Columbia University Grouper Deployment Guide

Version 1.x 2016-XX-XX

This document provides technical guidance for an initial deployment of Internet2 Grouper. Recommendations on configuration and operation are based on analysis and discussion of existing identity and access management architecture and potential use cases at Columbia University.

Table of Contents

1.0 Initial Install with the Grouper Installer

Grouper is installed, patched and upgraded using the Grouper Installer. The installer will download and configure all necessary components and will prompt for database connection strings, usernames and passwords, and a number of configuration elements and deployment choices.

The grouper installer can install all of the current grouper components which includes:

- grouper.apiBinary - used for running the grouper command line client and the grouper loader daemon
- grouper.ui - Java Servlet web-based user interface
- grouper.ws - SOAP and RESTful web service endpoints - used by the grouper client and for custom integration
- grouper.psp - deprecated provisioning plug-in for synchronizing group membership with external systems

Recommendation: grouper.psp is essentially deprecated. Advise against deploying this component. For Columbia's initial evaluation, group synchronization will be done with the Google Groups changelog consumer and custom integration with the local "grand unified forwarder" sendmail build scripts.

Recommendation: A typical production deployment would run the grouper loader daemon on a separate host than the grouper.ui and grouper.ws, since often times the grouper loader needs more privileged access to network and database resource than the user facing components. However, since the grouper installer requires access to the grouper.ui and grouper.ws you should consider deploying and maintaining them on the "loader box" and then moving those files to the grouper.ui and grouper.ws nodes as needed.

1.1 Download and Run the Grouper Installer

The latest Grouper Installer is available under Release Components on the Grouper Downloads wiki page. Download and run the installer:

- Make sure you have Java 1.6 or 1.7 SDK. Must be the SDK and not just the JRE.
- Make a folder where you want grouper installed
- Run: $ java -jar grouperInstaller.jar

Alternatively, you can pre-stage the install by downloading the required files first and then running the grouperInstaller.jar in the same directory:

- wget -r http://software.internet2.edu/grouper/release/2.2.2/
  - move all the files to the target host
- cd software.internet2.edu/grouper/release/2.2.2
- mv patches/grouper_v2_2_2* .
- java - jar grouperInstaller.jar

Most the of defaults can taken, except:

- Do you want to use the default and included hsqldb database (t|f)? [t]: f
  - database connection string, user and password for Oracle should be used
- Do you want to add quickstart subjects to DB (t|f)? f
- Do you want to add quickstart data to registry (t|f)? f
- Do you want to install the provisioning service provider (t|f)? [t]: f

Tip: The installer will initially create the grouper-ui and grouper-ws war files. However, in order to patch or upgrade the grouper-ui the installer must have access to the exploded war files.

Tip: GrouperSystem is the “root” user for grouper and has full administrative privileges. The password should be complex and treated as a privileged account. Additional subjects can be granted admin privilege by assigning them wheel group membership. Add additional members to the wheel group within grouper before configuring external authentication (such as CAS). This prevents you from being locked out of the Grouper UI.

Recommendation: Grouper passwords should be externalized and encrypted, especially in production deployments. This includes LDAP and database passwords.

Tip: Grouper needs full schema admin privileges at least for the initial install and upgrades. gsh can output DDL for these purposes if necessary to comply with organizational deployment policies. In any case, access to the underlying database by users is controlled via grouper application layer when interacting with grouper.ui and grouper.ws.

Tip: grouper.installer.properties can be created to run non-interactive Grouper installs.

Tip: Consider a grouper docker deployment to make it easy to create Grouper environments.

