Advanced Identity Management

Overview

Advanced Identity Management allows you to represent your organization's structure within Matrix and Element: creating a space for all its members, maintaining their membership in rooms and subspaces, managing power levels and more.

It is composed of two main parts:

- **Bridges** connect to an existing data source (LDAP, Azure AD or others) and extract the list of users and groups from it.
  Multiple **Bridges** exists, and more can be added by implementing the `Bridge` interface (see `src/bridging`).

  See Bridging for more details.
- **Provisioner** takes directory produced by a bridge, maps it to matrix spaces (see Space mapping) and enforces its presence on a Matrix server — enforces meaning that it will both create and modify it as needed, but also act as an Matrix Application Service that will automatically react to changes on the Matrix server and check them against the rules established prior.
  Provisioner is ignorant of its data source — it is not aware of the **Bridge** being used and is merely fed data from it.

  See Provisioning for more details.

In addition to that, Advanced Identity Management is also an Application Service. The Provisioner observers the events reported by the AS in case it needs to enforce its rules on entities that it didn't itself create: for example demote a room creator to their expected power level (see LDAP as a source of truth).

Example configuration

```
provisioner:
  # Optional. A list of rooms that'll get automatically created in in managed space.
  # The ID is required to enable GPS to track whether they were already created or not
  # – you can change it, but it'll cause new rooms to be generated.
  default_rooms:
  - id: 'general'
    properties: { name: 'General discussion' }
  # Optional. A list of userid patterns that will not get kicked from rooms
  # even if they don't belong to them according to LDAP.
  # This is useful for things like the auditbot.
  # Patterns listed here will be wrapped in ^ and $ before matching.
  allowed_users:
  - '@adminbot:.*'
  # Optional. Determines whether users will be automatically invited to rooms (default, public and space-
joinable)
  # when they gain access to them. Defaults to true. Users will still get invited to spaces regardless of this
setting.
  invite_to_public_rooms: false
  # Optional: A list of remote Advanced Identity Management we'll be federating with. Requests from other
remote users will be ignored.
  federation:
    federates_with:
    - '@aim_bot:consultancy.test'
  # Optional. When enabled, spaces that are no longer configured, and rooms belonging to those spaces will be
cleaned up.
  # This will likely become enabled by default in the future.
  # When disabled (or omitted), GS will log the rooms and spaces it would clean up if allowed to.
  gc:
    enabled: true

# Optional. If configured, Advanced Identity Management will synchronize user accounts (attributes and account
validity)
# found in the data directory to the specified list of targets.
userProvisioner:
  # Optional. Configure to enable user deprovisioning. Disabled by default.
  deprovisioning:
    enabled: false
    # Optional. When users get removed from the directory their accounts will only be deactivated,
    # but their erasure will be delayed by the specified time period, allowing them to be reactivated in the
meantime.
    # The format is <amount><unit>, with amount being numeric and unit being one of: [s, m, h, d], for seconds,
minutes,
    # hours or days respectively (for example: "24h", "31d" etc.).
    # The specified period will be translated into seconds, so won't account for things like DST, leap seconds etc.
    # Users will be deleted *no sooner* than that, but may be removed a bit later, depending on other Advanced
Identity Management operations.
    # By default set to 30 days.
    soft_delete_period: '30d'

# Configure the spaces you want Advanced Identity Management to manage on your Matrix server
spaces:
  # The internal ID of this space. Don't change it after you set it, or it will create a new one and abandon the old
one.
```

```
- id: main
  # The display name of the space, safe to change later.
  name: 'My Company'
  # The list of groups from your user directory to add as members to this space.
  # An empty string is a special name that means "all available users, regardless of group memberships".
  # You can set a power level for any of the groups. By default it's 0.
  # This space is going to contain all the users in the directory,
  # and those that are present in the "managers" groups will be the moderators (in the space and its child
rooms).
  groups:
    - externalId: ''
    - externalId: 'cn=managers,ou=employees,dc=element,dc=test'
      powerLevel: 50
- id: management
  name: 'Management'
  groups:
    - externalId: 'cn=managers,ou=employees,dc=element,dc=test'
  # You can confgure spaces that'll be federated between multiple GS servers.
  # Each Advanced Identity Management will only manage its local users.
- id: shared
  name: 'Federated space'
  groups:
    - externalId: 'cn=engineering,ou=employees,dc=element,dc=test'
  federatedGroups:
      # The external ID of the group on the foreign server.
      # This will be enforced by the Advanced Identity Management running on consultancy.test, not this instance
configured here.
      # The Advanced Identity Management running on consultancy.test needs to have our MXID
      # (@gpsbot:element.test by default) configured in its provisioner.federates_with config option.
    - externalId: 'ou=element-contractors,dc=consultancy,dc=test'
      # The MXID of the remote Advanced Identity Management bot.
      agent: '@aim_bot:consultancy.test'

source:
  type: 'ldap'
  # LDAP will be checked for changes every this many seconds
  check_interval_seconds: 60
  # The following can be copied straight from Synapse's homeserver.yaml
  # if you're already using its LDAP password provider
  uri: "ldap://element.test"
  # The base ou we specify here will become the root space
  base: "ou=employees,dc=element,dc=test"
