Solace JMS Integration with WebLogic Application Server 12c

Document Version 1.1

July 2016

This document is an integration guide for using Solace JMS (starting with version 7.2) as a JMS provider for WebLogic Sphere Application Server.

The WebLogic Application Server provides a comprehensive framework for application and integration middleware that is compliant with the Java Enterprise Edition computing platform. Solace provides a Java Connector Architecture (JCA) compliant Resource Adapter that may be deployed to the WebLogic application server providing enterprise applications with connectivity to the Solace message router.

Solace message routers unify many kinds of data movement so companies can efficiently and cost-effectively move all of the information associated with better serving customers and making smarter decisions. The Solace 3560 message router is the highest performance data movement technology available, with the capacity and robustness to support the most demanding enterprise messaging, big data, cloud computing and Internet of Things applications.

Solace Systems®

http://www.solacesystems.com

Table of Contents

1 Overview

This document demonstrates how to integrate Solace Java Message Service (JMS) with the WebLogic Application Server 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 both the application server and Solace JMS.

The target audience of this document is developers using the WebLogic Application Server with knowledge of both the WebLogic Application Server and JMS in general. As such this document focuses on the technical steps required to achieve the integration.

Note this document provides instructions on configuring and deploying the Solace JCA 1.5 resource adapter using the web console application of WebLogic, as well as instructions on configuring Solace as a Foreign JMS Provider Module. For detailed background on either Solace JMS or the WebLogic Application Server refer to the referenced documents below.

This document is divided into the following sections to cover the Solace JMS integration with WebLogic Application Server:

- Integrating with WebLogic Application Server
- Performance Considerations
- Working with Solace High Availability
- Debugging Tips
- Advanced Topics including:
    - Using SSL Communication
    - Working with XA Transactions
    - Working with 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
[WL-REF]	WebLogic Application Server Information	http://www.oracle.com/technetwork/middleware/web

Document ID	Document Title	Document Source
	Library	logic/documentation/index.html
[JCA-1.5]	Java Connector Architecture v1.5	Community Development of Java Technology Specifications (JSR)

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 WebLogic Application Server

Solace provides a JCA compliant resource adapter for integrating Java enterprise applications with the Solace JMS message router. There are two options for integrating Solace with WebLogic for use by Java enterprise applications including embedded and stand-alone deployment. This section will cover instructions on configuring the Resource Adapter Archive (RAR) file for stand-alone deployment, and configuring Solace as a Foreign JMS Provider

In order to illustrate WebLogic Application Server integration, the following sections will highlight the required WebLogic configuration changes and provide sample code for sending and receiving messages using Enterprise Java Beans.

This EJB sample consists of two enterprise beans, a Message Driven Bean and a Session Bean. The MDB is configured to receive a message on a 'requests' Queue. When the MDB receives a message it then calls a method of the Session Bean to send a reply message to a 'reply' Queue. The EJB sample requires configuration of various J2C entities in WebLogic to support usage of the Solace JCA compliant resource adapter.

The following steps are required to accomplish the above goals of sending and receiving messages using the Solace JMS message router.

3.1 Description of Resources Required

The Solace JCA 1.5 resource adapter is provided as a standalone RAR file and is versioned together with a specific release of the Solace JMS API. The JMS API libraries are bundled inside a single resource adapter RAR file for deployment to the WebLogic application server.

Resource	File Location
Solace JCA 1.5 resource adapter stand-alone RAR file	• Developer Portal: http://dev.solacesystems.com/downloads/ • Solace Customer SFTP: https://sftp.solacesystems.com/~<customer>/<version>/Topic_Routing/APIs/JMS/Current/<release>/sol-jms-ra-<release>.rar

Table 2 –Solace Resource Adapter Requirements

This integration guide will demonstrate the creation of Solace resources and the configuration of the WebLogic Application Server's managed resources. The section below outlines the resources that are created and used in the subsequent sections.

Solace Resource Naming Convention

To illustrate this integration example, all named resources created on the Solace appliance will have the following prefixes:

Resource	Prefix
Non-JNDI resource	**solace**_<resource name>
JNDI names	**JNDI/Sol/**<resource name>

Table 3 – Solace Resource Naming Convention

Solace Resources

The following Solace message router resources are required for the integration sample in this document.

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 a client will use when connecting to the Solace message router to send and receive message. This document uses a value of __IP:PORT__.
Message VPN	solace_VPN	A Message VPN, or virtual message broker, to scope the integration on the Solace message router.
Client Username	solace_user	The client username.
Client Password	solace_password	Optional client password.
Solace Queue	solace_requests	Solace destination for messages consumed by JEE enterprise application
Solace Queue	solace_replies	Solace destination for messages produced by JEE enterprise application
JNDI Connection Factory	JNDI/Sol/CF	The JNDI Connection factory for controlling Solace JMS connection properties
JNDI Queue Name	JNDI/Sol/Q/requests	The JNDI name of the queue used in the samples
JNDI Queue Name	JNDI/Sol/Q/replies	The JNDI name of the queue used in the samples

Table 4 – Solace Configuration Resources

Application Server Resource Naming Convention

To illustrate this integration example, all named resources created in WebLogic application server will have the following prefixes:

Resource	Prefix
Non-JNDI resource	**j2c**_<resource name>
JNDI names	**JNDI/J2C/**<resource name>

Table 5 – Application Server Resource Naming Convention

Application Server Resources

The following WebLogic application server resources are required for the integration example in this document.

Resource	JNDI Name	JNDI Value	Description
Resource Adapter	N/A	N/A	The *Solace JMS Resource Adapter* packaged as a Standalone RAR package (Implements a JCA compliant Resource Adapter)
J2C connection factory	j2c_cf	JNDI/J2C/CF	A J2C entity used to access a Solace javax.jms.ConnectionFactory (For Outbound messaging)
J2C administered object	j2c_request_queue	JNDI/J2C/Q/requests	A J2C entity used to perform a JNDI lookup of a javax.jms.Queue on the Solace message router
J2C administered object	j2c_reply_queue	JNDI/J2C/Q/replies	A J2C entity used to perform a JNDI lookup of a javax.jms.Queue on the Solace message router

Table 6 – WebLogic Configuration Resources

3.2 Solace JMS provider Configuration

The following entities on the Solace message router need to be configured at a minimum to enable JMS to send and receive messages within the WebLogic Application Server.

- 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 and sending 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 7.2 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 "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.

Creating a Message VPN

This section outlines how to create a message-VPN called "solace_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 WebLogic Application Server configuration when connecting to the Solace messaging appliance. 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_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_VPN
(config-message-spool)# max-spool-usage 2000
(config-message-spool)# exit
(config)#
```

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 for the XA-transactions capable Solace JCA Resource Adapter.

The chosen client username of "solace_user" will be required by the WebLogic Application Server when connecting to the Solace message router.

```
(config)# client-profile default message-vpn solace_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-guaranteed-endpoint-create
(config-client-profile)# message-spool allow-transacted-sessions
(config-client-profile)# exit
(config)#
(config)# create client-username solace_user message-vpn solace_VPN
(config-client-username)# acl-profile default
(config-client-username)# client-profile default
(config-client-username)# no shutdown
(config-client-username)# exit
(config)#
```

Setting up Guaranteed Messaging Endpoints

This integration guide shows receiving messages and sending reply messages within the WebLogic Application Server using two separate JMS Queues. For illustration purposes, these queues are chosen to be exclusive queues with a message spool quota of 2GB matching quota associated with the message VPN. The queue names chosen are "solace_requests" and "solace_replies".

```
(config)# message-spool message-vpn solace_VPN
(config-message-spool)# create queue solace_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)# create queue solace_replies
(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)#
```

Setting up Solace JNDI References

To enable the JMS clients to connect and look up the queue destination required by WebLogic Application Server, there are three JNDI objects required on the Solace message router:

- A connection factory: JNDI/Sol/CF
  - Note: Ensure 'direct-transport' is disabled for JMS persistent messaging.
- A queue destination: JNDI/Sol/Q/requests

- A queue destination: JNDI/Sol/Q/replies

They are configured as follows:

```
(config)# jndi message-vpn solace_VPN
(config-jndi)# create connection-factory JNDI/Sol/CF
(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/Sol/Q/requests
(config-jndi-queue)# property physical-name solace_requests
(config-jndi-queue)# exit
(config-jndi)#
(config-jndi)# create queue JNDI/Sol/Q/replies
(config-jndi-queue)# property physical-name solace_replies
(config-jndi-queue)# exit
(config-jndi)#
(config-jndi)# no shutdown
(config-jndi)# exit
(config)#
```

3.3 (Option 1) Deploying Solace JMS Resource Adapter

Solace provides a JCA compliant Resource Adapter that can be deployed to the WebLogic application server, allowing Enterprise Java Beans to connect to Solace through a standard JCA interface. This integration guide outlines the steps to deploy the resource adapter which is provided by Solace as a packaged stand-alone RAR file. This is the recommended way of integrating Solace with WebLogic as it provides support for XA transactions.

The following Java system property must be configured in the application server JVM properties:

```
-Dweblogic.mdb.JMSProviders.NeedContinuousPolling=soljms
```

This allows WebLogic to function correctly with a non-Oracle JMS provider.

Depending on your environment, this can be done by editing the startWebLogic.cmd or startWebLogic.sh file, which can be found in <WL_domain>/bin folder. Add the following at the start of the script:

JAVA_OPTIONS="-Dweblogic.mdb.JMSProviders.NeedContinuousPolling=soljms ${JAVA_OPTIONS}"

Add the Solace JMS jar files to the WebLogic libraries

The following steps will make the Solace resource adapter have access to the Solace JMS libraries.

Steps to place the Solace libraries into the WebLogic CLASSPATH:

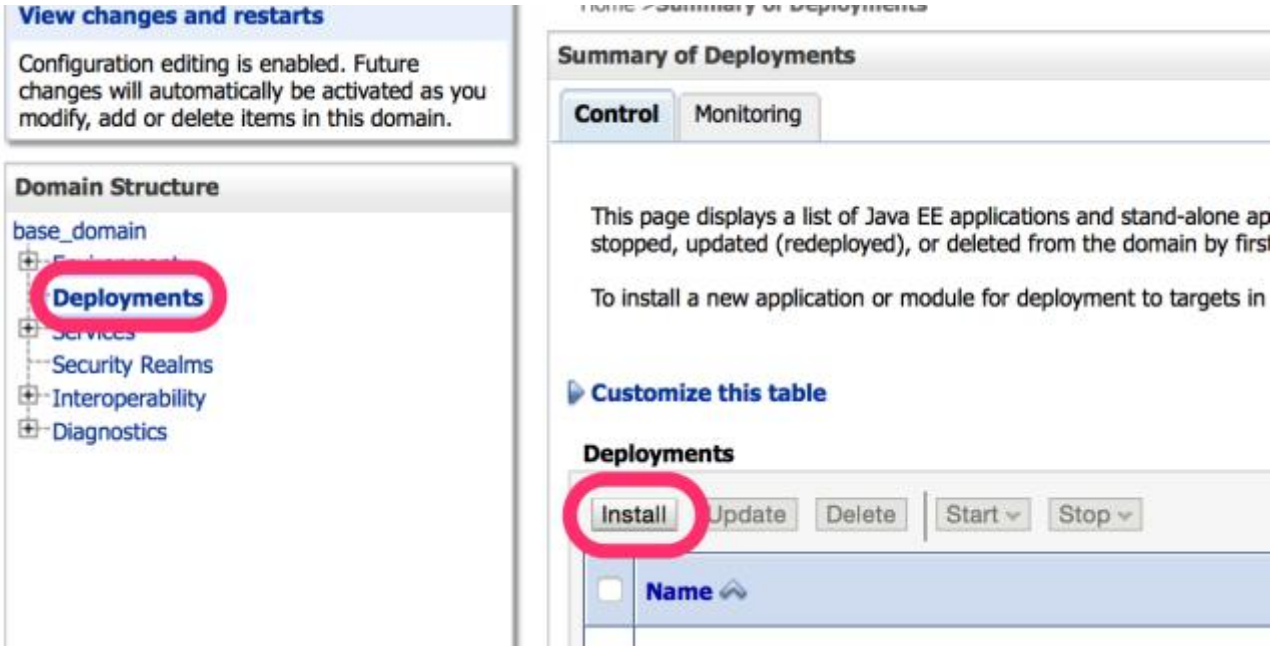
1. Expand the Solace JMS Resource Adapter .rar file.
2. Copy all of the .jar files to the <WL_domain>/lib folder on the WebLogic server. On startup, WebLogic automatically detects and appends files in this folder to the CLASSPATH.

Adding the Resource Adapter as a Deployment

The following steps will make the Solace resource adapter available to all enterprise applications (Refer to Section 3.1 Description of Resources Required for the file location of the Solace JCA 1.5 Resource Adapter RAR file).

Steps to deploy the Solace JMS Resource Adapter:

1. Log into the WebLogic Application Server administrative console. This is defaulted to http://localhost:7001/console
2. Click on the 'Deployments' link in the navigation pane and in the 'Deployments' table, click on the '*Install*' button



3. In the *Install Application Assistant* page, select the file path to the Solace JMS Resource Adapter RAR (.rar) file, then click on the '*Next*' button
4. Select "Install this deployment as an application" and click the 'Next' button
5. Assign the adapter a suitable name, such as sol-jms-ra-<version>
6. Click the 'Finish' link to commit the changes to the application server
7. Navigate to the 'Deployments' page again and click on the adapter name.

Configuring the connection to Solace JMS provider

Connecting to the Solace message router through the Solace JMS Resource Adapter requires configuration of additional resources in WebLogic. Two key pieces of information are required including connectivity information to the Solace message router and client authentication data.

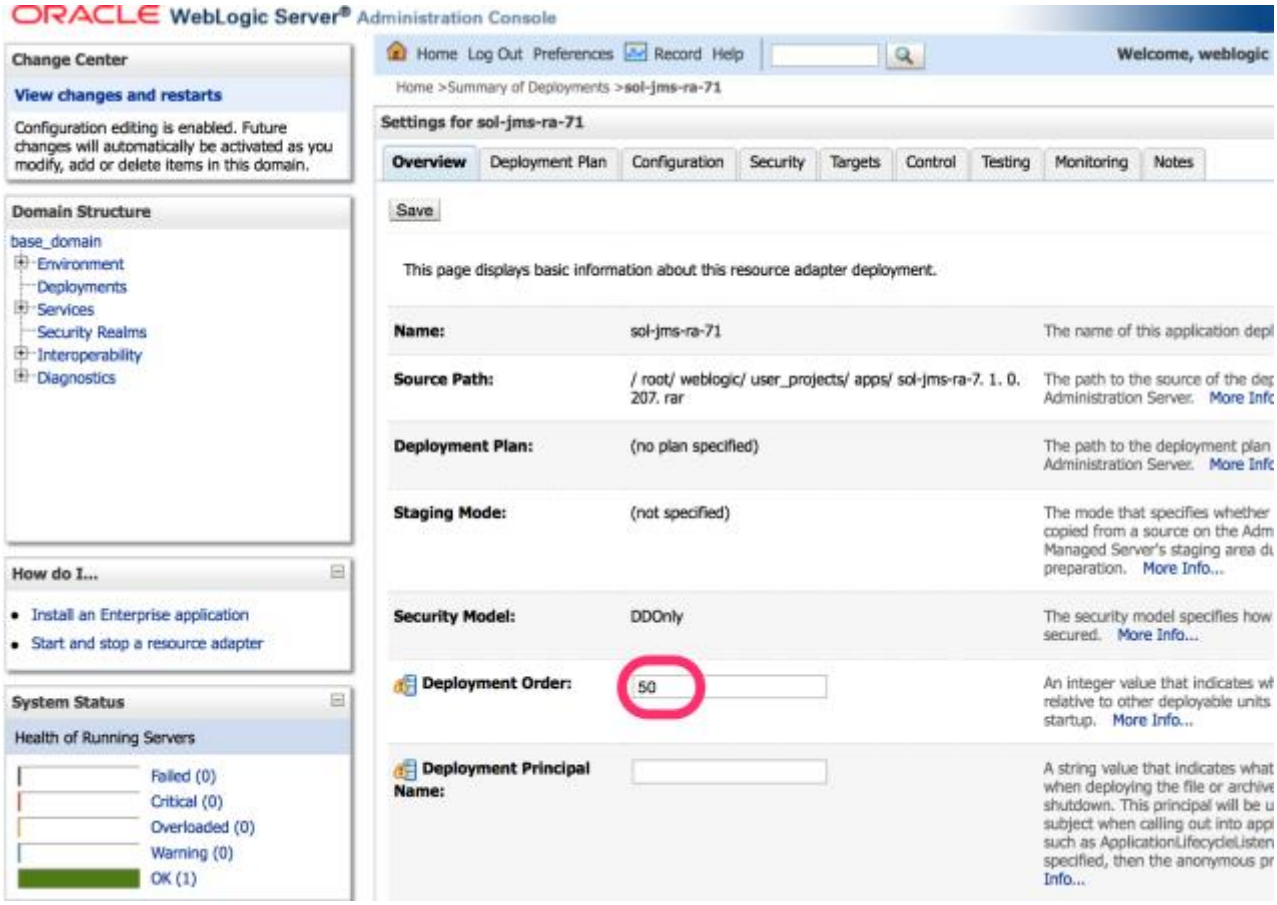
The above information is specified across one or more J2C entities depending on your application's JMS message flow (Inbound, Outbound, or both).

The Solace resource adapter includes several custom properties for specifying connectivity and authentication details to the Solace appliance. Setting these properties at the Resource Adapter level makes the information available to all child J2C entities like J2C connection factory, and J2C administered objects. The properties can also be overridden at the J2C entity level allowing connectivity to multiple Solace message routers.

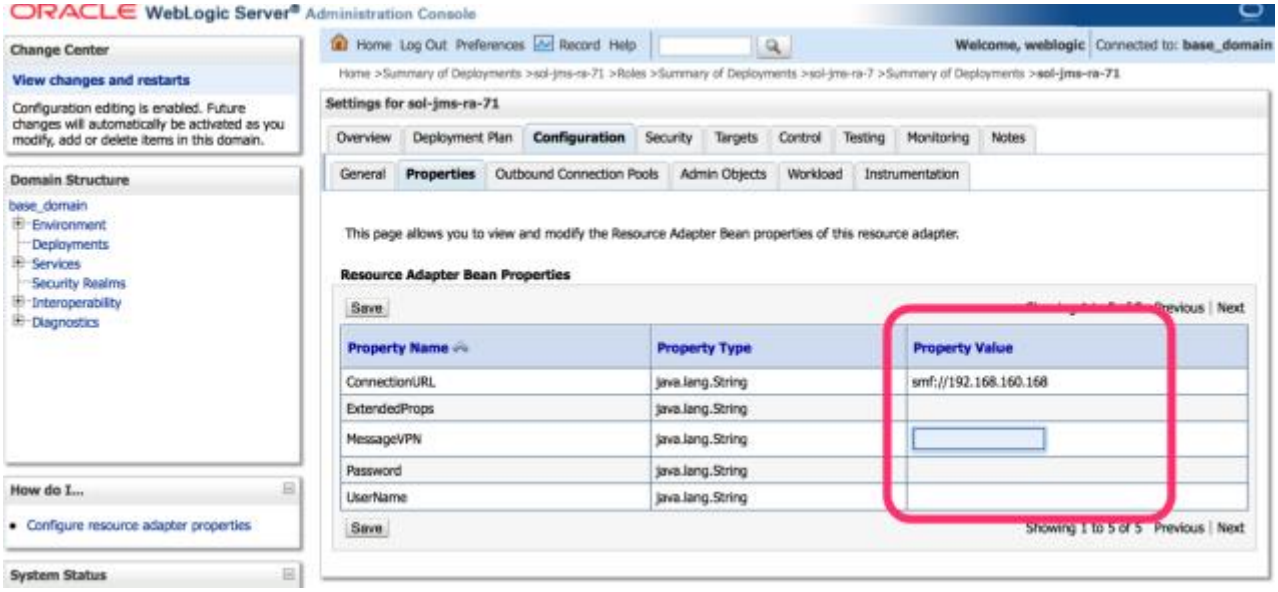
Please refer to [WL-REF] and [JCA-1.5] for more details on configuring general JEE authentication options. *Section 0 7.1* Authentication discusses configuration of Solace specific authentication in more detail.

Steps to configure the Solace JMS Resource Adapter:

1. Log into the WebLogic Application Server administrative console.
2. Click on the 'Deployments' link in the navigation pane
3. Click on the resource adapter to begin its configuration.