Tip: To drop all tables and recreate a fresh grouper database run: “gsh -registry -drop -runscript -noprompt”

Tip: To check if grouper loader daemon is running: “ps -ef | grep gsh | grep loader”

1.2 Configure grouper.properties and grouper.base.properties

Grouper uses a configuration overlay method to make it easier to deploy and upgrade grouper environments. This approach preserves a grouper project distribution properties files and allows an override properties files for local configuration. For the initial install update $GROUPER_HOME/grouper.apiBinary-X.X.X/conf/grouper.properties with the following:

```
# differentiate test vs dev vs prod for logging and emailing
grouper.env.name = dev

# the URL which will be used in emails to users.
#e.g. https://server.school.edu/grouper/
grouper.ui.url = {whatever the URL of dev is}

# auto-created wheel group for convenience
configuration.autocreate.system.groups = true

# A wheel group allows you to enable non-GrouperSystem subjects to act
# like a root user when interacting with the registry.
groups.wheel.use = true
```

2.0 Install gsh Wrapper

The default grouper shell command line client, gsh, lacks modern shell features and integration with scripting languages such as groovy or python. Shell Wrappers for Grouper provides these features and improves usability and opens up further operational efficiencies.

2.1 Install Groovy

- Groovy install instructions: http://groovy-lang.org/download.html#distro
- verify installation: $ groovysh -version

2.2 Install Shell Wrappers for Grouper

- git clone https://github.com/wgthom/groovysh4grouper
- follow install instructions: https://github.com/wgthom/groovysh4grouper/blob/master/groovy/README.groovy.md
- run $GROUPER_HOME/grouper.apiBinary-X.X.X/bin/gsh.groovy
- verify installation: $ groovy:000> findSubject("GrouperSystem")
    - ===> Subject id: GrouperSystem, sourceId: g:isa, name: GrouperSysAdmin
- to exit groovysh: $ groovy:000> :exit

Tip: gsh.groovy provides tab completion of all grouper API objects and a number of convenience function that are defined in groovysh.profile.

3.0 Configure Grouper Subject Sources

Grouper must be able to search and resolve entities (subjects, groups, etc) via the Subject API. Grouper can then be used to manage group membership assignments for these entities. This typically means connecting Grouper to either an enterprise directory such as OpenLDAP or to a relational database used for identity management. The key here is that any subject that you want to assign group membership to must be resolvable via the Subject API. Columbia will connect the Subject API to the local IDM database.

3.1 Configuration of Source Adapters

Configuration of Source Adapters provides considerations and install instructions.

Recommendation: Since Columbia has administrative control over the IDM database, recommend to follow the Penn subject source JDBC2 example.

3.2 Verify Source Adapter Configuration

- run $GROUPER_HOME/grouper.apiBinary-X.X.X/bin/gsh.groovy
- groovy:000> subj = SubjectFinder.findByIdOrIdentifier("{UNI}", false)
    - replace {UNI} with a resovable Id or Identifier, SubjectFinder.findByIdOrIdentifier(“wgt123”, false)
- groovy:000> subj.getAttributes()
    - should return all attributes configured in sources.xml

Tip: Make sure search terms are indexed.

Tip: $GROUPER_HOME/grouper.apiBinary-2.2.2/logs/grouper_error.log will have detailed information in the event the configuration is not working.

Tip: Grouper installer configures Tomcat to run the expanded war files for grouper.ui at $GROUPER_HOME/grouper.ui-X.X.X/dist/ and the grouper.ws at $GROUPER_HOME/grouper.ws-X.X.X/grouper-ws/build/dist/grouper-ws. The classpaths are $GROUPER_HOME/grouper.ui-X.X.X/dist/grouper/WEB-INF/classes and $GROUPER_HOME/grouper.ws-2.2.2/grouper-ws/build/dist/grouper-ws/WEB-INF/classes. Here you will find the respective copies of sources.xml and other configuration files.

Tip: grouper logs for the grouper.ui and grouper.ws defaults to $TOMCAT_HOME/logs/grouperUi and $TOMCAT_HOME/logs/grouperWs

Once you verify the sources.xml configuration with gsh.groovy, the configuration needs to be copied to the classpath of the grouper.ui and the grouper.ws webapps. After a restart of Tomcat you should be able to login to the grouper.ui with the GrouperSystem account and search for subjects in the IDM database.

