Syncing groups between group management systems

Syncing Groups between Group Management Systems

Grouper allow sharing a group between two Grouper management systems. This is accomplished through push, pull, and incremental_push. In the future we may add permissions.

First there are connections to other Groupers based on the Grouper client. These connections are stored in the grouper.properties. The sources in the remote and local can be matched up.

Second, there are links of group to group based on these connections.

The subjects are synced based on external subjects generally. A key point is there if the external subject does not exist in the destination, it can be added dynamically. Note that the remote Grouper can be v2.0+, it has been tested with v1.6, and probably works with previous versions, though it is most valuable when there are external subjects on both sides (generally v2.0+).

A config might look like this in the grouper-loader.properties to define the connection to the school (similar to how we define various DB connections for the loader):

```
#######################################
## Grouper client connections
## if this grouper needs to talk to another grouper, this is the client connection information
#######################################


# id of the source, should match the part in the property name
grouperClient.someOtherSchool.id = someOtherSchool

# url of web service, should include everything up to the first resource to access
# e.g. https://groups.school.edu/grouperWs/servicesRest
grouperClient.someOtherSchool.properties.grouperClient.webService.url = https://some.other.school.edu/grouperWs
/servicesRest

# login ID
grouperClient.someOtherSchool.properties.grouperClient.webService.login = someRemoteLogin

# password for shared secret authentication to web service
# or you can put a filename with an encrypted password
grouperClient.someOtherSchool.properties.grouperClient.webService.password = *********

# client version should match or be related to the server on the other end...
grouperClient.someOtherSchool.properties.grouperClient.webService.client.version = v2_0_000

# this is the subject to act as local, if blank, act as GrouperSystem, specify with SubjectFinder packed
string, e.g.
# subjectIdOrIdentifier  or  sourceId:::subjectId  or  :::subjectId  or  sourceId::::::
subjectIdentifier  or  ::::::subjectIdentifier
# sourceId::::::::subjectIdOrIdentifier  or  ::::::::subjectIdOrIdentifier
grouperClient.someOtherSchool.localActAsSubject =

# the id of this source, generally the same as the name in the property name.  This is mandatory
grouperClient.someOtherSchool.source.jdbc.id = jdbc

# the part between "grouperClient.someOtherSchool.source." and ".id" links up the configs,
# in this case, "jdbc", make sure it has no special chars.  sourceId can be blank if you dont want to specify
grouperClient.someOtherSchool.source.jdbc.local.sourceId = jdbc

# this is the identifier that goes between them, it is "id" or an attribute name.  subjects without this
attribute will not be processed
grouperClient.someOtherSchool.source.jdbc.local.read.subjectId = identifier

# this is the identifier to lookup to add a subject, should be "id" or "identifier" or "idOrIdentifier"
```

```
grouperClient.someOtherSchool.source.jdbc.local.write.subjectId = identifier

# sourceId of the remote system, can be blank
grouperClient.someOtherSchool.source.jdbc.remote.sourceId = jdbc

# this is the identifier that goes between them, it is "id" or an attribute name.  subjects without this
attribute will not be processed
grouperClient.someOtherSchool.source.jdbc.remote.read.subjectId =

# this is the identifier to lookup to add a subject, should be "id" or "identifier" or "idOrIdentifier"
grouperClient.someOtherSchool.source.jdbc.remote.write.subjectId =



#######################################
## Sync to/from another grouper
## Only sync one group to one other group, do not sync one group to
## two report groupers.  If you need to do this, add the group to another group
#######################################

# we need to know where our
# connection name in grouper client connections above
syncAnotherGrouper.testGroup0.connectionName = someOtherSchool

# incremental  or  push  or   pull  or  incremental_push.  Note, incremental push is cron'ed and incremental
(to make sure no discrepancies arise)
syncAnotherGrouper.testGroup0.syncType = incremental_push

# quartz cron  to schedule the pull or push (incremental is automatic as events happen) (e.g. 5am daily)
syncAnotherGrouper.testGroup0.cron =  0 0 5 * * ?

# local group which is being synced
syncAnotherGrouper.testGroup0.local.groupName = test:testGroup

# remote group at another grouper which is being synced
syncAnotherGrouper.testGroup0.remote.groupName = test2:testGroup2

# if subjects are external and should be created if not exist
syncAnotherGrouper.testGroup0.addExternalSubjectIfNotFound = true
```

This is using the grouper client, so the authentication is pluggable.

