# Solace JMS Integration with Mule v3.6

Document Version 1.2

November 2015

This document is an integration guide for using Solace JMS as a JMS provider within MuleSoft's Anypoint Platform Mule runtime engine.

Mule is the runtime engine of Anypoint Platform. It is a unified platform that combines data and application integration across legacy systems, SaaS applications, and APIs with hybrid deployment options for maximum flexibility.

The Solace message router supports persistent and non-persistent JMS messaging with high throughput and low, consistent latency. Thanks to very high capacity and built-in virtualization, each Solace message router can replace dozens of software-based JMS brokers in multi-tenant deployments. Since JMS is a standard API, client applications connect to Solace like any other JMS broker so companies whose applications are struggling with performance or reliability issues can easily overcome them by upgrading to Solace's hardware.

Solace Systems®



http://www.solacesystems.com

# Table of Contents

# 1 Overview

This document demonstrates how to integrate Solace Java Message Service (JMS) with MuleSoft's Anypoint Platform Mule runtime engine for production and consumption of JMS messages. The goal of this document is to outline best practices for this integration to enable efficient use of Solace JMS within the Mule runtime engine.

The target audience of this document is developers using Mule with knowledge of both the Mule runtime engine and JMS in general. As such this document focuses on the technical steps required to achieve the integration. For detailed background on either Solace JMS or the MuleSoft's Anypoint Platform Mule runtime engine refer to the referenced documents below.

It is important to note that there are several ways to connect Mule with Solace for messaging. This document specifically looks at JMS. The other connection methods are covered in other Solace documentation.

1. Solace JMS
2. Solace REST Messaging Interface
3. MQTT

This document is divided into the following sections to cover the Solace JMS integration with Mule:

- Integrating with Mule
- Performance Considerations
- Working with Solace High Availability
- Debugging Tips
- Advanced Topics including (TLS secure communications, XA Transactions, and Solace Disaster Recovery)

## 1.1 Related Documentation

These documents contain information related to the feature defined in this document

| Document ID | Document Title | Document Source |
|---|---|---|
| [Solace-Portal] | Solace Developer Portal | http://dev.solacesystems.com |
| [Solace-JMS-REF] | Solace JMS Messaging API Developer Guide | http://dev.solacesystems.com/docs/solace-jms-api-developer-guide |
| [Solace-JMS-API] | Solace JMS API Online Reference Documentation | http://dev.solacesystems.com/docs/solace-jms-api-online-reference |
| [Solace-FG] | Solace Messaging Platform – Feature Guide | http://dev.solacesystems.com/docs/messaging-platform-feature-guide |
| [Solace-FP] | Solace Messaging Platform – Feature Provisioning | http://dev.solacesystems.com/docs/messaging-platform-feature-provisioning |
| [Solace-CLI] | Solace Message Router Command Line Interface Reference | http://dev.solacesystems.com/docs/cli-reference |
| [Mule-REF] | Mule Runtime Engine Documentation | https://developer.mulesoft.com/docs/display/current/Home |
| [Mule-JMS] | Mule's JMS Transport Reference | https://developer.mulesoft.com/docs/display/current/JMS+Transport+Reference |

**Table 1 - Related Documents**

# 2 Why Solace

Solace technology efficiently moves information between all kinds of applications, users and devices, anywhere in the world, over all kinds of networks. Solace makes its state-of-the-art data movement capabilities available via hardware and software “message routers” that can meet the needs of any application or deployment environment. Solace’s unique solution offers unmatched capacity, performance, robustness and TCO so our customers can focus on seizing business opportunities instead of building and maintaining complex data distribution infrastructure.

## Superior Performance

Solace’s hardware and software messaging middleware products can cost-effectively meet the performance needs of any application, with feature parity and interoperability that lets companies start small and scale to support higher volume or more demanding requirements over time, and purpose-built appliances that offer 50-100x higher performance than any other technology for customers or applications that require extremely high capacity or low latency.

## Robustness

Solace offers high availability (HA) and disaster recovery (DR) without the need for 3rd party products, and fast failover times no other solution can match. Distributing data via dedicated TCP connections ensures an orderly, well-behaved system under load, and patented techniques ensure that the performance of publishers and high-speed consumers is never impacted by slow consumers.

## Simple Architecture

Modern enterprises run applications that demand many kinds of data movement such as persistent messaging, web streaming, WAN distribution and cloud-based communications. By supporting all kinds of data movement with a unified platform that can be deployed as a small-footprint software broker or high-capacity rack-mounted appliance, Solace lets architects design an end-to-end infrastructure that’s easy to build applications for, integrate with existing technologies, secure and scale.

## Simple Operations

Solace’s solution features a shared administration framework for all kinds of data movement, deployment models and network environments so it’s easy for IT staff to deploy, monitor, manage and upgrade their Solace-based messaging environment.

## Cost Savings

Solace reduces expenses with high-capacity hardware, flexible software, and the ability to deploy the right solution for each problem. Solace’s support for many kinds of messaging lets you replace multiple messaging products with just one, built-in HA, DR, WAN and Web functionality eliminate the need for third-party products.

# 3 Integrating with Mule

Mule support for JMS is explained in detail in the [Mule-JMS]. However, in summary, Mule enables JMS messaging via a Mule Transport Connector Component. The Mule JMS Transport allows messages to be sent or received using a JMS compliant message broker like the Solace Message Router.

In order to illustrate the Mule integration, the following sections will highlight the required Mule configuration changes and provide snippets of sample code for sending and receiving messages using Mule Flows. The full Mule configuration XML code can be found in the Section 7.1.

This integration guide demonstrates how to configure a Mule Flow to send and receive JMS messages using a shared JMS connection. Accomplishing this requires completion of the following steps.

- Step 1 - Configuration of the Solace Message Router.
- Step 2 – Configuring the Mule JMS Transport to connect to the Solace message router.
- Step 3 – Configuring a Mule Flow to send messages using Solace JMS.
- Step 4 – Configuring a Mule Flow to receive messages using Solace JMS.

## 3.1 Description of Resources Required

This integration guide will demonstrate creation of Solace resources and configuration of Mule Flows. This section outlines the resources that are created and used in the subsequent sections.

