Grouper Entra ID Provisioner (Legacy)

This page applies to Grouper 2.5. For Grouper v4+ see this page

The Grouper Azure provisioner is a changelog consumer that synchronizes Grouper groups and users to Microsoft Azure Active Directory/Office 365. This currently only supports security or Office 365 (unified) groups. Support for mail-enabled groups is unavailable due to lack of support in the Microsoft API.

The provisioner started as a Unicon project, before being incorporated into the Grouper project. The library included with Grouper container 2.5.23 includes many enhancements from the original:

- can now create Office 365 (unified) groups as well as security groups
- supports more than one provisioner
- custom names, descriptions using jexl expressions
- custom mail nicknames, using jexl, for unified groups
- custom UPN, can be derived from attribute, jexl, or customizable domain
- Public, Private, and HiddenMembership options for unified groups
- ability to set up multiple provisioners with different option groups

Initialize

The provisioning attribute to add to the provisionable object is, by default, `etc:attribute:office365:o365Sync`. If set on a folder, all groups under this folder (recursively) will be automatically provisioned. Alternatively, the attribute can be set on an individual group. It is not recommended to assign directly to a group, as there is currently a race condition that may occur in the current version of this code (i.e., if the changelog consumer runs after group creation but before attribute assignment, provisioning will not occur). It is also required to create attribute `etc:attribute:office365:o365Id`, which will store the Azure group Id on the Grouper group to be kept in sync.

Note that if the sync attribute is removed from a group or folder, it will not trigger the deletion of the remote group. It will only stop further sync of the group, until the attribute is added back. However, if a group is deleted in Grouper, deletion of the remote group will occur.

1. Set up stem for provisioning and ID attribute

```
GrouperSession grouperSession = GrouperSession.startRootSession();

/* the sync marker attribute */
AttributeDef provisioningMarkerAttributeDef = new AttributeDefSave(grouperSession).
assignCreateParentStemsIfNotExist(true).assignName("etc:attribute:office365:o365SyncDef").assignToStem(true).
assignToGroup(true).save();
AttributeDefName provisioningMarkerAttributeName = new AttributeDefNameSave(grouperSession,
provisioningMarkerAttributeDef).assignName("etc:attribute:office365:o365Sync").save();

/* the Azure groupId to store with provisioned groups */
AttributeDef o365Id = new AttributeDefSave(grouperSession).assignCreateParentStemsIfNotExist(true).assignName
("etc:attribute:office365:o365IdDef").assignToGroup(true).assignValueType(AttributeDefValueType.string).save();
AttributeDefName o365IdName = new AttributeDefNameSave(grouperSession, o365Id).assignName("etc:attribute:
office365:o365Id").save();

/* add the marker attribute at the folder level */
rootStem = addStem("", "test", "test");
rootStem.getAttributeDelegate().assignAttribute(provisioningMarkerAttributeName);
```

2. Configure loader job in grouper-loader.properties. Note that you will need to set up an application with access to your domain. See documentation at http://graph.microsoft.io/en-us/docs, and Notes for Developers below.

```
changeLog.consumer.o365.class = edu.internet2.middleware.grouper.changeLog.consumer.Office365ChangeLogConsumer
# fire every 5 seconds
changeLog.consumer.o365.quartzCron = 0,5,10,15,20,25,30,35,40,45,50,55 * * * * ?
changeLog.consumer.o365.syncAttributeName = etc:attribute:office365:o365Sync
changeLog.consumer.o365.retryOnError = true
changeLog.consumer.o365.tenantId = @o365.tenantId@
changeLog.consumer.o365.clientId = @o365.clientId@
changeLog.consumer.o365.clientSecret = @o365.clientSecret@
#changeLog.consumer.o365.domain =
#changeLog.consumer.o365.idAttribute =
#changeLog.consumer.o365.upnAttribute =
#changeLog.consumer.o365.groupJexl =
#changeLog.consumer.o365.mailNicknameJexl =
#changeLog.consumer.o365.descriptionJexl =
#changeLog.consumer.o365.subjectJexl =
#changeLog.consumer.o365.groupType = [Security* | Unified]
#changeLog.consumer.o365.visibility = [Public* | Private | HiddenMembership]
#changeLog.consumer.o365.proxyType = [http | socks]
#changeLog.consumer.o365.proxyHost =
#changeLog.consumer.o365.proxyPort =
```

