Configuring the Relay Server with Apache HTTP Server 2.x and SSL on Linux

Purpose

The purpose of this page is to discuss configuring the Relay Server with Apache HTTP Server 2.x and SSL on Linux.

Note: Apache HTTP Server on Windows, Mac OS X, and Unix, is not supported for Relay Server installations.

Introduction

Secure data transmission and data availability are integral components of corporate information systems in today's wireless and occasionally connected environments. The need for data security is at an all time high as data breaches and sensitive information thefts are becoming a common occurrence, and combined with the demand for always available data, corporate data systems are being pushed to new limits.

MobiLink already provides a highly scalable session-based synchronization system over HTTP(S) that allows bidirectional data transfer between a main database, called the consolidated database, and many remote databases. The consolidated database can be one of several databases, including SAP SQL Anywhere, SAP HANA, SAP Adaptive Server Enterprise (ASE), SAP IQ, Oracle, Microsoft SQL Server, MySQL and IBM DB2 (LUW).

As new demands are being placed on IT and Web Infrastructure, SQL Anywhere’s MobiLink synchronization technology is more flexible than ever in developing a highly secure and always available solution for mobilizing enterprise applications to front line employees. The Relay Server was introduced with SQL Anywhere 11, which is an HTTP relay that eases integration with existing IT and Web Infrastructure to provide a secure gateway for data synchronization. Using the Relay Server in a multi-tenant environment is a cost effective way to secure a wide spectrum of services required for enterprise business.

Since the initial release of Relay Server, many additional SAP applications (including SAP Afaria and SAP Mobile Platform) can now use the Relay Server as a reverse-proxy HTTP relay for HTTP and HTTPS traffic.

Software Required

For the supported web servers matrix for Relay Server, see: [http://scn.sap.com/docs/DOC-40860](http://scn.sap.com/docs/DOC-40860)

- A Linux operating system
- The manual compilation instructions below for Apache HTTP Server were tested on Ubuntu Server 14. Individual operating systems may be slightly different in Apache HTTP Server configuration and installation if you are installing Apache HTTP Server via a package manager.
- Apache HTTP Server 2.2.x or Apache HTTP Server 2.4.x
  - Apache HTTP Server 2.2.x instructions follow below. The instructions can also be adapted for an Apache HTTP Server 2.4.x configuration
- A version of OpenSSL that is compatible with the Apache HTTP Server version. For security reasons, try to use the highest build version of OpenSSL currently available.
- SAP SQL Anywhere 11* or later, with MobiLink synchronization server and Relay Server components
  - In SQL Anywhere 11, the optional RSA encryption option must have also been purchased for SSL connections. SQL Anywhere 12 and up include this option by default.
  - or
  - SAP Afaria, with the Relay Server components
  - or
  - SAP Mobile Platform, with the Relay Server Components

The Relay Server

The Relay Server Architecture

The Relay Server is a set of web extensions for Microsoft Internet Information Services (IIS) and Apache Web servers that enables secure communications for Web traffic. The Relay Server provides multi-tenant load balancing and failover, and can be easily integrated with other SAP products (SAP Mobile Platform 2.x, SAP Afaria). The Relay Server can be easily integrated into existing IT and Web infrastructure without requiring changes to existing corporate firewalls and IT policies. It is designed to handle incoming HTTP and HTTPS requests from SAP HTTP clients through multiple connector pieces called the Outbound Enabler and Relay Server State Manager. Multiple backend HTTP services can be provided when using the Relay Server, including MobiLink, Afaria, and SAP Mobile Platform. This configuration does not require new ports to be enabled on either the internal or external firewall if the Relay Server is located in the corporate DMZ. The figure below shows the typical setup using a single Relay Server and MobiLink server.
The Outbound Enabler

The Outbound Enabler is a service that is designed specifically for use as the connector between the back-end server and the Relay Server. The figure above shows the MobiLink server as the back-end server communicating with the Outbound Enabler. The Outbound Enabler acts as a double connection piece that establishes on-demand connections with the back-end server, and a pair of up/down channel connections with the Relay Server in the DMZ.

The Relay Server forwards client requests to the Outbound Enabler using the established up channel. The Outbound Enabler then relays the client request to the back-end server, in this case the MobiLink server. Once the Outbound Enabler receives the response from the MobiLink server, it forwards the response back to the Relay Server using the established down channel. The Relay Server then relays the response to the client.

