticos.
 Metodologías ágiles
 Patrones de diseño
 TDD

BPM (jBPM o Bonita)
 Generación de informes (JasperReport)
 ESB (Open ESB)



E-mai

Contr

Entra

[Inicio](#) [Quiénes somos](#) [Tutoriales](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Charlas](#)

» [Estás en: Inicio](#) [Tutoriales](#) [Trabajando con Mule ESB](#)



Juan Alonso Ramos

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Informática de Gestión e Ingeniero en Informática, especialidad en Ingeniería del Software

Puedes encontrarme en [Autentia](#): Ofrecemos de servicios soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

[Ver todos los tutoriales del autor](#)

**Cat
Aut**

Fecha de publicación del tutorial: 2012-02-17

Trabajando con Mule ESB

Tutorial visitado 1 veces [Descargar en PDF](#)

» Au
las r
Adrr

» X
Myb
Hibe

» Pr
Lag;

» Ct
prep
apai

» ¡¡¡
traíc

Histi

Índice de contenidos.

- [1. Introducción](#)
- [2. Entorno](#)
- [3. Crear el Web Service de SOAP](#)
- [4. Fichero mule-config.xml](#)
- [5. Envío de correos](#)
- [6. Conclusiones](#)

1. Introducción

En este tutorial vamos a hacer un ejemplo de uso de Mule ESB. En otros tutoriales vimos unos [primeros pasos con Mule ESB](#), también cómo montar un [proyecto de Mule ESB con Maven](#), y cómo crear un proyecto de Mule con [Mule Studio](#).

En esta ocasión vamos a profundizar un poco más en esta tecnología creando una aplicación que levantará un web service de SOAP que recibirá peticiones para consultar un catálogo de vehículos por id de vehículo. El web service hará la consulta de vehículos almacenados en un mapa y enviará al usuario por correo electrónico la información del vehículo consultada. Para trabajar un poco más con esta tecnología haremos uso de los transformadores de Mule e introduciremos uno en el flujo que se encargará de componer el correo electrónico que se envía al usuario.

2. Entorno

- MacBook Pro 15' (2.4 GHz Intel Core i5, 8GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Snow Leopard 10.6.8
- JDK 1.6.0_29
- Mule 3.2.0

3. Crear el Web Service de SOAP

Para empezar crearemos un nuevo proyecto utilizando el [arquetipo de Mule](#) ya que nos ahorra mucho tiempo.

A continuación crearemos el servicio web que se encargue de atender las peticiones HTTP y hacer la consulta de los vehículos en el catálogo. Primero crearemos el interfaz 'SearchCar' que expone el método 'search' el cual recibe dos parámetros: el id del vehículo y el mail del usuario que realiza la petición para enviarle por correo electrónico la información del vehículo consultado.

```
1 package com.autentia.tutoriales;
```



Últi

» Aq
Maq

» El
de F
foto

» Tr
core

» Ar

**Últi
Aut**

```

1 package com.autentia.tutoriales;
2
3 import javax.jws.WebParam;
4 import javax.jws.WebResult;
5 import javax.jws.WebService;
6
7 @WebService
8 public interface SearchCar {
9
10     @WebResult(name="id")
11     Car search(@WebParam(name="id") Long id,
12              @WebParam(name="mail") String customer);
13 }

```

La forma más sencilla de crear el servicio web es anotándolo mediante `@WebService`. Esta anotación pertenece al API de JAX-WS (Java API for XML Web Services) incluida en Java 6. El método 'search' recibe los parámetros id y mail en la petición.

Una vez creado el interfaz del web service creamos una implementación. Implementamos el interfaz creado anteriormente y también el interfaz de Initialisable del API de Mule. Mediante la implementación de este interfaz, Mule se encargará de invocar al método initialize durante el arranque de la aplicación el servidor. En este método creamos el catálogo de coches para la consulta.

```

1 package com.autentia.tutoriales;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 import javax.jws.WebService;
7
8 import org.mule.api.lifecycle.Initialisable;
9 import org.mule.api.lifecycle.InitialisationException;
10
11 @WebService(serviceName="searchCarService", endpointInterface="com.autentia.tutoriales.S
12 public class SearchCarService implements SearchCar, Initialisable {
13
14     final Map<Long, Car> cars = new HashMap<Long, Car>();
15
16     @Override
17     public void initialise() throws InitialisationException {
18         cars.put(1L, new Car(1L, "Renault", "Megane", 18500D));
19         cars.put(2L, new Car(2L, "Ford", "Focus", 17500D));
20         cars.put(3L, new Car(3L, "Alfa Romeo", "159", 24000D));
21         cars.put(4L, new Car(4L, "BMW", "Serie 1", 38900D));
22         cars.put(5L, new Car(5L, "Volkswagen", "Golf", 24200D));
23     }
24
25     @Override
26     public Car search(Long id, String customerMail) {
27         final Car car = cars.get(id);
28
29         car.setCustomerMail(customerMail);
30         return car;
31     }

```

El método 'initialise' será invocado tras la instanciación del servicio. En este caso lo utilizaremos para crear el catálogo de vehículos del ejemplo.

