Grouper provisioning framework

This page is your starting point for the Grouper provisioning framework in Grouper v4+.
See the Versioning page for clarification on Grouper version numbering.

For background, see these blogs

- New Provisioning Framework blog - January 2021
- News from the Grouper Project blog - November 2021
- Major Improvements to the Grouper Provisioning Framework - February 2023

Provisioning Glossary

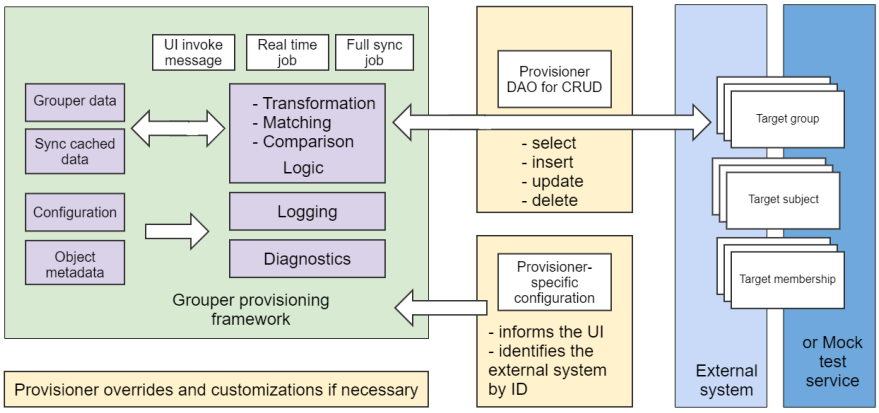
- Overview of features
- High level design
- Data Model diagram
- Provisioning tasks (for more detail see this page)
- Targets
- Export a config
- Provisioning in LDAP/AD
- Provisioning to SQL
- Provisioning to SCIM
- Provisioning to Box
- Provisioning to Azure
- See Also

Overview of features

Key features of the provisioning framework:

- All parts of provisioning configuration are performed in the Grouper UI with helpful documentation, wizard-like interfaces, descriptive validations, and diagnostic tests.
- All provisioners have consistent configuration concepts and terms so adding the next provisioner will be easy.
- Configuring new provisioners or editing existing provisioners do not require Grouper to be redeployed/restarted.
- Provisioning configuration starts with an “external system”, which is the connection information to connect to the target to provisioning to.
- External systems can be re-used among provisioners, or for other parts of Grouper (e.g. loader, “custom UI”). The provisioner itself is configured next.
- Scaffolding (start with) can help you get started with provisioning configuration.
- There are the daemon jobs for the full or incremental sync which are scheduled.
- All provisioners have a standard object model. Translating data from Grouper to the target format is consistent across all provisioners.
- You configure which data from Grouper gets sent to the target and how it is formatted
- There are provisioning-specific screens to identify which objects (groups, users, memberships, attributes) are sent to the target.
- Provisioner specific metadata can be assigned to Grouper objects to inform provisioning actions.
- When the provisioner is up and running, data propagation is verified and errors info is readily available

