

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 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 its 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.
-

Revision #4

Created 15 May 2025 09:22:30 by Gaël Goinvic

Updated 27 May 2025 06:41:06 by Gaël Goinvic