4. In the 'Overview' tab, modify the deployment order to ensure that the adapter has a lower value than any application that will depend on the JMS connection provided by the RA. This is to ensure that the RA is deployed first.



5. In the 'Configuration > Properties' tab, edit the connectivity properties for the Solace JMS Resource Adapter. Values entered here will be inherited by the adapter's outbound connection pools and admin objects. Values are entered by clicking in the "Property Value" column and confirming each value with the Enter key.
    a. Update the value for the properties 'ConnectionURL', 'UserName', 'Password', and 'MessageVPN':



    i. Click on the '*ConnectionURL*' property and specify the value '***smf://__IP:Port__***' (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP).
    ii. Click on the '*MessageVPN*' property and specify the value corresponding to the Solace message VPN ('***solace_VPN***' for this example). Press Enter to input the change.
    iii. Click on the '*UserName*' property and specify the value corresponding to the Solace username ('***solace_user***' for this example). Press Enter to input the change.
    iv. Click on the 'Password' property and specify the value for the Solace username if required (**Note**, in this example, Section 0 Configuring Client Usernames & Profiles specified a Solace authentication type of 'none', so the password here will be ignored).
    v. Click on the '*Save*' button to commit the changes to the application server.
    vi. When the server prompts for a Deployment Plan to be created, configure its desired location. More details on deployment plan files can be found at the following location:

Configuring Applications for Production Deployment:
http://docs.oracle.com/cd/E12839_01/web.1111/e13702/config.htm#DEPGD169

The following table summarizes the values used for the resource adapter's bean properties.

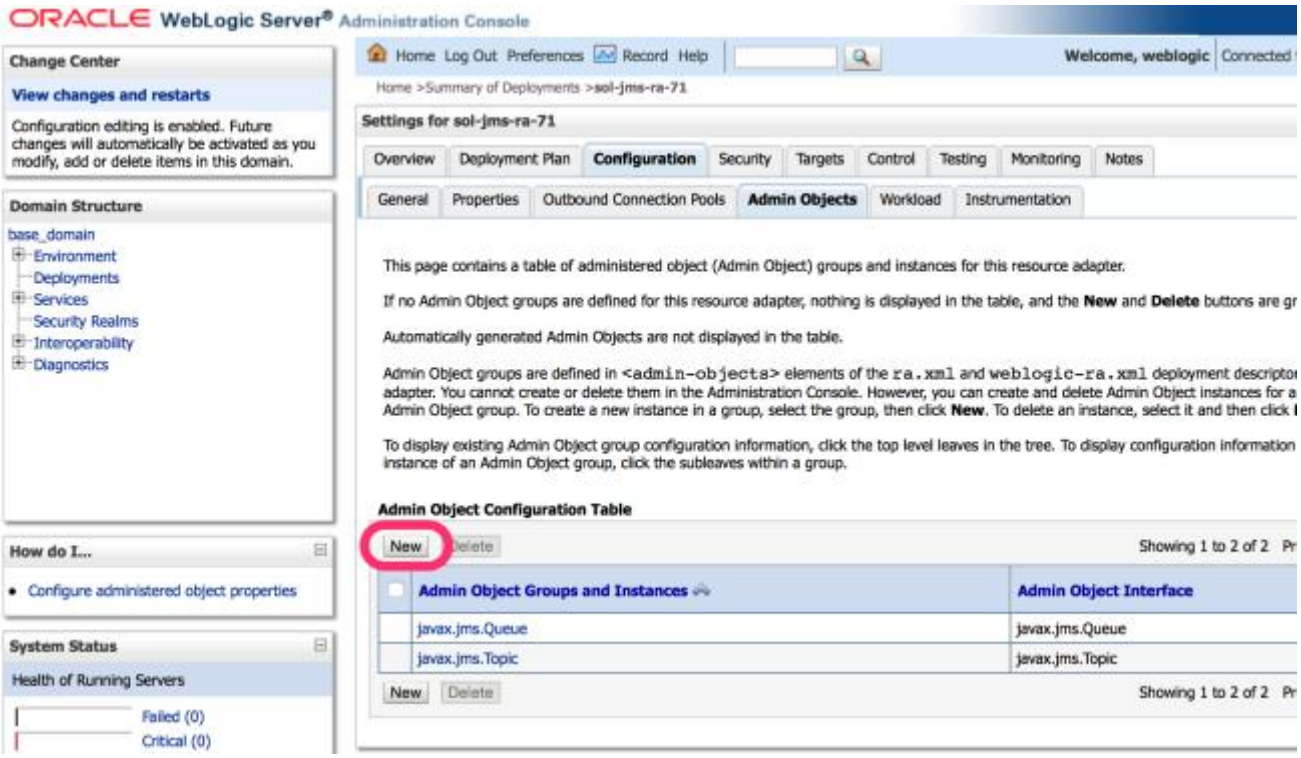
Name	Value	Description
ConnectionURL	smf://__IP:Port__	The connection URL to the Solace message router of the form: smf://__IP:Port__ (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP)
messageVPN	solace_VPN	A Message VPN, or virtual message broker, to scope the integration on the Solace message router.

UserName	solace_user	The client username credentials on the Solace appliance
Password	default	Password of the Client Username on the Solace appliance
ExtendedProps		Comma-seperated list for advanced control of the connection. For this example, leave empty. Supported values include: • Solace_JMS_Authentication_Scheme • Solace_JMS_CompressionLevel • Solace_JMS_JNDI_ConnectRetries • Solace_JMS_JNDI_ConnectRetriesPerHost • Solace_JMS_JNDI_ConnectTimeout • Solace_JMS_JNDI_ReadTimeout • Solace_JMS_JNDI_ReconnectRetries • Solace_JMS_JNDI_ReconnectRetryWait • Solace_JMS_SSL_ValidateCertificateDate • Solace_JMS_SSL_ValidateCertificate • Solace_JMS_SSL_CipherSuites • Solace_JMS_SSL_KeyStore • Solace_JMS_SSL_KeyStoreFormat • Solace_JMS_SSL_KeyStorePassword • Solace_JMS_SSL_PrivateKeyAlias • Solace_JMS_SSL_PrivateKeyPassword • Solace_JMS_SSL_ExcludedProtocols • Solace_JMS_SSL_TrustStore • Solace_JMS_SSL_TrustStoreFormat • Solace_JMS_SSL_TrustStorePassword • Solace_JMS_SSL_TrustedCommonNameList Ex: Solace_JMS_CompressionLevel=9

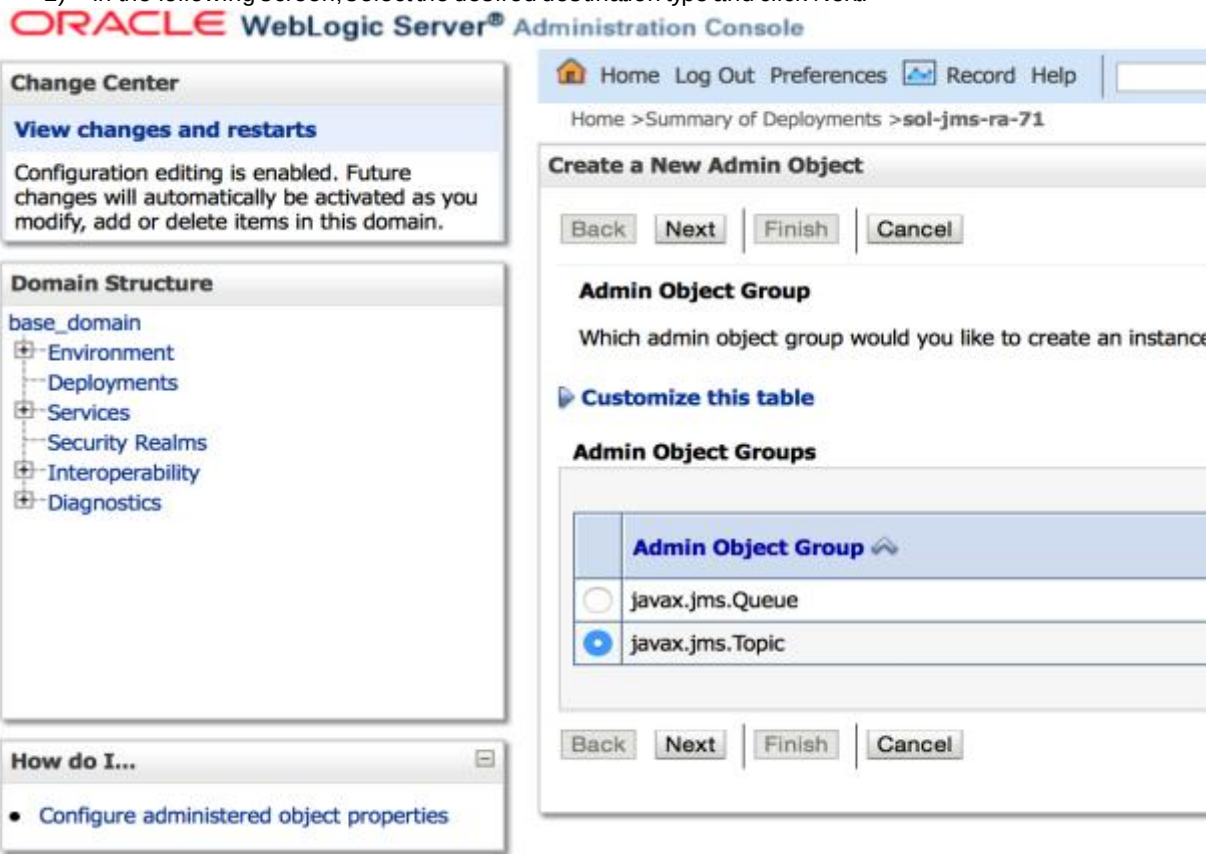
Table 7 – Resource Adapter Custom Properties

Configuring destinations

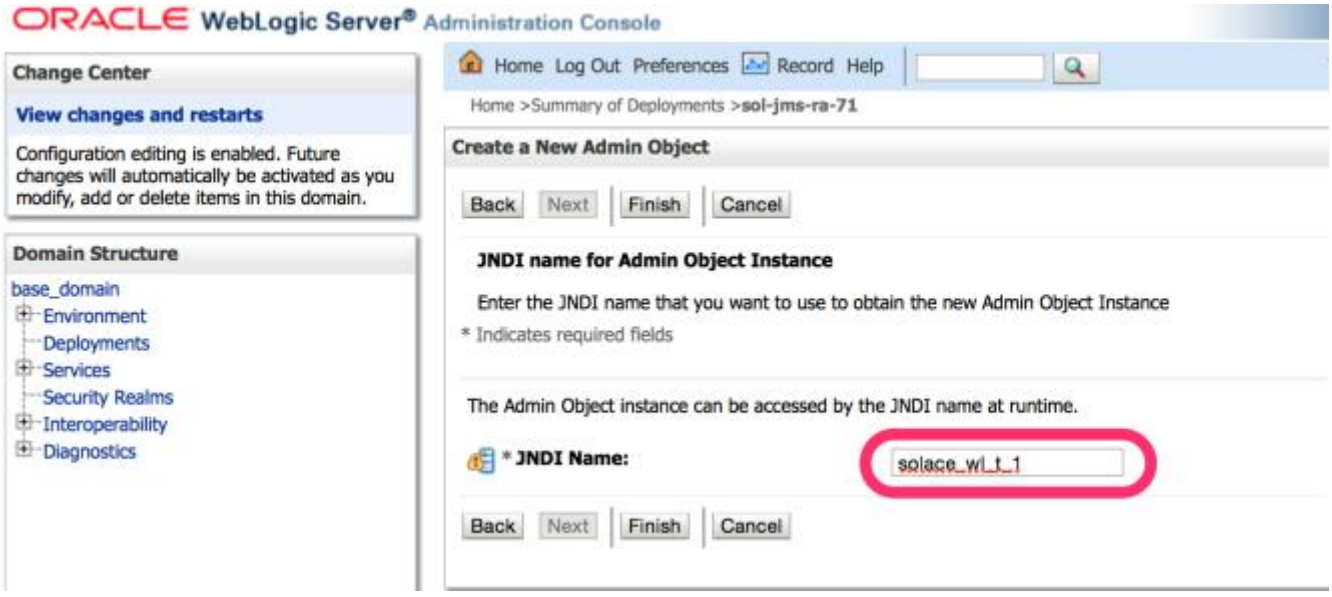
1) To create a new Destination, in the list of Destination types, click the New button.