The Outbound Enabler and clients are serviced by the Relay Server using two different Web extensions. The Relay Server client extension is used for client connections and the server extension is used for the Outbound Enabler.

If a Relay Server farm exists, the Outbound Enabler can establish an HTTP or HTTPS connection with the Relay Server to retrieve a list of all available Relay Servers. This is done by supplying the server with a URL that maps to the Relay Server's server extension. If there is a load balancer handling a Relay Server farm, the server URL would map through the load balancer to the Relay Server's server extension.

The Outbound Enabler, after retrieving information regarding the available Relay Servers, creates the up/down channel pairs with each Relay Server. Connect, transfer, and disconnect activities of the Outbound Enabler are then driven on demand by the clients and back-end servers.

In the case of MobiLink 16 and higher, the Outbound Enabler is built-in and can be accessed via the `mlsrv16 -x oe()` switch.

Starting the Relay Server

The Relay Server architecture is broken down into three phases:

- Start-up phase
- Ready phase
- Working phase

The start-up phase, shown in the next figure, is initiated when the Outbound Enabler establishes a connection with the Web server running the Relay Server (or load balancer, if multiple Web servers exist). The Outbound Enabler then retrieves a list of all available Relay Servers in the Relay Server farm.
The ready phase, shown in the figure below, is initiated when the Outbound Enabler has the list of all available Relay Servers, at which point the Outbound Enabler detects if the back-end server is available and then establishes a direct connection with each Relay Server, bypassing the load balancer if one exists.
The working phase, shown in the next image, is initiated when an HTTP client establishes a connection with the Web server. In the next figure, the connection is drawn with a hashed mark to represent a stateless HTTP/S connection. All communication between the Relay Server and client is through the Web server, or load balancer if one exists.

Performance Tip: If a load balancer is being used to handle multiple Relay Servers in a farm environment, the load balancer does not need to maintain HTTP session affinity between the client and the Relay Server. Individual HTTP requests within the same HTTP session may go through different Relay Servers in the farm. A load balancer performance gain may be obtained by turning off the HTTP affinity feature.
If a back-end server farm is being used, all three phases (Start-up, Ready, and Working) occur regardless of the number of back-end servers.

1. Configuring the Relay Server components with the Apache HTTP Server

About the Apache HTTP Server and the Relay Server

The Apache 2.2 installation instructions do not provide any specific information regarding the Linux user to install Apache under. This document includes instructions on installing Apache under the root user. Following the instructions outlined below will ensure that all the sufficient permissions between Apache Web server, the Relay Server State Manager, and the Relay Server Web extensions exist.

If the root user is not used to install Apache, there may be issues that need to be addressed when configuring the Apache Web server for use with the Relay Server. For instance, only the root user has permissions to bind to any port below 1024. This means the Apache Web server cannot be started on the default ports of 80 and 443 unless started by the root user.

Apache HTTP Server workers utilize a specific user from the httpd.conf information, which by default is the 'daemon' operating system user in order to not allow security issues while web requests are executing.

Installing Apache HTTP Server 2.2.x

The full Apache 2.2.x web server installation instructions can be found on Apache's website: http://httpd.apache.org/docs/2.2/install.html

Compiling Apache 2.2.x from Source

- To download, compile, and install Apache on any Linux distribution (Ubuntu instructions are shown below).
  1. Log in to the Linux machine as the root user, or use su. Alternatively, you may prefix all commands with sudo, if you do not wish to log in
Some distributions do not have a default password and/or shell set for the root user. You may be able to set a password from your current user with sudo access by using:

```
sudo passwd root
```

2. Install and configure the GNU C Compiler (gcc) tools and make tools if they are not already installed. You can use a package manager for this:

```
apt-get install gcc make expat
```

3. Install and configure the OpenSSL and OpenSSL development libraries if they are not already installed. You can use a package manager for this, or download and compile them from source.

```
apt-get install openssl libssl-dev
```

4. Download the latest version of the Apache 2.2.x source - from the Apache HTTPD download area: http://httpd.apache.org/download.cgi#
apache22