We should look at real time proxying of the getMembers() call to the remote site.

Proof of concept setup

Add a group to grouper1

```
gsh 0% grouperSession = GrouperSession.startRootSession();
gsh 1% new GroupSave(grouperSession).assignName("aStem:anotherGroup").assignCreateParentStemsIfNotExist(true).
save();
```

Add group to grouper 2

```
gsh 0% grouperSession = GrouperSession.startRootSession();
gsh 1% new GroupSave(grouperSession).assignName("aStem2:anotherGroup2").assignCreateParentStemsIfNotExist(true).
save();
```

Test grouper1 with grouper client

```
C:\temp\grouperClient_v2_0>java -jar grouperClient.jar --operation=addMemberWs --groupName=aStem:anotherGroup --
subjectIds=GrouperSystem
C:\temp\grouperClient_v2_0>java -jar grouperClient.jar --operation=getMembersWs --groupNames=aStem:anotherGroup
```

Test grouper2 with grouper client

```
C:\temp\grouperClient_v2_0>java -jar grouperClient.jar --operation=groupSaveWs --name=aStem2:anotherGroup2 --
createParentStemsIfNotExist=true
C:\temp\grouperClient_v2_0>java -jar grouperClient.jar --operation=addMemberWs --groupName=aStem2:anotherGroup2
--subjectIds=GrouperSystem
C:\temp\grouperClient_v2_0>java -jar grouperClient.jar --operation=getMembersWs --groupNames=aStem2:
anotherGroup2
```

Save this file into subjects.sql and run it to load test subjects

Create a connection from grouper1 to grouper2 in the grouper.properties

```
#######################################
## Grouper client connections
## if this grouper needs to talk to another grouper, this is the client connection information
#######################################


# id of the source, should match the part in the property name
grouperClient.localhostGrouper2.id = localhostGrouper2

# url of web service, should include everything up to the first resource to access
# e.g. https://groups.school.edu/grouperWs/servicesRest
grouperClient.localhostGrouper2.properties.grouperClient.webService.url = http://localhost:8091/grouperWs2
/servicesRest

# login ID
grouperClient.localhostGrouper2.properties.grouperClient.webService.login = GrouperSystem

# password for shared secret authentication to web service
# or you can put a filename with an encrypted password
grouperClient.localhostGrouper2.properties.grouperClient.webService.password = **********

# client version should match or be related to the server on the other end...
grouperClient.localhostGrouper2.properties.grouperClient.webService.client.version = v2_0_000

# the id of this source, generally the same as the name in the property name.  This is mandatory
grouperClient.localhostGrouper2.source.jdbc.id = jdbc

# the part between "grouperClient.someOtherSchool.source." and ".id" links up the configs,
# in this case, "jdbc", make sure it has no special chars.  sourceId can be blank if you dont want to specify
grouperClient.localhostGrouper2.source.jdbc.local.sourceId = jdbc

# this is the identifier that goes between them, it is "id" or an attribute name.  subjects without this
attribute will not be processed
grouperClient.localhostGrouper2.source.jdbc.local.read.subjectId = id

# this is the identifier to lookup to add a subject, should be "id" or "identifier" or "idOrIdentifier"
grouperClient.localhostGrouper2.source.jdbc.remote.write.subjectId = idOrIdentifier

# if subjects are external and should be created if not exist
#grouperClient.someOtherSchool.source.jdbc.addExternalSubjectIfNotFound = true




#######################################
## Sync to/from another grouper
#######################################

# we need to know where our
# connection name in grouper client connections above
syncAnotherGrouper.anotherGroup.connectionName = localhostGrouper2

# incremental  or  push  or   pull  or  incremental_push.  Note, incremental push is cron'ed and incremental
(to make sure no discrepancies arise)
syncAnotherGrouper.anotherGroup.syncType = incremental_push

# quartz cron  to schedule the pull or push (incremental is automatic as events happen) (e.g. 5am daily)
#syncAnotherGrouper.testGroup0.cron =  0 0 5 * * ?

# local group which is being synced
syncAnotherGrouper.anotherGroup.local.groupName = aStem:anotherGroup

# remote group at another grouper which is being synced
syncAnotherGrouper.anotherGroup.remote.groupName = aStem2:anotherGroup2
```