4.0 CAS Authentication for grouper.ui

Implementing CAS Authentication for Grouper is known to work well and is currently supported by the project. However, the yale-cas-client itself is no longer an active project and has been superseded by CAS Client for Java 3.1. Supporting the newer CAS client while out of scope for this engagement could be achieved with some work.

5.0 Authentication and Authorization for grouper.ws

grouper.ws by default uses container based authentication and expects users to have a role assignment of “grouper_user”. Users of the web service must be resolvable subjects by the grouper Subject API. Other options for authentication are possible, and are listed in Grouper WS Authentication.

Generally the steps to configure authentication and authorization are:

1. Create a service account (non-person DN and password) in LDAP
2. Configure container managed authentication in Tomcat that will authenticate the user and also return the “grouper_user” role

Tomcat 8 JNDIRealm Configuration for a user with an attribute memberOf=grouper_user:

```
<Realm className="org.apache.catalina.realm.JNDIRealm"
        connectionURL="ldaps://localhost"
        userBase="ou=people,dc=example,dc=edu"
        userSearch="(uid={0})"
        userSubtree="true"
        connectionName="cn=admin,dc=example,dc=edu"
        connectionPassword="password"
```

```
userRoleName="memberOf"  />
```

Tip: For development purposes the tomcat-user.xml can also used to set up a local user/password and roles. However, the subject must still be resolvable by Grouper via the Subject API.

6.0 Group and Folder Design

Grouper mostly leaves the approach to group and folder design up to the deployer. However the community has provided a number of examples and suggestions for folder and group design.

Some common themes among these approaches include:

- a root folder for the institution like “columbia” or “cu”
- “etc” folders for configuration and admin groups
- a folder for reference groups (i.e. institutional affiliations)
- a folder for applications
- a folder for organizational hierarchies
- a folder for class rosters
- use of folders hierarchy to provide delegated management

Recommendation: Keep things as simple as possible and add structure and groups as use cases demand. A reasonable starting structure for Columbia based on initial use cases would be:

- cu - top level folder to organize Columbia namespace folder and groups
- cu:app - enterprise applications folders and authorization groups
- cu:org - organization hierarchy and groups
- cu:ref - reference groups
- etc - top level folder for Grouper config groups, sysadmin group, loader jobs
- test - testing folder for the IAM team

7.0 Grouper Loader

Grouper loader base behavior including available group membership sources is configured in grouper-loader.base.properties and grouper-loader.properties. SQL and LDAP sources for group loader jobs can be configured here, such as a connection to the IDM database for reference groups.

In grouper-loader.properties add:

```
####################################
## DB connections
####################################
# specify the db connection with user, pass, url, and driver class
# the string after "db." is the name of the connection, and it should not have
# spaces or other special chars in it. pass should be encrypted and externalized
db.idm.user =
db.idm.pass =
db.idm.url =

# Loader jobs can omit the subject source id if a default is specified
```

default.subject.source.id = idm

Tip: Additional Grouper daemon tasks and options, such as change log consumers and daily reports are also configured in grouper-loader.properties. See grouper-loader.base.properties for a full set.

Tip: Grouper provides a status URL for Grouper Diagnostics at http://{hostname}/grouper/status?diagnosticType=[trivial|db|all]

Tip: Grouper Loader logs the results of loader jobs in the Grouper database in the grouper_loader_log table. Grouper loader also logs to grouper_error.log. grouper_error.log logging levels is controlled by log4j.properties.

