NativeSPODBCStorageService

The SP includes a plugin for storing persistent information, including user sessions, in a database using an ODBC driver.

Before trying to use this extension, strongly consider whether it's really necessary. Most applications worth clustering are not going to be able to rely on the SP's built-in session functionality. It's there primarily for simple web sites and smaller applications.

Even if you do need clustered sessions, a better option may be to run a single `shibd` instance and share it using a private network between the machines. It's a single point of failure, but so is a database in most cases.

Included below are the basic steps for getting a database session storage system setup. *(If you have experience with a specific database, please provide any information down below.)* The thing to remember is that the plugin is written to work with a fully ACID database that is providing 100% serializable transactions. This means taking locks on reads to prevent another transaction from updating the record until the plugin is done with it. To my knowledge, only a few databases do that *(and Oracle isn't one of them)*. The result will be a lot of trial and error to adjust other settings to compensate, so expect problems.

Database Setup

The odbc-store.so extension requires a database to store the shared sessions. The odbc-store.so extension does *not* automatically create any tables for itself. The rough SQL DDL for the database is the following:

```
CREATE TABLE version (
    major int NOT NULL,
    minor int NOT NULL
    )

CREATE TABLE strings (
    context varchar(255) not null,
    id varchar(255) not null,
    expires datetime not null,
    version smallint not null,
    value varchar(255) not null,
    PRIMARY KEY (context, id)
    )

CREATE TABLE texts (
    context varchar(255) not null,
    id varchar(255) not null,
    expires datetime not null,
    version smallint not null,
    value text not null,
    PRIMARY KEY (context, id)
    )
```

You may need to adjust data types when moving between database systems, but this has to be done carefully to maintain compatibility. The SQL itself is not yet pulled out of the plugin code, so you can't change it without changing the code.

Once the tables exist, you need to prepopulate the `version` table with the right information, which at the moment should be 1 and 0:

```
insert into version values (1,0)
```

ODBC Connection Strings

The primary piece of information the SP configuration will need is the connection string to use. The string has to be self-contained, and needs to identify the database server, the database or tablespace, and the username and password to connect with. All of this is driver specific, and may involve setting up other files on the machine to create an alias for the database server. See below for specific database tips.

SP Configuration

The default configuration file comes with some comments to help set this up. You'll need to uncomment the `<OutOfProcess>`/`<Extensions>` element containing the `odbc-store.so` extension. Then you'll need to comment out the four elements grouped together that manage in-memory storage and uncomment the four below them that are labeled with the ODBC comment.

shibboleth2.xml

```
<OutOfProcess logger="shibd.logger">
<Extensions>
  <Library path="odbc-store.so" fatal="true"/>
</Extensions>
</OutOfProcess>
```

Database Notes

Finally, apart from adjusting any of the timeout controls, you'll have to put the right connection string into the `<ConnectionString>` element.

Microsoft SQL Server

This is the primary test platform, and has been tested under only light load on both Windows and CentOS 5 (x86_64). On Linux, the FreeTDS client and ODBC driver from the OS were used.

The simplest way to set this up is to use the SQL Server or FreeTDS client and define a server alias so you can give the server a logical name. A typical connection string would then be:

```
DRIVER=drivername;SERVER=dbserver;UID=shibboleth;PWD=password;DATABASE=shibboleth;APP=Shibboleth
```

Creating the database with the default DDL worked fine.

MySQL

A bug report indicates that shibd is unstable on Red Hat 6 using the MySQL driver. The stack traces suggest crashes in the driver layer, and since the same SP code is stable on Red Hat 5, that points to a regression in the Red Hat supplied code.

Tested under light load on CentOS 5 (x86_64) using TCP to connect to a database on the same box. The latest ODBC driver from MySQL was used (although the installation of the RPM wasn't perfect).

