StorageConfiguration

File(s): *conf/global.xml, conf/idp.properties*

Format: Native Spring

Overview

The IdP provides a number of general-purpose storage facilities that can be used by core subsystems like session management and consent. Broadly speaking, there are two kinds of storage plugins: client-side and server-side. Client-side plugins have the advantage of requiring no additional software or configuration and make clustering very robust and simple, but they only support a subset of use cases. Server-side plugins (aside from the simple case of storing data in memory) support all use cases, but require additional software and configuration, and usually create additional points of failure in a clustered deployment.

The IdP ships with 3 preconfigured org.opensaml.storage.StorageService beans:

- **shibboleth.ClientSessionStorageService** (of type ClientStorageService)
  - Stores data in a browser session cookie or HTML local storage[3.2]
- **shibboleth.ClientPersistentStorageService** (of type ClientStorageService)
  - Stores data in a long-lived browser cookie or HTML local storage[3.2]
- **shibboleth.StorageService** (of type MemoryStorageService)
  - Stores data in server memory, does not survive restarts and is not replicated across a cluster

There are additional storage service plugins included in the software (JPA, memcached) but they are not predefined. Using them requires defining beans yourself and setting various properties to point to them.

By default, the **shibboleth.ClientSessionStorageService** bean, which stores data in the client, is used to store IdP session data, but that can be modified via the *idp.properties* file:

Example changing IdP session storage

```
idp.session.StorageService = shibboleth.StorageService
```

There are additional properties that can be used to change how other data is stored on a per-use case basis, but note that some components can't rely on client-side storage options.

Various properties controlling data storage

```
# These all require server-side storage
#idp.replayCache.StorageService = shibboleth.StorageService
#idp.artifact.StorageService = shibboleth.StorageService
#idp.cas.StorageService=shibboleth.StorageService

# This defaults to longer-lived client-side storage
#idp.consent.StorageService = shibboleth.ClientPersistentStorageService
```

Reference

Beans

The following beans (most of which are internal to the system) can be used in various properties to control what storage instances are used for specific purposes. You can define your own beans also (e.g. in *global.xml*).

Bean ID	Type	Function

shibboleth.StorageService	MemoryStorageService	Default server-side storage, stores data in memory
shibboleth.ClientSessionStorageService	ClientStorageService	Default client-side storage for per-session data, stores data in session cookies or HTML local storage [3.2]
shibboleth.ClientPersistentStorageService	ClientStorageService	Default client-side storage for long-lived data, stores data in persistent cookies or HTML local storage [3.2]
shibboleth.ClientStorageServices	List<StorageService>	Enumeration of ClientStorageService plugins used, ensures proper load/save of data

Properties

Property	Type	Default	Function
idp.cookie.secure	Boolean	false	Whether cookies created by the software include the "secure" attribute; the default is mostly an accident, you should strongly consider setting this
idp.cookie.httpOnly	Boolean	true	Whether cookies created by the software include the "httpOnly" attribute (excepting a few user-preference cookies that are explicitly meant to be accessed by JavaScript)
idp.cookie.domain	String		Optional domain to attach to cookies
idp.cookie.path	String		Optional path to attach to cookies
idp.cookie.maxAge	Integer	31536000	Lifetime of non-session cookies
idp.storage.cleanupInterval	Duration	PT10M	Interval of background thread sweeping server-side storage for expired records
idp.storage.htmlLocalStorage	Boolean	false	Whether to use HTML Local Storage (if available) instead of cookies
idp.storage.clientSessionStorageName [3.3]	String	shib_idp_session_ss	Name of cookie or HTML storage key used by the default per-session instance of the client storage service
idp.storage.clientPersistentStorageName [3.3]	String	shib_idp_persistent_ss	Name of cookie or HTML storage key used by the default persistent instance of the client storage service
idp.session.StorageService	Bean ID of a StorageService	shibboleth.ClientSessionStorageService	Storage back-end to use for IdP sessions, authentication results, and optionally tracking of SP usage for logout
idp.consent.StorageService	Bean ID of a StorageService	shibboleth.ClientPersistentStorageService	Storage back-end to use for consent and terms-of-use records
idp.replayCache.StorageService	Bean ID of a StorageService	shibboleth.StorageService	Storage back-end to use for message replay checking (must be server-side)
idp.replayCache.strict [3.4]	Boolean	true	Whether storage errors during replay checks should be treated as a replay
idp.artifact.StorageService	Bean ID of a StorageService	shibboleth.StorageService	Storage back-end to use for short-lived SAML Artifact mappings (must be server-side)
idp.cas.StorageService	Bean ID of a StorageService	shibboleth.StorageService	Storage back-end to use for CAS ticket mappings (must be server-side)