2) In the following screen, select the desired destination type and click Next.



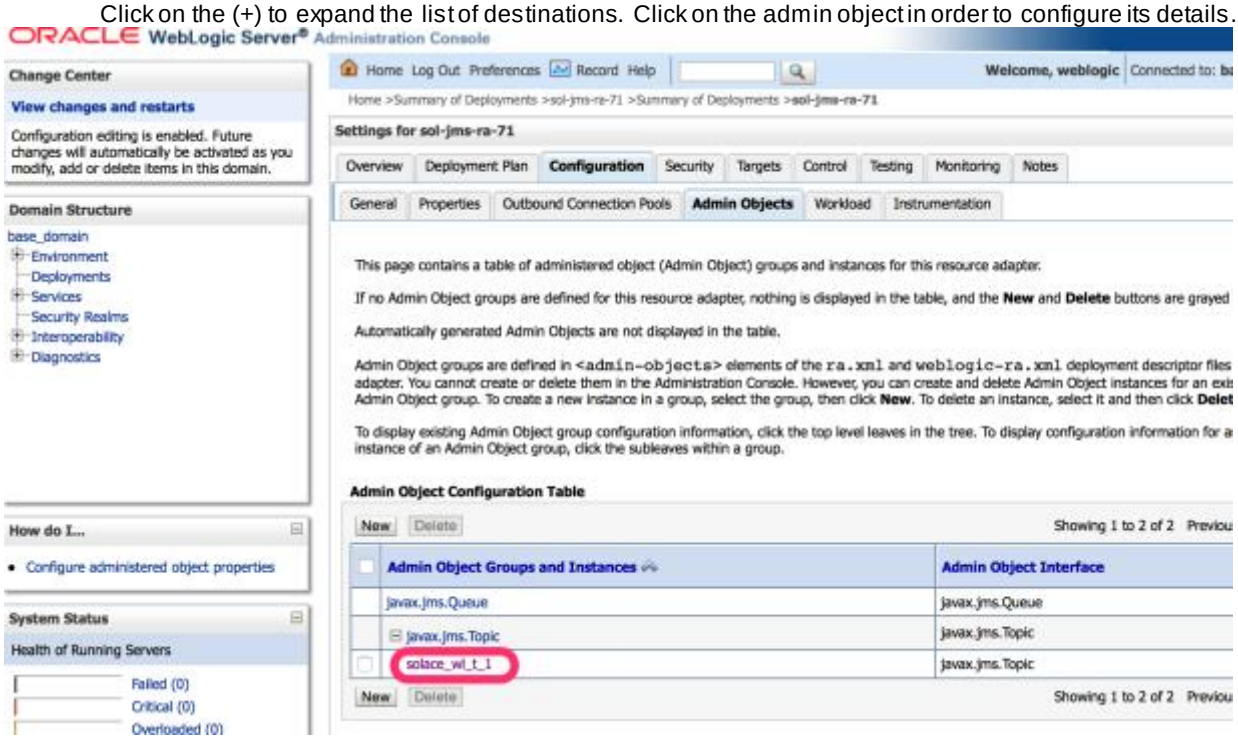
3) The next screen allows you to specify the JNDI name for the destination inside of the WebLogic server. A WebLogic client will later use this string to look up the destination from the WebLogic JNDI. Finish the setup with the Finish button.



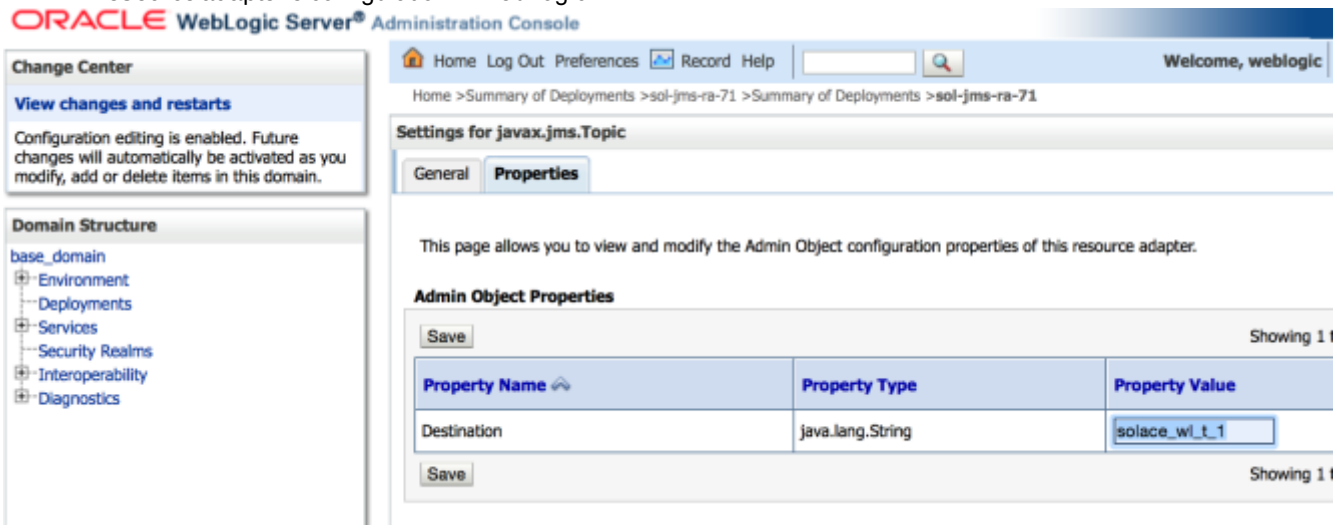
Enter the local JNDI name for the destination and click Finish.

S.No	Key	Value
2	(Local) JNDI Name	This is the JNDI name that MDBs and other resources deployed on WebLogic will use to publish and subscribe to Destinations

4) Going back to the Admin Objects tab, a navigable tree component (+) is visible on the destination type type. Click on the (+) to expand the list of destinations. Click on the admin object in order to configure its details.



5) In the Properties section, the connectivity to the Solace appliance is configured. Enter the remote JNDI name as configured at the Solace appliance for the destination to connect to and confirm the settings with the Save button. Similarly to the connection factory configuration, fields left empty will inherit the configuration from the resource adapter's configuration in WebLogic.



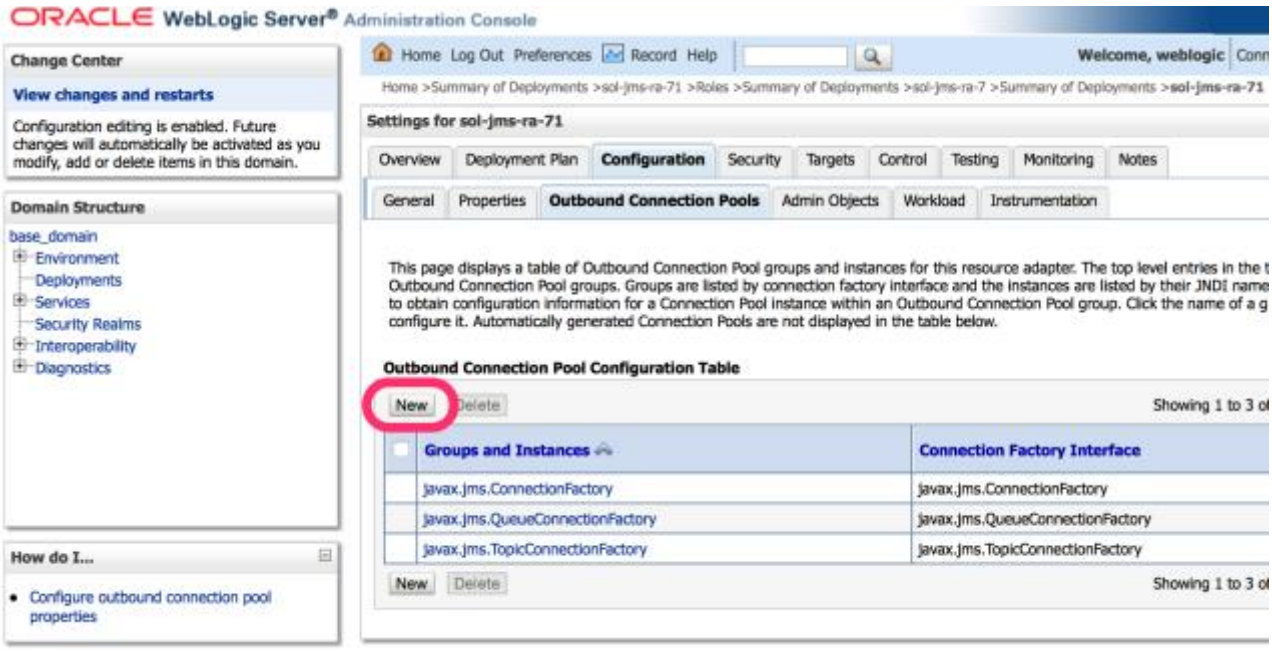
Enter the remote JNDI name for the destination.

S.No	Key	Value
3	Remote JNDI Name	This is the JNDI name for the Destination as configured on Solace for the Endpoint. This can be configured using CLI SolAdmin from the JMS tab

Configuring connection factories

Steps to create a J2C Connection Factory:

1. Log into the WebLogic Application Server administrative console.
2. Click on the 'Deployment > *<resource adapter>* > Configuration > Outbound Connection Pools
3. In the Outbound Connection Pool Configuration Table, click on the 'New' button.



4. Specify type of connection factory to use (javax.jms.ConnectionFactory in this example) and click Next.
5. Specify a unique *JNDI Name* for this J2C Connection Factory (‘***j2c_cf***’ for this example) and click Finish.
6. In the *Outbound Connection Pool Configuration Table*, there is now a navigable tree component (+) on the connection factory type. Click on the (+) to expand the list and click on the connection factory.
7. Edit the J2C Connection Factory:
    a. Specify values for two custom properties, ‘ConnectionFactoryJndiName’ and ‘ConnectionValidationEnabled’.
    b. Click on the “Property Value” column for ‘*ConnectionFactoryJndiName*’ and specify the value ‘***JNDI/Sol/CF***’. (Note, this is the value configured on the Solace message router in *Section 0 Setting up Solace JNDI References*).
    c. Click on the “Property Value” column for ‘*ConnectionValidationEnabled*’ and specify the value ‘**true**’.
    d. Ensure that other necessary values, such as ConnectionURL, MessageVPN and UserName were correctly inherited from the RA’s configuration (done in step 3.4)
    e. Click on the ‘save’ button to commit your changes to the application server.

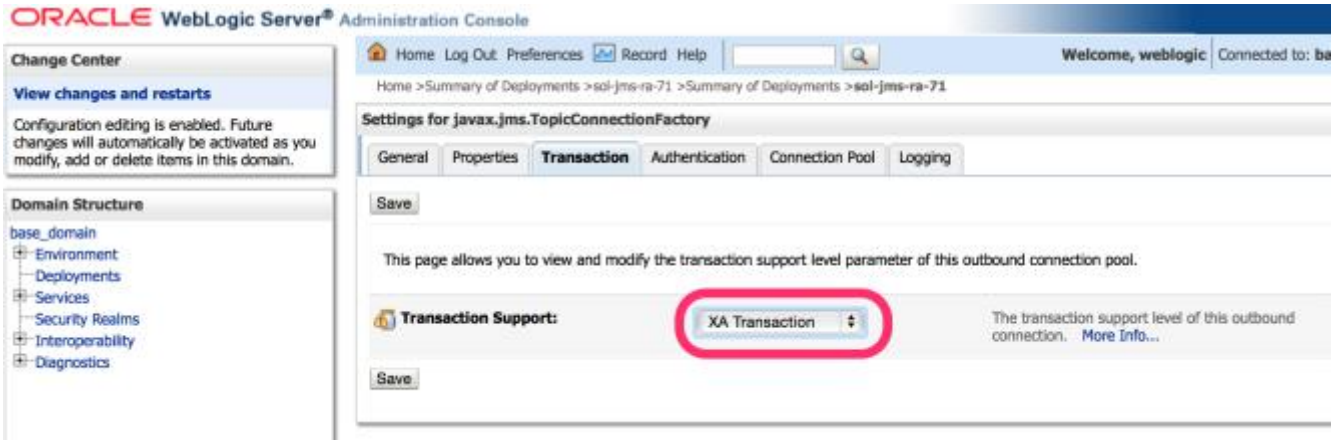
The following table summarizes the values used for the J2C connection factory custom properties.

Name	Value	Description
ConnectionFactoryJndiName	JNDI/Sol/CF	The JNDI name of the JMS connection factory as configured on the Solace message router.
ConnectionValidationEnabled	true	Enable the validation of connections. While not strictly necessary, it is recommended to set it to “true”.