### 3.1.1 Solace Resources

The following Solace message router resources are required.

| Resource | Value | Description |
|---|---|---|
| Solace message router IP:Port | __IP:Port__ | The IP address and port of the Solace message router message backbone. This is the address clients use when connecting to the Solace message router to send and receive message. This document uses a value of __IP:PORT__. |
| Message VPN | Solace_Mule_VPN | A Message VPN, or virtual message broker, to scope the integration on the Solace message router. |
| Client Username | mule_user | The client username. |
| Client Password | mule_password | Optional client password. |
| Solace Queue | Q/requests | Solace destination of messages produced and consumed |
| JNDI Connection Factory | JNDI/CF/mule | The JNDI Connection factory for controlling Solace JMS connection properties |
| JNDI Queue Name | JNDI/Q/requests | The JNDI name of the queue used in the samples |

**Table 2 – Solace Configuration Resources**

### 3.1.2 Mule Configuration

The following Mule configuration is referenced in the integration steps. These items are explained in detail in the [Mule-REF]. The “Appendix – Mule Configuration Reference” contains the full Mule configuration file for these resources and how each of these resources relates to integration with Solace is explained in the subsequent sections as these resources are introduced. Also, the configuration makes use of several Mule properties which are also shown below. These are normally set in the “`mule-app.properties`” file in the Mule project.

| Resource | Value |
|---|---|
| JMS Connector (Global Element) | SolaceJMS |

**Table 3 – Mule Configuration Resources**

| Resource | Value | Description |
|---|---|---|
| solace.host | __IP:PORT__ | The host address for JNDI look up of the Solace connection. Normally this is the messasge-backbone IP of the Solace message router. |
| solace.msgvpn | Solace_Mule_VPN | The Solace message VPN name. |
| solace.username | mule_user | The Solace client username |
| solace.jndi.cf | JNDI/CF/mule | JNDI connection factory name |
| solace.jndi.queue | JNDI/Q/requests | Solace JNDI Queue name |

**Table 4 – Mule Configuration Properties**

## 3.2 Step 1 – Configuring the Solace message router

The Solace message router needs to be configured with the following configuration objects at a minimum to enable JMS to send and receive messages within a Mule Flow.

- A Message VPN, or virtual message broker, to scope the integration on the Solace message router.
- Client connectivity configurations like usernames and profiles
- Guaranteed messaging endpoints for receiving messages.
- Appropriate JNDI mappings enabling JMS clients to connect to the Solace message router configuration.

For reference, the CLI commands in the following sections are from SolOS version 7.1 but will generally be forward compatible. For more details related to Solace message router CLI see [Solace-CLI]. Wherever possible, default values will be used to minimize the required configuration. The CLI commands listed also assume that the CLI user has a Global Access Level set to Admin. For details on CLI access levels please see [Solace-FG] section “CLI User Authentication and Authorization”.

Also note that this configuration can also be easily performed using SolAdmin, Solace’s GUI management tool. This is in fact the recommended approach for configuring a Solace message router. This document uses CLI as the reference to remain concise.

### 3.2.1 Creating a Message VPN

This section outlines how to create a message-VPN called “Solace_Mule_VPN” on the Solace message router with authentication disabled and 2GB of message spool quota for Guaranteed Messaging. This message-VPN name is required in the Mule configuration when connecting to the Solace message router. In practice appropriate values for authentication, message spool and other message-VPN properties should be chosen depending on the end application’s use case.

```
(config)# create message-vpn Solace_Mule_VPN
(config-msg-vpn)# authentication
(config-msg-vpn-auth)# user-class client
(config-msg-vpn-auth-user-class)# basic auth-type none
(config-msg-vpn-auth-user-class)# exit
(config-msg-vpn-auth)# exit
(config-msg-vpn)# no shutdown
(config-msg-vpn)# exit
(config)#
(config)# message-spool message-vpn Solace_Mule_VPN
(config-message-spool)# max-spool-usage 2000
(config-message-spool)# exit
(config)#
```

### 3.2.2 Configuring Client Usernames & Profiles

This section outlines how to update the default client-profile and how to create a client username for connecting to the Solace message router. For the client-profile, it is important to enable guaranteed messaging for JMS messaging and transacted sessions if using transactions.

The chosen client username of “mule_user” will be required by the Mule when connecting to the Solace message router.

```
(config)# client-profile default message-vpn Solace_Mule_VPN
(config-client-profile)# message-spool allow-guaranteed-message-receive
(config-client-profile)# message-spool allow-guaranteed-message-send
(config-client-profile)# message-spool allow-transacted-sessions
(config-client-profile)# exit
(config)#
(config)# create client-username mule_user message-vpn Solace_Mule_VPN
(config-client-username)# acl-profile default
(config-client-username)# client-profile default
(config-client-username)# no shutdown
(config-client-username)# exit
(config)#
```

### 3.2.3 Setting up Guaranteed Messaging Endpoints

This integration guide shows receiving messages from a single JMS Queue in a Mule Flow. For illustration purposes, this queue is chosen to be an exclusive queue with a message spool quota of 2GB matching quota associated with the message VPN. The queue name chosen is “Q/requests”.

```
(config)# message-spool message-vpn Solace_Mule_VPN
(config-message-spool)# create queue Q/requests
(config-message-spool-queue)# access-type exclusive
(config-message-spool-queue)# max-spool-usage 2000
(config-message-spool-queue)# permission all delete
(config-message-spool-queue)# no shutdown
(config-message-spool-queue)# exit
(config-message-spool)# exit
(config)#
```

### 3.2.4 Setting up Solace JNDI References

To enable the JMS clients to connect and look up the Queue destination required by Mule, there are two JNDI objects required on the Solace message router:

- A connection factory: JNDI/CF/mule
- A queue destination: JNDI/Q/requests

They are configured as follows:

```
(config)# jndi message-vpn Solace_Mule_VPN
(config-jndi)# create connection-factory JNDI/CF/mule
(config-jndi-connection-factory)# property-list messaging-properties
(config-jndi-connection-factory-pl)# property default-delivery-mode persistent
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# property-list transport-properties
(config-jndi-connection-factory-pl)# property direct-transport false
(config-jndi-connection-factory-pl)# property "reconnect-retry-wait" "3000"
(config-jndi-connection-factory-pl)# property "reconnect-retries" "20"
(config-jndi-connection-factory-pl)# property "connect-retries-per-host" "5"
(config-jndi-connection-factory-pl)# property "connect-retries" "1"
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)#
(config-jndi)# create queue JNDI/Q/requests
(config-jndi-queue)# property physical-name Q/requests
(config-jndi-queue)# exit
(config-jndi)#
(config-jndi)# no shutdown
(config-jndi)# exit
(config)#
```

## 3.3 Step 2 – Connecting using the Mule JMS Transport

### 3.3.1 Installing the Solace JMS jars into a Mule project

There are several different supported methods for including third-party libraries in mule application. The following is taken from [Mule-REF]:

> **Packaging Third-Party Libraries**
>
> CloudHub provides services in the platform that support connecting over many different protocols and transforming many data formats. All the standard Mule libraries are included and available to your application. If you have the need to include a different third party library, you can package that library in the <mule application zip>/lib folder of your application package. Any libraries that are packaged in your Mule application lib folder are available on the classpath to your application at run time.