ClientStorageService

Versions of the IdP prior to V3.2.0 included a cookie-backed storage plugin used by default for IdP sessions and authentication results. A drop-in replacement for this plugin is used in subsequent versions that includes support for HTML Local Storage along with cookies if the client supports that feature. The new plugin is functionally quite different in its behavior but no configuration changes are required to use it, apart from the decision to enable the Local Storage feature.

There actually is one configuration change you can make: the original deployment descriptor in **webapp/WEB-INF/web.xml** contains a servlet filter declaration that supported the old cookie-based storage plugin. That filter has been stubbed out in the new version so if you want to gain some infinitesimal efficiency gains, you can copy the file to your *edit-webapp* tree, remove the following markup, and rebuild the warfile:

Superfluous web.xml Content in V3.2.0+

```
    <!-- Automates the unpack and pack of the cookie-based storage model. -->
    <filter>
        <filter-name>ClientSessionStorageServiceFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        <init-param>
            <param-name>targetBeanName</param-name>
            <param-value>shibboleth.ClientSessionStorageService</param-value>
        </init-param>
    </filter>
    <!-- Automates the unpack and pack of the cookie-based storage model. -->
    <filter>
        <filter-name>ClientPersistentStorageServiceFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        <init-param>
            <param-name>targetBeanName</param-name>
            <param-value>shibboleth.ClientPersistentStorageService</param-value>
        </init-param>
    </filter>
...

    <filter-mapping>
        <filter-name>ClientSessionStorageServiceFilter</filter-name>
        <url-pattern>/profile/Logout</url-pattern>
        <url-pattern>/profile/Shibboleth/SSO</url-pattern>
        <url-pattern>/profile/SAML2/Unsolicited/SSO</url-pattern>
        <url-pattern>/profile/SAML2/Redirect/SSO</url-pattern>
        <url-pattern>/profile/SAML2/POST/SSO</url-pattern>
        <url-pattern>/profile/SAML2/POST-SimpleSign/SSO</url-pattern>
        <url-pattern>/profile/SAML2/Redirect/SLO</url-pattern>
        <url-pattern>/profile/SAML2/POST/SLO</url-pattern>
        <url-pattern>/profile/SAML2/POST-SimpleSign/SLO</url-pattern>
        <url-pattern>/profile/cas/login</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>ClientPersistentStorageServiceFilter</filter-name>
        <url-pattern>/profile/Shibboleth/SSO</url-pattern>
        <url-pattern>/profile/SAML2/Unsolicited/SSO</url-pattern>
        <url-pattern>/profile/SAML2/Redirect/SSO</url-pattern>
        <url-pattern>/profile/SAML2/POST/SSO</url-pattern>
        <url-pattern>/profile/SAML2/POST-SimpleSign/SSO</url-pattern>
        <url-pattern>/profile/cas/login</url-pattern>
    </filter-mapping>
```

We have not enabled it by default to maintain consistency with the original release, but it will be enabled in future versions. There are visual impacts from the use of the feature and it requires JavaScript be enabled, because reading and writing to the client requires an explicit page be rendered.

Enabling this feature is very simple: just set the **idp.storage.htmlLocalStorage** property to true.