7.1 SIMPLE_SQL Loader Jobs

SIMPLE_SQL loader jobs provide an easy way to pull in a single reference group from an external database like IDM. SIMPLE_SQL Loader jobs are configured directly on groups which are created "in-line" within the grouper folder/group hierarchy. (e.g. cu:ref:affiliation:student)

Once the loader job sources are available, configure a new loader job with the Admin UI. Using the web UI the basic steps are:

1. If needed, create a folder to hold the reference group (e.g. "cu:ref:affiliation")
2. Create a new reference group (e.g. "cu:ref:affiliation:student")
3. Edit the group to mark it as a grouperLoader job
4. Edit Attributes on the group to configure the loader job
    a. grouperLoaderDbName = idm
    b. grouperLoaderQuartzCron = 0 5 7 * * ? (7:05am every day)
    c. grouperLoaderQuery = select subject_id from idm_student_v
    d. grouperLoaderScheduleType = CRON
    e. grouperLoaderType = SQL_SIMPLE

Test the new loader job using the gsh.groovy to run the job:

1. start the gsh.groovy
2. job = GroupFinder.findByName(grouperSession, "cu:ref:affiliation:student");
3. loaderRunOneJob(job);

This should return something like: "loader ran successfully, inserted 288 memberships, deleted 0 memberships, total membership count: 288"

Tip: The Grouper Loader daemon requires a restart to pick up the new job.

Recommendation: Do not use include/exclude for reference groups. If possible rely on loader source systems for "truth" about membership in core institutional constituencies. Do consider include/exclude groups for distributed access management, and exceptions to authorization policy and class rosters. The Grouper Loader classlist example from Penn provides an example of the latter.

Recommendation: Create SQL views for membership assignments so that grouper loader job configuration is simple and stable. This way you can change the SQL query in the view without have to update grouper configuration.

7.2 SQL_GROUP_LIST Loader Jobs

SQL_GROUP_LIST loader jobs provide the capability to create multiple groups and folders from a single query. The result set of the SQL query must contains a group_name in addition to subject_id. Unlike SQL_SIMPLE, SQL_GROUP_LIST loader jobs are configured in separate "loader groups" rather than the target groups themselves.

Recommendation: Keep loader jobs that create multiple folder/groups in "etc:loader".

SQL_GROUP_LIST are often used to create organizational hierarchies and class rosters. The Grouper Loader classlist example from Penn provides a robust example of combining SQL_GROUP_LIST loader jobs and include/exclude groups to provide up to date class membership for students, instructors, assistants (to instructors), guests, and all members. Guests are ad-hoc groups, and provide an entry point for authorized subjects that is not dependent on institutional source systems (i.e. SIS system).

Tip: The grouperLoaderGroupsLike configuration options tells grouper to remove managed groups that no longer have membership assignments returned by the SQL query.

Tip: The group audit report show how and when membership changes over time.

8.0 Columbia Grouper Use Cases

8.1 Simple Email Distribution List Management

Columbia maintains simple email distribution lists where membership is managed by the respective list owners. The proposed grouper solution has two parts; 1) model each distribution list as a special group, and 2) update the grand unified forwarder (GUF) to read list membership directly from grouper via either the grouper client or a custom change log consumer process.

8.1.1 Email Distribution Lists Modeled as Grouper Groups

The proposed solution models the distributions lists a special Grouper groups. The group name maps to the email distribution list address, and membership in the list signifies that the subjects should receive any email sent to the distribution list address. Management of the membership of each list is delegated to the list (i.e. group) owners and is done via the standard grouper.ui. List owners are designated by direct permission assignment on each email distribution group.

A folder/group structure would look like this:
cu:apps:guf:etc:guf_admin - group of members who have admin privileges over cu:apps:guf
cu:apps:guf:guf_lists: - folder to contain distribution list groups
cu:apps:guf:guf_lists:mail_list1 - example group whose members receive all email sent to mail_list1@columbia.edu.

Privileges to manage membership of each list are directly assigned to users by guf_admins using the grouper.ui. Basic steps to assign group privileges are:

1. select the target group (e.g. cu:app:guf:guf_lists:mail_list1)
2. click on Privileges tab
3. click on + Add members button

4. added Member name or ID, and assign READ and UPDATE privilege
5. click Add.

The Privileges tab should now show this member with the Update privilege. This group will now show up under “Groups I manage” for the list owner in the grouper.ui and the user will be able to add and remove members from the group, and see current membership. When adding members to the group, Default Privileges assigns basic membership.