  # Optional. An LDAP filter to use when searching for entries
  filter: '(!(ou=Domain Controllers))'
  # Make sure the account you use here has enough permissions to perform searches on your `base`
  bind_dn: "ELEMENT\\administrator"
  bind_password: "donkey.8"
  # Needs `uid` to be able to determine Matrix localparts for users
  # and `name`s to pick the right names for spaces
  attributes:
    uid: "sAMAccountName"
    name: "name"
  # If the LDAP server requires a client certificate, enable this option.
  # cert:
    # The path to the file
    # file: "./my-cert-file.pem"
    # OR the PEM-encoded cert itself
    # cert: "foobar"
    # Passphrase for the cert, if required.
    # passphrase: "passphrase"

# For Microsoft Graph:
# source:
#   type: 'ms-graph-ad'
#
#   # This is the "Tenant ID" from your Azure Active Directory Overview
#   tenant_id: 'b9355cb3-feed-dead-beef-9cc325f0335b'
#
#   # Register your app in "App registrations". This will be its "Application (client) ID"
#   client_id: '5c955b66-18b3-42de-bb5a-13b5a202d4fc'
#
#   # Go to "Certificates & secrets", and click on "New client secret".
#   # This will be the "Value" of the created secret (not the "Secret ID").
#   client_secret: 'yOb7Q~Km~~YMKzpeq73swJj3kOeJpUwXSZamr'
#   # For the bridge to be able to operate correctly, navigate to API permissions and unsure
#   # it has access to GroupMember.Read.All and User.Read.All
#   # Application permissions for Microsoft Graph. Remember to grant the admin consent for those.
#
#   # Optional. The url to reach Graph on. Override if your deployment uses a specific graph endpoint.
#   base_url: 'https://graph.microsoft.com/'
```

```
#
#   # Optional. Specific scopes to set for graph to use.
#   scopes: ['https://graph.microsoft.com/.default']

# For SCIM:
#   type: 'scim'
#   # HTTP port that the SCIM server will listen on
#   port: 8040
#   # Optional URL prefix for all routes
#   base_url: '/scim/v2'
#   client:
#     # Unique ID for the SCIM client.
#     # This will be used to keep track of the managed Space and User/Group storage in Matrix.
#     id: 'element-ad'
#     # You can set up multiple client tokens with different permission levels.
#     rbac:
#         # Bearer token for the client, as per RFC 6750
#       - token: 'foo-bar-baz'
#         # What's the token allowed to do: in this case, everything (read+write on all endpoints).
#         # The format for these is 'access:scope', access being 'read', 'write' or '*' for both,
#         # scope being 'users', 'groups' or '*' for everything.
#         roles: ['*:*']
#         # You can specify permissions for anyone who presents a valid Matrix access_token for an admin user
#       - synapse_user: 'admin'
#         # ...and assign more fine-tuned permissions to it
#         roles: ['read:*', 'write:groups']
#     attributeMapping:
#       # The SCIM user attribute that'll be used as the Matrix username for provisioned users
#       username: 'externalId'
#   # Should SCIM user creation register a Matrix account for the user.
#   # Possible values are 'yes', 'no' and 'if-missing'
#   # - 'yes' will register Matrix accounts on the server upon a SCIM create user request,
#   #   and error out if the user with that username already exists.
#   # - 'if-missing' will register Matrix accounts unless they exist already.
#   #   This is useful if some users have their user accounts created independently before the SCIM bridge was
set up.
#   # - 'no' will not create user accounts, only work with existing ones.
#   register_users: 'no'
#   # Optional: Should SCIM responses wait for Matrix provisioning to complete.
#   # It is recommended to leave it as false. HTTP responses will be sent quicker,
#   # and Matrix provisioning may still fail in the background (to be retried later).
#   synchronous_provisioning: false
#   # Optional: Configure a mailer to send email notifications to newly registered, activated and deactivated
users.
#   # mailer:
#   #   # The email address emails will be sent from
#   #   from: 'element@element.com'
#   #   # Path to a directory with email templates.
#   #   # Each template should be a directory containing 'subject.pug', 'text.pug' and 'html.pug',
#   #   # all using https://pugjs.org/ as a template language.
#   #   # Advanced Identity Management ships with standard, Element-branded templates in templates/
#   #   templates_path: './templates'
#   #   # SMTP transport configuration, as per https://nodemailer.com/smtp/,
#   #   # except that we default `secure` to `true` and `port` to 465.
#   #   transport:
#   #     host: 'smtp.example.com'
#   #     auth:
#   #       user: 'mailer'
#   #       pass: 'mailerpass.8'

# Optional. Configure this to gather usage statistics.
# See telemetry spec at https://gitlab.matrix.org/new-vector/modular/telemetry-schema
# for details on what's being gathered and sent.
telemetry:
  # Identifier of this Advanced Identity Management instance
  instance_id: 'foo'
  # Every this many seconds (and on startup) telemetry will be recorded (and optionally sent)
  send_interval: 3600
  # Optional: the EMS endpoint to submit telemetry entries to.
  # This is optional as it wouldn't work for airgapped environments,
  # and by default no telemetry is sent (but it is still gathered).
  endpoint: 'https://ems.com/telemetry'
  # Optional: how many times should we retry sending telemetry if it fails. Defaults to 3
  retry_count: 3
  # Optional: how long should we wait between retries. Defaults to 60, in seconds
  retry_interval: 60

# Optional
logging:
  # Allowed levels are: error, warn, info, http, verbose, debug, silly - case sensitive.
# "info" will typically notify of all "write" actions (affecting the state of the homeserver),
# while "debug" will also be reporting checks performed that didn't result in any changes.
level: "info"
# Optional. Allowed formats are are:
# - pretty: the default. A timestamped, colorized output suitable for humans
# - json:   logging a json object containing a `level`, `message`, `timestamp` and optionally a `label`
format: "json"
```