Solace JMS requires the following libraries be available in the Mule application to function correctly:

- Commons-lang-<version>.jar
- Commons-logging-<version>.jar
- Geronimo-jms_1.1_spec-<version>.jar
- Sol-common-<version>.jar
- Sol-jcsmp-<version>.jar
- Sol-jms-<version>.jar

### 3.3.2 Configuring the Mule JMS Transport

The following configuration is required to successfully establish a connection from Mule to the Solace message router.

The example configuration below uses XML-based container configuration to illustrate the integration. The "___IP:PORT___" should be updated to reference the actual Solace message router message-backbone VRF IP.

In Solace JMS, the "java.naming.security.principal" often uses the format <username>@<message-vpn>. This allows specification of the Solace message router client username ("mule_user") and message-vpn ("Solace_Mule_VPN") created in the previous section. Both of these items are mandatory in order to connect to the Solace message router.

The "java.naming.security.credentials" is optional and provides the Solace message router client password for use when authenticating with the Solace message router. In this example a password is not used and so this parameter is left commented in the configuration. For further details on authentication see Section 7.1.

As outlined in [Mule-JMS], there are two different ways to configure the Mule JMS connector JNDI settings.

1. Using connector properties
2. Using a JndiNameResolver

Of the two, Mule is migrating towards the JndiNameResolver. However both will work with Solace.

### 3.3.3 Using Connector Properties

To connect Solace using connector properties, the Mule JMS connector should be configured as follows:

```
<jms:connector name="SolaceJMS" specification="1.1" username="${solace.username}"
       validateConnections="true"
       persistentDelivery="true"
       cacheJmsSessions="true"
       eagerConsumer="true"
       doc:name="JMS"
       forceJndiDestinations="true"
       jndiDestinations="true"
       connectionFactoryJndiName="${solace.jndi.cf}"
       jndiInitialFactory="com.solacesystems.jndi.SolJNDIInitialContextFactory"
       jndiProviderUrl="smf://${solace.host}">
       <spring:property name="jndiProviderProperties">
              <spring:map>
                     <spring:entry key="java.naming.security.principal"
                            value="${solace.username}@${solace.msgvpn}" />
              </spring:map>
       </spring:property>
</jms:connector>
```

The following table explains key configuration and its purpose when connecting to the Solace message router.

| Mule Configuration Item | Description |
|---|---|
| **jms:connector** | This is the global configuration element which contains all of the JMS connectivity configuration. |
| **connectionFactoryJndiName** | The connection factory configured in Solace JNDI. |
| **persistentDelivery** | Forces Mule to look up all destinations in JNDI. Disable this for fastest performance. |
| **forceJndiDestinations** | Forces Mule to return errors if JNDI destination lookups fail. Otherwise if true, Mule will auto create destination if lookup fails. |
| **jndiInitialFactory** | The Solace JNDI initial context factory class |
| **jndiProviderUrl** | The IP and port address of the Solace message router messaging interface. |
| **jndiProviderProperties** | The username, password, and message-vpn for the JNDI connection to Solace. |

**Table 5 – Mule JMS Connector – Properties approach**

### 3.3.4 Using a JndiNameResolver

To connect Solace using a JndiNameResolver, the Mule JMS connector can be configured to use one of two name resolvers. From [Mule_JMS] these are:

SimpleJndiNameResolver: Uses a JNDI context instance to search for the names. That instance is maintained opened during the full lifecycle of the name resolver.

CachedJndiNameResolver: Uses a simple cache in order to store previously resolved names. A JNDI context instance is created for each request that is sent to the JNDI server and then the instance is freed. The cache can be cleaned up restarting the name resolver.

The following example shows how to use a CachedJndiNameResolver:

```
<jms:connector name="SolaceJMS" specification="1.1"
       username="${solace.username}" validateConnections="true"
       connectionFactoryJndiName="${solace.jndi.cf}"
       persistentDelivery="true"
       cacheJmsSessions="true"
       eagerConsumer="true"
       jndiDestinations="true"
       forceJndiDestinations="true"
       doc:name="JMS">
       <jms:custom-jndi-name-resolver
           class="org.mule.transport.jms.jndi.CachedJndiNameResolver">
            <spring:property name="jndiInitialFactory"
value="com.solacesystems.jndi.SolJNDIInitialContextFactory"/>
            <spring:property name="jndiProviderUrl" value="smf://${solace.host}"/>
            <spring:property name="jndiProviderProperties">
               <spring:map>
                      <spring:entry key="java.naming.security.principal"
                                    value="${solace.username}@${solace.msgvpn}" />
               </spring:map>
            </spring:property>
       </jms:custom-jndi-name-resolver>
</jms:connector>
```

The following table explains key configuration and its purpose when connecting to the Solace message router.

| Mule Configuration Item | Description |
|---|---|
| **jms:connector** | This is the global configuration element which contains all of the JMS connectivity configuration. |
| **connectionFactoryJndiName** | The connection factory configured in Solace JNDI. |
| **persistentDelivery** | Forces Mule to look up all destinations in JNDI. Disable this for fastest performance. |
| **forceJndiDestinations** | Forces Mule to return errors if JNDI destination lookups fail. Otherwise if true, Mule will auto create destination if lookup fails. |
| **custom-jndi-name-resolver** | The Solace JNDI connection details |

| Mule Configuration Item | Description |
|---|---|
| **jndiInitialFactory** | The Solace JNDI initial context factory class |
| **jndiProviderUrl** | The IP and port address of the Solace message router messaging interface. |
| **jndiProviderProperties** | The username, password, and message-vpn for the JNDI connection to Solace. |

**Table 6 – Mule JMS Connector – Properties approach**

## 3.4 Step 3 – Mule – Sending Messages to Solace

Once the Solace JMS Mule connector is configured, it is very simple to send messages to Solace. All that is required is to add the `<jms:outbound-enpoint>` configuration to a Mule flow. The following section demonstrate a simple Mule flow that will listen for an HTTP POST and send a message based on the query parameters. For more details on all of the properties supported by `<jms:outbound-enpoint>` see [Mule-JMS].