Make sure the new default change log consumer is in the grouper-loader.properties

```
changeLog.consumer.syncGroups.class = edu.internet2.middleware.grouper.client.GroupSyncConsumer
changeLog.consumer.syncGroups.quartzCron =
```

Add a member to grouper1 in the group configured:

```
gsh 12% addMember("aStem:anotherGroup", "bawi");
true
```

See the web service call go to grouper2:

```
<WsRestAddMemberRequest>
  <wsGroupLookup>
   <groupName>aStem2:anotherGroup2</groupName>
  </wsGroupLookup>
  <subjectLookups>
   <WsSubjectLookup>
    <subjectId>bawi</subjectId>
    <subjectIdentifier>bawi</subjectIdentifier>
   </WsSubjectLookup>
  </subjectLookups>
</WsRestAddMemberRequest>
```

Check grouper2 for the member in the remote group name

```
gsh 2% hasMember("aStem2:anotherGroup2", "bawi")
true
```

Delete it from the original

```
gsh 13% delMember("aStem:anotherGroup", "bawi");
```

See the web service call go to grouper2:

```
<WsRestDeleteMemberRequest>
  <wsGroupLookup>
    <groupName>aStem2:anotherGroup2</groupName>
  </wsGroupLookup>
  <subjectLookups>
    <WsSubjectLookup>
      <subjectId>bawi</subjectId>
      <subjectIdentifier>bawi</subjectIdentifier>
    </WsSubjectLookup>
  </subjectLookups>
</WsRestDeleteMemberRequest>
```

Check grouper2 for the member in the remote group name

```
gsh 3% hasMember("aStem2:anotherGroup2", "bawi")
false
```

Unit test

There is a unit test: GroupSyncDaemonTest

This unit test requires a WS server running, and a login, and privileges, so it does not run by default. To run this, set these params in the grouper.properties

```
# if the group sync should be tested... note you need the demo server available to test this, or change some
settings...
junit.test.groupSync = false
junit.test.groupSync.url = https://grouperdemo.internet2.edu/grouper-ws_v2_0_0/servicesRest
junit.test.groupSync.user = remoteUser
junit.test.groupSync.password = R:/pass/grouperDemoRemoteUser.pass
#folder where the user can create/stem which the user can use to run tests
junit.test.groupSync.folder = test2:whateverFolder
#this is true unless testing to an older grouper which doesnt support this
junit.test.groupSync.addExternalSubjectIfNotExist = true
junit.test.groupSync.createRemoteFolderIfNotExist = true
```

Make sure the endpoint has a folder available and that the login user has privileges to use it:

```
[mchyzer@i2midev1 bin]$ sudo htpasswd /etc/httpd/conf.d/users.pass remoteUser

gsh 0% grouperSession = GrouperSession.startRootSession();
gsh 1% addSubject("remoteUser", "application", "remoteUser");
gsh 2% addMember("etc:webServiceClientUsers", "remoteUser");
gsh 3% addRootStem("test2", "test2");
gsh 4% addStem("test2", "whateverFolder", "whateverFolder");
gsh 5% grantPriv("test2:whateverFolder", "remoteUser", NamingPrivilege.STEM);
gsh 6% grantPriv("test2:whateverFolder", "remoteUser", NamingPrivilege.CREATE);
```