You may use wget if you are downloading the source from the console. e.g.
```
wget http://www.us.apache.org/dist/httpd/httpd-2.2.29.tar.gz
```

5. From the root shell, run the following commands:

```
tar -zxvf httpd-2.2.29.tar.gz
cd httpd-2.2.29
./configure --enable-ssl=shared --with-included-apr --prefix=<apache-dir>
    • <apache-dir> default location is /usr/local/apache2 for the Apache install if --prefix is not specified. The user installing Apache must have full read, modify, and execute permissions on the <apache-dir> directory. By default, the root user has the necessary permissions.

make
make install
```

Apache HTTP server will be installed to the <apache-dir> directory location defined when configuring the source tree. If the --enable-ssl=shared option is not set, mod_ssl modules will not be installed for use by Apache.

This should result in the following directories being created with the Apache HTTP Server binaries installed (with the default Apache HTTP Server directory prefix):

```
/usr/local/apache2/conf
/usr/local/apache2/conf/extra
/usr/local/apache2/conf/original
/usr/local/apache2/conf/original/extra
/usr/local/apache2/htdocs
/usr/local/apache2/error
/usr/local/apache2/icons
/usr/local/apache2/logs
/usr/local/apache2/cgi-bin
/usr/local/apache2/manual
/usr/local/apache2/modules
```

The instructions below assume a default prefix of /usr/local/apache2.

**Using a Package Manager (apt, yum, rpm, dpkg, etc.)**

- Consult the documentation for your package manager on your distribution. Be aware that not every operating system installs the Apache Web Server components in the same directories as the default configuration above.

**Deploying the Relay Server Web Extensions to Apache HTTP Server**

The Relay Server requires the following configuration file, resource file, and shared objects that match your Apache HTTP server installation (the specific Apache HTTP Server web extension modules are in the Relay Server deployment package in relayserver/apache22 or relayserver/apache24). Note that you need to use either the 32-bit or the 64-bit version of the web extensions that match the architecture of Apache HTTP Server. The 64-bit instructions follow below.
The Relay Server files either need to be copied into the `<apache-dir>/modules` directory or linked so that they can be referenced by Apache. The instructions below copy the files directly. If you are using a full SQL Anywhere install, you can skip the `"./generate_minimal_sa_config.sh"` commands and copy the files out of the SQL Anywhere install tree instead.

1. `mkdir rs`
2. `tar -C rs -zxf sa16_relayserver_linux_x86+x64.1600_2087_l10n.tar.gz`
3. `mv rs /usr/local/apache2/modules`
4. `cd /usr/local/apache2/modules/rs/bin64`
5. `./generate_minimal_sa_config.sh`
6. `cd .. /bin32`
7. `./generate_minimal_sa_config.sh`
8. `cd ..`
9. `ls -al`

```
    drwxr-xr-x 11 root root 4096 Feb 20 15:53 .
    drwxr-xr-x  3 root root 4096 Apr  2 16:39 ..
    drwxr-xr-x  4 root root 4096 Feb 20 15:53 bin32
    drwxr-xr-x  2 root root 4096 Feb 20 15:53 bin32s
    drwxr-xr-x  4 root root 4096 Feb 20 15:53 bin64
    drwxr-xr-x  2 root root 4096 Feb 20 15:53 bin64s
    drwxr-xr-x  2 root root 4096 Feb 20 15:53 java
    drwxr-xr-x  2 root root 4096 Feb 20 15:53 lib32
    drwxr-xr-x  2 root root 4096 Feb 20 15:53 lib64
    drwxr-xr-x  5 root root 4096 Feb 20 15:53 relayserver
    drwxr-xr-x  2 root root 4096 Feb 20 15:53 res
   -r-r-r-r--  1 root root 910 Feb 11 21:30 RS_README.txt
   -r-xr-xr-x  1 root root 197746 Feb 20 15:18 uninstall.sh
```

All files should now be accessible underneath `/usr/local/apache2/modules/rs`

**Configuring the Apache 2.2 HTTP Server**

To configure the Apache 2.2 HTTP server:

1. Open the Apache HTTP server configuration file (httpd.conf) in a text editor (e.g. vi /usr/local/apache2/conf/httpd.conf)
2. Add the following sections to the file:

```
LoadModule iarelayserver_client_module modules/rs/relayserver/apache22/bin64/mod_rs_ap_client.so
LoadModule iarelayserver_server_module modules/rs/relayserver/apache22/bin64/mod_rs_ap_server.so
LoadModule iarelayserver_admin_module modules/rs/relayserver/apache22/bin64/mod_rs_ap_admin.so
LoadModule iarelayserver_monitor_module modules/rs/relayserver/apache22/bin64/mod_rs_ap_monitor.so

<LocationMatch /cli/iarelayserver/* >
  SetHandler iarelayserver-client-handler
</LocationMatch>
<LocationMatch /srv/iarelayserver/* >
  SetHandler iarelayserver-server-handler
  RSConfigFile modules/rs/relayserver/rs.config
</LocationMatch>
```
The `<LocationMatch>` entries will load both the Relay Server client and server modules. The `<LocationMatch>` and `<Location>` entries allow the client and server modules to be invoked using different URLs. The `RSConfigure` directive specifies the location of the Relay Server configuration (`rs.config`) file.

In the `httpd.conf` file, confirm the following already exists:

```
# Listen: Allows you to bind Apache to specific IP addresses and/or ports, instead of the default. See also the <VirtualHost> directive.
# Change this to Listen on specific IP addresses as shown below to prevent Apache from glomming onto all bound IP addresses.
#Listen 12.34.56.78:80
Listen 80
```

The `Listen` directive tells the Apache Web server to listen on port 80 for all incoming HTTP requests. If a port other than 80 is desired, change the `Listen` directive.

```
# If you wish httpd to run as a different user or group, you must run httpd as root initially and it will switch.
# User/Group: The name (or #number) of the user/group to run httpd as. It is usually good practice to create a dedicated user and group for running httpd, as with most system services.
User daemon
Group daemon
```

The `User` directive tells Apache which operating system user to use when responding to web requests. This information is important when configuring the `rshost` process below.

### Sourcing the Linux Environment Variables to run the Relay Server and Apache

To run the Relay Server and Apache on Linux, some environment variables first need to be set. To properly source the Linux environment for use by the Relay Server and Apache, the following must be set:

1. **PATH** environment variables to Apache and rshost
2. **LD_LIBRARY_PATH** to locate the `lib32` or `lib64` shared libraries for Relay Server
3. **TMP**, **TEMP**, or **TMPDIR** environment variable

### Generated `sa_config.sh` creation

To ensure the environment is set correctly each time the Relay Server is run, it is recommended to store the environment variables in a text file and use the Linux source command to execute the file. In the steps above, the command `./generate_minimal_sa_config.sh` generated an `/bin32/sa_config.sh` and `/bin64/sa_config.sh` script file with the appropriate variables set.

If you installed a full copy of SQL Anywhere, the `sa_config.sh` files exist already.

### Manual `sa_config.sh` creation

In a text file called `/usr/local/apache2/modules/rs/bin64/sa_config.sh (used for Bash Shell), add the following information:

```
PATH="/usr/local/apache2/modules/rs/bin64:/usr/local/apache2/bin:${PATH}"
export PATH
LD_LIBRARY_PATH="/usr/local/apache2/modules/rs/lib64:${LD_LIBRARY_PATH}"
export LD_LIBRARY_PATH
TMP="/tmp"
export TMP
```

To source the environment, on the command line run:

```
./ /usr/local/apache2/modules/rs/bin64/sa_config.sh
```
The source command is documented here. You will then need to source this file in your shell for the rshost process. You can do this automatically by running the /bin64s version which references the environment script first.

Once the sa_config.sh files are in place, this then allows all of the binaries in the /bin32s and /bin64s to be run without first sourcing the sa_config.sh script manually.

Apache envvars

The `<apache-dir>/bin/envvars` file is used to set optional environment variables for use in just the Apache environment. The Relay Server state manager requires the use of the TMP, TEMP, or TMPDIR environment variables to store log information and share state information with Apache. This requires Apache to have the same TMP, TEMP, or TMPDIR location set for its environment.