Custom configuration parameters

tenantId, clientId, clientSecret

Replace `@o365.tenantId@`, `@o365.clientId@` and `@o365.clientSecret@` with appropriate values from the application configuration. Note that the clientSecret is sensitive information.

domain

Property domain defines the domain name to be used with user principals. If not defined, the tenantId property will be used to construct user principals.

idAttribute

Optional property `idAttribute` specifies what attribute is used to build the Azure user principal, and will default to "uid" if not set. The Azure principal will get built as idattribute + "@" + domain or tenantId. Whatever attribute is used must be available as a key in the set returned in `subject.getAttributes()`; i.e., if you want to use the subject's id or identifier, it needs to be defined in the subject attributes too.

upnAttribute

If subjects reliably have the Azure principal name as one of their attributes, setting upnAttribute will use this value as is to identify Azure users, rather than calculating it through other methods.

groupJexl, mailNicknameJexl, and descriptionJexl

Optional *Jexl properties support jexl2 expressions in deriving custom values. Variable `group` is provided to the group jexl expressions, while `subject` and `subjectIdValue` are for subjectJexl. Variables common to both are `consumerName`, `tenantId`, `domain`, and `idAttribute`. Brackets are not needed around the expression.

property	associated Azure property	default	default example	Max length in Azure
groupJexl	displayName	group.name	app:azure:provisioned:groupTypes:o365Sync:group1	256
mailNicknameJexl	mailNickname	group.uuid	ec3d7d285be34e3f921a6e89a1514a94	64
descriptionJexl	description	group.uuid	ec3d7d285be34e3f921a6e89a1514a94	1024
subjectJexl	userPrincipalName	subjectIdValue@domain	wreed@example.onmicrosoft.com	1024?

Examples:

```
group.name.replaceAll("^app:test:grp-2607:provision:", "").replaceAll(":", " / ")
  => groupTypes / o365Sync / group5

(group.name.length() <= 64 + 11) ? "GROUPER-SECURITY-" + group.name.replaceAll("^app:test:grp-2607:provision:",
"").replaceAll(":", "-") : group.name.length().toString() + ":" + group.id
  => GROUPER-SECURITY-groupTypes-o365Sync-group5
  => Note: the length checking is to ensure the mail nickname does not exceed 64 characters

group.parentStem.parentStem.displayExtension + " - " + group.parentStem.displayExtension + " - " + group.
displayExtension + (group.description ? ": " + group.description : "")
  => Group Types - Security Groups - Group 5: Testing of Azure Security Groups

subject.getAttributeValue('azurePrincipal') ?: subjectIdValue + '@' + domain
  => vsteele@custom.domain.edu
  => (if the subject has a pre-defined principal, use it, otherwise default to the constructed one)

subjectIdValue + '@' + ({'WGA': 'wga.school.edu', 'WGR': 'west-gr.edu'}.get(subject.getAttributeValue
('school')) ?: 'domain.edu')
  => snelson@wga.school.edu
  => (map schools to domains, with a fallback value)
```

To aid in developing jexl expressions, a minimal consumer instance can be bootstrapped enough to test jexl expressions using gsh. All the documented variables, except for consumerName (which is only defined while processing changelog entries), are available during evaluation.

```
import edu.internet2.middleware.grouper.changeLog.consumer.Office365ChangeLogConsumer
import edu.internet2.middleware.grouper.changeLog.ChangeLogProcessorMetadata

def changeLogProcessorMetadata = new ChangeLogProcessorMetadata()
changeLogProcessorMetadata.consumerName = "o365" // should match the consumer definition in grouper-loader

def consumer = new Office365ChangeLogConsumer()
consumer.initialize(changeLogProcessorMetadata)

def gs = GrouperSession.startRootSession()
def group = GroupFinder.findByName(gs, "app:test:grp-2607:provision:groupTypes:o365Sync:group7", true)
consumer.evaluateGroupJexlExpression(group, "group.name + ' (' + group.displayExtension + ')'",
defaultIfExpressionNull="nothing defined")

def subject = SubjectFinder.findById("800000023")
consumer.evaluateSubjectJexlExpression(subject, "subject.getAttributeValue('azurePrincipal') ?: subjectIdValue
+ '@' + domain")
```