Bridging

Bridging directories

Bridges' job is to turn the contents of an external data directory into a data structure that can then be then constructed on the Matrix server by the Provisioner. See State representation for the details description of the data structure being produced.

See specific bridges (in the sidebar) to learn more about how GS interprets the contents of specific data sources.

Bridges run continously and trigger provisioning whenever they observer changes in the data source.

LDAP

The LDAP bridge will periodically (according to its configuration) fetch the LDAP tree from the server (filtering out the things it doesn't find interesting).

To enable maximum flexibility it "flattens" the LDAP tree so that the users' (and groups') place in directory tree doesn't matter.

Groups, OrgUnits and Domains (if found), will all be flattened and treated like a container for users. It makes it possible to use their DNs[^note] (fully qualified names) to assign users to spaces and power levels in those spaces.

[^note]: CNs are also allowed here for backwards compatibility reasons, but **only for groups**. It is however advised to avoid using CNs and use DNs instead, since they are guaranteed to be unique across the LDAP tree. GS' behaviour is undefined when mapping groups with duplicate names.

For example, for the following LDAP tree:

```
- Company (Domain) (`dc=company`)
  - Alfred (User) (`cn=alfred,cn=company`)
  - Engineering (OrgUnit) (`ou=engineering,cn=company`)
    - Barbara (User) (`cn=barbara,ou=engineering,cn=company`)
    - Moderators (Group) (`cn=moderators,ou=engineering,cn=company`)
      - Charlie (User) (`cn=charlie,ou=engineering,cn=company`)
```

Company, Engineering and Moderators will all be treated as if they were an group. We could then use the following space mapping configuration with it:

```
spaces:
  id: root
  name: "Company"
  groups:
    - externalId: `dc=company` # or leave it empty with the same result
  subspaces:
    - id: engineering
      name: Engineering
      groups:
        - externalId: `ou=engineering,cn=company`
        - externalId: `cn=moderators,ou=engineering,cn=company`
          powerLevel: 50
```