No configuration is required, but you may want to change the look and feel of the templates that are displayed to the client while data is being read or written. These pages don't require any user interaction as long as JavaScript is enabled, but they tend to be visible at least briefly, particularly the first time through. They're somewhat similar to the templates displayed when SAML messages are handed off to the browser.

Much of that look is obviously controlled by style sheets and message properties, but the "visible" portions have been moved into *views/client-storage* as of V3.3 (to avoid losing your changes on upgrades).

Note that this feature is safe to enable globally. The implementation is written to check for this capability in each client, and to back off to cookies.

As to **why** you would do this, there are really two main reasons:

- Consent
- Logout

The consent feature is very limited when cookies are used because the number of records it can store is extremely small. If Local Storage is available, that limit is essentially ignored. If you're comfortable with tracking consent per-device, this is a much more practical way to deploy it at most sites than with a database. Of course, many deployers are not comfortable with per-device consent.

The main reason for this feature is that by default, the IdP's session manager is configured not to track or index the sessions created with SPs, because that information also does not fit reliably in a cookie. That makes the single-logout feature unusable since the IdP doesn't know what SPs to communicate with. Turning on the Local Storage feature is necessary but not sufficient to allow at least some form of single logout to work without moving session storage to the server. You also will need to enable a couple of additional session management properties **(idp.session.trackSPSessions** and **idp.session.secondaryServiceIndex)**. There are two properties because the latter is more a SAML-specific need that may not extend to other protocols in the future.

JPAStorageService

The JPA storage facility uses Hibernate ORM for searching and persistence using a relational database for storage. Example schemas are shown below.

Whatever you do, you MUST ensure the context and id columns are case-sensitively handled and compared. That is a requirement of the API that will be using the database.

MySQL

```
CREATE TABLE `StorageRecords` (
  `context` varchar(255) NOT NULL,
  `id` varchar(255) NOT NULL,
  `expires` bigint(20) DEFAULT NULL,
  `value` longtext NOT NULL,
  `version` bigint(20) NOT NULL,
  PRIMARY KEY (`context`,`id`)
)
```

PostgreSQL or H2

```
CREATE TABLE storagerecords (
  context varchar(255) NOT NULL,
  id varchar(255) NOT NULL,
  expires bigint DEFAULT NULL,
  value text NOT NULL,
  version bigint NOT NULL,
  PRIMARY KEY (context, id)
);
```

Oracle

```
CREATE TABLE storagerecords (
  context varchar2(255) NOT NULL,
  id varchar2(255) NOT NULL,
  expires number(19,0),
  value clob NOT NULL,
  version number(19,0) NOT NULL,
  PRIMARY KEY (context, id)
);
```

In order to configure this service you must provide Spring bean configuration for the JPAStorageService that includes the driver, URL, and credentials for your database. You are required to provide a jar containing the driver for your particular database. In addition, we recommend the use of a DataSource that provides connection pooling, which may require installing an additional library as well.

The following libraries provide connection pooling functionality:

- Commons DBCP
- Tomcat JDBC Pool
- BoneCP
- HikariCP

In the DB-specific examples below use of HikariCP is demonstrated (`class="com.zaxxer.hikari.HikariDataSource", p:jdbcUrl="..."` in the DataSource bean). When using other Connection Pool implementations change the class and properties appropriately, e.g.:

- Apache Commons: `class="org.apache.commons.dbcp.BasicDataSource", p:url="..."`
- Tomcat DBCP2: `class="org.apache.tomcat.dbcp.dbcp2.BasicDataSource", p:url="..."`
- Tomcat JDBC Pool: `class="org.apache.tomcat.jdbc.pool.DataSource", p:url="..."`

Installation

Place the driver jar and connection pooling jar in **edit-webapp/WEB-INF/lib** then execute *bin/build.sh* or *bin/build.bat* as appropriate for your environment.