`The defaultIfExpressionNull parameter is intended mainly for internal code to avoid extra tests for a null expression.`

groupType, visibility

The optional `groupType` property can set the provisioned groups as either a security group (`groupType = Security`) or an Office 365 group (`groupType = Unified`). If not set, the default will be a security group. Mail-enabled groups are not currently available, as they cannot be set through the Microsoft web service API.

For a Unified group provisioner only, the `visibility` property sets the Office 365 visibility. Possible values are Public (default), Private, or HiddenMembership*. See Microsoft's documentation on the option and also the GitHub documentation for more information.

proxyType, proxyHost, proxyPort

If the daemon server requires a proxy to access the internet, a HTTP or SOCKS proxy can be defined using `proxyType`, `proxyHost`, and `proxyPort`. Currently, the SOCKS5 proxy only supports anonymous access.

Fallback methods for subject principal name calculations

Methods for determining the subject principal will try properties in the following order, when they are defined. If a method returns null or blank, the next method will be tried.

1. upnAttribute
2. subjectJexl
3. idAttribute (before appending "@" + domain)

If all three methods return blank values, a runtime exception will be thrown.

Multiple Provisioners

It is possible to set up multiple Azure provisioners, each with different settings. One scenario for this would be to have one folder for security groups and another for Office 365 groups. Or, different folders can have different Jexl expressions, etc. To distinguish them, they need separate consumer attributes created. Then, each loader configuration would reference their respective attribute names, and folders would set the distinguishing syncAttributeName attribute to select a provisioner. Other required properties need to be repeated for each provisioner For example:

```
# Creates security groups
changeLog.consumer.o365.class = edu.internet2.middleware.grouper.changeLog.consumer.Office365ChangeLogConsumer
changeLog.consumer.o365.tenantId = my-tenant.onmicrosoft.com
changeLog.consumer.o365.clientId = ...
changeLog.consumer.o365.clientSecret = ...
...
changeLog.consumer.o365.syncAttributeName = etc:attribute:office365:o365Sync
changeLog.consumer.o365.groupJexl = ...

# Creates Office 365 groups
changeLog.consumer.o365Unified.class = edu.internet2.middleware.grouper.changeLog.consumer.
Office365ChangeLogConsumer
changeLog.consumer.o365Unified.tenantId = my-tenant.onmicrosoft.com
changeLog.consumer.o365Unified.clientId = ...
changeLog.consumer.o365Unified.clientSecret = ...
...
changeLog.consumer.o365Unified.syncAttributeName = etc:attribute:office365:o365SyncUnified
changeLog.consumer.o365Unified.groupType = Unified
changeLog.consumer.o365Unified.visibility = Private
```

Install

The Java library and its dependencies are included with the Grouper container >=2.5.26. But it should be usable with earlier Grouper 2.4 or 2.5 versions by downloading and adding the required libraries. Those libraries are:

- https://repo1.maven.org/maven2/edu/internet2/middleware/grouper/grouper-azure-provisioner/2.5.23/grouper-azure-provisioner-2.5.23.jar
- https://repo1.maven.org/maven2/com/squareup/okio/okio/1.17.2/okio-1.17.2.jar
- https://repo1.maven.org/maven2/com/squareup/retrofit2/converter-moshi/2.7.2/converter-moshi-2.7.2.jar
- https://repo1.maven.org/maven2/com/squareup/retrofit2/retrofit/2.7.2/retrofit-2.7.2.jar
- https://repo1.maven.org/maven2/com/squareup/moshi/moshi/1.8.0/moshi-1.8.0.jar
- https://repo1.maven.org/maven2/com/squareup/okhttp3/logging-interceptor/3.14.7/logging-interceptor-3.14.7.jar
- https://repo1.maven.org/maven2/commons-logging/commons-logging/1.2/commons-logging-1.2.jar
- https://repo1.maven.org/maven2/com/squareup/okhttp3/okhttp/3.14.7/okhttp-3.14.7.jar
- https://repo1.maven.org/maven2/org/apache/commons/commons-jexl/2.1.1/commons-jexl-2.1.1.jar
- https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.2/gson-2.8.2.jar

