Advanced Software Design Team

Fujitsu America Inc.
1250 East Arques Avenue
Sunnyvale, CA 94085

SSOFI Provider – High Availability Configuration

Updated: 24 May 2013
Editor: Keith D Swenson

This is an architecture specification that outlines the major modules and interfaces between them. This does not represent a specific release level. The solution is undergoing continual development and improvement. Most of what is described here is already available in finished form, but some capabilities are projections of future capabilities under development. Consult with specific release documentation to determine precisely what is available in a given release.

Table of Contents

1 Setting

Customers want to use software in a clustered configuration so that they get high availability (HA) of the software. If one server fails, all traffic rolls over to the other cluster node without stopping. If such an application was to use a SSO approach to log in, then it is important that the SSO server be HA as well.

Interstage BPM has some special requirements on the OpenID provider: the user id MUST be a specified part of the OpenID URL. We cannot guarantee that other, third party, OpenID implementations will offer this pattern. So, to make sure that Interstage BPM can work with OpenID, we must offer a version of SSOFI that can be installed and run in a cluster configuration.

When configured in a cluster, the various node instances of the provider might receive any request at any time. The first request from the browser might go to one node, then next to a different one. The way that OpenID works is that the browser must get a token from the provider, and the server must call the provider to verify that the token is valid. The problem exists if the token is set up on one node, and the request for validation arrives at another.

2 Solution Overview

Non clustered versions of SSOFI keep session information in memory. When a user logs in, a session is created. When a user attempts to authentication with a particular application, a unique token is generated for that application, and stored in memory as well.

What is important is that all instances of the provider have the same session information. Two approaches for this are given.

2.1 Store in database

All of the information necessary can be stored in a database. For every request, the session information is retrieved from the DB. If anything is changed, the DB row is updated.

2.2 Replicated Memory

Some application servers offer a replicated session capability that automatically copies session information from each node to all other nodes so that all nodes have the same session information.

2.3 Tradeoffs

The advantage of the DB approach is that no exotic capability is needed. All app servers have standard ways to access a DB, and all conceivable clustered application servers will have a clustered HA DB to work off of. We have plenty of experience with DB and this can be quite reliably done. If a server crashes and restarts, it will always get consistent data from the DB.

The disadvantage of the DB approach is that the DB settings must be configured, and the DB must be initialized.

The advantage of the replicated memory approach is that no DB server is needed, and you don't have to initialize the database.

The disadvantage of the shared memory approach is that while it is supported in WebSphere, I don't think there is a standard for this, and it might not be supported in all environments. The setup for this might be as complex as setting up a DB, and we have no experience with troubleshooting these configurations. It is not clear how initialization of this memory is handled.

2.4 Desired Approach

The conservative approach is to store the session information in a DB. This has an additional advantage over the current implementation that if the server is stopped and restarted, the current user sessions will be preserved.

3 Details

3.1 What is stored

SSOFI provider is an OpenID provider, but at the same time it is interfacing with other systems to authenticate the user as well. It can authenticate the user using LDAP (which can be done as a simple invocation) and as Email (which requires sometimes multiple interactions to send an email to reset the password). Some of the information stored in the session is to support basic OpenID exchange, and other information for this other capability. The stored information in the session is for all of this.

The current SSOFI provider stores the following information about every session stored in the AuthAttemptRecord class:

- Paramlist – a ParameterList object used by the openid4java library.
- return_to - where the entire exchange will return to once done logging in – actually this is also in the savedParams below.
- identity - the originally passed identity to VERIFY – actually this is also in the savedParams below.
- regEmail - the email address supplied by the user for registering a new email address
- regMagicNo - the generated magic number which the user must receive in the email msg
- regEmailConfirmed – marked when registration is successful
- savedParams – a Properties object holding all the parameters from the original request. This is used by the openid4java library to implement the protocol.
- errMsg - if something goes wrong, this holds what should be displayed to the user the next time they access the provider. It is cleared after being displayed once.

Most of this could be simply serialized as a blob value – only the session id is needed to set and update this.

There would need to be a session id value that would be the unique key to get and update this record.

There would also need to be a timestamp that is updated when the user accesses the session. If the timestamp is too old upon reading, the current record is discarded and the application starts with a clean record. This should be outside the blob so that if the session times out the blob does not need to be decomposed, and so that it can be updated separately.

It turns out that if the entire AuthAttemptRecord is serialized out to a file, and if the file name is the session id, and the timestamp on the file is the timestamp of the session, then this can all be stored in a file system without too much trouble.

3.2 Code Structure

The SSOFI provider should be immediately redesigned to get and update through a session manager class that can isolate all the detail of this.

The AuthAttemptRecord class should be made a public class. Currently it is a private class used only inside one other class. This should be renamed "AuthSession" because it is used for more than just authenticate attempts.

There should be three implementations of the session manager:

- One that simply stores the AuthSession object in the J2EE servlet session
- One that stores all the sessions in a simple files on the file system, which might be a shared file system for clustered implementation.
- One that stores all the sessions in an actual DB

4 Conclusion

This is not a big change to the current SSOFI provider. It should be relatively easy to retrofit this for clustered use.