The `<apache-dir>/bin/envvars` file should have the following entries:

```
PATH="/usr/local/apache2/modules/rs/relayserver/apache22/bin64:/usr/local/apache2/bin:$PATH"
export PATH
LD_LIBRARY_PATH="/usr/local/apache2/modules/rs/lib64:${LD_LIBRARY_PATH}"
export LD_LIBRARY_PATH
TMP="/tmp"
export TMP
```

The `<apache-dir>/bin/envvars` file can also just source the other script file created above:

```
. "/usr/local/apache2/modules/rs/bin64/sa_config.sh"
```

Starting the Relay Server State Manager (rshost) as a Service

In order to start rshost, a Relay Server Configuration File (rs.config) must be present. For now, we’ll copy the sample configuration file that comes with the Relay Server package, and then start rshost. Feel free to edit the rs.config file at this time.

```
cp /usr/local/apache2/modules/rs/relayserver/rs.config.sample
/usr/local/apache2/modules/rs/relayserver/rs.config
chmod u+w /usr/local/apache2/modules/rs/relayserver/rs.config
```

**WARNING:** rshost MUST be executed by the same user that Apache is using in the `httpd.conf` "User" directive above. If you don’t configure the service this way, you will see similar errors in the Apache error log, similar to:

```
[error] [client 127.0.0.1] Failed to attach to Relay Server state store, pid: 23864
```

Once the Relay Server State Manager (rshost) and Apache HTTP Server processes are set up and running below, check to ensure that the users running "httpd -k" Apache children processes are the same user running the "rshost" process:

```
ps -eo "%U %c" | grep 'rshost|httpd'
```

```
root  httpd
daemon httpd
daemon httpd
daemon httpd
daemon httpd
daemon httpd
daemon httpd
daemon httpd
```

For other possible causes for this error and to help resolve this message, see the following KBAs:

- **KBA 2043096** - Getting a "400 Bad Request" when testing the status pages of the Relay Server running on Linux/Apache - SQL Anywhere
- **KBA 2024319** - What does the following RSOE error indicate: "HTTP Error 400.0 - Failed to attach to Relay Server state store"

The following command installs the rshost process as a system service, using dbsvc:

```
/<apache-prefix>/modules/rs/bin64s/dbsvc -y -a <apache-user> -t rshost -w RelayServer -q -qc -f /<your-config-directory>/rs.config -os 100K -ot /tmp/rs.log
```

e.g.
You can now start the rshost service:

```
/usr/local/apache2/modules/rs/bin64s/dbsvc -u RelayServer
```

For testing purposes, you can also directly launch the rshost binary via sudo as a particular user:

```
sudo -u daemon "/usr/local/apache2/modules/rs/bin64s/rshost" -q -qc -f
```

**Starting the Apache Web Server**

The Apache Web server and Relay Server rely on the environment to be properly set to ensure shared memory state can be accessed by both processes. This requires the Apache Web server to be started as follows (as root):

```
/usr/local/apache2/bin/apachectl –k start
```

By starting the Apache Web server using apachectl, this will ensure the `<apache-dir>/bin/envvars` file is properly loaded into the environment. Starting the Apache Web server using httpd will not automatically load the `<apache-dir>/bin/envvars` file. Using `httpd does not provide sufficient environment setup for the Relay Server.`

**Installing Apache HTTP Server as a system service**

If you have installed Apache HTTP Server via a package manager, it is likely also configured to run as a service by default. See your operating system documentation for further instructions on configuring the service.

If you have compiled Apache HTTP Server from source, you will want to configure the HTTP service using your boot initialization system (init.d, systemd, etc.)

For init.d:

1. touch /etc/init.d/apache2
2. chmod 755 /etc/init.d/apache2
3. vi /etc/init.d/apache2
   ```
   #!/bin/bash
   /usr/local/apache2/bin/apachectl $@
   ```

   Install the init.d script:

   **On Ubuntu / Debian**
   ```
   sudo update-rc.d apache2 defaults
   ```

   **On Redhat / Fedora / CentOS**
   ```
   chkconfig --add apache2
   ```

   **On Gentoo**
   ```
   rc-update add apache2 defaults
   ```

**Performance Tips**

The `LogLevel` directive controls the amount of verbose logging the Apache HTTP server records while running. The Apache HTTP Server documentation recommends that the `LogLevel` directive be set to at least to `crit` when running a production system ( [httpdocs/2.2/mod/core.html#loglevel](http://httpd.apache.org/docs/2.2/mod/core.html#loglevel) ). A lower log setting is recommended to prevent the Apache HTTP server from constantly writing to the `access_log` and `error_log` files.