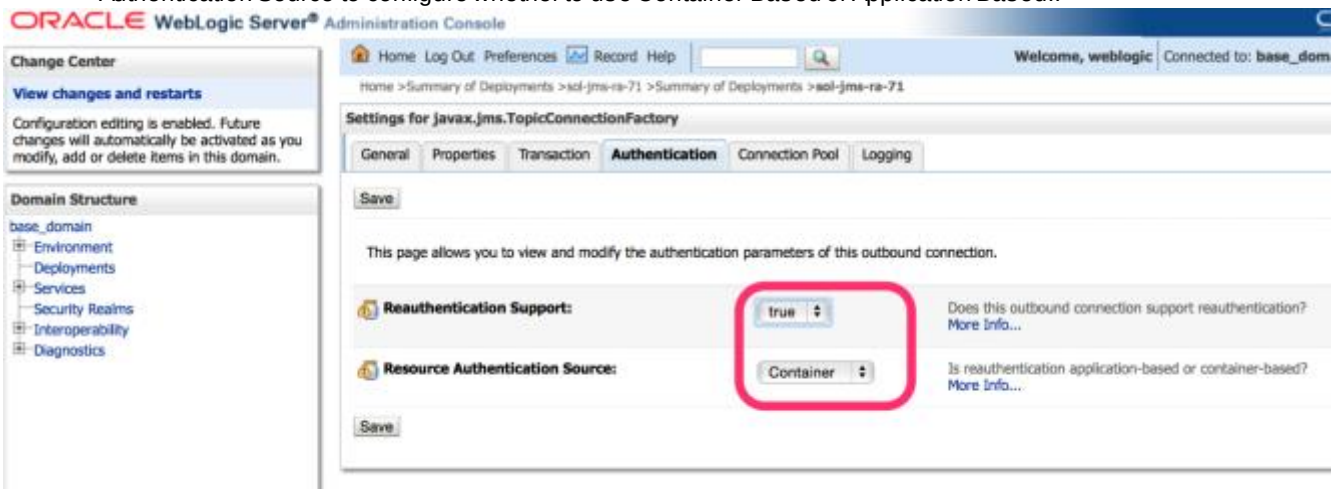
Table 8 – J2C connection factory properties

8. Use the Transaction tab to set the transactions support for the connection. Select the desired level of transaction support and confirm with Save. Note: The transaction support should match with your client using the connection. E.g. a WebApp client usually will not have transaction support while EJB will have it,

depending on the settings of their deployment. Transaction support = "XA Transaction" requires the use of an XA Connection Factory.



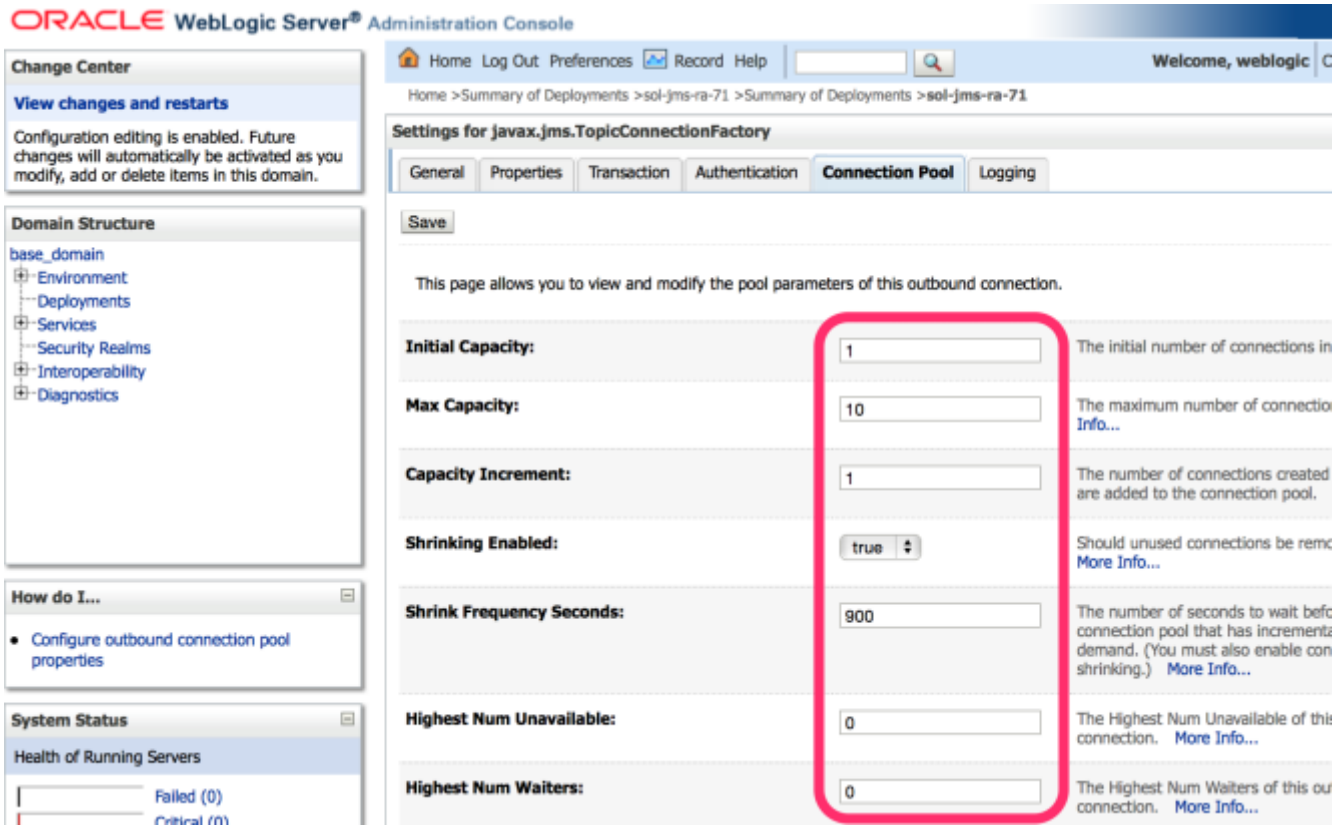
1. Under the Authentication tab, you can configure the Reauthentication Support and the Resource Authentication Source to configure whether to use Container Based or Application Based..



Configuring connection pooling for a connection factory

The Connection Pool tab allows the configuration of the amount of concurrent connections that are run between the WebLogic server and the Solace appliance. Those connections are shared between the requests coming from the applications, therefore the resources can be used more efficiently (limited number of connections) and optimized for performance (pre-initialized connections).

The amounts to be set vary, depending on the demand of the WebLogic application(s). Capacity planning and load testing should identify the appropriate values. E.g. if only every 3rd request is using messaging, the pool size can be reduced approximately by that factor from the maximum amount of concurrent requests.



The most relevant values are:

S.No	Key	Value
1	Initial Capacity	Number of connections to the Solace appliance initially available in the pool for this connection factory.
2	Max Capacity	Maximum number of connection created for this pool.
3	Capacity Increment	The number of connections created when the demand exceeds the current amount of connections. A higher value meets demand spikes more efficiently but can use more resources.
4	Shrinking enabled	Connections can be released if demand is low, freeing up resources.
5	Highest Num Unavailable	Size of a list for connections that failed creation. Attempts to reconnect are periodically made for entries on this list.
6	Highest Num of Waiters	Number of clients that can wait for a connection if all available connections are currently in use. This controls the “overflow” size for client requests before an “unavailable resource” exception is thrown.

Save and return to the resource adapter configuration page.

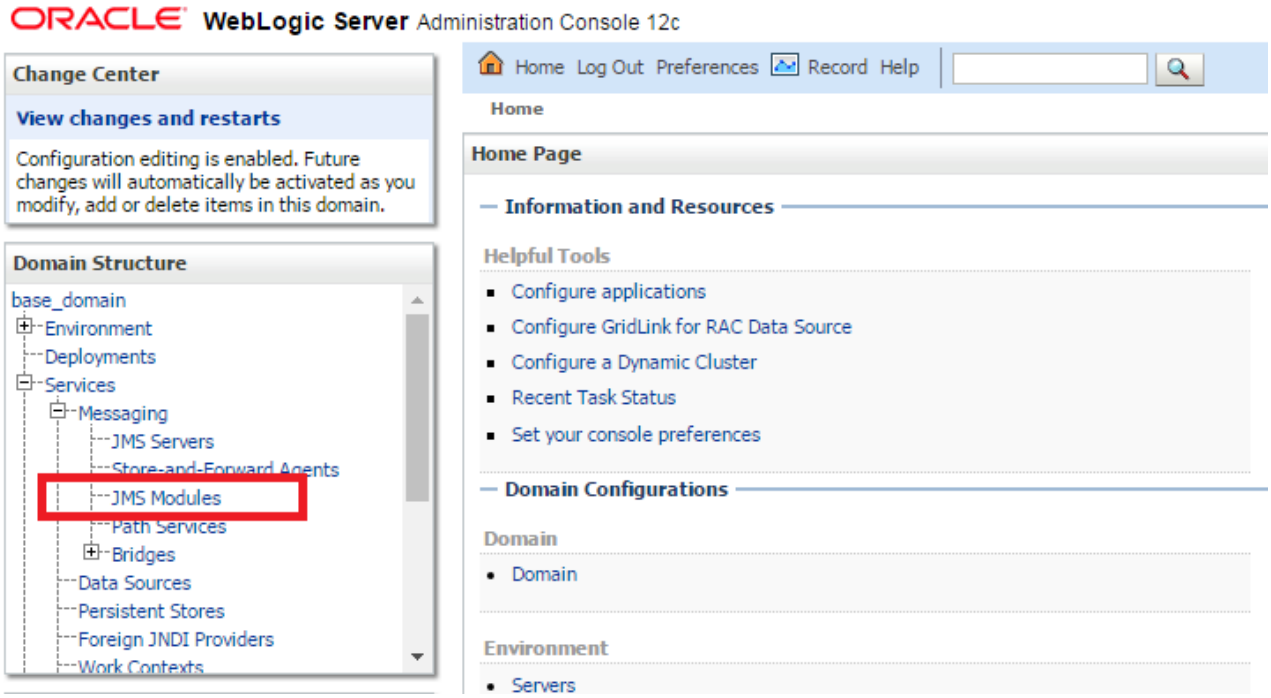
In the 'Deployments' page, select the checkbox for the resource adapter and click on Update. Select "Redeploy this application using the following deployment files" and click on Finish. This redeploys the adapter so that all of the above changes become active.

3.4 (Option 2) Deploying Solace as a Foreign JMS Provider Module

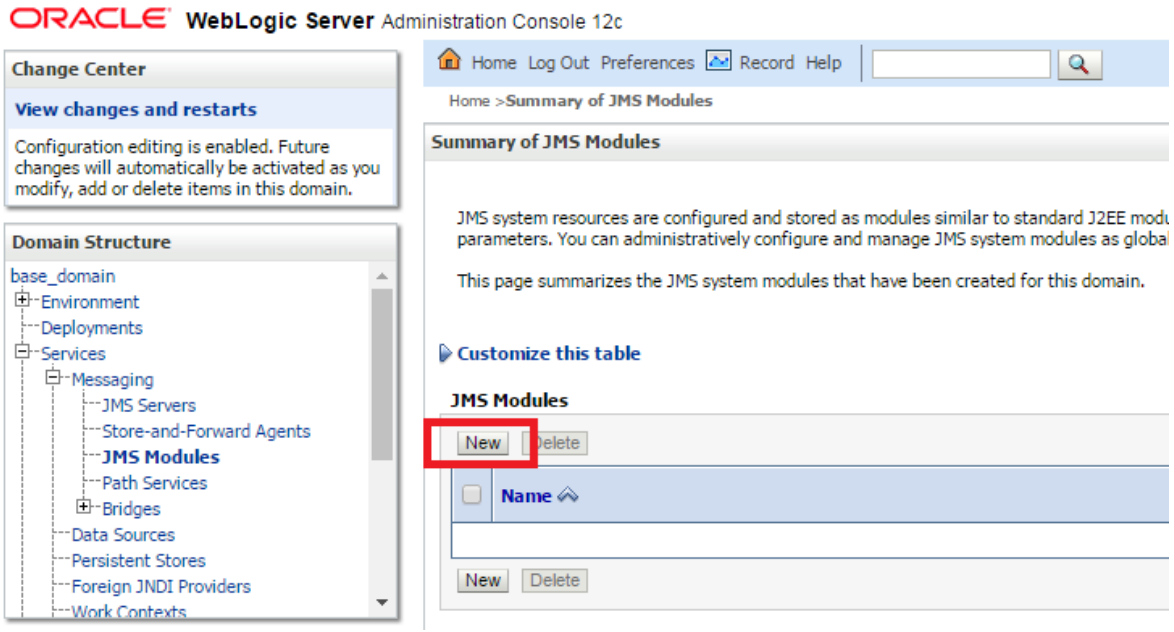
If a resource adapter is not used, the following steps should be followed in order to set up Solace as a foreign JMS provider on the WebLogic server. This option does not support XA transactions.

Adding Solace as a new Foreign Server JMS Module

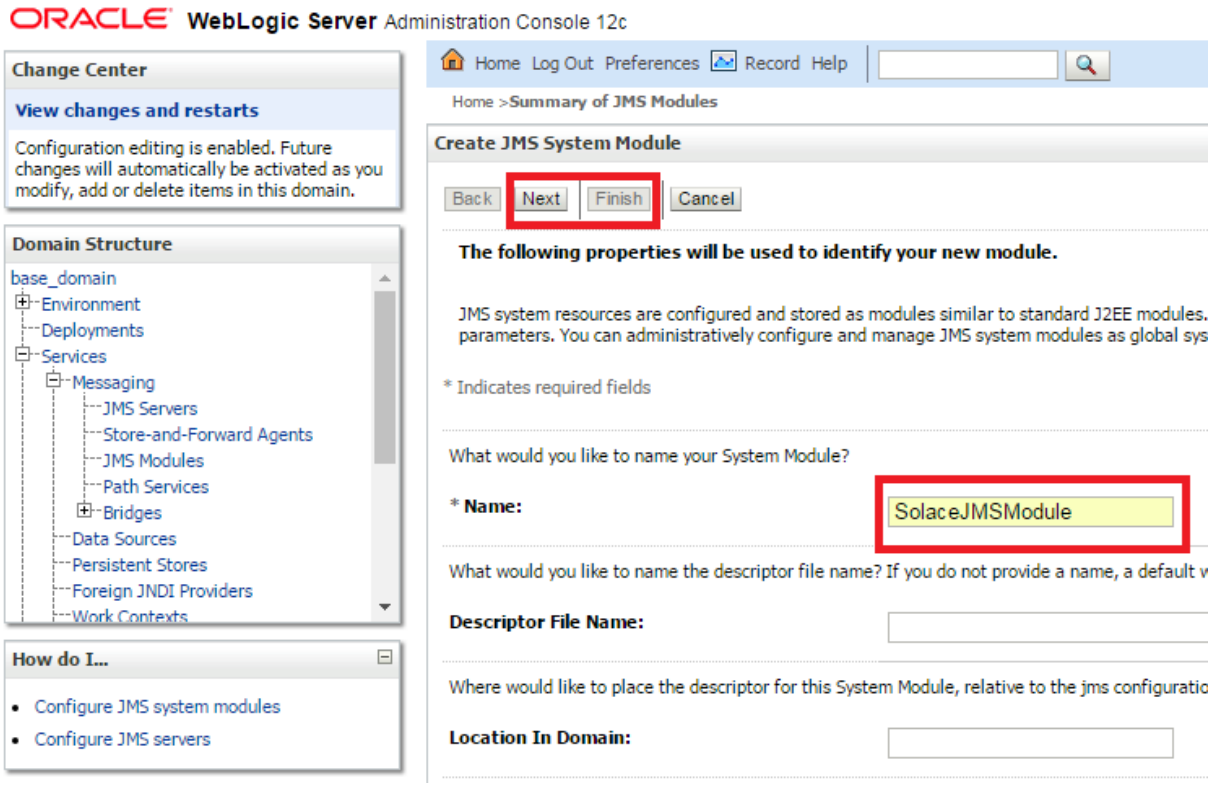
1) Log in to the WebLogic admin console at the administration port once WebLogic is running. From the left menu, Open the Services > Messaging menu, and click on the JMS Modules link.



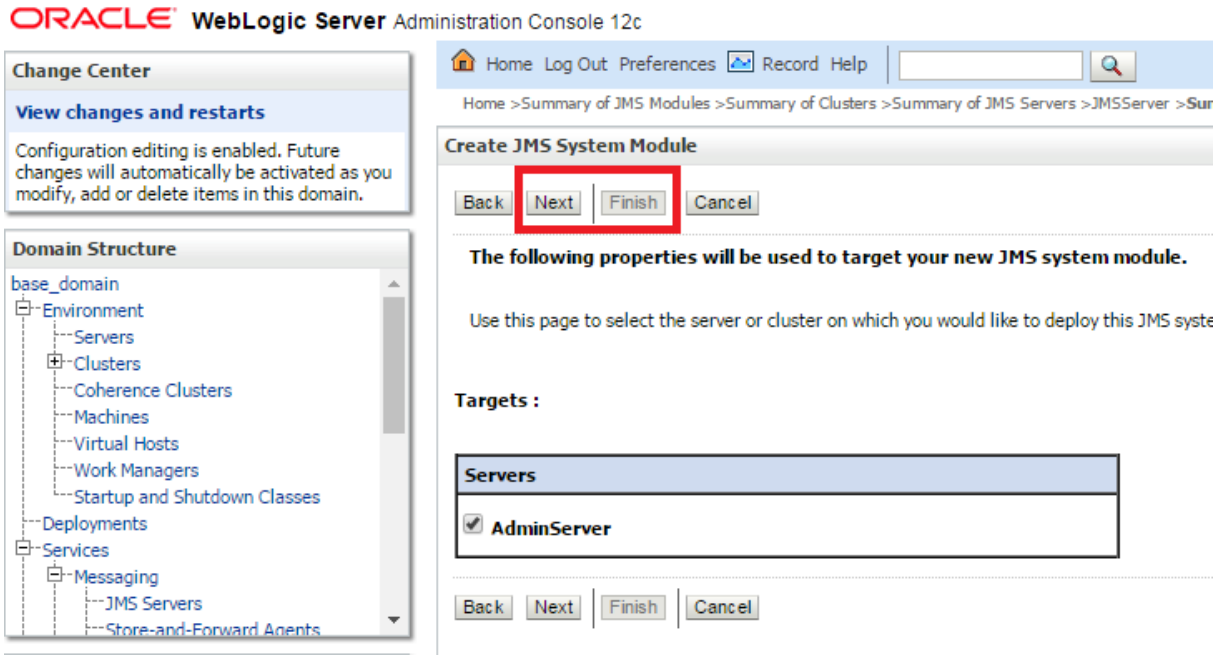