High level design

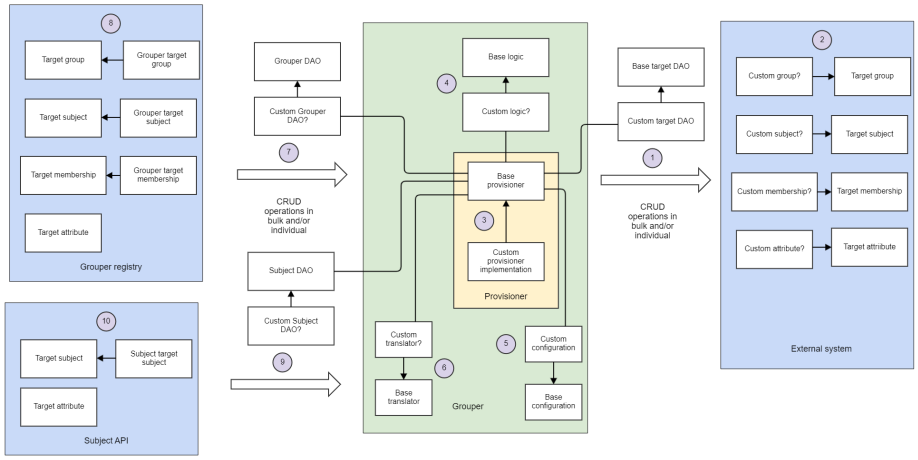


Notes on High Level Design diagram

1. The provisioner has CRUD (Create Read Update Delete) operations implemented for the endpoint.
    a. Note: there is a concept of an "external system" in Grouper. Perhaps some or all of the CRUD is implemented in the "External System". Perhaps the external system builds on top of an External System
    b. Its possible to integrate ConnID here or wrap around it in the future
2. There is an object model for TargetGroup, TargetSubject, TargetMembership, TargetAttribute.
    a. The Provisioner instance can use the built in Target objects or can subclass those as needed
    b. Note: "Target attribute" is just the object model for being able to have arbitrary fields in target groups, subjects, and memberships. Its not really related to the attribute framework though it could probably be translated from it
3. The Custom Provisioner Implementation extends the Base implementation, and has methods for whatever is necessary. The Base provisioner has built in logic:
    a. Deal with Grouper objects
    b. Compute what needs to be inserted/updated/deleted
    c. Do the logging and error handling
    d. Interface with "Sync" objects
    e. Triggered from real time change log, full sync daemon, and UI to fix one-offs
    f. Interface with UI so provisioning state is known by users or people troubleshooting
4. The provisioner uses standard configuration
    a. The UI will configure the provisioner via wizard. Has standard validation and documentation. This is a pass through to grouper-loader.properties configs in the database. You can configure directly in config files instead if you like (not recommended)
        i. Option for grouperIsAuthoritative to delete objects not in Grouper
    b. There is standard configuration for external systems with wizards
    c. Configuration for daemons (real time and full sync)
5. The real time change log consumer has a workflow built for provisioning
    a. Compute which events really need to be processed
    b. Convert between individual events and group-sync or full-sync
    c. Reduces the amount of logic needed in provisioners
    d. Check message queue for UI events to fix a membership or group etc
    e. Check message queue for "async" computations. i.e. a full sync that runs in the background and doesn't block other processes and sends messages to fix certain suggested objects
    f. Note, this runs as Java in the Grouper Daemon, not as an async provisioner. This is so we can tightly couple and keep track of things and not require extra processes
6. The real time change log for provisioners will consult the "sync" tables to see if a task is already done, see if it is provisionable, etc
7. The full sync daemon runs as asynchronous or synchronous. Generally it is probably ok to run daily off hours as synchronous, but maybe an async run will be sufficient.
    a. Synchronous runs will block other processing, and uses the "sync tables" to block. This allows daemons to run on multiple servers and not need to run multiple times
    b. Asynchronous runs will not block and will send messages for individual objects to check, or will see there are so many updates it requires a blocking full sync
    c. Full syncs will bulk fetch from Grouper and the Target system (if the API supports it) and process everything in a short amount of time but using a lot of resources (e.g. memory). Note its possible that there is batch size to reduce the resources needed and not need as much resources.
    d. Full syncs will also try to batch inserts/updates/deletes (if API supports it)
    e. Note, this runs as Java in the Grouper Daemon, not as an async provisioner. This is so we can tightly couple and keep track of things and not require extra processes
8. Full syncs will bulk fetch from sync tables to consult and true up what Grouper thinks the state of the target system is
    a. Errors will trigger a recalc where cached data is ignored and updated as the real underlying data is fetched from grouper, subject source, and target system. There are options to take into consideration e.g. grouperIsAuthoritative
9. There is a new format for provisioning attributes on groups and folders. This format precalculates inheritance and has more options than the legacy PSPNG attributes
    a. Grouper will be able to migrate from one format to the new
    b. All provisioners will use standard provisioning attributes
    c. It is possible to use these attributes to override where an object is provisioned in the target (e.g. point a group or folder at a different OU in ldap
    d. This data (if provisionable and where) is copied to the "sync tables" so that there are no foreign keys and PIT is not needed when a group /member/membership is deleted