### 3.4.1 Configuration

The configuration below sends JMS messages to the Solace message router.

```
<http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081"
doc:name="HTTP Listener Configuration"/>

<flow name="sendingToSolace">
    <http:listener config-ref="HTTP_Listener_Configuration" path="/sendMessage"
       doc:name="HTTP"/>
    <set-payload value="#[message.inboundProperties.'http.query.params'.message]"
       doc:name="Set Payload"/>
    <jms:outbound-endpoint
       queue="${solace.jndi.queue}"
       connector-ref="SolaceJMS" doc:name="Send to Solace"/>
</flow>
```

The following table explains the configuration and its purpose when publishing to the Solace message router.

| Bean Id | Description |
|---|---|
| **http:listener-config** | The configuration for the HTTP server. |
| **http:listener** | Listens for HTTP request messages on the /sendMessage path using the configuration found in `<http:listener-config>` |
| **set-payload** | Used for demo purposes to set the message payload from the query param “message” |
| **jms:outbound-endpoint** | Sends a JMS message to a destination using a pre-existing SolaceJMS connector. |

**Table 7 - Solace Publish Configuration**

### 3.4.2 Sending a Test Message

To test the above flow once the Mule worker is started, send and HTTP request with the following format:

```
http://localhost:8081/sendMessage?message=Test_message_contents
```

This will send a message with contents "Test_message_contents" to the Solace queue named "Q/requests" which is looked up in JNDI using the name "JNDI/Q/requests".

## 3.5 Step 4 – Mule – Receiving Messages from Solace

In order to receive messages from Solace, a <jms:inbound-endpoint> is used. The following is a simple flow that will receive messages and print them to the screen. For more details on all of the properties supported by <jms:inbound-endpoint> see [Mule-JMS]

### 3.5.1 Configuration

The configuration below receives messages from a Solace queue that is looked up in JNDI.

```
<flow name="receivingFromSolace">
    <jms:inbound-endpoint connector-ref="SolaceJMS"
                          doc:name="JMS" queue="${solace.jndi.queue}"/>
    <logger level="INFO" message="Processing Solace JMS message: #[payload]"
                          doc:name="Logger" />
</flow>
```

The following table explains the configuration and its purpose when receiving messages from the Solace message router.

| Bean Id | Description |
|---|---|
| **jms:inbound-endpoint** | Connects to the Solace queue as looked up in JNDI "JNDI/Q/requests". The receives messages. |
| **logger** | Prints the message payload to the log. |

**Table 8 - Solace Receive Configuration**

# 4 Performance Considerations

The standard JMS API allows clients to send and receive persistent messages at high rates if used efficiently. In order to use the Solace JMS API efficiently, some JMS objects should be cached. Mule makes it possible to create these objects up front and cache them for re-use and in many cases makes this the default behaviour. This section outlines how to tune Mule through configuration to properly re-use the following JMS Objects:

- JMS Connections, Sessions, MessageProducers & MessageConsumers
- Destinations

Failure to correctly cache these objects can result in a new connection being established to the Solace appliance for each message sent. This results in low overall performance and is not a recommended method of operating. It is possible to detect this scenario by monitoring the Solace event logs for frequent client connection and disconnection events.

## 4.1 Caching JMS Connections

As of Mule 3.6, JMS connections will cache Sessions and Producers by default. This replaces the previous method of configuring “Caching connection factory” in the Mule configuration. Whether or not JMS sessions are cached within a connection can be controlled using the `cacheJmsSessions` boolean property. This property is true by default in Mule 3.6.

On the consumer side, it is possible to control the amount of receive consumers throug the use of another jms connector property:

- numberOfConsumers
- numberOfConcurrentTransactedReceivers

The properties are mutually exclusive. Use the appropriate property depending on whether or not the messages are being received within a JMS transaction.

## 4.2 Resolving and Caching JMS Destinations on Send

When working with Solace JMS and using the Solace Appliance as the JNDI provider, it is also important to know that each JNDI lookup of a destination will result in a JNDI request to and response from the Solace appliance. As such, for efficient integration with Solace JMS, destinations should be cached and reused as much as possible. This is very important for producers to consider when sending messages.

Mule provides two easy ways to achieve this. The first is to use a `CachedJndiNameResolver` for all the JNDI lookups. This `CachedJndiNameResolver` would be a good option for applications that use a group of destinations and send large numbers of messages across this group of destinations. In this scenario, the JNDI destination lookup would occur once for each unique destination and then subsequent publishes would use the destination from the local cache avoiding the cost of the JNDI lookup. An example of a `<jms:connector>` using a `CachedJndiNameResolver` can be found in section 3.3.4 ”Using a JndiNameResolver”.

Alternatively, applications can often elect to bypass JNDI entirely for Topic and Queue lookups. In this cases, Topics and Queues use a provider specific String format which is mapped directly to the correct provider specific destination. To do this simply set `jndiDestinations="false"` in the `<jms:connector>` properties.

# 5 Working with Solace High Availability (HA)

The [Solace-JMS-REF] section “Establishing Connection and Creating Sessions” provides details on how to enable the Solace JMS connection to automatically reconnect to the standby appliance in the case of a HA failover of a Solace message router. By default Solace JMS connections will reconnect to the standby appliance in the case of an HA failover.

In general the Solace documentation contains the following note regarding reconnection:

> Note: When using HA redundant appliances, a fail-over from one appliance to its mate will typically occur in under 30 seconds, however, applications should attempt to reconnect for at least five minutes.