2) On the landing page, click **New** to add a new JMS Module



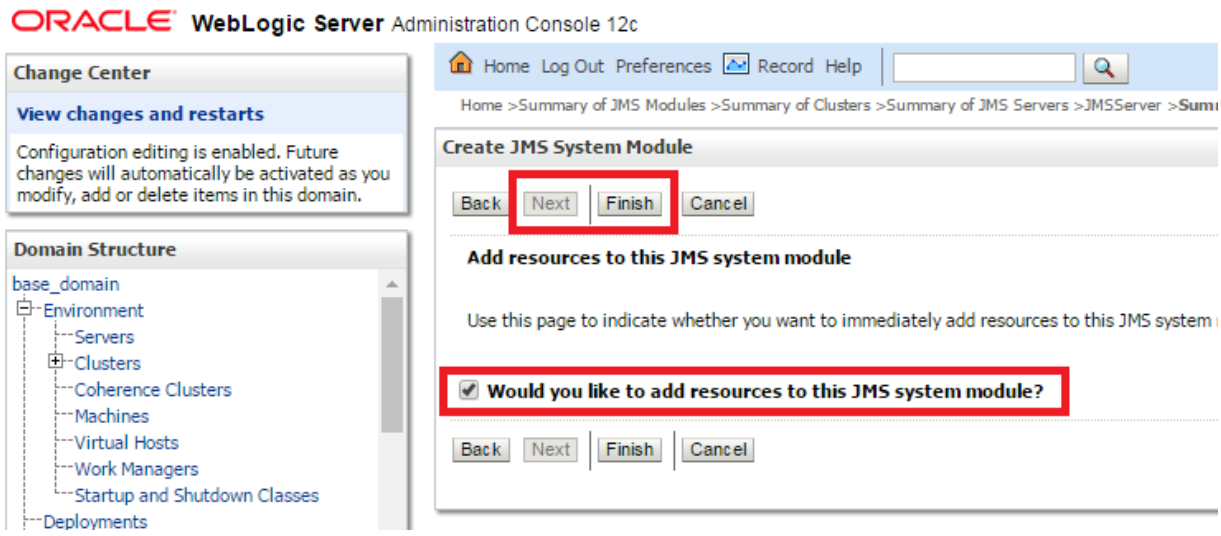
3) 3) Provide the name for the JMS Module and click next.



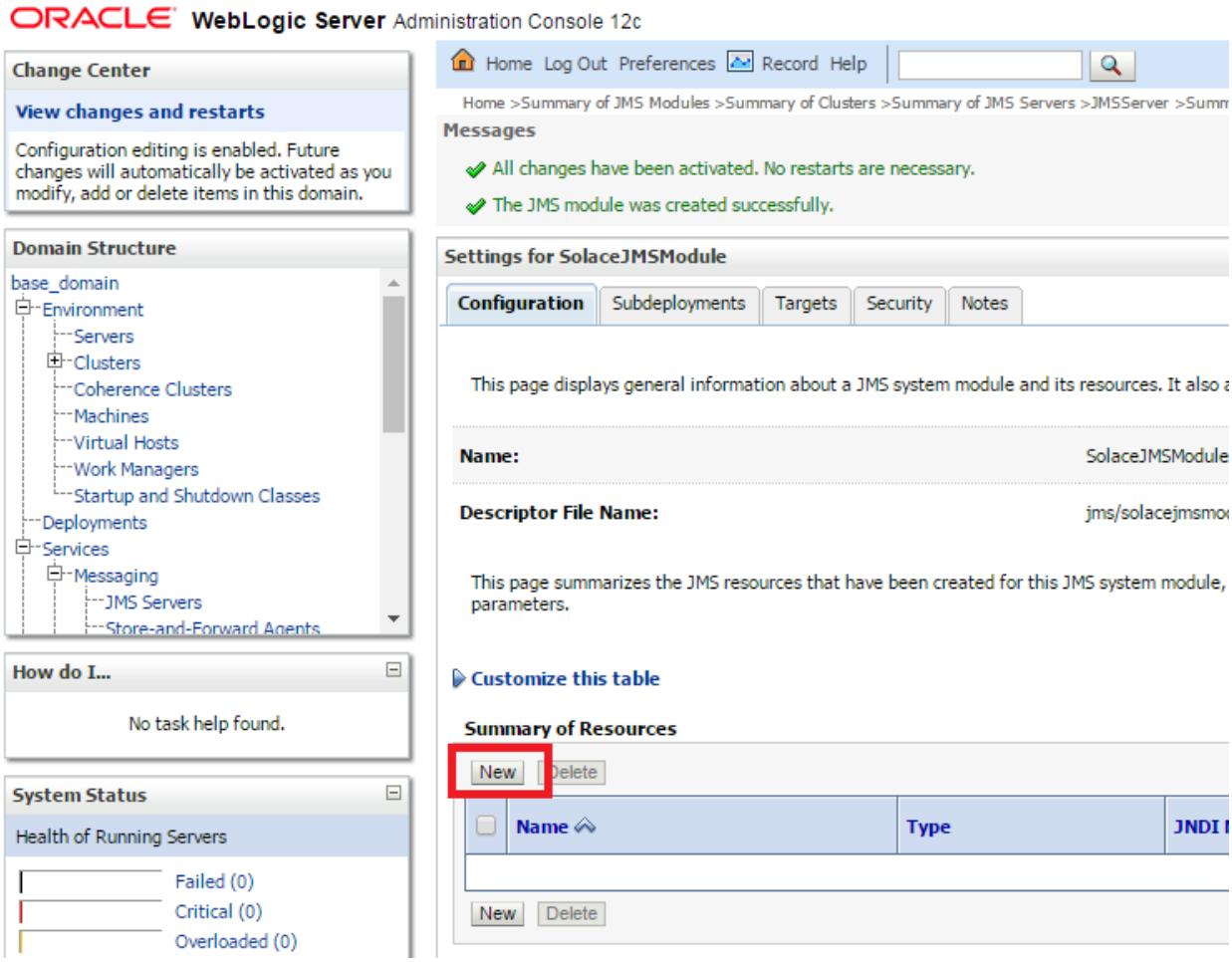
4) Target the mapping as desired for the JMS module, depending on which server(s) the applications will be deployed on, and click next.



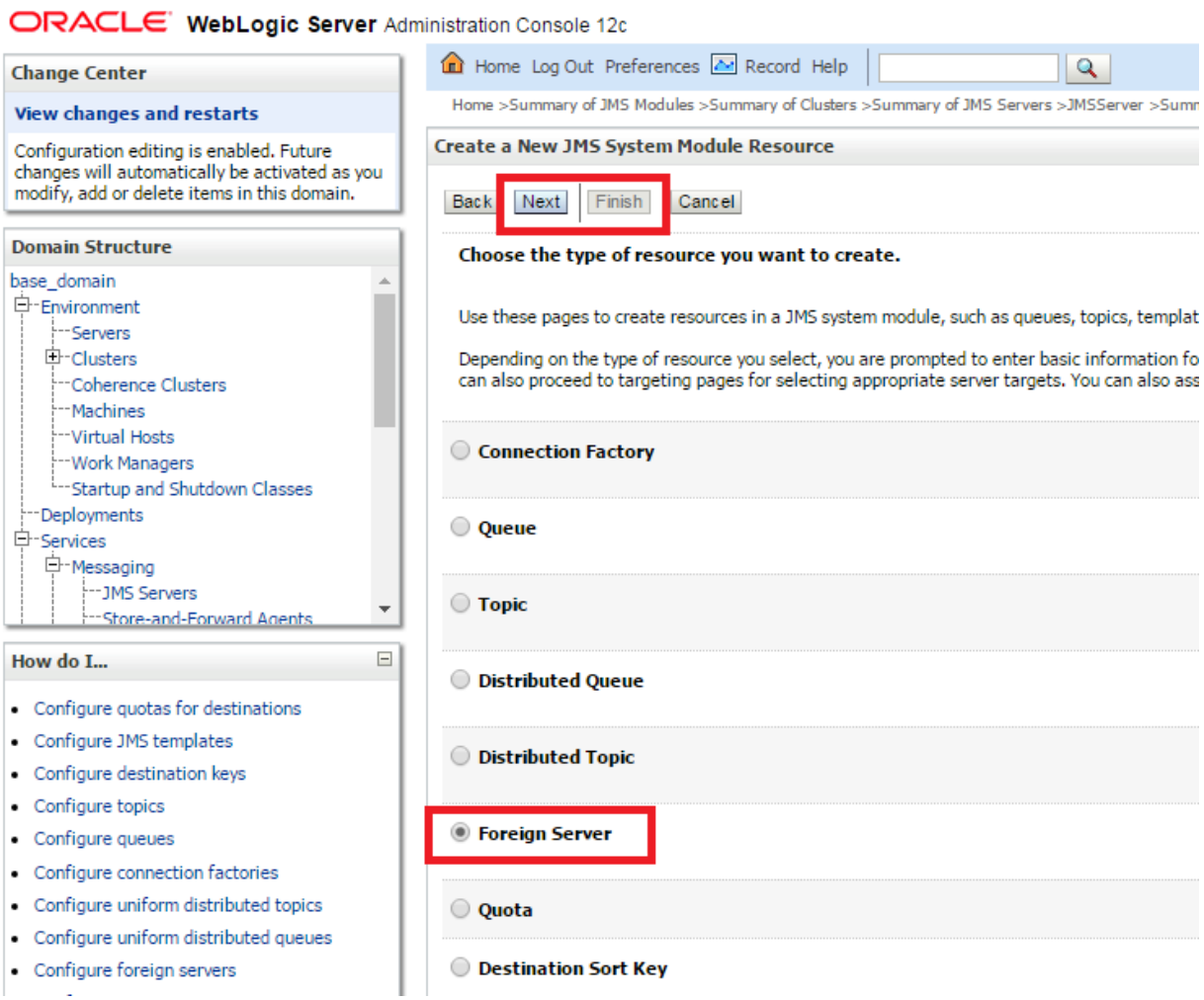
5) Click Finish to end the flow, but select the checkbox to add additional resources. This will be done in the next steps.



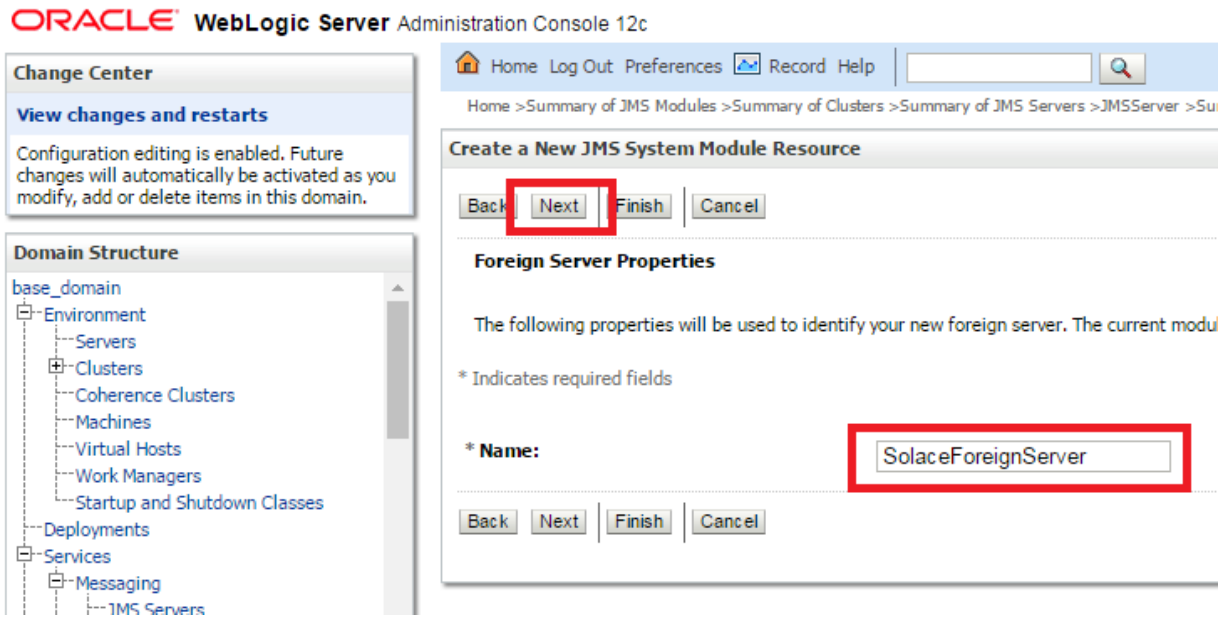
6) Click new to add a Foreign Server



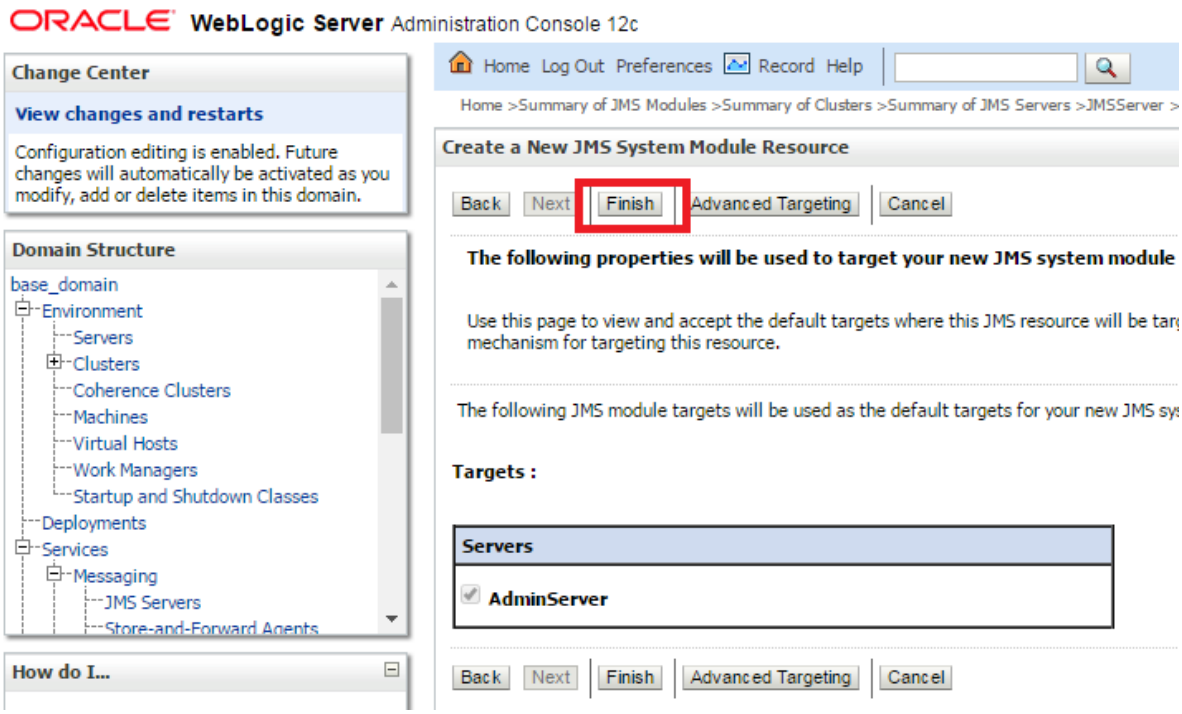
7) Choose Foreign Server from the Radio Button List to add Solace as anew Foreign Server



8) Provide a name for the Foreign Server and click Next.

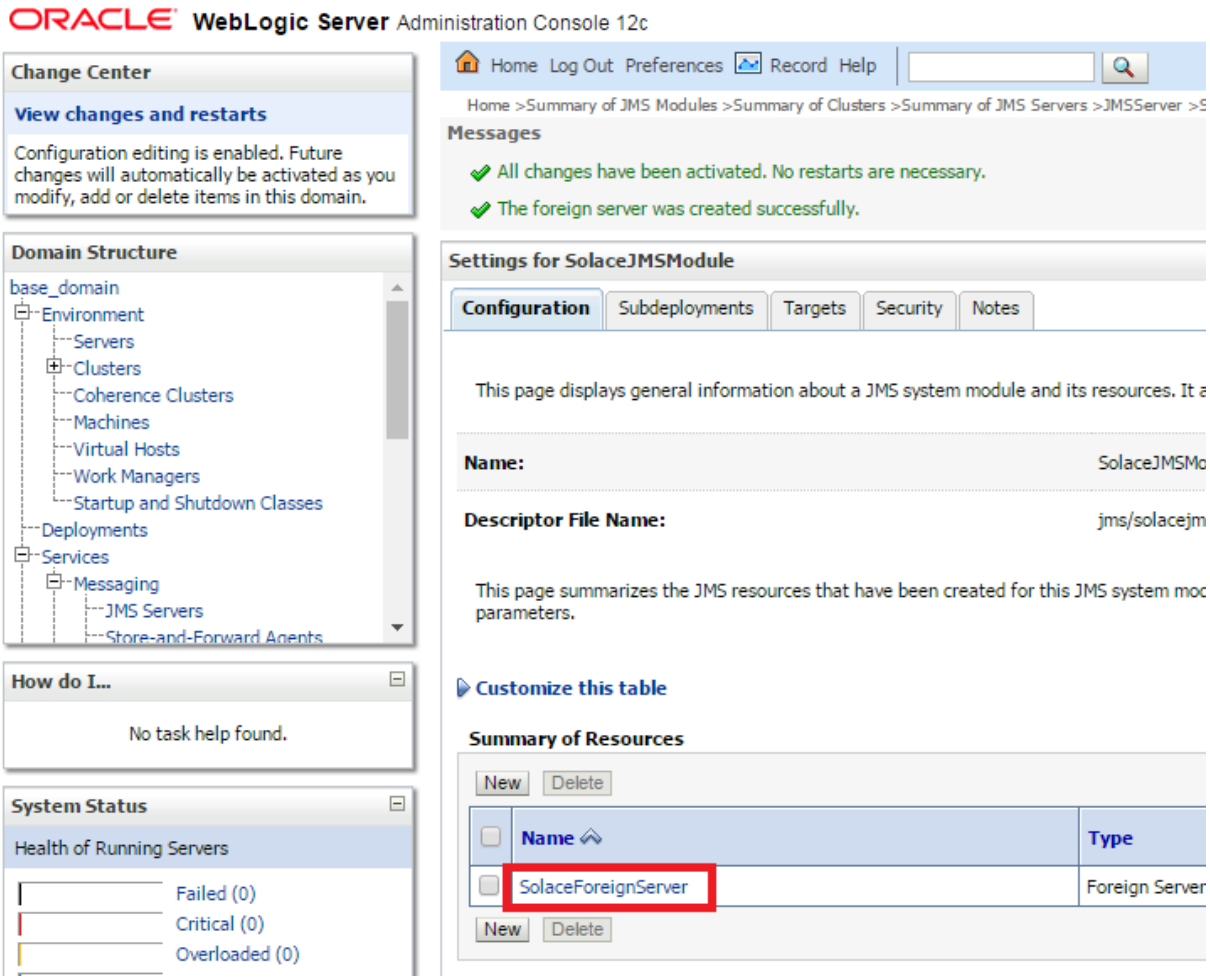


9) The appropriate targets will automatically get assigned. Click Finish to complete the Foreign Server addition.

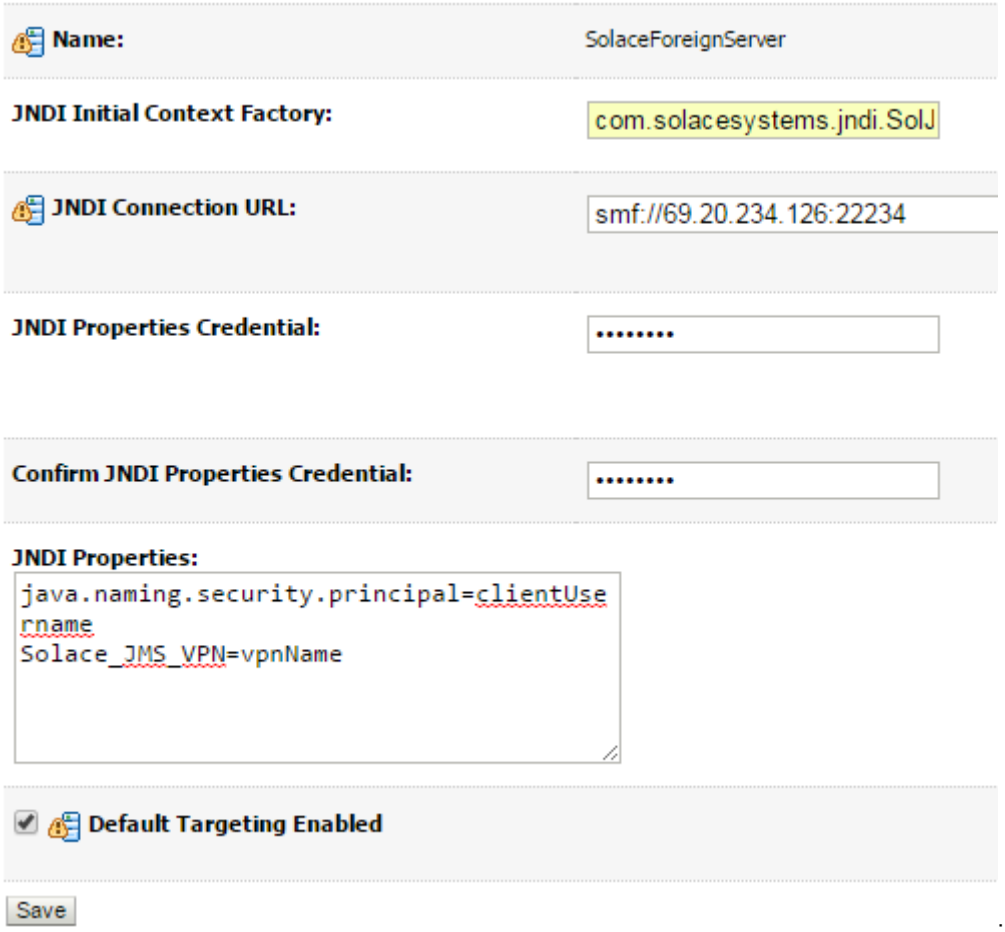


Configuring the connection to Solace

1) Click on the newly created Foreign Server to open the configuration page. The General Tab will open.



2) Enter the following values in the fields and click Save
:

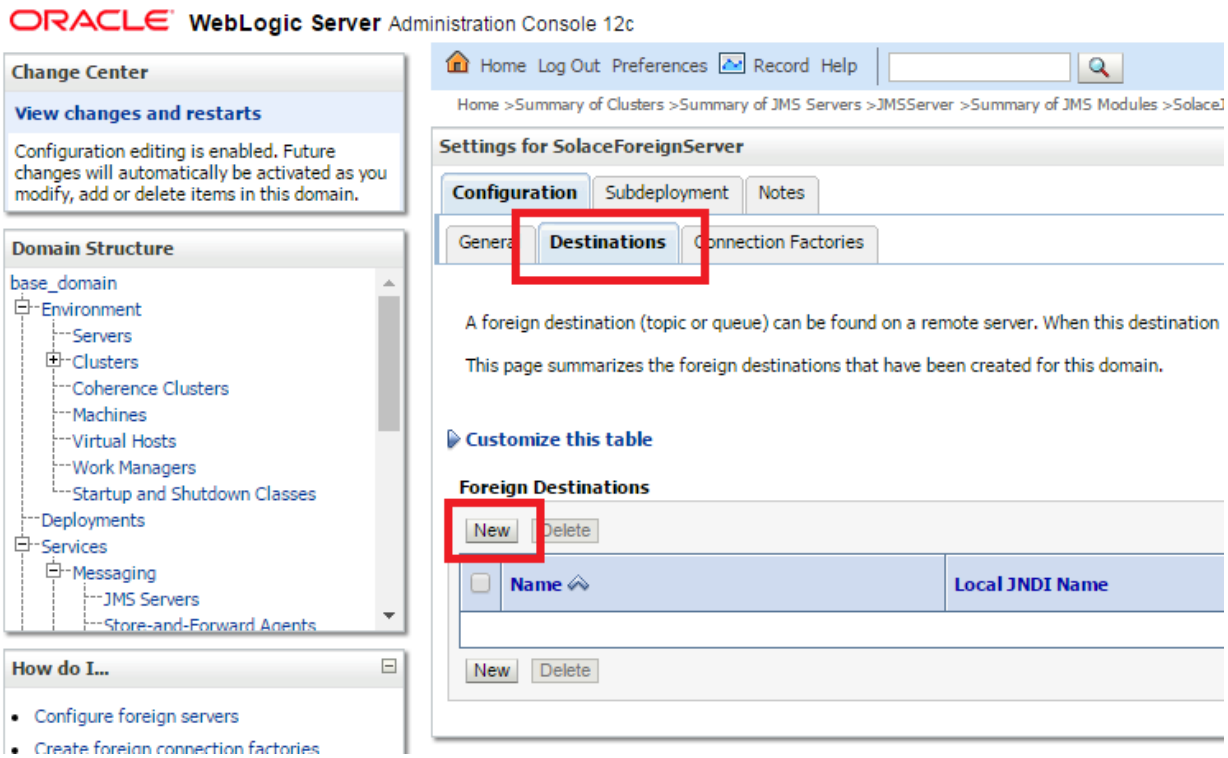


:

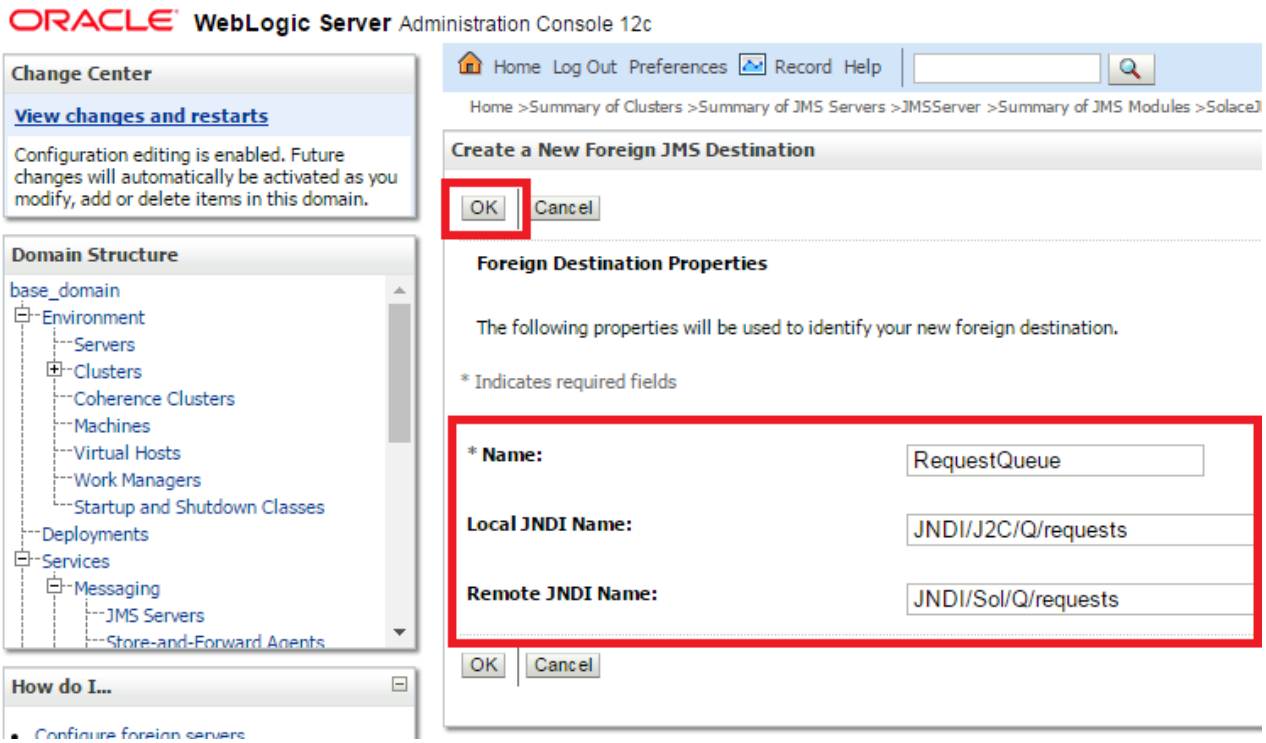
S.No	Key	Value
1	JNDI Initial Context Factory	com.solacesystems.jndi.SolJNDIInitialContextFactory
2	JNDI Connection URL	smf://<data ip of solace>:<data port> *e.g. smf://69.20.234.126:22234*
3	JNDI Properties Credentials	<password of the Client Username for Solace>
4	JNDI Properties	Individual custom properties. Properties are delimited by the Enter key and are set as Key=Value pairs. The below 2 properties must be set: java.naming.security.principal=<client_username> Solace_JMS_VPN=<vpn_name>

Configuring destinations

1) Click on the Destinations Tab to add new Destinations (Queues or Topics)



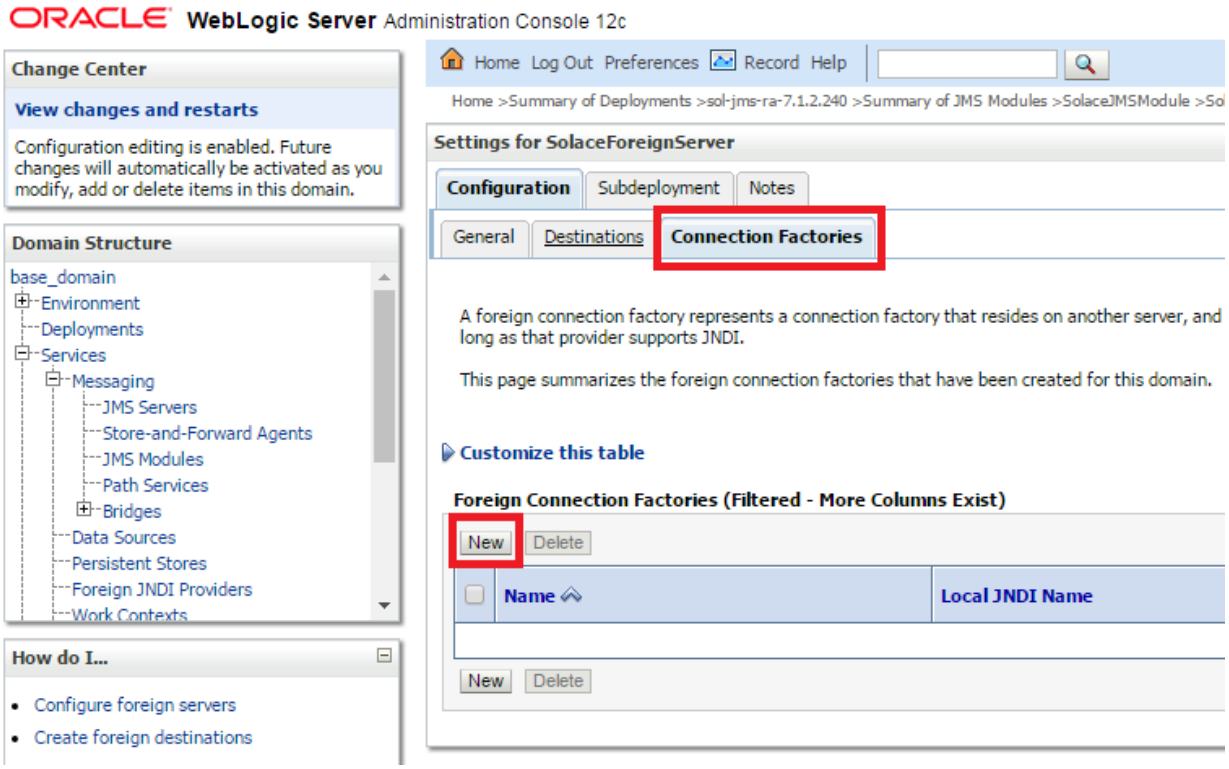
2) Enter the Local and Remote JNDI Names of the Destination



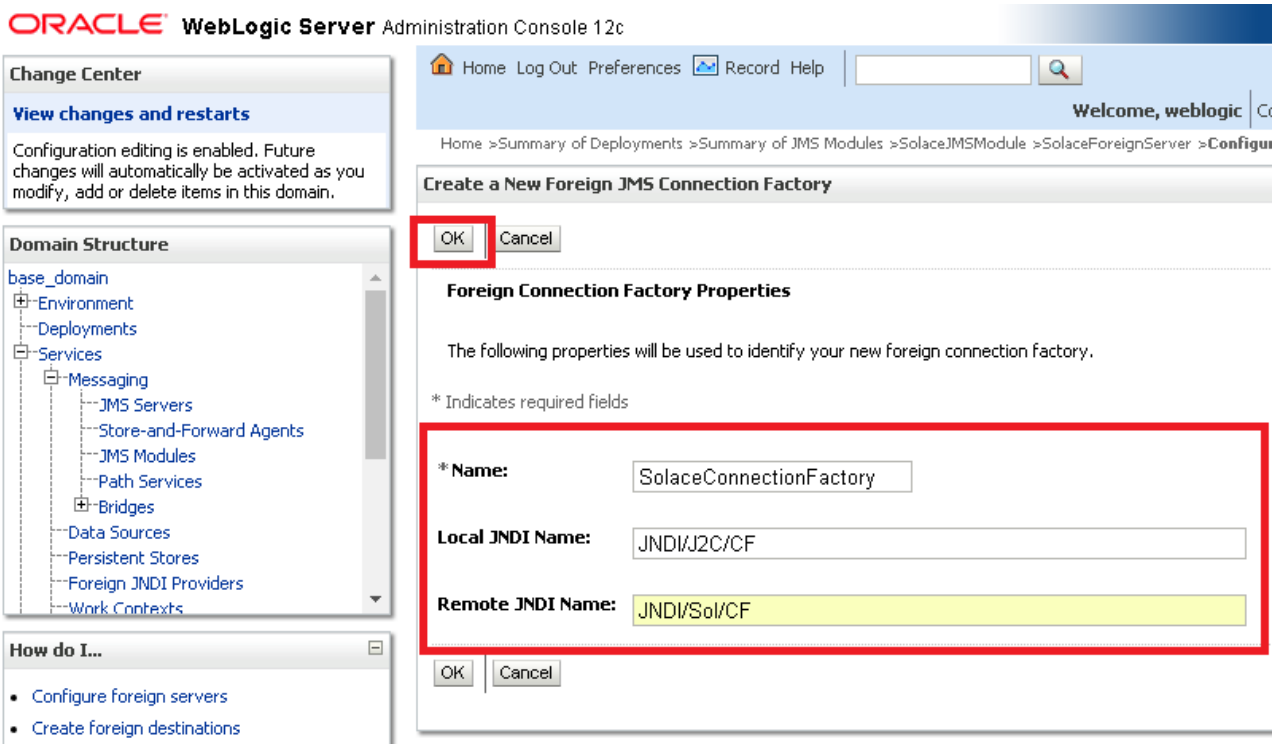
S.No	Key	Value
1	Name	Desired local name of the Destination
2	Local JNDI Name	This is the JNDI name which MDBs and other resources deployed on WebLogic will use to publish and subscribe to Destinations
3	Remote JNDI Name	This is the JNDI name for the Destination as configured on the Solace appliance for the Endpoint. This can be configured using SolAdmin from the JMS Tab

Configuring connection factories

1) Click on the Connection Factories tab, and click New to add a new foreign connection factory



2) Enter the values for Name, Local and Remote JNDI Names



S.No	Key	Value
1	Name	Any text name, not used in code
2	Local JNDI Name	This is the JNDI name which MDBs and other resources deployed on WebLogic will use to lookup the connection factory to connect to Solace
3	Remote JNDI Name	This is the JNDI name for the Connection Factory as configured on the Solace appliance. This can be configured using CLI or through SolAdmin from the JMS Tab

3.5 Sample MDB Code

The sample code below shows the implementation of a message-driven bean (ConsumerMDB) which listens for JMS messages to arrive on the configured Solace JCA destination. Upon receiving a message, the MDB calls the method sendMessage() of the ProducerSB session bean which in turn sends a reply message to a 'reply' Queue destination.

This sample code can be used with the Solace Resource Adapter or Solace as a Foreign Server, assuming the configuration is correct.