10. Grouper tables have Grouper's state for groups and memberships. Again as these are deleted or moved, the sync tables can facilitate efficient and accurate provisioning
11. Sync tables keep a copy of state that does not have a foreign key to groups/members/memberships. There is a table for
    a. Provisioner
    b. Job (e.g. full or real time)
    c. Sync group: cache state from Grouper and Target. Is provisionable. Some stats. Timestamps. Error messages
    d. Sync member: cache state from Grouper and Target. Is provisoinable. Some stats. Timestamps. Error messages
    e. Sync membership: cache state from Grouper and Target. Is provisionable. Some stats. Timestamps. Error messages
    f. Log: keep log messages temporarily about a sync of a job/group/member/membership
12. Provisioning targets that need subject data might not use the subject ID or identifier in the member table, or might use other attributes
    a. The new USDU will resolve all subjects from all subject sources nightly, and while that data is being processed, the sync tables can be updated for what that provisioner needs
    b. Might need a different identifier
    c. Might need other attributes stored in JSON
13. The provisioner uses the sync table data to make provisioning more efficient, communicate and keep track of errors, log progress, survive Grouper object deletion
14. There is translation between the Grouper id's and names, the Target id's and names, and the Subject API id's and names
    a. Hopefully most of that can be handled with EL scripts
    b. Whatever is needed for the translation needs to be cached in sync tables (that is configurable)

Data Model diagram



Notes on Data Model diagram

1. The DAO is how Java talks to the target. This would implement CRUD operations and hopefully do so in bulk
2. "Target beans" can be subclassed if needed
3. The provisioner specifies the class names of all the mandatory and optional subclasses
4. The logic is what takes the beans and decides what has changed and runs the actions to sync or notify or log
5. The configuration class will translate from the properties file to java, and validate the configuration
6. The translator will translate from Grouper objects to Target objects (and vise versa), based on configuration and potentially cache
7. The Grouper DAO will get objects from Grouper efficiently based on which field is being retrieved
8. The target classes can be subclassed for Grouper
9. The Subject DAO will get objects from the subject source
10. The target objects for subjects can be subclassed

Provisioning tasks (for more detail see this page)

Task	Description	Status
Ldap Dao	New interface implementation for efficient LDAP operations	In progress
External systems in UI	Endpoints, usernames, passwords, with wizards in UI	In progress
New USDU daemon	Resolve all subjects, update unresolvable status, and cache provisioning sync attributes	Almost started
Provisioning configuration in UI	Mark folders and groups as provisionable and provide specific configuration	Done
Provisioning change log consumer	Look at "sync" objects and provide provisioner with "processed" view of work to do	Done
Shadow objects in database	"sync" tables keep track of target system status and cache attribute	Done

Targets

- LDAP
- SQL
- SCIM
- google
- drop box (SCIM?) (todo)
- box
- Duo
- azure / o365
- zoom (todo)
- slack (SCIM?) (todo)
- servicenow (jeffW) (todo)
- canvas (Unicon) (todo)
- adobe (jeffW, liam) (todo)
- teamDynamix (liam) (todo)
- remedy
- papercut (liam) (todo)
- tableau (liam) (todo)
- salesforce (liam) (todo)

Export a config

If you are reporting an issue with the provisioning framework, please follow these steps:

1. Ensure you are on a recent v2.6.* release of Grouper
2. Report your version number
3. Send sanitized configs for the provisioner either to the list or to a Grouper developer in a private slack
    a. Export your grouper-loader.properties from the UI
    b. Search for your provisioner config id
    c. Send that block of configs
    d. No need to send the daemon configs
    e. If there is a problem making calls to the target you might want to send the sanitized (no passwords or keys) configs of the external system for the provisioner

Provisioning in LDAP/AD

See the info here.

Provisioning to SQL

See the info here

Provisioning to SCIM

See this page

Provisioning to Box

See this page

Provisioning to Azure

See this page

Provisioning to other systems

You could implement your own provisioner or use the provisioning framework to send messages

See Also

- Grouper provisioners
- Grouper provisioning examples
- Grouper provisioning features
- Grouper provisioning in UI
- Grouper provisioning technical
- Grouper provisioning glossary