In section 3.2.4 Setting up Solace JNDI References, the Solace CLI commands correctly configured the required JNDI properties to reasonable values. These commands are repeated here for completeness.

```
config)# jndi message-vpn Solace_Mule_VPN
(config-jndi)# create connection-factory JNDI/CF/mule
(config-jndi-connection-factory)# property-list transport-properties
(config-jndi-connection-factory-pl)# property "reconnect-retry-wait" "3000"
(config-jndi-connection-factory-pl)# property "reconnect-retries" "20"
(config-jndi-connection-factory-pl)# property "connect-retries-per-host" "5"
(config-jndi-connection-factory-pl)# property "connect-retries" "1"
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)# exit
(config)#
```

# 6 Debugging Tips for Solace JMS API Integration

The key component for debugging integration issues with the Solace JMS API is the API logging that can be enabled. How to enable logging in the Solace API is described below.

## 6.1 Logging with Mule

As of Mule 3.6, Mule uses log4j2 to control applications logging. An overview can be found here:

- https://developer.mulesoft.com/docs/display/current/Logging+in+Mule

The following is an excerpt giving an overview of the logging in Mule:

> **Logging in Mule** (Mule 3.6.0 and Later)
>
> You can configure what gets logged, where it gets logged, and how by editing a configuration file that sits in your project.
>
> Mule logs multiple messages and specific elements in your applications' flows, in order to help you debug and keep track of events. You can also include the <logger> Element anywhere in a flow and set it up to output any message you want. By creating a configuration file, you can define what kinds of messages will be logged, in what way (asynchronously or synchronously), and where they get logged (e.g. to the console, to disk, to an endpoint or to a database).
>
> For logging, Mule ESB uses slf4j, which is a logging facade that discovers and uses a logging strategy from the classpath, such as Log4J2 or the JDK Logger. By default, Mule includes Log4J2, which is configured with a file called log4j2.xml.
>
> The Mule server has a log4j2.xml in its conf directory, which you can customize when running the server in standalone mode. Additionally, all the examples included with Mule have log4j2.xml files in their conf directories.

## 6.2 How to enable Solace JMS API logging

Since the Solace JMS API also makes use of the Jakarta Commons Logging API (JCL), configuring the Solace JMS API logging is very similar to configuring any other Mule logging. The following example shows how to enable debug logging in the Solace JMS API using log4j2.

One note to consider is that since the Solace JMS API has a dependency on the Solace Java API (JCSMP) both of the following logging components should be enabled and tuned when debugging to get full information. For example to set both to debug level:

```
<AsyncLogger name="com.solacesystems.jcsmp" level="DEBUG"/>
<AsyncLogger name="com.solacesystems.jms" level="DEBUG"/>
```

As outlined in the [Mule-REF] there are multiple options for enabling logging within Mule. In order to enable Solace JMS API logging, a user must do two things:

- Put Log4j2 on the classpath (For example: `src/main/resources`)

- Create a log4j2.xml configuration file to control logging

Below is an example log4j2 xml configuration file that will enable debug logging within the Solace JMS API.

```
<Configuration>
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%-5p %d [%t] %c: %m%n"/>
        </Console>
    </Appenders>

    <Loggers>
       <!-- Solace classes -->
       <AsyncLogger name="com.solacesystems.jcsmp" level="DEBUG"/>
       <AsyncLogger name="com.solacesystems.jms" level="DEBUG"/>

        <AsyncRoot level="INFO">
            <AppenderRef ref="Console"/>
        </AsyncRoot>
    </Loggers>
</Configuration>
```

With this you can get output in a format similar to the following which can help in understanding what is happening within the Solace JMS API.

```
INFO  2015-06-08 13:22:40,372 [main] com.solacesystems.jms.impl.ConsumerFactory: Creating
durable consumer for queue Q/requests
DEBUG 2015-06-08 13:22:40,372 [main] com.solacesystems.jcsmp.impl.RequestResponseTask:
RequestResponseTask ([BRT resource=Q/requests existingFH=null flowType=CONSUMER
counter=3]) startTimer
DEBUG 2015-06-08 13:22:40,372 [Context_2_ReactorThread]
com.solacesystems.jcsmp.impl.flow.BindRequestTask: Executing response handler.
DEBUG 2015-06-08 13:22:40,372 [Context_2_ReactorThread]
com.solacesystems.jcsmp.impl.RequestResponseTask: RequestResponseTask ([BRT
resource=Q/requests existingFH=null flowType=CONSUMER counter=3]) cancelTimer
DEBUG 2015-06-08 13:22:40,372 [Context_2_ReactorThread]
com.solacesystems.jcsmp.impl.timers.SubAckTimedTask: Creating SubAckTimedTask.
```

# 7 Advanced Topics

## 7.1 Authentication

JMS Client authentication is handled by the Solace appliance. The Solace appliance supports a variety of authentications schemes as described in [Solace-FG] in the Section “Client Authentication and Authorization”. The required JMS authentication properties can be set in the `jndiProviderProperties` configuration depending on which authentication scheme is being used. The following example shows how to enable basic authentication using JMS connection properties.

```
<jms:connector name="SolaceJMS" specification="1.1"
       username="${solace.username}"
       validateConnections="true"
       persistentDelivery="true"
       cacheJmsSessions="true"
       eagerConsumer="true"
       doc:name="JMS"
       forceJndiDestinations="true"
       jndiDestinations="true"
       connectionFactoryJndiName="${solace.jndi.cf}"
       jndiInitialFactory="com.solacesystems.jndi.SolJNDIInitialContextFactory"
       jndiProviderUrl="smf://${solace.host}">
       <spring:property name="jndiProviderProperties">
              <spring:map>
                     <spring:entry key="java.naming.security.principal"
                            value="${solace.username}@${solace.msgvpn}" />
                     <spring:entry key="java.naming.security.credentials"
                            value="${solace.password}" />
              </spring:map>
       </spring:property>
</jms:connector>
```

## 7.2 Using SSL Communication

This section outlines how to update the Solace message router and Mule configuration to enable secure connections between Mule and the Solace message router. For the purposes of illustration, this section uses a server certificate on the Solace message router and basic client authentication. It is possible to configure Solace JMS to use client certificates instead of basic authentication. This is done using configuration steps that are very similar to those outlined in this document. The [Solace-FP] and [Solace-JMS-REF] outline the extra configuration items required to switch from basic authentication to client certificates.