8.1.2 Integration with GUF

GUF is a Java program that currently reads information from the IDM database and creates configuration files that manage all aspects of email delivery for columbia.edu addresses, including distributions lists. The Grouper solution proposes that GUF will use the Grouper Client to map group names in cu:app:guf:mail_list: to email distribution list configuration, and configure those email distribution addresses to forward email to all members of those groups.

The Grouper Client can be used as a Java API by adding the grouperClient.jar and grouper.client.properties to the GUF classpath. The application can then use the classes in the edu.internet2.middleware.grouperClient.api package to query for folders, group names, and membership.

8.1.3 Email Distribution List Migration

Migrating from the existing system to the grouper based solution could be done with the following steps:

- Create and run a temporary grouper loader job to populate initial groups for each mailist list
- Remove grouper loader job
- Assign UPDATE and READ permissions to group owners

8.1.4 Email Distribution List Maintenance

The Unresolvable Subject Deletion Utility (USDU) can be used to remove any previously added group members that are not longer resolvable by the Subject API.

8.2 Institutional Reference Groups

Institutional reference groups provide membership assignment for a variety of campus affiliations, constituents, and organizational roles. Reference groups are often created with Grouper Loader jobs, but can also be maintained by hand when no authoritative source exist for the membership set. The Columbia IDM database will provide a single version of truth regarding the initial set of reference groups.

Examples of institutional reference groups include:

- faculty (full-time, part-time, adjunct, course lecturer,...)
- student (full-time, part-time, undergraduate, graduate, transfer, class year,...)
- staff (full-time, part-time,...)

8.2.1 Affiliation Reference Groups

Based on initial analysis of IDM “exploder” the follow groups could be pulled as grouper reference groups:

- consultant (3621 people)
- researcher (4558 people)
- student (45304 people)

These could be create as SIMPLE_SQL loader jobs resulting in the following reference groups:

- **cu:ref:affiliation:consultant**
- **cu:ref:affiliation:researcher**
- **cu:ref:affiliation:student**

Recommendation: Let the use cases drive what reference groups are created and when.

8.2.2 Authorization Groups

Reference groups are particularly suited to enforce authorization policy for services. Since they are based on authoritative sources and are kept up to date, authorization configuration will accurately reflect policy automatically. As members come into or fall out of reference groups, authorization membership is updated accordingly.

Authorization groups are typically managed using include/exclude and composite groups. An application called *service* would have the following folder/group structure:

cu:apps:*service_name* - folder for service_name groups and policy

cu:apps:*service_name*:etc - folder for configuration and admins, and possibly service specific ad hoc groups

cu:apps:*service_name*:etc:*service_name*_admin - group of users who have admin privileges for all the objects in service_name

cu:apps:*service_name*:*service_name*_role - composite group that provides a list of authorized subjects for the service in a particular role.

cu:apps:*service_name*:*service_name*_role_allow - authorized reference groups are added as sub groups and provide an initial statement of access policy

cu:apps:*service_name*:*service_name*_role_deny - the deny group is used to override the allow group and can be done for exceptions or other policy such as an IAM override

Tip: Grouper Rules can be used to inherit privileges on groups created under a specific folder. To have admin privileges inherent to groups and folders created under a specific folder, run these command in gsh:

- grouperSession = GrouperSession.startRootSession();
- folder = StemFinder.findByName(grouperSession, "{path:to:folder}")
- admins = GroupFinder.findByName(grouperSession, "{path:to:admin_group}”);
- RuleApi.inheritGroupPrivileges(SubjectFinder.findRootSubject(), folder, Stem.Scope.SUB, admins.toSubject(), Privilege.getInstances("admin"));
- RuleApi.inheritFolderPrivileges(SubjectFinder.findRootSubject(), folder, Stem.Scope.SUB, admins.toSubject(), Privilege.getInstances("stem, create"));

- RuleApi.runRulesForOwner(folder);

Recommendation: A Grouper Hook could be implemented so that for any folder created under cu:apps would automatically create an etc:*service_name*_admin group and apply the inherit privileges rule.