El método 'search' es el encargado de hacer la búsqueda del vehículo en el catálogo a través del id. También se encargará de recoger el email del cliente al que enviarle la información del vehículo consultado. La clase Car es un simple POJO que almacena el id del vehículo, el fabricante, modelo, precio y mail del usuario que realiza la consulta.

4. Fichero mule-config.xml

En este fichero es donde se configura todo el flujo de Mule. Contiene el servicio web de búsqueda de vehículos donde se indica la dirección HTTP donde está desplegado nuestro servicio de consulta de vehículos en el catálogo.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:xsi="http://www.w3.org/2001
3   xmlns:context="http://www.springframework.org/schema/context"
4   xmlns:vm="http://www.mulesoft.org/schema/mule/vm" xmlns:cxf="http://www.mulesoft.org
5   xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp" xmlns:smtps="http://www.muleso
6   xmlns:email="http://www.mulesoft.org/schema/mule/email"
7   xmlns:servlet="http://www.mulesoft.org/schema/mule/servlet"
8   xsi:schemaLocation="
9     http://www.springframework.org/schema/context http://www.springframework.org/sch
10    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core
11    http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.
12    http://www.mulesoft.org/schema/mule/cxf http://www.mulesoft.org/schema/mule/cxf/
13    http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/schema/mule/smt
14    http://www.mulesoft.org/schema/mule/smtps http://www.mulesoft.org/schema/mule/sm
15    http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/em
16    http://www.mulesoft.org/schema/mule/servlet http://www.mulesoft.org/schema/mule/
17
18 <flow name="SearchCarService">
19   <composite-source>
20     <inbound-endpoint address="http://localhost:8081/cars" exchange-pattern="req

```

```

21         <cxfservice:jaxws-service serviceClass="com.autentia.tutoriales.SearchCar" />
22     </inbound-endpoint>
23 </composite-source>
24
25     <component>
26         <singleton-object class="com.autentia.tutoriales.SearchCarService" />
27     </component>
28 </flow>
29 </mule>

```

Como endpoint de entrada configuramos el SearchCar a través de la etiqueta 'jaxws-service'.

Indicamos a mule a través de la etiqueta 'singleton-object' que cree un Singleton del servicio de búsqueda de vehículos.

5. Envío de correos

Para configurar el servicio de envío de correos utilizaremos el servicio smtps que nos proporciona Mule. Para incluirlo en nuestro flujo añadimos al fichero config-mule.xml, después del flujo anterior, lo siguiente:

```

1 <context:property-placeholder location="mail.properties" />
2
3 <flow name="EmailService">
4     <vm:inbound-endpoint path="emailService" exchange-pattern="one-way" />
5     <smtps:outbound-endpoint user="{user}"
6         password="{password}"
7         host="{host}"
8         from="{from}" />
9     <custom-transformer class="com.autentia.tutoriales.EmailTransformer" />
10    <email:string-to-email-transformer />
11 </smtps:outbound-endpoint>
12 </flow>

```

Configuramos un nuevo flujo llamado EmailService que contienen un endpoint de entrada y un servicio de envío de correos que proporciona Mule. En el servicio SMTP debemos indicar la configuración de la cuenta con la que se envían los correos. Estas propiedades se externalizan en el fichero mail.properties que dejaremos en src/main/resources junto al mule-config.xml.

Para completar el servicio de envío de correos hemos introducido un transformer que será invocado antes del envío del correo y se encargará de componer el mensaje que se enviará al cliente. Para crear este transformer debemos crear una clase que extienda de org.mule.transformer.AbstractMessageTransformer y que implemente el método transformMessage:

```

1 package com.autentia.tutoriales;
2
3 import org.mule.api.MuleMessage;
4 import org.mule.api.transformer.TransformerException;
5 import org.mule.transformer.AbstractMessageTransformer;
6 import org.mule.transport.email.MailProperties;
7
8 public class EmailTransformer extends AbstractMessageTransformer {
9
10    @Override
11    public Object transformMessage(MuleMessage message, String outputEncoding) throws Tr
12        final Car car = (Car) message.getPayload();
13
14        final StringBuilder mailMessage = new StringBuilder("A continuación le enviamos
15        mailMessage.append("Fabricante: " + car.getManufacturer()).append("\n");
16        mailMessage.append("Modelo: " + car.getModel()).append("\n");
17        mailMessage.append("Precio: " + car.getPrice()).append(" euros \n");
18        mailMessage.append("Recibe un cordial saludo. ");
19
20        message.setOutboundProperty(MailProperties.SUBJECT_PROPERTY, "Información del ve
21        message.setOutboundProperty(MailProperties.TO_ADDRESSES_PROPERTY, car.getCustom
22
23        return mailMessage.toString();
24    }
25 }

```

Lo más destacado de esta clase es el parámetro 'message' de la clase MuleMessage que contiene el parámetro devuelto por el método search del servicio web, en nuestro caso un objeto de la clase Car. A través de estos mensajes es la manera que tenemos de comunicar los distintos elementos dentro del flujo de Mule.

Por último debemos comunicar el servicio web anterior con el servicio de envío de correo por lo que añadimos al mule-config.xml, dentro del flujo SearchCarService un outbound-endpoint invocando al servicio de correo. El fichero mule-config.xml al completo quedaría así:

```

1 ...
2 <flow name="SearchCarService">
3     <composite-source>
4         <inbound-endpoint address="http://localhost:8081/cars" exchange-pattern="req
5         <cxfservice:jaxws-service serviceClass="com.autentia.tutoriales.SearchCar" />
6     </inbound-endpoint>
7 </composite-source>
8
9 </component>

```

```

9      <component>
10         <singleton-object class="com.autentia.tutoriales.SearchCarService" />
11     </component>
12
13     <async>
14         <vm:outbound-endpoint path="emailService" exchange-pattern="one-way" />
15     </async>
16 </flow>
17
18 <context:property-placeholder location="mail.properties" />
19
20 <flow name="EmailService">
21     <vm:inbound-endpoint path="emailService" exchange-pattern="one-way" />
22     <smtpts:outbound-endpoint user="{user}"
23                             password="{password}"
24                             host="{host}"
25                             from="{from}">
26         <custom-transformer class="com.autentia.tutoriales.EmailTransformer" />
27         <email:string-to-email-transformer />
28     </smtpts:outbound-endpoint>
29 </flow>
30 ...

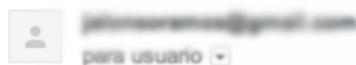
```

Para el servicio de envío de correos hemos configurado la tarea como asíncrona ya que no nos importa que no sea realice de forma inmediata tras la petición del cliente.

Para desplegar la aplicación en el servidor de Mule, podemos hacerlo seleccionando el fichero mule-config.xml desde Eclipse (si tenemos instalado el [plugin de Mule](#)) y con botón derecho Run As > Mule Server.

Para probar la aplicación invocaremos al servicio web a través de la URL

<http://localhost:8081/cars/search/id/2/mail/jalonso@autentia.com> donde indicaremos un id del vehículo a buscar y el email donde queremos que nos envíe la respuesta. Si todo ha ido bien en la cuenta de correo indicada habrá llegado un correo con la información que le hemos solicitado.



A continuación le enviamos los datos del vehiculo que nos ha solicitado:

Fabricante: Ford
 Modelo: Focus
 Precio: 17500.0 euros
 Recibe un cordial saludo.

6.Conclusiones

Con este caso práctico de un flujo de Mule hemos podido ver que con poco código y un poco de configuración obtenemos mucha funcionalidad. Es por ello que consideramos esta tecnología, al igual que otras [similares](#), de gran ayuda para un gran número de aplicaciones de negocio que necesiten comunicar diferentes servicios.

Espero que te haya servido de ayuda.

Un saludo. Juan.

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

Por favor, vota +1 o compártelo si te pareció interesante

Share |

0

¿Te gusta [adictosaltrabajo.com](#)? Síguenos a través de:



Anímate y coméntanos lo que pienses sobre este **TUTORIAL**:

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda,

podemos ofrecerte un mejor servicio.

[Enviar comentario](#)

(Sólo para usuarios registrados)

» **Regístrate** y accede a esta y otras ventajas «



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

IMPULSA

[Impulsores](#)

[Comunidad](#)

[¿Ayuda?](#)

0 personas han traído clicks a esta página

sin clicks

+ + + + + + + +

powered by [karmacracy](#)

Copyright 2003-2012 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) |