To change a Mule application from using a plain text connection to a secure connection, first the Solace message router configuration must be updated as outlined in Section 7.2.1 and the Solace JMS configuration within Mule must be updated as outlined in Section 7.2.2.

### 7.2.1 Configuring the Solace message router

To enable secure connections to the Solace message router, the following configuration must be updated on the Solace message router.

- Server Certificate
- TLS/SSL Service Listen Port
- Enable TLS/SSL over SMF in the Message VPN

The following sections outline how to configure these items.

#### 7.2.1.1 Configure the Server Certificate

Before, starting, here is some background detail on the server certificate required by the Solace message router. This is from the [Solace-FP] section “Setting a Server Certificate”

> To enable the exchange of information through TLS/SSL-encrypted SMF service, you must set the TLS/SSL server certificate file that the Solace message router is to use. This server certificate is presented to a client during the TLS/SSL handshakes. A server certificate used by an appliance must be an x509v3 certificate and it must include a private key. The server certificate and key use an RSA algorithm for private key generation, encryption and decryption, and they both must be encoded with a Privacy Enhanced Mail (PEM) format.
>
> The single server certificate file set for the appliance can have a maximum chain depth of three (that is, the single certificate file can contain up to three certificates in a chain that can be used for the certificate verification).

To configure the server certificate, first copy the server certificate to the Solace message router. For the purposes of this example, assume the server certificate file is named “mycert.pem”.

```
# copy sftp://[<username>@]<ip-addr>/<remote-pathname>/mycert.pem /certs
<username>@<ip-addr>'s password:
#
```

Then set the server certificate for the Solace message router.

```
(config)# ssl server-certificate mycert.pem
(config)#
```

#### 7.2.1.2 Configure TLS/SSL Service Listen Port

By default, the Solace message router accepts secure messaging client connections on port 55443. If this port is acceptable then no further configuration is required and this section can be skipped. If a non-default port is desired, then follow the steps below. Note this configuration change will disrupt service to all clients of the Solace message router and should therefore be performed during a maintenance window when this client disconnection is acceptable. This example assumes that the new port should be 55403.

```
(config)# service smf
(config-service-smf)# shutdown
All SMF and WEB clients will be disconnected.
Do you want to continue (y/n)? y
(config-service-smf)# listen-port 55403 ssl
(config-service-smf)# no shutdown
(config-service-smf)# exit
(config)#
```

#### 7.2.1.3 Enable TLS/SSL within the Message VPN

By default within Solace message VPNs both the plain-text and SSL services are enabled. If the Message VPN defaults remain unchanged, then this section can be skipped. However, if within the current application VPN, this service has been disabled, then for secure communication to succeed it should be enabled. The steps below show how to enable SSL within the SMF service to allow secure client connections from the Spring Framework.

```
(config)# message-vpn Solace_Mule_VPN
(config-msg-vpn)# service smf
(config-msg-vpn-service-smf)# ssl
(config-msg-vpn-service-ssl)# no shutdown
(config-msg-vpn-service-ssl)# end
#
```

### 7.2.2 Configuring Mule

The configuration in the Spring Framework requires updating the `SolaceJndiTemplate` bean in two ways.

- Updating the provider URL to specify the protocol as secure (smfs)
- Adding the required parameters for the secure connection

#### 7.2.2.1 Updating the provider URL

In order to signal to the Solace JMS API that the connection should be a secure connection, the protocol must be updated in the URI scheme. The Solace JMS API has a URI format as follows:

```
<URI Scheme>://[username]:[password]@<IP address>[:port]
```

Recall from Section 3.3, originally, the “jndiProviderUrl” was as follows:

```
<spring:property name="jndiProviderUrl" value="smf://${solace.host}"/>
```

This `jndiProviderUrl` was specified with URI scheme of “smf” which is the plaint-text method of communicating with the Solace message router as outlined in Section 3.1.2. To enable secure connectivity, the `jndiProviderUrl` must be updated to a URI scheme of “**smfs**”.

```
<spring:property name="jndiProviderUrl" value="smfs://${solace.host}"/>
```

#### 7.2.2.2 Adding SSL Related Configuration

Additionally, the Solace JMS API must be able to validate the server certificate of the Solace message router in order to establish a secure connection. To do this, the following trust store parameters need to be provided.

First the Solace JMS API must be given a location of a trust store file so that it can verify the credentials of the Solace message router server certificate during connection establishment. This parameter takes a URL or Path to the trust store file.

```
<spring:entry key="Solace_JMS_SSL_TrustStore" value="${solace.truststorePath}" />
```

It is also required to provide a trust store password. This password allows the Solace JMS API to validate the integrity of the contents of the trust store. This is done through the following parameter.

```
<spring:entry key="Solace_JMS_SSL_TrustStorePassword"
              value="${solace.truststorePassword}" />
```

There are multiple formats for the trust store file. By default Solace JMS assumes a format of Java Key Store (JKS). So if the trust store file follows the JKS format then this parameter may be omitted. Solace JMS supports two formats for the trust store: “jks” for Java Key Store or “pkcs12”. Setting the trust store format is done through the following parameter.

```
<spring:entry key="Solace_JMS_SSL_TrustStoreFormat" value="jks" />
```

And finally, the authentication scheme must be selected. Solace JMS supports the following authentication schemes for secure connections:

- AUTHENTICATION_SCHEME_BASIC
- AUTHENTICATION_SCHEME_CLIENT_CERTIFICATE

This integration example will use basic authentication. So the required parameter is as follows:

```
<spring:entry key="Solace_JMS_Authentication_Scheme"
              value="AUTHENTICATION_SCHEME_BASIC" />
```

#### 7.2.2.3 Spring Configuration Bean

The following example outlines all of the required configuration changes to the Mule JMS connector where the user should substitute appropriate values for the following or use properties.