Tip: To inspect Grouper Rules, run the follow command in gsh:
- Print out the rules for a folder or group: RulesApi.rulesToString({group|folder})
- Print out all rules: RuleApi.rulesToSring()

8.3 Google Groups for Authorization

The Google Apps Grouper Provisioner is a Grouper change log consumer and FullSync agent that provisions and de-provisions Grouper groups and subjects to a Google Apps for Education/Business domain. This project is relatively new and a number of schools are working on a production deployment. The code has been incorporated in the latest grouper source repository, but must be built in a separate step. Also the project documentation is in the Unicon Google App Grouper Provisioner Wiki github repository.

The Google Apps Provisioner uses a Grouper Attribute assignment to determine which groups to sync. The first time the provisioner starts up via the grouper loader it will create a new folder "etc:attribute:googleProvisioner" and a Grouper Attribute for each Google changelog consumer that has been configured. The attribute will be named syncToGoogle{consumerName}, where {consumerName} matches what was configured in grouper-loader.properties.

Tip: Multiple changelog consumers can be configued in grouper-loader.properties and are referenced by name. (e.g. a changelog consumer named "orgGroups" would have a series of properties configured as changeLog.consumer.orgGroups.*)

8.3.1 Building the Google Apps Grouper Provisioner

Unfortunately at present the google provisioner is not included in the binary distribution and must be compiled from source.

- Download source from github
  - wget https://github.com/Internet2/grouper/archive/GROUPER_2_2_2.zip
  - unzip GROUPER_2_2_2.zip
  - cd grouper-GROUPER_2_2_2/grouper-misc/googleapps-grouper-provisioner/
- Build using Maven
  - mvn package dependency:copy-dependencies -DskipTests -DincludeScope=runtime
  - results in target/google-apps-provisioner-1.1.0-SNAPSHOT.jar and depended jars in target/dependency/
- copied all of those jars to grouper.apiBinary/lib/custom/

8.3.2 Google Apps Grouper Provisioner Deployment

The main steps to deploy are:
1. Set up Google API Connections (might need new specific guidance)

2. Configure the changelog consumer in grouper-loader.properties (Step 3 of the Installation instructions)
3. Configure additional Provisioner Properties so that the behavior suits the use case.
4. Start the grouper loader (this will add the grouper attributes in etc:attribute:googleProvisioner that are needed to mark any object for provisioning)
5. Mark Grouper folder or groups for provisioning

Once the attribute is added to a Grouper folder or group, the next time the change log consumer gets executed or the FullSync is run the groups will get propagated to Google.

Tip: If a folder is marked for Google provisioning, any group under that folder or sub folder will also be provisioned to Google.

8.3.3 Google Provisioner Properties for Columbia Use Case

The initial Columbia use case is to synchronize organizational groups for the purposes of access control to content within Google Apps. The follow example configuration assumes a changelog consumer named “orgGroups” which be used to sync institutional organizations like CUIT from Grouper to Google.

The Google provisioner has a lot of flexibility in terms of properties that control its behavior. The full list of properties and options is available at Provisioner-Properties section of the project wiki. The following recommendations for properties settings assumes organizational groups in Grouper will be marked specifically for provisioning to Google, and that Grouper will be authoritative for membership.

Note: in the grouper-loader.properties each property is be prefixed with “changeLog.consumer.orgGroups.”

groupIdentifierExpression controls how Grouper group names will be translated into Google group names (e.g. cu:org:admin:cuit -> org-groups-admin-cuit@columbia.edu).

- groupIdentifierExpression=org-groups-${groupPath.replace(“cu:org:”,””)}

googleGroupFilter specifies which groups the Google Provisioner is authoritative for based on a regex matching of group name. Any Google group name that matches the regex will be controlled by the Google Provisioner and must exist in Grouper otherwise it will be deleted when the FullSync agent runs. The regex should complement the groupIdentifierExpressions.