```
@TransactionManagement(value = TransactionManagementType.BEAN)
@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName="connectionFactoryJndiName",
            propertyValue="JNDI/Sol/CF"),
        @ActivationConfigProperty(
            propertyName="destinationType",
            propertyValue="javax.jms.Queue"),
         @ActivationConfigProperty(
            propertyName="destination",
            propertyValue="JNDI/Sol/Q/requests")
    }
)
public class ConsumerMDB implements MessageListener {

    @EJB(beanName = "ProducerSB", beanInterface=ProducerSBRemote.class)
    Producer sb;

    public ConsumerMDB() { }

    public void onMessage(Message message) {
        String msg = message.toString();

        System.out.println(Thread.currentThread().getName() +
           " - ConsumerMDB: received message: " + msg);

        try {
               // Send reply message
               sb.sendMessage();

        } catch (JMSException e) {
                     throw new EJBException("Error while sending reply message", e);
        }
     }
}
```

```
@Stateless(name = "ProducerSB")
@TransactionManagement(value = TransactionManagementType.BEAN)
@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)

public class ProducerSB implements ProducerSBRemote, ProducerSBLocal
{
	@Resource(name = "myCF")
	ConnectionFactory myCF;

	@Resource(name = "myReplyQueue")
	Queue myReplyQueue;

	public ProducerSB() { }

	@TransactionAttribute(value = TransactionAttributeType.NOT_SUPPORTED)
	@Override
	public void sendMessage() throws JMSException {

		System.out.println("Sending reply message");
		Connection conn = null;
		Session session = null;
		MessageProducer prod = null;

		try {
			conn = myCF.createConnection();
			session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
			prod = session.createProducer(myReplyQueue);

			ObjectMessage msg = session.createObjectMessage();
			msg.setObject("Hello world!");
			prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
		}
		finally {
			if (prod != null) prod.close();
			if (session != null) session.close();
			if (conn != null) conn.close();
		}
	}
}
```

The sample above requires configuration of JNDI mapped-names to the resource names referenced in the EJB code. The mapping, as well as additional MDB properties can be defined in the WebLogic EJB deployment descriptor file, weblogic-ejb-jar.xml:

```
<?xml version='1.0' encoding='UTF-8'?>
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar
http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.0/weblogic-ejb-jar.xsd ">
        <weblogic-enterprise-bean>
                <ejb-name>ConsumerMDB</ejb-name>
                <message-driven-descriptor>
                        <pool>
                                <max-beans-in-free-pool>10</max-beans-in-free-pool>
                                <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
                        </pool>
                </message-driven-descriptor>
                <transaction-descriptor>
                        <trans-timeout-seconds>10</trans-timeout-seconds>
                </transaction-descriptor>
        </weblogic-enterprise-bean>

        <weblogic-enterprise-bean>
                <ejb-name>ProducerSB</ejb-name>
                <stateless-session-descriptor></stateless-session-descriptor>
                <resource-description>
                        <res-ref-name>myCF</res-ref-name>
                        <jndi-name>myCF</jndi-name>
                </resource-description>
                <resource-env-description>
                        <resource-env-ref-name>myReplyQueue</resource-env-ref-name>
                        <jndi-name>myReplyQueue</jndi-name>
                </resource-env-description>
        </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

4 Performance Considerations

For general tuning including EJBs / MDBs, refer to https://docs.oracle.com/middleware/11119/wls/PERFM/toc.htm

In particular, enabling concurrent MDB instances may not mean that messages are processed in parallel. When using Exclusive Queues or Durable Topics, messages are processed by a single instance of an MDB even if there are multiple configured instances.

If using XA, it is strongly recommended to set total time for low level connection retries to be longer than the transaction timeout, but shorter than the maximum duration of XA calls. These settings can be found in 3 locations:

1) The connection factory on the Solace Appliance controls the low-level connection configuration, such as "Number of Times to Attempt Reconnect" and "Interval Between Reconnect Attempts"
2) The WebLogic server's bean configuration controls the transaction configuration. This can be found in Deployments - <application name> - <bean name> - Configuration. The Transaction Timeout is set at the bottom of the page. Other fine-tuning options are available here, such as the Max Beans in Free Pool, the Max Messages in a Transaction, etc.
3) The WebLogic server's Java Transaction API (JTA), located at Services – JTA – Configuration – JTA, controls several more XA configuration parameters. In particular, the Maximum Duration of XA Calls can be changed here.

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 less than 30 seconds, however, applications should attempt to reconnect for at least five minutes.