- ${solace.host}
- ${solace.truststorePath}
- ${solace.truststorePassword}

The example use the JndiNameResolver format for the JMS connector. The connection properties format also works and follows a similar format in terms of required changes.

```
<jms:connector name="SolaceJMS" specification="1.1"
       username="${solace.username}" validateConnections="true"
       connectionFactoryJndiName="${solace.jndi.cf}"
       persistentDelivery="true"
       cacheJmsSessions="true"
       eagerConsumer="true"
       jndiDestinations="true"
       forceJndiDestinations="true"
       doc:name="JMS">
       <jms:custom-jndi-name-resolver
           class="org.mule.transport.jms.jndi.CachedJndiNameResolver">
            <spring:property name="jndiInitialFactory"
value="com.solacesystems.jndi.SolJNDIInitialContextFactory"/>
            <spring:property name="jndiProviderUrl" value="smfs://${solace.host}"/>
            <spring:property name="jndiProviderProperties">
               <spring:map>
                 <spring:entry key="java.naming.security.principal"
                              value="${solace.username}@${solace.msgvpn}" />

                 <!-- SSL Related Configuration -->
                 <spring:entry key="Solace_JMS_Authentication_Scheme"
                              value="AUTHENTICATION_SCHEME_BASIC" />
                 <spring:entry key="Solace_JMS_SSL_TrustStore"
                              value="${solace.truststorePath}" />
                 <spring:entry key="Solace_JMS_SSL_TrustStoreFormat"
                              value="jks" />
                 <spring:entry key="Solace_JMS_SSL_TrustStorePassword"
                              value="${solace.truststorePassword}" />

               </spring:map>
            </spring:property>
       </jms:custom-jndi-name-resolver>
</jms:connector>
```

## 7.3 Working with XA Transactions

This section demonstrates how to configure the Solace message router to support the XA transaction processing capabilities of Mule. In addition, code examples are provided showing JMS message consumption and production over XA transactions within a Mule flow.

XA transactions are supported in the general-availability release of SolOS version 7.1.

In addition to the standard XA Recovery functionality provided through the Mule, SolOS version 7.1 provides XA transaction administration facilities in the event that customers must perform manual failure recovery. Refer to the document [Solace-FP], specifically the section “Making Heuristic Decisions on Transactions” for full details on administering and configuring XA Transaction support on the Solace Message Router.

### 7.3.1 Enabling XA Support for JMS Connection Factories – Solace Message Router

To enable XA transaction support for specific JMS connection factories the customer must configure XA support for the respective JNDI connection factory on the Solace Message Router:

```
(config)# jndi message-vpn Solace_Mule_VPN
(config-jndi)# connection-factory JNDI/CF/mule
(config-jndi-connection-factory)# property-list messaging-properties
(config-jndi-connection-factory-pl)# property xa true
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)#
```

### 7.3.2 Using XA Transactions in Mule Flows

The [Mule-REF] has a good summary of the Mule support for transactions. Specifically the following link:

- https://developer.mulesoft.com/docs/display/current/Transaction+Management

To use XA transactions in a Mule flow, you need to configure the following items:

- A Mule Transaction Manager in the global context.
- Apply transactions to connectors or through scopes.

#### 7.3.2.1 Mule Transaction Manager

Currently the recommended Mule transaction manager is Bitronix which is included in the Mule distrbituion. To enable a Bitronix transaction manager, simply add the following to the Mule configuration.

```
<bti:transaction-manager/>
```

For more details refer to the Mule documentation here:

- https://developer.mulesoft.com/docs/display/current/Using+Bitronix+to+Manage+Transactions

#### 7.3.2.2 Solace requirements for XA transactions in the JMS Connector

To integrate correctly with the XA transactions support provided in the general-availability release of SolOS version 7.1, it is important to set the `numberOfConcurrentTransactedReceivers` to 1. If more than one concurrent transacted received is required in an application, use multiple JMS connectors. It is expected that this requirement will be removed in a future version of SolOS. Setting the number of concurrent transacted receviers is done as follows:

```
<jms:connector name="SolaceJMS" specification="1.1"
       username="${solace.username}"
       connectionFactoryJndiName="${solace.jndi.cf}"
       <!-- ... Snip unrelated generic configuration ... -->
       numberOfConcurrentTransactedReceivers="1"
       doc:name="JMS">
       <jms:custom-jndi-name-resolver
           class="org.mule.transport.jms.jndi.CachedJndiNameResolver">
            <spring:property name="jndiInitialFactory"
value="com.solacesystems.jndi.SolJNDIInitialContextFactory"/>
            <spring:property name="jndiProviderUrl" value="smfs://${solace.host}"/>
            <spring:property name="jndiProviderProperties">
               <spring:map>
                 <spring:entry key="java.naming.security.principal"
                               value="${solace.username}@${solace.msgvpn}" />
                 </spring:map>
            </spring:property>
       </jms:custom-jndi-name-resolver>
</jms:connector>
```

#### 7.3.2.3 Applying a Transaction to a Connector

In addition to configuring the JMS connector, the inbound endpoint of the flow must be updated to use XA transactions. Adding the `<xa-transaction>` child element to a `<jms:inbound-enboint>` is sufficient.

```
<flow name="receivingFromSolace">
    <jms:inbound-endpoint connector-ref="SolaceJMS"
           doc:name="JMS" queue="${solace.jndi.queue}">
           <xa-transaction action="BEGIN_OR_JOIN" timeout="60000"/>
    </jms:inbound-endpoint>
</flow>
```

#### 7.3.2.4 Applying a Transaction as a Scope

Transaction scopes are a good way to group outbound actions together in an XA transaction. To enable a transaction scope in a flow simply next the operations within an `<ee:xa-transactional>` element. The following is an example:

```
<flow name="sendingToSolace">
    <http:listener config-ref="HTTP_Listener_Configuration" path="/sendMessage"
doc:name="HTTP"/>
    <ee:xa-transactional action="BEGIN_OR_JOIN" doc:name="Transactional">
       <set-payload value="#[message.inboundProperties.'http.query.params'.message]"
            doc:name="Set Payload"/>
        <jms:outbound-endpoint queue="${solace.jndi.queue}"
            connector-ref="SolaceJMS_cached"
            doc:name="Send to Solace"/>
    </ee:xa-transactional>
</flow>
```

## 7.4 Working with the Solace Disaster Recovery Solution

The [Solace- FG] section “Data Center Replication” contains a sub-section on “Application Implementation” which details items that need to be considered when working with Solace’s Data Center Replication feature.

In general a disaster recovery event is a major outage which requires manual intervention to failover. Often there is significant manual involvement again to bring up all the applications in the DR site to resume normal operations. This is why the Solace disaster recovery features involves a manual step to switch over to the DR location although this can be automated if required.