In a production environment, the Relay Server should be run with minimal to no verbosity. This is set via the Options in the Relay Server configuration file (rs.config). The verbosity level should be set to 0.

For more information about deploying the Relay Server to Apache HTTP server, see the [documentation](http://httpdocs/2.2/mod/core.html#loglevel).
2. Configuring Apache HTTP Server to use SSL

The first requirement for configuring Apache HTTP Server to use SSL encryption is to obtain a security certificate. You can either create a self-signed certificate or obtain one from a third-party certificate vendor.

To create a self-signed certificate, you can use the Certificate Creation Utility (createcert) provided with SQL Anywhere or use OpenSSL.

Using createcert from SQL Anywhere to create a self-signed certificate

Run the following commands with the following inputs:

cd /usr/local/apache2/conf
/opt/sqlanywhere16/bin64s/createcert

SQL Anywhere X.509 Certificate Generator Version 16.0.0.2087

Warning: The certificate will not be compatible with older versions of the software including version 12.0.1 prior to build 3994 and version 16.0 prior to build 1691. Use the -3des switch if you require compatibility.

Enter RSA key length (512-16384): 2048
Generating key pair...
Country Code: CA
State/Province: ON
Locality: Waterloo
Organization: SAP
Organizational Unit: SAP
Common Name: test.server.com
Enter file path of signer's certificate: <enter>
Certificate will be a self-signed root
Serial number [generate GUID]: <enter>
Generated serial number:
de9cd2c1270f4af980f525ca4a779b4a
Certificate valid for how many years (1-100): 10
Certificate Authority (Y/N) [N]: N
1. Digital Signature
2. Nonrepudiation
3. Key Encipherment
4. Data Encipherment
5. Key Agreement
6. Certificate Signing
7. CRL Signing
8. Encipher Only
9. Decipher Only
Key Usage [1,3,4,5]: 1,3,4,5,6
Enter file path to save certificate: server.crt
Enter file path to save private key: server.key
Enter password to protect private key: password
Enter file path to save identity: ident.crt

Using OpenSSL to create a self-signed certificate for Apache HTTP Server

Run the following commands with the following inputs:

cd /usr/local/apache2/conf
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -keyout server.key -out server.crt

Generating a 2048 bit RSA private key
............+++ 
................+++
writing new private key to 'server.key'

You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,
If you enter '.' , the field will be left blank.
-----
Configuring Apache HTTP Server for SSL Communications

1. The Apache HTTP Server needs to be configured for secure communications. In the `httpd.conf` file found in the `<apache-dir>/conf` directory, add or confirm that the following entries exist:

```
# Dynamic Shared Object (DSO) Support
# Example:
# LoadModule foo_module modules/mod_foo.so
 LoadModule ssl_module modules/mod_ssl.so
# Secure (SSL/TLS) connections
Include conf/extra/httpd-ssl.conf
```

2. In the `httpd-ssl.conf` file found in the `<apache-dir>/conf/extra` directory, add or confirm that the following entries exist:

```
# When we also provide SSL we have to listen to the standard HTTP port (see above) and to the
# HTTPS port
# Note: Configurations that use IPv6 but not IPv4-mapped addresses need two Listen directives:
"Listen [::]:443" and
"Listen 0.0.0.0:443"

Listen 443
# SSL Engine Switch:
# Enable/Disable SSL for this virtual host.
SSLEngine on

# Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate. If the certificate is encrypted, then
# you will be prompted for a
# pass phrase. Note that a kill -HUP will prompt again. Keep in mind that if you have both an
# RSA and a DSA certificate
# you can configure both in parallel (to also allow the use of DSA ciphers, etc.)
SSLCertificateFile "/usr/local/apache2/conf/server.crt"

# Server Private Key:
# If the key is not combined with the certificate, use this directive to point at the key file.
# Keep in mind that if
# you've both a RSA and a DSA private key you can configure both in parallel (to also allow the
# use of DSA ciphers, etc.)
SSLCertificateKeyFile "/usr/local/apache2/conf/server.key"
```

3. Restart the Apache HTTP Server:

```
service apache2 restart
```

4. Verify the HTTPS connection to Apache HTTP Server

1. Open an Internet Browser.
2. In the URL Address field enter `https://servername` (where `servername` is the host name for your Apache HTTP Server) and press Enter.
3. There should be a security warning prompt as the browser will not trust this site by default. Click 'Continue to this Website'. The main root page (by default: 'It works!') of Apache HTTP server should appear, indicating that the HTTPS certificate was installed correctly.