Example Configuration

When using the helm chart, the authentication schema is automatically used to configure GroupSync LDAP source. If you want to override some settings, you can always implement the following configuration:

```
source:
  type: 'ldap'
  # LDAP will be checked for changes every this many seconds
  check_interval_seconds: 60
  # The following can be copied straight from Synapse's homeserver.yaml
  # if you're already using its LDAP password provider
  uri: "ldap://element.test"
  # The base ou we specify here will become the root space
  base: "ou=employees,dc=element,dc=test"
  # Optional. An LDAP filter to use when searching for entries
  filter: '(!(ou=Domain Controllers))'
  # Make sure the account you use here has enough permissions to perform searches on your `base`
  bind_dn: "ELEMENT\\administrator"
  bind_password: "donkey.8"
  # Needs `uid` to be able to determine Matrix localparts for users
  # and `name`s to pick the right names for spaces
  attributes:
    uid: "sAMAccountName"
```

```
      name: "name"
    # If the LDAP server requires a client certificate, enable this option.
    # cert:
      # The path to the file
      # file: "./my-cert-file.pem"
      # OR the PEM-encoded cert itself
      # cert: "foobar"
      # Passphrase for the cert, if required.
      # passphrase: "passphrase"
```

Microsoft Graph

The MsGraph bridge will periodically (according to its configuration) perform the following API calls:

- `/organization`, to determine the name of the organization
- `/users` to get the list of users in the org
- `/groups` to get the list of groups in the org
- `/groups/<id>/members` to get a list of members for a particular group

In order to perform the queries successfully, Advanced Identity Management's Application needs to have the following permissions granted in Azure:

- `User.Read.All`
- `GroupMember.Read.All`

It emits a list of users and groups as-is, without performing any transformations on them.

Example Configuration

Using MS-Graph requires the following GroupSync configuration :

```
source:
  type: 'ms-graph-ad'

  # This is the "Tenant ID" from your Azure Active Directory Overview
  tenant_id: 'b9355cb3-feed-dead-beef-9cc325f0335b'
```

```
# Register your app in "App registrations". This will be its "Application (client) ID"
client_id: '5c955b66-18b3-42de-bb5a-13b5a202d4fc'

# Go to "Certificates & secrets", and click on "New client secret".
# This will be the "Value" of the created secret (not the "Secret ID").
client_secret: 'yOb7Q~Km~~YMKzpeq73swJj3kOeJpUwXSZamr'
# For the bridge to be able to operate correctly, navigate to API permissions and unsure
# it has access to GroupMember.Read.All and User.Read.All
# Application permissions for Microsoft Graph. Remember to grant the admin consent for those.

# Optional. The url to reach Graph on. Override if your deployment uses a specific graph endpoint.
base_url: 'https://graph.microsoft.com/'

# Optional. Specific scopes to set for graph to use.
scopes: ['https://graph.microsoft.com/.default']
```

SCIM

The SCIM bridge maintains an HTTP service that conforms to the SCIM protocol (RFC 7644) and provisions a Matrix server with the SCIM resources sent to it.

Configuration

The following options are available when configuring the SCIM bridge:

- `base_url` (optional) - the URL prefix for each route. For instance, with `base_url` set to `/scim/azure` the requests need to be hitting `/scim/azure/Users` etc. Useful when running behind a non-rewriting proxy. Set to an empty string by default.
- `client` – a structure with the following fields:
  - `id` - a string specifying the name of the client, e.g. the name of the organization. Must be unique across SCIM bridge instances running on the server. Changing this value after some SCIM resources have been provisioned is equivalent to creating a new user/group database and a new Matrix Space. The value will also be the default name of the organization Space (which can later be changed by Space Moderators).
  - `token` - access token for the SCIM client, as per RFC 6750
  - `attributeMapping` - a structure with the following fields:

    - `username` - the SCIM user attribute to use when determining the Matrix username. If you're using OIDC, make sure it matches its setup.
- `synchronous_provisioning` (optional) - a boolean flag. If set to true, SCIM responses won't be sent before the Matrix provisioning finishes, and any Matrix errors may cause SCIM requests to fail and potentially leave the server in an invalid state. Useful for testing. False by default, and it's strongly recommended to leave it that way.