After a DR switch over of the messaging infrastructure, then applications can reconnect and continue operating. Since most applications will co-exist in the same datacenter as the Solace message routers, these applications will have to be restarted in the DR location. These applications can start up and connect to the DR Solace message routers through DNS which avoids the need for any reconfiguration of the applications in the event of a DR switchover. There is essentially nothing to be done to prepare these applications for a DR failover. It is sufficient to update the DNS and start the applications in the DR location.

However, it is possible to have applications transition automatically in a DR failover for applications that required this. This integration guide will outline these details as they apply to integration with Mule and is divided into the following subsections:

- Disaster Recovery Behavior Notes and Mule Connector Reconnection
- Configuring a Host List within the Mule JMS Connector

### 7.4.1 Disaster Recovery Behavior Notes

When a disaster recovery switch-over occurs, the Solace JMS API must establish a new connection to the Solace message routers in the standby data center. Because this is a new connection there are some special considerations worth noting. The [Solace-FG] contains the following notes:

> Java and JMS APIs
>
> For client applications using the Java or JMS APIs, any sessions on which the clients have published Guaranteed messages will be destroyed after the switch-over. To indicate the disconnect and loss of publisher flow:

- The Java API will generate an exception from the `JCSMPStreamingPublishCorrelatingEventHandler.handleErrorEx()` that contains a subcode of `JCSMPErrorResponseSubcodeEx.UNKNOWN_FLOW_NAME`.
- The JMS API will generate an exception from the `javax.jms.ExceptionListener` that contains the error code `SolJMSErrorCodes.EC_UNKNOWN_FLOW_NAME_ERROR`.

Upon receiving these exceptions the client application will know to create a new session.

After a new session is established, the client application can republish any Guaranteed messages that had been sent but not acked on the previous session, as these message might not have been persisted and replicated.

To avoid out-of-order messages, the application must maintain an unacked list that is added to before message publish and removed from on receiving an ack from the appliance. If a connection is re-established to a different host in the hostlist, the unacked list must be resent before any new messages are published.

Note: When sending persistent messages using the JMS API, a producer's send message will not return until an acknowledgment is received from the appliance. Once received, it is safe to remove messages from the unacked list.

Alternatively, if the application has a way of determining the last replicated message—perhaps by reading from a last value queue—then the application can use that to determine where to start publishing.

For integration with Mule it is recommended to use a reconnection strategy as outlined in [Mule-REF]. The most basic reconnection strategy is the infite retry. To enable this, simply add the `reconnect-forever` element to the JMS connector in the Mule configuration.

```
<jms:connector name="SolaceJMS" specification="1.1"
       username="${solace.username}"
       connectionFactoryJndiName="${solace.jndi.cf}"
       <!-- ... Snip unrelated generic configuration ... -->
       doc:name="JMS">
       <reconnect-forever/>
       <jms:custom-jndi-name-resolver
           class="org.mule.transport.jms.jndi.CachedJndiNameResolver">
            <spring:property name="jndiInitialFactory"
value="com.solacesystems.jndi.SolJNDIInitialContextFactory"/>
           <spring:property name="jndiProviderUrl" value="smfs://${solace.host}"/>
           <spring:property name="jndiProviderProperties">
              <spring:map>
                <spring:entry key="java.naming.security.principal"
                              value="${solace.username}@${solace.msgvpn}" />
                </spring:map>
           </spring:property>
       </jms:custom-jndi-name-resolver>
</jms:connector>
```

### 7.4.2 Configuring a Host List within the Mule JMS Connector

As described in [Solace-FG], the host list provides the address of the backup data center. This can be configured within the Mule JMS Connector through the `jndiProviderUrl` configuration property value as follows:

```
<spring:property name="jndiProviderUrl"
                 value="smf://${solace.host1},smf://${solace.host2}"/>
```

The active site and standby site addresses are provided as a comma-separated list. When connecting, the Solace JMS connection will first try the active site and if it is unable to successfully connect to the active site, then it will try the standby site. This is discussed in much more detail in the referenced Solace documentation.

# 8 Appendix – Mule Configuration Reference

## 8.1 MuleConfiguration.xml

```
<?xml version="1.0" encoding="UTF-8"?>


<mule xmlns:wmq="http://www.mulesoft.org/schema/mule/ee/wmq"
xmlns:http="http://www.mulesoft.org/schema/mule/http"
xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
xmlns="http://www.mulesoft.org/schema/mule/core"
xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
       xmlns:spring="http://www.springframework.org/schema/beans" version="EE-3.6.1"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-current.xsd
http://www.mulesoft.org/schema/mule/core
http://www.mulesoft.org/schema/mule/core/current/mule.xsd
http://www.mulesoft.org/schema/mule/http
http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd
http://www.mulesoft.org/schema/mule/jms
http://www.mulesoft.org/schema/mule/jms/current/mule-jms.xsd">
    <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0"
port="${http.port}" doc:name="HTTP Listener Configuration"/>
<jms:connector name="SolaceJMS" specification="1.1" username="${solace.username}"
validateConnections="true" connectionFactoryJndiName="${solace.jndi.cf}"
persistentDelivery="true" cacheJmsSessions="true" eagerConsumer="true" doc:name="JMS"
forceJndiDestinations="true" jndiDestinations="true"
jndiInitialFactory="com.solacesystems.jndi.SolJNDIInitialContextFactory"
jndiProviderUrl="smf://${solace.host}">
<spring:property name="jndiProviderProperties">
       <spring:map>
              <spring:entry key="java.naming.security.principal"
                     value="${solace.username}@${solace.msgvpn}" />
       </spring:map>
</spring:property>
</jms:connector>
    <flow name="sendingToSolace">
              <http:listener config-ref="HTTP_Listener_Configuration"
path="/sendMessage" doc:name="HTTP"/>
              <set-payload
value="#[message.inboundProperties.'http.query.params'.message]" doc:name="Set Payload"/>
       <jms:outbound-endpoint queue="${solace.jndi.queue}" connector-ref="SolaceJMS"
doc:name="Send to Solace"/>
```

```
    </flow>
    <flow name="receivingFromSolace">
                <jms:inbound-endpoint connector-ref="SolaceJMS" doc:name="JMS"
queue="${solace.jndi.queue}"/>
        <logger level="INFO" message="Processing Solace JMS message: #[payload]"
doc:name="Logger" />
    </flow>
</mule>
```