3. Understanding the Relay Server Configuration File
The **Relay Server Configuration File** is used to define all Relay Servers, back-end servers, and the back-end server farms in the environment. The configuration file, `rs.config`, is divided into four sections:

- Options
- Relay Server
- Backend farm
- Backend server

**Note:** After changing the `rs.config` file, the Relay Server State Manager (`rshost`) needs to be updated with the changes. This is done by either restarting the `rshost` process, or by executing `rshost -u -f rs.config` on the command line. To have the `rshost` window close on completion of the update, add the `-qc` switch to the command line.

If you are dynamically updating the configuration, you must run the update command as the same operating system user that started the original `rshost` process, with the same environment settings.

### Options Section

The options section is used to specify properties that apply to each Relay Server and determine how the Relay Server is started. A sample of the section is shown below:

```
#-------------------------------------
# Relay Server options
#-------------------------------------
[options]
verbosity  = 2
```

For more information about the Options section, see the documentation.

### Relay Server Section

The Relay Server section is used to define each Relay Server that exists in the environment. Each entry in the Relay Server section identifies a unique Relay Server.

```
#-----------------
# Relay Server peers
#-----------------
[relay_server]
enable   = yes
host     = server1.sap.com
http_port = 80
https_port = 443
description = Relay Server Definition 1
[relay_server]
enable   = yes
host     = server2.sap.com
http_port = 80
https_port = 443
description = Relay Server Definition 2
```

The above example shows there are two Relay Servers in the farm: `server1.sap.com` and `server2.sap.com`. Each is defined to listen on port 80 for HTTP requests and port 443 for HTTPS requests.

For more information on the Relay Server section, see the documentation.

### Backend Farm Section

The backend farm section specifies the properties of the MobiLink server farm. If multiple MobiLink server farms exist in the environment, there will be multiple entries in this section. A sample of the section is shown below:

```
#-----------------
# Backend farms
#-----------------
[backend_farm]
enable    = yes
id        = MLServerFarm
```
client_security  = off
backend_security = off
description    = The is the MobiLink server farm entry

The above example shows there is only one MobiLink server farm, which is identified by the ID MobiLink.Server.

For more information about the backend farm section, see the documentation.

**Backend Server Section**

The Backend Server section defines a back-end end connection to the machine running the MobiLink server. Each entry in the Backend Server section defines a single MobiLink server. The definition information is used by the Outbound Enabler when it connects to the Relay Server. A sample of the section is shown below:

```
#-----------------
# Backend servers
#-----------------
[backend_server]
   enable = yes
   farm   = MLServerFarm
   id     = MLServer1
   mac    = 00-0C-29-7A-C2-AB
   token = 7b2493b0-d0d4-464f-b0de-24643e1e0feb

[backend_server]
   enable = yes
   farm   = MLServerFarm
   id     = MLServer2
   mac    = 00-0C-29-7A-C2-AB
   token = 9a2633e0-a1b5-321b-d0ac-2667f1a2abc
```

The above example shows that there are two MobiLink servers in the MLServerFarm server farm. Each MobiLink server runs on the same machine, identified by a unique id and the MAC address of the machine's network adapter. There is also an optional unique security token to authenticate back-end server connections with the Relay Server.

For more information about the Backend Server section, see the documentation.

**Applying the Relay Server Configuration File to the Relay Server State Manager Dynamically**

The changes to the rs.config file need to be applied to the Relay Server State Manager. Without stopping the manager, this can be accomplished by running the following command as the same operating system user that originally started the rshost process:

```
rshost /usr/local/apache2/modules/rs/bin64/rshost -qc -u -f /usr/local/apache2/modules/rs/relayserver/rs.conf
```

- **rshost** is the Relay Server state manager
- **-qc** shuts down the Relay Server state manager window upon completion
- **-u** updates the configuration of the running Relay Server
- **-f rs.config** specifies the location of the configuration file

**4. Using the Outbound Enabler**

The Outbound Enabler is designed to act as a double connector between the Relay Server and the back-end server. The Outbound Enabler establishes a connection with the Relay Server using HTTP or HTTPS, and the back-end server using HTTP or HTTPS. If you are using the built-in Outbound Enabler with MobiLink 16, the communication is done in-process.