Example configuration

Configuring the SCIM bridge requires to configure the following values. When using ESS Helm Chart, you need to set `groupSync.enableSCIM` to expose the SCIM ingress. It will be abailable under GroupSync ingress if it is enabled, or Synapse ingress at `/scim/v2` path.

```
source:
  type: 'scim'
  client:
    # Unique ID for the SCIM client.
    # This will be used to keep track of the managed Space and User/Group storage in Matrix.
    id: 'element-ad'
    # You can set up multiple client tokens with different permission levels.
    rbac:
        # Bearer token for the client, as per RFC 6750
      - token: 'foo-bar-baz'
        # What's the token allowed to do: in this case, everything (read+write on all endpoints).
        # The format for these is 'access:scope', access being 'read', 'write' or '*' for both,
        # scope being 'users', 'groups' or '*' for everything.
        roles: ['*:*']
        # You can specify permissions for anyone who presents a valid Matrix access_token for an admin user
      - synapse_user: 'admin'
        # ...and assign more fine-tuned permissions to it
        roles: ['read:*', 'write:groups']
    attributeMapping:
      # The SCIM user attribute that'll be used as the Matrix username for provisioned users
      username: 'externalId'
  # Should SCIM user creation register a Matrix account for the user.
  # Possible values are 'yes', 'no' and 'if-missing'
  # - 'yes' will register Matrix accounts on the server upon a SCIM create user request,
  #   and error out if the user with that username already exists.
  # - 'if-missing' will register Matrix accounts unless they exist already.
  #   This is useful if some users have their user accounts created independently before the SCIM bridge was set
up.
  # - 'no' will not create user accounts, only work with existing ones.
  register_users: 'no'
  # Optional: Should SCIM responses wait for Matrix provisioning to complete.
  # It is recommended to leave it as false. HTTP responses will be sent quicker,
  # and Matrix provisioning may still fail in the background (to be retried later).
  synchronous_provisioning: false
  # Optional: Configure a mailer to send email notifications to newly registered, activated and deactivated users.
  # mailer:
  #   # The email address emails will be sent from
  #   from: 'element@element.com'
  #   # Path to a directory with email templates.
  #   # Each template should be a directory containing 'subject.pug', 'text.pug' and 'html.pug',
  #   # all using https://pugjs.org/ as a template language.
  #   # Group sync ships with standard, Element-branded templates in templates/
  #   templates_path: './templates'
  #   # SMTP transport configuration, as per https://nodemailer.com/smtp/,
  #   # except that we default `secure` to `true` and `port` to 465.
  #   transport:
  #     host: 'smtp.example.com'
  #     auth:
  #       user: 'mailer'
  #       pass: 'mailerpass.8'
```

Space Mapping

This mechanism allows us to configure spaces that Advanced Identity Management will maintain.

Configuration

We define each space giving it a name (which will be displayed in Element), a unique ID (which allows Advanced Identity Management to track the Space even if it gets renamed), and a list of groups whose users will become the members of the Space. Users needs to be a member of *any* configured group, not all of them.

You can pick any ID you want, but if you change it later Advanced Identity Management will create a brand new space and abandon the old ones, likely confusing the users.

In order to limit space membership to a specific Group, we include its Group ID.

Each group may optionally include a powerLevel setting, allowing specific groups to have elevated permissions in the space.

A special group ID of `''` (an empty string) indicates that all users from the server, regardless of their group membership, should become the members of the Space.

In addition to regular groups, you may also make a space federated by specifying `federatedGroups` and a remote Advanced Identity Management server. See Federation for more details.

An optional list of subspaces may also be configured, each using the same configuration format and behaviour (recursively).

If a space has subspaces configured, its members list will be composed of the members of the space itself **any any of its subspaces, recursively** -- so a subspace's member list is always a subset of its parent space's member list. This may change in the future, so it's advised not to rely on this when configuring your spaces.

```
spaces:
  id: root
  name: 'Company'
  groups:
    - externalId: 'element-users'
```

With `powerLevel` option allows us to give users extra permissions. This is equivalent to the `group_power_level` setting[^note].

```
spaces:
  id: root
  name: 'Company'
  groups:
    # regular users
    - externalId: 'element-users'
    # moderators
    - externalId: 'element-moderators'
      powerLevel: 50
```

In case of Power Level conflicts, the highest power level will be used. With the following configuration:

```
spaces:
  id: root
  name: 'Company'
  groups:
    - externalId: 'moderators'
      powerLevel: 50
    - externalId: 'admins'
      powerLevel: 100
```

A user who's a member of both `moderators` and `admins` will end up with Power Level of 100.

Subspaces can be configured analogically:

```
spaces:
  id: shared
  name: "Element Corp"
  groups:
  - externalId: 'matrix-mods'
    powerLevel: 50
  - externalId: ''
  subspaces:
  - id: london
    name: "London Office"
    groups:
    - externalId: 'london-matrix-mods'
      powerLevel: 50
    - externalId: 'london-employees'
```

Provisioning

Provisioning

The role of the provisioner is to take the expected state representation produced by Bridges and *ensure* that the server state matches these expectations. The provisioner will try to do as little as possible to go from the existing to the desired state — in particular, running a Provisioner twice will result in no operations being performed on the second run.

Provisioning will typically be triggered by the bridge, either on its startup or whenever it becomes aware of changes in the data source.