The following configuration should be placed in *conf/global.xml*:

DB-independent Configuration

```
<bean id="shibboleth.JPAStorageService"
    class="org.opensaml.storage.impl.JPAStorageService"
    p:cleanupInterval="%{idp.storage.cleanupInterval:PT10M}"
    c:factory-ref="shibboleth.JPAStorageService.EntityManagerFactory" />

<bean id="shibboleth.JPAStorageService.EntityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="storageservice" />
    <property name="packagesToScan" value="org.opensaml.storage.impl" />
    <property name="dataSource" ref="shibboleth.JPAStorageService.DataSource" />
    <property name="jpaVendorAdapter" ref="shibboleth.JPAStorageService.JPAVendorAdapter" />
    <property name="jpaDialect">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaDialect" />
    </property>
</bean>
```


The specific examples that follow should NOT be assumed to be functional, as they likely are the product of different sources, varying amounts of testing (including none), and may not be current. Drivers get updated frequently and JDBC and database bugs appear and disappear with regularity. When in doubt, always grab new ones when problems appear.

Postgres Configuration

```
<!-- Postgres configuration -->
<bean id="shibboleth.JPAStorageService.JPAVendorAdapter"
    class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="POSTGRESQL" />
</bean>
<bean id="shibboleth.JPAStorageService.DataSource"
    class="com.zaxxer.hikari.HikariDataSource" destroy-method="close" lazy-init="true"
    p:driverClassName="org.postgresql.Driver"
    p:jdbcUrl="jdbc:postgresql://localhost:5432/storageservice"
    p:username="shib"
    p:password="p@ssw0rd" />
```

MySQL Configuration

```
<bean id="shibboleth.JPAStorageService.JPAVendorAdapter"
    class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="MYSQL" />
</bean>
<bean id="shibboleth.JPAStorageService.DataSource"
    class="com.zaxxer.hikari.HikariDataSource" destroy-method="close" lazy-init="true"
    p:driverClassName="com.mysql.jdbc.Driver"
    p:jdbcUrl="jdbc:mysql://localhost:3306/storageservice"
    p:username="shib"
    p:password="p@ssw0rd" />
```

Oracle Configuration

```
     <bean id="shibboleth.JPAStorageService.JPAVendorAdapter"
        class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="database" value="ORACLE" />
    </bean>
    <bean id="shibboleth.JPAStorageService.DataSource"
        class="com.zaxxer.hikari.HikariDataSource" destroy-method="close" lazy-init="true"
        p:driverClassName="oracle.jdbc.OracleDriver"
        p:jdbcUrl="jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=yes)(ADDRESS=(PROTOCOL=TCP)(HOST=ora-scan-
address)(port=1521))(CONNECT_DATA=(SERVICE_NAME=shib)))"
        p:username="shibboleth"
        p:password="p@ssw0rd" />
```

H2 Configuration

```
    <!-- H2 configuration -->
    <bean id="shibboleth.JPAStorageService.JPAVendorAdapter"
        class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="database" value="H2" />
    </bean>
    <bean id="shibboleth.JPAStorageService.DataSource"
        class="com.zaxxer.hikari.HikariDataSource" destroy-method="close" lazy-init="true"
        p:driverClassName="org.h2.Driver"
        p:jdbcUrl="jdbc:h2:file:/opt/shibboleth-idp/db/h2" />
```

Further Configuration

The LocalContainerEntityManagerFactoryBean contains more configuration options and is designed to eliminate the need for a persistence.xml file. If you need to alter the database schema you can deploy a custom mapping file which overrides column names and types.