- googleGroupFilter=org-groups-.*

ignoreExtraGoogleMembers makes Grouper authoritative for the membership of the groups it has provisioned. Meaning that if the membership of the group changes in Google the next time the fullSync is run it will remove or add members so that it is in sync with Grouper membership.

- ignoreExtraGoogleMembers=false

handleDeletedGroup tells the provisior what to do when previously provisioned groups which match the googleGroupFilter are deleted in Grouper. For this use case we’d want those groups to be deleted.

- handleDeletedGroup=delete

whoCanManage determines who has admin privilieges for provisioned group in Google. For this use case we want this set to none as the Grouper is authoritative for membership and settings.

- whoCanManage=none

retryOnError is a queue to the provisions to retry a provisioning operation when it encounters errors. Setting this to true may cause other operations to block. Recommend setting this to false and let FullSync fix any discrepancies.

- retryOnError=false

8.3.4 Caching Controls

Calls to Google APIs are limited by service account. The provisioner reduces the number of group and user look-up calls by caching the group and user objects. This behavior can be tuned in the caching control properties.

```
# This is required for googleFullSync to work
changeLog.consumer.orgGroups.prefillGoogleCachesForFullSync = true
```

8.3.5 Google Group Settings

Google Group Settings controls what the initial Google Group settings are for a group that is provisioned from Grouper. They are used when the group is initially created, but are not synchronized should they diverge in Google or Grouper. It is possible to create authorization groups without the email list functionality, but that would require some development work on the provisioner and some testing.

8.3.6 FullSync Agent

The FullSync agent will synchronize all groups marked for Google provisioning to Google Groups. The FullSync agent is run by the command line and can be put on a cron job. The FullSync command is: $GROUPER_HOME/grouper.apiBinary/bin/googleAppsFullSync.

- Copy the files at https://github.com/Unicon/googleapps-grouper-provisioner/tree/master/bin to $GROUPER_HOME/grouper.apiBinary/bin

8.3.7 Account Provisioning with GADS

Similar to the way that authorization groups can be managed in Grouper by making use of reference groups and include/exclude logic, provisioning groups can also be setup. Membership in the provision group signifies authorization to use the service and that an account should be created. Account creation can be done via a custom change log consumer or an LDAP based provisioning tool. This examples presumes the use of Google Apps Directory Sync (GADS) to provision accounts in Google by way of an LDAP group managed by Grouper.

cu:apps:google:etc:google_admins - admins over google policy in grouper
cu:apps:google:etc:adhoc - for adhoc groups that contribute to google account policy
cu:apps:google:google_account - this goes to ldap and is used by GADS to provision accounts
cu:apps:google:google_account_allow - add reference groups here to implement default access policy
cu:apps:google:google_account_deny - add groups or individual assignments to disable google accounts

9.0 Operational Considerations

9.1 Java Version

Grouper was designed and tested on Java 6. Java 7 is generally supported by the project team and unit tests for the project all run on Java 7. Grouper is also known to run on Java 8 and Tomcat 8 without the PSP. The Java version checks which display error messages on start up can be ignored or configured not to show.

Recommendation: Deploy on Java 8 and Tomcat 8.

9.2 Running Grouper Loader as a Linux Service

If deploying on Linux the Grouper Loader and be setup to run as a linux service so that it auto-starts when the machine boots up.

9.3 Email Notifications

Grouper can be configured to send various alerts and messages typically to the Grouper administrator or an email list of administrators. You find smtp settings in grouper.base.properties. This should copied over to grouper.properties and set there.

Tip: To send a test email from gsh run:
gsh 0% new GrouperEmail().setTo("something@somewhere.edu").setBody("email body").setSubject("email subject").send();

9.4 Other Community Use Cases

Other community use cases and examples are available on the project wiki as Use Cases By Category.