See Usage Scenarios for examples of provisioning actions in response to data source changes.

Example Configuration

```
provisioner:
  # Optional. A list of rooms that'll get automatically created in in managed space.
  # The ID is required to enable GPS to track whether they were already created or not
  # – you can change it, but it'll cause new rooms to be generated.
  default_rooms:
  - id: 'general'
    properties: { name: 'General discussion' }
  # Optional. A list of userid patterns that will not get kicked from rooms
  # even if they don't belong to them according to LDAP.
  # This is useful for things like the auditbot.
  # Patterns listed here will be wrapped in ^ and $ before matching.
  allowed_users:
  - '@adminbot:.*'
  # Optional. Determines whether users will be automatically invited to rooms (default, public and space-joinable)
  # when they gain access to them. Defaults to true. Users will still get invited to spaces regardless of this setting.
  invite_to_public_rooms: false
  # Optional: A list of remote Advanced Identity Management we'll be federating with. Requests from other
remote users will be ignored.
  federation:
    federates_with:
    - '@gs_bot:consultancy.test'
```

State representation

Both users and power level targets are currently only represented as a localpart: Advanced Identity Management is meant to manage a single server, where each organization member has an account on the server being provisioned.

Advanced Identity Management is *not* involved in the registration of user accounts themselves — this is typically handled by Synapse's authentication provider. Some bridges may take this responsibility upon themselves — for example the SCIM bridge, when new User accounts are being sent to it. Still, even in that case, Provisioner is not responsible for ensuring that the accounts exist before it starts managing them.

User provisioning

Advanced Identity Management can be configured to synchronize user accounts found in the bridged data directory to a specified list of targets.

Currently the only supported target is Synapse, and the synchronization is limited to user attributes for already existing accounts.

If you are using the helm chart, this can be configured through `groupSync.syncedUserAttributes`.

Attribute sync

When users in a data directory change, Advanced Identity Management will ensure that the attributes match those in Synapse (and in the future, other user provisioning targets). Advanced Identity Management will only update users if any updates need to be performed, and only update the attributes it needs to.

Supported attributes are:

- `displayName`

  Refers to a `displayname` attribute in Synapse.
  In LDAP, displayName is obtained from the value of an attribute configured as `name` in the attribute mapping.
  In Azure AD and SCIM its value is taken from the `displayName` attribute of a given user.
- `emails`
  Refers to Synapse's `threepids` for `medium: "email"`. When updating this attribute, all threepids other than `"email"` will be left intact.
  In LDAP, the value of this attribute is determined by the value of the attribute configured as `mail` in attribute mapping.
  In Azure AD, the value of this attribute is taken from the `mail` attribute in Azure AD. This is limited to just one email address per user.
  In SCIM, the value of this attribute is taken from the `emails` attribute for a given user.

Advanced Identity Management can be configured to sync all of them, or a limited set. See the example config for more details.

Federation

Advanced Identity Management supports closed federation — as in, one where all participating servers are known in advance.

Each federated server maintains its own Advanced Identity Management instance, crucially its own Provisioner. Each Provisioner is responsible for managing users belonging to its homeserver, and ignores those that belong to another homeserver and another Provisioner.

The servers are are equal, and any of them may invite other Advanced Identity Management servers to any of its spaces -- but they do need to be on a preconfigured list of servers (see `federates_with` option in the example config).

When a Advanced Identity Management server wishes to federated with another, it should specify which of its spaces should include a remote Advanced Identity Management server, and which of its groups should be invited.

Example

Let's say we have an organization with two servers -- dallas.example.com and berlin.example.com. Both use Advanced Identity Management with their own data directories.

Dallas has 3 users: Alice, Bob and Cyril. Alice is additionally in a group called "dallas-management".

Berlin has 3 users too: Dave, Eve and Francis. Dave is a member of "berlin-management" group.

We'll set up a space federated between the two, so that users from both servers will end up there, which both managers having a power level of 50.

Federation whitelist

Both provisioners need to have to be aware of the other participating server, so for Dallas we need:

```
provisioner:
  federates_with: ['@groupsync:berlin.example.com']
```

And for Berlin:

```
provisioner:
  federates_with: ['@groupsync:dallas.example.com']
```

Without having that configured, each Advanced Identity Management will ignore the requests sent by the other one, in order to not accidentally expose information to an untrusted party.

Note that Matrix IDs in the `federates_with` section must match the other servers: both the `server_name`s and the `sender_localpart`s, so that Advanced Identity Managements know how to invite to rooms and spaces as co-conspirators.