In Section 0 - 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_VPN
(config-jndi)# create connection-factory JNDI/Sol/CF
(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)#
```

In addition to configuring the above properties for connection factories, care should be taken to configure connection properties for performing JNDI lookups to the Solace message router. These settings can be configured in the WebLogic application server globally by setting them at the Solace resource adapter level or within individual J2C entities.

To configure JNDI connection properties for JNDI lookups, set the corresponding Solace JMS property values (as a semi-colon separated list of name=value pairs) through the 'ExtendedProps' custom property of the Solace resource adapter or J2C administered objects.

JMS Property Name	Example Value
Solace_JMS_JNDI_ConnectRetries	1
Solace_JMS_JNDI_ConnectRetriesPerHost	5
Solace_JMS_JNDI_ConnectTimeout	30000 (milliseconds)
Solace_JMS_JNDI_ReadTimeout	10000 (milliseconds)
Solace_JMS_JNDI_ReconnectRetries	20
Solace_JMS_JNDI_ReconnectRetryWait	3000 (milliseconds)

Table 9 – JMS JNDI Connection Properties

6 Debugging Tips for Solace JMS API Integration

The key component for debugging integration issues with the Solace JMS API is to enable API logging. Enabling API logging from WebLogic Application Server is described below..

How to enable Solace JMS API logging

The Solace JMS API and Solace Resource Adapter use Jarkarta Commons Logging to support different logging frameworks.

When using log4j with WebLogic, the directory in which the log4j.properties file and log4j jar file exist is on the classpath. A sample log4j.properties setup is shown below:

```
# Log4j configuration properties used
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %-5p [%c] %m%n

# Second appender is set by default to level
log4j.appender.A2=org.apache.log4j.net.SocketAppender
log4j.appender.A2.RemoteHost=localhost
log4j.appender.A2.Port=4445

# Categories
log4j.additivity=false

log4j.logger.com.solacesystems=ERROR, A1
log4j.additivity.com.solacesystems=false

log4j.logger.com.solacesystems.jcsmp=WARN, A1
log4j.additivity.com.solacesystems.jcsmp=false

log4j.logger.com.solacesystems.jms=WARN, A1
log4j.additivity.com.solacesystems.jms=false
```

7 Advanced Topics

7.1 Authentication

The integration example illustrated in *Section 3* of this guide uses the authentication information specified in the custom properties of the Solace resource adapter. These authentication properties are used whenever Application Managed authentication is specified for a JCA resource. No matter the authentication mode (Application-Managed or Container-Managed) specified for a resource, the Solace 'MessageVPN' information for a connection is always retrieved from the Solace resource adapter configured properties (or the configured properties of the respective J2C resource).

WebLogic supports configuration of Container-Managed authentication for J2C resources. The administrator of an EJB application can configure Solace message router sign-on credentials using a J2C authentication alias that is assigned to either a J2C activation specification or J2C connection factory . Solace JCA resource adapter supports authentication using the '*DefaultPrincipalMapping* 'mapping configuration alias. Refer to [WL-REF] for more details on configuring J2C authentication data aliases.

The Solace message router supports a variety of client authentications schemes as described in [Solace-FG] in the Section "Client Authentication and Authorization". The Solace JCA resource adapter supports a subset of these schemes including 'Basic' authentication and 'SSL Client Certificate' authentication. The default authentication scheme used by the Solace JMS Resource Adapter is AUTHENTICATION_SCHEME_BASIC.

The value of the Solace resource adapter custom property '*extendedProps*' is used to specify an alternate authentication scheme such as 'AUTHENTICATION_SCHEME_CLIENT_CERTIFICATE' . The value of the custom property 'extendedProps' consists of a semi-colon separated list of Solace JMS property / value pairs (SOLACE_PROPERTY=value). You can specify the required properties for an alternate authentication scheme using this technique. Refer to the document [Solace-JMS-API] for further details on the required JMS properties for configuring SSL client certificate authentication.

Although the authentication scheme AUTHENTICATION_SCHEME_BASIC is the default scheme, that scheme could also have been specified using the 'extendedProps' custom property of the resource adapter.

Custom Property	Value
ExtendedProps	Solace_JMS_Authentication_Scheme=AUTHENTICATION_SCHEME_BASIC

Table 10 – Specifying the Solace Authentication Scheme

7.2 Using SSL Communication

This section outlines how to update the Solace message router and WebLogic Application Server configuration to switch the client connection to using secure connections with 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 WebLogic Application Server from using a plain text connection to a secure connection, first the Solace message router configuration must be updated as outlined in Section 0 and the Solace JMS configuration within the WebLogic Application Server must be updated as outlined in Section 0.

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 WebLogic Application Server.

```
(config)# message-vpn solace_VPN
(config-msg-vpn)# service smf
(config-msg-vpn-service-smf)# ssl
(config-msg-vpn-service-ssl)# no shutdown
(config-msg-vpn-service-ssl)# exit
(config-msg-vpn-service-smf)# exit
(config-msg-vpn-service)# exit
(config-msg-vpn)# exit
(config)#
```

7.2.2 Configuring the WebLogic Application Server

Secure connections to the Solace JMS provider require configuring SSL parameters on one or more J2C entities. While using the Solace Resource Adapter, these two parameters include changes to the Solace J2C custom properties 'ConnectionURL' and 'ExtendedProps'. Note that the property values for 'ConnectionURL' and 'ExtendedProps' are inherited by J2C connection factory,,and J2C administered objects from their parent Resource Adapter. Thus, unless you are connecting to multiple Solace message routers, a best practice is to configure values for 'ConnectionURL' and 'ExtendedProps' in the Solace Resource Adapter, otherwise the SSL related changes should be duplicated across custom properties for all of the J2C entities you want to secure.

The required SSL parameters include modifications to the URL scheme of 'ConnectionURL' (from 'smf' to 'smfs'), and setting additional SSL attributes through the custom property 'ExtendedProps'. The following sections describe the required changes in more detail.

While using Solace as a Foreign Server module, the 'ConnectionURL ' parameter refered to above maps to 'JNDI Connection URL' in the general properties configuration page of the Foreign Server in the WebLogic Administration Console. 'ExtendedProps' maps to 'JNDI Properties'.

7.2.2.1 Updating the JMS provider URL (ConnectionURL)

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 "ConnectionURL" was as follows:

```
smf://___IP:PORT___
```

This specified a URI scheme of "smf" which is the plaint-text method of communicating with the Solace message router. This should be updated to "smfs" to switch to secure communication giving you the following configuration:

```
smfs://___IP:PORT___
```

7.2.2.2 Specifying other SSL Related Configuration

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.

Specifying a value for the parameter 'Solace_JMS_SSL_TrustStore' is accomplished by modifying the Solace J2C custom property 'ExtendedProps'. The value for the property is comprised of a semi-colon separated list of Solace JMS parameters.

```
Solace_JMS_SSL_TrustStore=___Path_or_URL___
```

A trust store password may also be specified. This password allows the Solace JMS API to validate the integrity of the contents of the trust store. This is done through the Solace JMS parameter 'Solace_JMS_SSL_TrustStorePassword'.

```
Solace_JMS_SSL_TrustStorePassword=___Password___
```

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 parameter '*Solace_JMS_SSL_TrustStoreFormat*':

```
Solace_JMS_SSL_TrustStoreFormat=jks
```

In a similar fashion, the authentication scheme used to connect to Solace may be specified using the parameter '*Solace_JMS_Authentication_Scheme'*

- AUTHENTICATION_SCHEME_BASIC
- AUTHENTICATION_SCHEME_CLIENT_CERTIFICATE

The integration examples in this guide use basic authentication (the default authentication scheme):

```
Solace_JMS_Authentication_Scheme=AUTHENTICATION_SCHEME_BASIC
```

7.3 Working with XA Transactions

This section demonstrates how to configure the Solace message router to support the XA transaction processing capabilities of the Solace JCA Resource Adapter. In addition, code examples are provided showing JMS message consumption and production over XA transactions using both Container-Managed-Transactions (CMT) and Bean-Managed-Transaction (BMT) configuration.

XA transactions are supported in the general-availability release of SolOS version 7.1 and above. The Solace JCA Resource Adapter provides XA Transaction support in version 7.2 and above.

In addition to the standard XA Recovery functionality provided through the Solace JCA Resource Adapter, SolOS version 7.1 provides XA transaction administration facilities in the event that customers must perform manual failure recovery. Refer to the document [Solace-JMS-REF] for full details on administering and configuring XA Transaction support on the Solace Message Router.

Enabling XA Support for JMS Connection Factories

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_VPN
(config-jndi)# connection-factory JNDI/Sol/CF
(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)#
```

XA Transactions – Sample Code

The following examples demonstrate how to receive and send messages using XA transactions. Examples are given for both Bean-Managed and Container-Managed Transactions (BMT and CMT respectively).

Receiving messages from Solace over XA transaction – CMT Sample Code

The following code is similar to the example from *Section 3.5* but specifies Container-Managed XA Transaction support for inbound messages. In this example, the Message-Driven-Bean (MDB) - 'XAConsumerMDB' is configured such that the EJB container will provision and start an XA transaction prior to calling the onMessage() method and finalize or rollback the transaction when onMessage() exits (Rollback typically occurs when an unchecked exception is caught by the Container).

```
@TransactionManagement(value = TransactionManagementType.CONTAINER)
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)

public class XAConsumerMDB implements MessageListener {

    @EJB(beanName = "XAProducerSB", beanInterface=Producer.class)
    Producer sb;

    public XAConsumerMDB() { }

    public void onMessage(Message message) {
        String msg = message.toString();

        System.out.println(Thread.currentThread().getName() + " - XAConsumerMDB: received
message: " + msg);

        try {
              // Send reply message
              sb.sendMessage();

        } catch (JMSException e) {
                    throw new EJBException("Error while sending reply message", e);
        }
     }
}
```

Sending Messages to Solace over XA Transaction – CMT Sample Code

The following code is similar to the EJB example from *Section 3.5* but configures Container-Managed XA Transaction support for outbound messaging. In this example, the Session Bean 'XAProducerSB' method 'SendMessage()' requires that the caller have an existing XA Transaction context. In this example, the 'SendMessage()' method is called from the MDB - 'XAConsumerMDB' in the above example where the EJB container has created an XA Transaction context for the inbound message. When the method sendMessage() completes the EJB container will either finalize the XA transaction or perform a rollback operation.

```
Stateless(name = "XAProducerSB")
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class XAProducerSB implements Producer, ProducerLocal
{
	@Resource(name = "myCF")
	ConnectionFactory myCF;

	@Resource(name = "myReplyQueue")
	Queue myReplyQueue;

	public XAProducerSB() { }

	@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
	@Override
	public void sendMessage() throws JMSException {

		System.out.println("Sending reply message");
		Connection conn = null;
		Session session = null;
		MessageProducer prod = null;

		try {
			conn = myCF.createConnection();
			session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
			prod = session.createProducer(myReplyQueue);

			ObjectMessage msg = session.createObjectMessage();
			msg.setObject("Hello world!");
			prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
		}
		finally {
			if (prod != null) prod.close();
			if (session != null) session.close();
			if (conn != null) conn.close();
		}
	}
}
```

Sending Messages to Solace over XA Transaction – BMT Sample Code

EJB code can use the *UserTransaction* interface (Bean-Managed) to provision and control the lifecycle of an XA transaction. The EJB container will not provision XA transactions when the EJB class's 'TransactionManagement' type is designated as 'BEAN' managed. In the following example, the session Bean 'XAProducerBMTSB' starts a new XA Transaction and performs an explicit 'commit()' operation after successfully sending the message. If a runtime error is detected, then an explicit 'rollback()' operation is executed. If the rollback operation fails, then the EJB code throws an EJBException() to allow the EJB container to handle the error.

```
@Stateless(name = "XAProducerBMTSB")
@TransactionManagement(value=TransactionManagementType.BEAN)
public class XAProducerBMTSB implements Producer, ProducerLocal
{
        @Resource(name = "myCF")
        ConnectionFactory myCF;
        @Resource(name = "myReplyQueue")
        Queue myReplyQueue;
         @Resource
        SessionContext sessionContext;

        public XAProducerBMTSB () { }
        @Override
        public void sendMessage() throws JMSException {
                System.out.println("Sending reply message");
                Connection conn = null;
                Session session = null;
                MessageProducer prod = null;
                 UserTransaction ux = sessionContext.getUserTransaction();
                try {
                        ux.begin();
                        conn = myCF.createConnection();
                        session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
                        prod = session.createProducer(myReplyQueue);
                        ObjectMessage msg = session.createObjectMessage();
                        msg.setObject("Hello world!");
                        prod.send(msg, DeliveryMode.PERSISTENT, 0, 0);
                        ux.commit();
                } catch (Exception e) {
                        e.printStackTrace();
                        try {
                           ux.rollback();
                        } catch (Exception ex) {
                           throw new EJBException(
                              "rollback failed: " + ex.getMessage(), ex);
                        }
                }
                finally {
                        if (prod != null) prod.close();
                        if (session != null) session.close();
```

```
                    if (conn != null) conn.close();
                }
            }
}
```

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. This integration guide will show how the following items required to have a WebLogic Application Server successfully connect to a backup data center using the Solace Data Center Replication feature.

Configuring a Host List within the WebLogic Application Server

As described in [Solace-FG], the host list provides the address of the backup data center. This is configured within the WebLogic application server through the `ConnectionURL` custom property value (of a respective J2C entity) as follows:

```
smf://__IP_active_site:PORT__,smf://__IP_standby_site:PORT__
```

The active site and standby site addresses are provided as a comma-separated list of 'Connection URIs'. 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

Configuring reasonable JMS Reconnection Properties within Solace JNDI

In order to enable applications to successfully reconnect to the standby site in the event of a data center failure, it is required that the Solace JMS connection be configured to attempt connection reconnection for a sufficiently long time to enable the manual switch-over to occur. This time is application specific depending on individual disaster recovery procedures and can range from minutes to hours depending on the application. In general it is best to tune the reconnection by changing the "reconnect retries" parameter within the Solace JNDI to a value large enough to cover the maximum time to detect and execute a disaster recovery switch over. If this time is unknown, it is also possible to use a value of "-1" to force the Solace JMS API to reconnect indefinitely.

The reconnect retries is tuned in the Solace message router CLI as follows:

```
(config)# jndi message-vpn solace_VPN
(config-jndi)# connection-factory JNDI/Sol/CF
(config-jndi-connection-factory)# property-list transport-properties
(config-jndi-connection-factory-pl)# property "reconnect-retries" "-1"
(config-jndi-connection-factory-pl)# exit
(config-jndi-connection-factory)# exit
(config-jndi)# exit
(config)#
```

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 WebLogic, it's important to consider this interaction in the context of a Message Driven Bean and Session Bean.

Receiving Messages in a Message Driven Bean

There is no special processing required during a disaster recovery switch-over specifically for applications receiving messages. After successfully reconnecting to the standby site, it is possible that the application will receive some duplicate messages. The application should apply normal duplicate detection handling for these messages.

Sending Messages from a Session Bean

For WebLogic applications that are sending messages, there is nothing specifically required to reconnect the Solace JMS connection. However, any messages that were in the process of being sent will receive an error from the Solace Resource Adapter. These messages must be retransmitted as possibly duplicated. The application should catch and handle any of the following exceptions:

- javax.resource.spi.SecurityException
- javax.resource.ResourceException or one of its subclasses
- javax.jms.JMSException

Appendix - Solace Resource Adapter and J2C Custom Properties

Resource Adapter Custom Properties (Inherited by J2C connection factory, and J2C administered objects). Note, please refer to [Solace-JMS-REF] for a full list properties supported by the Solace Resource Adapter.

Name	Example Value	Description
ConnectionURL	smf://__IP:Port__	The connection URL to the Solace message router of the form: smf://__IP:Port__ (Update the value '__IP:Port__' with the actual Solace message router message-backbone VRF IP)
ExtendedProps	PROPNAME1=VALUE; PROPNAME2=VALUE; …	Semi-colon separated list of Solace-specific extended properties
messageVPN	<Solace message VPN name>	A Message VPN, or virtual message broker, to scope the integration on the Solace message router.
UserName	<Solace username>	The client Solace username credentials
Password	*****	Client password

J2C connection factory Custom Properties

Name	Example Value	Description
ClientId		A JMS client ID.
ConnectionFactoryJndiName	JNDI/Sol/CF	The JNDI name of the JMS connection factory as configured on the Solace message router.
ConnectionValidationEnabled	true	When set to true, the application server will be notified of any connection error event emitted through a JMS Connection's Exception listener

J2C administered object Custom Properties

Name	Example Value	Description
Destination	JNDI/Sol/Q/requests	The JNDI name of the JMS destination as configured on the Solace message router.