The connection string may look like as simple as this:

```
DRIVER=MySql;OPTION=65536
```

The cryptic OPTION argument above tells ODBC to look for MySql configuration files for connection data. Hence you should create a MySql config file (ie. `/etc/mysql/conf.d/odbc.cnf`) accordingly, and make it readable by the user running *shibd* (eg. `_shibd`) only.

```
[odbc]
host = dbhost
user = shibboleth
password = secret
database = shibboleth
```

Alternatively, you can also specify the DSN in the main config file, though it's insecure:

```
DRIVER=MySQL;SERVER=sp.example.org;DATABASE=shibboleth;USER=root;PASSWORD=password
```

Creating the database with the default DDL worked fine. I used the InnoDB MySQL table type, which is somewhat (but not fully) transactional.

In general, MySQL presents a lot of problems because it doesn't take transactions seriously. It claims to support ACID behavior, but it mixes code for data storage with higher level code that doesn't honor those requirements, and tells you bluntly that many errors will result in only partial transaction rollback. It also has storage engines that operate like Oracle does, by not taking locks when told to. As such, I don't know how the plugin will behave under exceptional conditions.

PostgreSQL

- In PostgreSQL the **datetime** data type is called **timestamp** which requires a small change to the reference SQL from above:

shibboleth-sp.sql

```
CREATE TABLE version (
    major int NOT NULL,
    minor int NOT NULL
    );
INSERT INTO version VALUES (1,0);

CREATE TABLE strings (
    context varchar(255) NOT NULL,
    id      varchar(255) NOT NULL,
    expires timestamp    NOT NULL,
    version smallint     NOT NULL,
    value   varchar(255) NOT NULL,
    PRIMARY KEY (context, id)
    );

CREATE TABLE texts (
    context varchar(255) NOT NULL,
    id      varchar(255) NOT NULL,
    expires timestamp    NOT NULL,
    version smallint     NOT NULL,
    value   text         NOT NULL,
    PRIMARY KEY (context, id)
    );
```

- **GOTCHA:** be sure that your pg_hba.conf file is setup to allow IPv4 MD5 authentication from the network location of your Shibboleth-SP host:

~postgres/data/pg_hba.conf

```
local   all             postgres                        peer
host    all             all             127.0.0.1/32 md5
host    all             all             10.0.0.0/8   md5
```

- On **RHEL/CentOS Linux**, the **odbc-store.so** extension *(/usr/lib64/shibboleth/odbc-store.so)* uses **/usr/lib64/libpq.so**.
- You will need to install the **postgresql-libs** package which provides **/usr/lib64/libpq.so.5** and create a symlink *(symbolic link)* from the installed **libpq** to the location where **odbc-store.so** expects it:

Install postgresql-libs and create symlink

```
yum install -y postgresql-libs
ln -s /usr/lib64/libpq.so.5 /usr/lib64/libpq.so
```

- Configure the connection string in the **shibboleth2.xml** file:

shibboleth2.xml

```
  <StorageService type="ODBC" id="db" cleanupInterval="900">
    <ConnectionString><![CDATA[
Driver=PostgreSQL;Server=127.0.0.1;Port=5432;Database=shibboleth-sp;Uid=shibboleth-sp;Password=shibboleth-sp-
password
]]></ConnectionString>
  </StorageService>
  <SessionCache type="StorageService" StorageService="db" cacheTimeout="3600" inprocTimeout="900"
cleanupInterval="900" />
  <ReplayCache StorageService="db" />
  <ArtifactMap StorageService="db" artifactTTL="180" />
```

- After adding the "**db**" **storage service**, configure the **session initiator** and **logout initiator** to use **ss:db** *(db storage service)* for the **relay state**.

shibboleth2.xml

```
<SessionInitiator type="Chaining" Location="/Login" isDefault="true" id="Default"
        relayState="ss:db" forceAuthn="true" entityID="">
    <SessionInitiator type="SAML2" acsIndex="1" template="bindingTemplate.html"/>
</SessionInitiator>

<LogoutInitiator type="Chaining" Location="/Logout" relayState="ss:db">
    <LogoutInitiator type="SAML2" template="bindingTemplate.html"/>
    <LogoutInitiator type="Local"/>
</LogoutInitiator>
```

Oracle