Federated space mapping

While the space will be replicated on both servers, with both being equally responsible for it, we need to pick a server to create it on. Let's do it on the Dallas server:

```
- id: shared
  name: 'Federated space'
  groups:
    - externalId: '' # include all our users
    - externalId: 'dallas-managers'
      powerLevel: 50
  federatedGroups:
      # The MXID of the remote Advanced Identity Management bot.
      agent: '@groupsync:berlin.example.com'
    - externalId: '' # include all users known to the Berlin Advanced Identity Management
    - externalId: 'berlin-managers'
      powerLevel: 50
```

No additional configuration is needed on the Berlin server.

Once this configuration is applied, the following thing will happen:

- The Dallas GS will create the `Federated space`, invite its users, (Alice, Bob and Cyril) and make Alice a moderator (PL50).

- The Dallas GS will invite the Berlin GS (`@groupsync:berlin.example.com`) to that Space, and store its expectations for it in a Matrix State Event.
- The Berlin GS will receive an invitation to `Federated space` and recognize it as coming from a federating GS (inviter is on the `federates_with` list).
- The Berlin GS will join a space and read the State Event to figure out which of its users should be members of the `Federated Space`
- The Berlin GS will invite all it users (Dave, Eve and Francis) to `Federated space` and make Dave a moderator (PL50).
- When enforcing memberships rules, both servers will only consider users from its own server: The Dallas GS will never touch Berlin users and vice versa.

Room Cleanup

Room cleanup

After each provisioning cycle, Advanced Identity Management will clean up the rooms and spaces that it no longer needs to manage. Spaces in Matrix are still rooms, but we treat them a little differently during the cleanup, matching their distinct uses.

Internally, room cleanup is refered to as Room GC (Garbage Collection).

This is meant to be a reversible process (in case it was performed accidentally), so we avoid information loss when possible.

Space cleanup

Spaces are cleaned up when they are no longer configured -- once they are removed from Space mapping configuration, Advanced Identity Management will abandon them by kicking every member and then leaving itself

- resulting in an empty room that will eventually get cleanup up entirely by the homeserver.

The kicking of the users is done so that the deconfigured spaces don't show up in their clients anymore. The rooms inside those spaces remain accessible though, so no conversations are being lost.

We don't draw a distinction here between GS- and user-created spaces, because GS doesn't care about user-created spaces at all. It never joins them and it never manages them, so they will never be part of the cleanup process.

Room cleanup

Rooms are cleaned up when they're no longer accessible from any of the spaces that Advanced Identity Management manages. This can happen in a few cases:

1. The room belonged to a space that was cleaned up up by Advanced Identity Management
2. The room has been removed from a space managed by Advanced Identity Management
3. The room is made private (but remains in a managed space)

Notably, none of these apply if a default rooms gets deconfigured in Advanced Identity Management. Those get created in each GS-managed space, but after their creation they're treated like any other space-public[^note] room.

When a room is cleaned up, GS cleans up its room metadata (this is stored in state events) and leaves the room. All the room members remain in the room so that the conversation is preserved and can continue if needed. Room moderators can then tombstone the room if they so desire, or add it to a different space.

If the room was not originally created by Advanced Identity Management, we give PL 100 back to its original creator (having taken it away back when we took control of it). If a room was created by Advanced Identity Management, its power levels are not touched. Advanced Identity Management remains a room admin in case it needs to take control of the room again in the future (e.g. because it gets added to a different managed space).

[^note]: Space-public meaning: with `join_rule: restricted`, allowing space members to join.

Configuration

Room and Space cleanup can be configured through :

```
provisioner:
  # Optional. When enabled, spaces that are no longer configured, and rooms belonging to those spaces will be cleaned up.
  # This will likely become enabled by default in the future.
  # When disabled (or omitted), GS will log the rooms and spaces it would clean up if allowed to.
  gc:
    enabled: true
```

Usage Scenarios

Usage Scenarios

With LDAP bridge as an example data source.

Onboarding

- User logs in to Element, either using OpenID or with a user+password with LDAP integration
- User automatically gets invited to spaces matching their LDAP Organizational Unit memberships, which are nested the way they are nested in LDAP
- For each space they're in, user gets invited to every room that's configured to be joinable by space members
- **Result**: user learns their place in the company structure, discovers their peers and becomes aware of all the public conversations happening in the company

For the following sections, we assume a hierarchy of LDAP Organizational Units:

- Employees
  - Engineering
  - Support

Restructuring

- In the previously described OrgUnit hierarchy, assume a user Evan belonging to Engineering
- Evan is being moved from Engineering to the more generic Employees
- GS kicks evan from the Engineering space and from all the public rooms in it
- Evan is added to the Support OrgUnit
- GS invites Evan to the Support Matrix space and all its space-public rooms
- **Result**: Moving users around within the company hierarchy is represented by moving them around in the Matrix space.