If you have run the test against 2.0+, and want to clear out the groups and users, run this in GSH:

```
grouperSession = GrouperSession.startRootSession();
delGroup("test2:whateverFolder:remoteTestPull");
delGroup("test2:whateverFolder:remoteTestPush");
delGroup("test2:whateverFolder:remoteTestPushIncremental");
externalSubject = GrouperDAOFactory.getFactory().getExternalSubject().findByIdentifier("testuser1@internet2.
edu", true, null);
if (externalSubject != null) {externalSubject.delete();}
externalSubject = GrouperDAOFactory.getFactory().getExternalSubject().findByIdentifier("testuser2@internet2.
edu", true, null);
if (externalSubject != null) {externalSubject.delete();}
externalSubject = GrouperDAOFactory.getFactory().getExternalSubject().findByIdentifier("testuser3@internet2.
edu", true, null);
if (externalSubject != null) {externalSubject.delete();}
externalSubject = GrouperDAOFactory.getFactory().getExternalSubject().findByIdentifier("testuser4@internet2.
edu", true, null);
if (externalSubject != null) {externalSubject.delete();}
externalSubject = GrouperDAOFactory.getFactory().getExternalSubject().findByIdentifier("testuser7@internet2.
edu", true, null);
if (externalSubject != null) {externalSubject.delete();}
```

Point that somewhere, give the login privileges, and you can test the pull, push, or incremental_push. Note, if you arent running v2.0 with external members on the remote WS server, then you need to make sure the following subjects are available:

```
gsh 0% addSubject("testuser1@internet2.edu", "person", "Test User1 at Internet2");
gsh 0% addSubject("testuser2@internet2.edu", "person", "Test User2 at Internet2");
gsh 0% addSubject("testuser3@internet2.edu", "person", "Test User3 at Internet2");
gsh 0% addSubject("testuser4@internet2.edu", "person", "Test User4 at Internet2");
gsh 0% addSubject("testuser7@internet2.edu", "person", "Test User7 at Internet2");

INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser1@internet2.edu', 'loginid', 'testuser1@internet2.edu', 'testuser1@internet2.edu');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser1@internet2.edu', 'name', 'Test User1 at Internet2', 'test user1 at internet2');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser1@internet2.edu', 'description', 'Test User1 at Internet2', 'test user1 at internet2');

INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser2@internet2.edu', 'loginid', 'testuser2@internet2.edu', 'testuser2@internet2.edu');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser2@internet2.edu', 'name', 'Test User2 at Internet2', 'test user2 at internet2');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser2@internet2.edu', 'description', 'Test User2 at Internet2', 'test user2 at internet2');

INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser3@internet2.edu', 'loginid', 'testuser3@internet2.edu', 'testuser3@internet2.edu');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser3@internet2.edu', 'name', 'Test User3 at Internet2', 'test user3 at internet2');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser3@internet2.edu', 'description', 'Test User3 at Internet2', 'test user3 at internet2');

INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser4@internet2.edu', 'loginid', 'testuser4@internet2.edu', 'testuser4@internet2.edu');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser4@internet2.edu', 'name', 'Test User4 at Internet2', 'test user4 at internet2');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser4@internet2.edu', 'description', 'Test User4 at Internet2', 'test user4 at internet2');

INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser7@internet2.edu', 'loginid', 'testuser7@internet2.edu', 'testuser7@internet2.edu');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser7@internet2.edu', 'name', 'Test User7 at Internet2', 'test user7 at internet2');
INSERT INTO subjectattribute (subjectId, NAME, VALUE, searchValue)
VALUES ('testuser7@internet2.edu', 'description', 'Test User7 at Internet2', 'test user7 at internet2');
COMMIT;
```

Note: in the future, we would like this provisioning to occur with SPML similar to ldappcng, speak the common Groups API (up and coming). Therefore anything that speaks the (standard) API could be an endpoint. Perhaps the design of web services and xmpp could use SPML as the data format.