The Outbound Enabler also reports back to the Relay Server if it cannot establish a connection with the back-end server. At that time, the Outbound Enabler terminates the connection with the Relay Server and does not accept any new requests from the Relay Server until the back-end server is back online.

A sample Outbound Enabler command line is shown below:

```
$opt/sqlanywhere16/bin64/rsoe -cr
"https=1;host=relaysserver.sap.com;port=443;url_suffix=/srv/iarelayserver/;trusted_certificates=server.crt"
-cs "host=localhost;port=80" -f MLServerFarm -id MLServer1
   -t 7b2493b0-d0d4-464f-b0de-24643e1e0feb
```

- **rsoe** is the Outbound Enabler utility
- **-cr <connection string>** is the connection to the Relay Server (HTTP or HTTPS)
- **url_suffix=/srv/iarelayserver/** specifies the URL path to the server extension for the Relay Server
- **-cs <connection string>** is the connection to the MobiLink server (HTTP)
- `MLServerFarm` is the farm name for the back-end server group
- `id MLServer1` is the name assigned for the back-end server
- `-t 7b2493b0-d0d4-464f-b0de-24643e1e0feb` is the security token passed to the Relay Server

**Note:** the `-cs` switch defaults to `host=localhost;port=80` if it is not specified on the Outbound Enabler command line.

There are additional switches that can be provided to the Outbound Enabler. For more information about the Outbound Enabler, see the documentation.

### 5. Synchronizing HTTP clients with the Relay Server

In order to synchronize a web client, the client must specify a `url_suffix` that targets the appropriate server farm they wish to use.

E.g. For a SQL Anywhere MobiLink client, using the above Relay Server information (where loadbalancer.sap.com points to server1.sap.com and server2.sap.com) with `MLServerFarm`:

```bash
dbmlsync -c "DSN=rem1" -e "CTP=HTTPS;ADR='host=loadbalancer.sap.com;port=443;url_suffix=/cli/iarelayserver/MLServerFarm;trusted_certificates=client.crt'" -mp sql -v+ -o rem1.txt -k
```

At this point you should see a client request at a back-end server (in this case, MobiLink) and confirm that the communication was successful.

### 6. Troubleshooting connectivity issues with the Relay Server

There are 5 places you can look for additional information for problems in connectivity with the Relay Server:

1. The client application HTTP request log. This varies by client application. In the above command used with `dbmlsync`, all information is logged via the `-v+` switch.
2. The Apache HTTP server logs (typically found in `/logs/access_log` and `error_log`)—these will typically point to an issue with the Apache HTTP server configuration and whether the Apache HTTP server was able to forward on the HTTP request appropriately.
3. The Relay Server State Manager (`rshost`) log. This is configured via the `rshost -o` switch and the Relay Server Configuration File in the `Options` section.
4. The Outbound Enabler (`rsoe`) log. This is configured via the `rsoe -o output.log -v` switches. If you are using the built-in MobiLink Outbound Enabler, the log is part of the usual MobiLink output log (`mlsrv16 -x oe(config=oe.config) -o output.log`).
5. The back-end server HTTP request log. This is specific to the back-end server.

### Summary

The SAP Relay Server technology eases integration with existing IT and Web Infrastructure to provide a secure gateway for data synchronization. Using the Relay Server in a multi-tenant environment is a cost effective way to secure a wide spectrum of services required for enterprise business.

This document has outlined how to configure the Relay Server using an Apache HTTP server to complete secure communications (using MobiLink as an example). After finishing this document you should have the understanding to setup a complete synchronization system using SSL communication between the back-end servers, HTTP web clients, and Apache HTTP Server using the Relay Server feature.

### Related Content

### Related Documents

- Deploying the Relay Server components to Apache on Linux
- Options section
- Relay Server section
- Backend farm section
- Backend server section
- Outbound Enabler

### Related SAP Notes/KBAs