Offboarding

- In the previously described OrgUnit hierarchy, assume a user Evan belonging to Engineering
- Evan is being removed from LDAP entirely, or moved outside of Employees, which is the root space managed by GS
- Evan's Matrix account is being deactivated, preventing them from logging in.
- Evan is subject to the user deletion flow.

- **Result**: GS will automatically erase a user from Matrix if they no longer belong to the LDAP space managed by it.

Permission management

- GS is configured to assign a power level of 50 to every user in groups called moderators or engineering-moderators
- In the previously described OrgUnit hierarchy, we have users Evan and Brenda belonging to Engineering (and therefore also to Employees).
- We create two LDAP Security Groups: moderators in Employees and engineering-moderators in Engineering
- We make Evan a member of moderators, and Brenda a member of engineering-moderators
- GS assigns a power level of 50 to Evan in the Employees Matrix space and all rooms contained within it – except its subspaces (including Engineering) where Evan’s power level is still the default 0
- GS assigns a power level of 50 to Brenda in the Engineering and all its child rooms. Brenda still has a default power level of 0 in Employees
- The spaces managed by GS allow its moderators (PL 50) to create child rooms and spaces
- **Result**: LDAP Security Groups can be used to manage Matrix power levels in a granular and configurable manner

LDAP as a source of truth

- In a space hierarchy managed by GS, a user acquires a power level higher than the one described in their LDAP security group
- In response to that, GS demotes the user back to the power level that they should have according to LDAP. If the user is currently an Admin, GS will temporarily take over their account and make them demote themselves.
- A user, for any reason, ends up in a room that they shouldn’t be in – for example, a room they were a member of now becomes a part of the company Space.
- In response to that, GS immediately kicks them from the room they weren’t supposed to be in.
- **Result**: LDAP is the source of truth, and any changes in Matrix that don’t conform to the rules established in LDAP get automatically corrected.

User Deletion

When user are no longer desired on the AIM-managed server, they fall under the user deletion flow described here.

Deletion criteria

They are considered no longer desired if they are present in the root space of the organization, but they're not on the list of users who are supposed to be provisioned. In a GS-managed server this will only happen if a user was first added to the directory (and provisioned), and then removed from it.

Potential issues

If a user who was never part of the directory finds themselves in the company space somehow, they will be subject to the user deletion flow as well. This should never happen, as GS will not ever invite them to the root space by itself. However, a user with Admin (PL 100) permissions could invite them to the root space manually, which would make GS consider them a deletion candidate -- despite them never being part of a the directory and not being managed by GS before that point.

Configuration

User deletion can happen either instantly, or delayed by a configurable "grace period". In practice, the instant deletion is implemented the exact same way as the delayed deletion, but with grace period set to 0 seconds. It's controlled by the `user_soft_delete_period` parameter in the config, and leaving it unset is the same as setting it to `"0s"`.

```
userProvisioner:
  # Optional. Configure to enable user deprovisioning. Disabled by default.
  deprovisioning:
    enabled: false
    # Optional. When users get removed from the directory their accounts will only be deactivated,
    # but their erasure will be delayed by the specified time period, allowing them to be reactivated in the meantime.
    # The format is <amount><unit>, with amount being numeric and unit being one of: [s, m, h, d], for seconds, minutes,
    # hours or days respectively (for example: "24h", "31d" etc.).
    # The specified period will be translated into seconds, so won't account for things like DST, leap seconds etc.
    # Users will be deleted *no sooner* than that, but may be removed a bit later, depending on other Advanced Identity Management operations.
    # By default set to 30 days.
    soft_delete_period: '30d'
```

Actions performed instantly

When a user is found to be undesirable, their Matrix account is deactivated, preventing them from logging in. They are not being removed from any rooms though, and their permissions and power levels stay the same. It's equivalent to using a deactivate user Synapse API (with `erase` set to `false`).

If a grace period is configured and the user gets added back to the directory before they get deleted completely, their account will get reactivated.

The user then gets put on the "grace period" list.

NOTE: Historically Advanced Identity Management would reset the password for the user to a random value, but modern versions will just lock the account.

Actions performed after the grace period

Once the grace period expires for a user, they get erased from the server. This is done under the hood by the Synapse API deactivate user, with `erase` set to `true`.