ORM Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
                                     http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">
  <package>org.opensaml.storage.impl</package>

  <!-- changes the table name and column names -->
  <entity class="JPAStorageRecord" access="PROPERTY">
    <table name="StorageService.shibrecords"/>
    <id-class class="JPAStorageRecord$RecordId"/>
    <attributes>
      <id name="context">
        <column name="record_ctx" nullable="false"/>
      </id>
      <id name="key">
        <column name="record_id" nullable="false"/>
      </id>
      <basic name="value">
        <column name="record_val" nullable="false"/>
        <lob/> <!-- without this the default type is varchar -->
      </basic>
      <basic name="expiration">
        <column name="record_exp" nullable="true"/>
      </basic>
      <basic name="version">
        <column name="record_ver" nullable="false"/>
      </basic>
    </attributes>
  </entity>
  <embeddable class="JPAStorageRecord$RecordId" access="PROPERTY">
    <attributes>
      <basic name="context">
        <column name="record_ctx" nullable="false"/>
      </basic>
      <basic name="key">
        <column name="record_id" nullable="false"/>
      </basic>
    </attributes>
  </embeddable>
</entity-mappings>
```

Place your custom orm.xml file in *edit-webapp/WEB-INF/classes/META-INF/orm.xml* then rebuild your war. While you can configure a custom name and path for this file it **must** be located on your web application classpath. File system paths are not supported.

Postgres LOB Concerns

Switch identified an issue with the Postgres JDBC driver and the storage of LOBs related to the default mapping. Deployers can experience data loss when the Postgres vacuumlo command is run. It is recommend that a custom orm.xml file be used to override the value type:

Postgres ORM

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/persistence
/orm_1_0.xsd" version="1.0">
<package>org.opensaml.storage.impl</package>
<entity class="JPAStorageRecord" access="PROPERTY">
  <attributes>
    <basic name="value">
      <column name="value" nullable="false"/>
    </basic>
  </attributes>
</entity>
</entity-mappings>
```

See the Switch Installation Docs for more details.

MemcachedStorageService

Requirements: memcached v1.4.14 or later

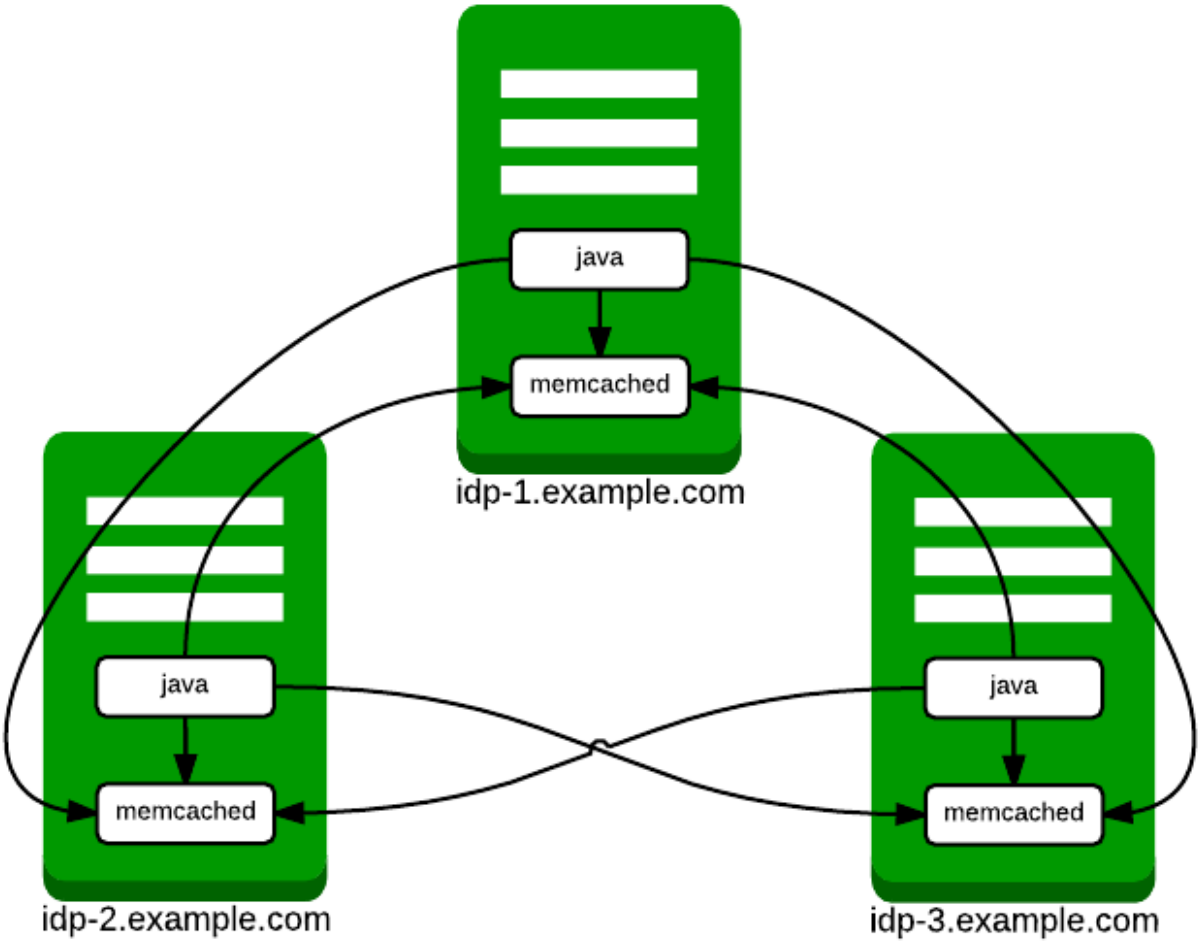
The memcached-based storage facility in IdPv3 is based on the spymemcached library, which has a number of compelling features for HA deployments:

- Optimized IO for high throughput
- Memcached failover facility
- Stable hashing algorithm supports memcached pool resizing

The failover facility merits further discussion. Failover is enabled by specifying multiple memcached hosts and `failureMode="Redistribute"`. When the client encounters an unreachable host in redistribute mode, it will temporarily remove the unavailable host from the pool of available hosts. Any keys that hash to the unavailable host will be written or retrieved from a backup host. The high-level effect of this behavior on the IdP session management service is that a node failure will cause loss of IdP session, which would impact users as an unexpected authentication. The IdP session management service, however, would be fully functional during a host failure/recovery event. Also note that this behavior requires no state sharing (i.e. *repcache*) between memcached nodes.

Bear in mind that different storage use cases have different failover properties. While the replay cache would be similarly unimpacted, the artifact map failing to retrieve a previously stored artifact mapping would result in a failed login to the service to which the artifact was sent.

The following architecture is strongly recommended for HA deployments:



Thus every IdP node runs a memcached service and the Java process running the IdP software connects to every memcached service. The following configuration example assumes the recommended architecture above and should be placed in *conf/global.xml* .

MemcachedStorageService Configuration

```
 <bean id="shibboleth.MemcachedStorageService"
      class="org.opensaml.storage.impl.memcached.MemcachedStorageService"
      c:timeout="2">
    <constructor-arg name="client">
        <bean class="net.spy.memcached.spring.MemcachedClientFactoryBean"
              p:servers="idp-1.example.com:11211,idp-2.example.com:11211,idp-3.example.com:11211"
              p:protocol="BINARY"
              p:locatorType="CONSISTENT"
              p:failureMode="Redistribute">
            <property name="hashAlg">
                <util:constant static-field="net.spy.memcached.DefaultHashAlgorithm.FNV1_64_HASH" />
            </property>
            <property name="transcoder">
                <!-- DO NOT MODIFY THIS PROPERTY -->
                <bean class="org.opensaml.storage.impl.memcached.StorageRecordTranscoder" />
            </property>
        </bean>
    </constructor-arg>
</bean>
```

Once a MemcachedStorageService bean has been defined as above, it can be used with subsystems that require a StorageService component. The following configuration snippet from *conf/idp.properties* indicates how to use memcached for session storage.

Memcached for IdP Sessions

```
idp.session.StorageService = shibboleth.MemcachedStorageService
```