Grouper containers 2.5.16 through 2.5.25 ship with an earlier version of the grouper-azure-provisioner.jar. If you happen to be running these versions, you should upgrade to 2.5.26 or greater.

Office 365 Notes for Developers

Login to the app management console:

https://apps.dev.microsoft.com/

For first time use, create Office 365 account first:

- Enroll in Office 365 (30-day trail account is fine), during enrollment, note the tenant value of the form "abcdef.onmicrosoft.com"

- At Office 365 management portal, click "Azure AD" link
- In Azure AD portal, click "Azure Activity Directory"
- Under "Manage" tab, click "App registrations"
- In intro text "To view and manage your registrations for converged applications, please visit the Microsoft Application Console.", click that link

Once in the App Mgmt Console, create an app:

- Note the "Application Id" that is generated (this will be the client id for OAuth2 tokens)
- Use "Generate New Password" when creating keys (this is the client secret for OAuth2 tokens)
- Under "Platforms" tab, click "Add Platform" and specify a redirect URL like "http://localhost/grouper"
- Under "Microsoft Graph Permissions", in "Application Permissions" specify required Graph API permissions:
    - User.Read (may already be set)
    - User.ReadWrite.All
    - Group.Create
    - Group.ReadWrite.All
    - GroupMember.ReadWrite.All
- At bottom of page, click "Save"
- IMPORTANT: after permissions are added (or modified) admin consent must be granted at the URL template:

https://login.microsoftonline.com/$TENANT/adminconsent?client_id=$APPLICATION_ID

Graph API Notes

To get a token for making Graph API calls, do the following:

- POST http method
- URL is of the form https://login.microsoftonline.com/$TENANT/oauth2/v2.0/token

Use the following request parameters:

- client_secret - specify password generated from "Generate New Password" above
- client_id - specify Application Id that's generated when creating app
- grant_type - specify client_credentials, this value implies "Application Permissions" (as opposed to "Delegated Permissions")
- redirect_uri - specify http://localhost/grouper
- scope - specify https://graph.microsoft.com/.default

Official description of relevant group fields (from Microsoft)

Maximum lengths for displayName, description, and mailNickname for provisioned groups must be within the length limits as described by Microsoft (https://github.com/microsoftgraph/microsoft-graph-docs/blob/master/api-reference/v1.0/api/group-post-groups.md), otherwise group creation will fail.

Property	Type	Description
displayName	string	The name to display in the address book for the group. Maximum length: 256 characters. Required.
description	string	A description for the group. Max. length: 1024 characters. Optional.
mailEnabled	boolean	Set to true for mail-enabled groups. Required.
mailNickname	string	The mail alias for the group. Max. length: 64 characters. Required.
securityEnabled	boolean	Set to true for security-enabled groups, including Office 365 groups. Required.
owners	string collection	This property represents the owners for the group at creation time. Optional.
members	string collection	This property represents the members for the group at creation time. Optional.
visibility	String	Specifies the visibility of an Office 365 group. Possible values are: Private, Public, HiddenMembership, or empty (which is interpreted as Public).

Acknowledgments

The Office365/Azure provisioner for Grouper originated as a project from Unicon, Inc. Primary developers were Bill Thompson, Jj!, and John Gasper, with other contributions by Chris Hyzer and Russ Trotter. The source project can be found at https://github.com/Unicon/office365-and-azure-ad-grouper-provisioner.

This project includes contributions from Kansas State University, project https://github.com/kstateome/office365-and-azure-ad-grouper-provisioner.git. Principal development was done by David Malia, with contribution by Kurt Zoglmann.