Migration of Static Permissions Upon Upgrade

Overview

The purpose of this spike is to investigate how to implement static permission migration. That is, how to handle when permissions & permission sets defined in module descriptors are renamed, removed, or updated (e.g. to include new permissions) during a module upgrade. This does not cover the case of user-defined module permission sets.

TL;DR

The following statements apply to permissions defined in module descriptors. NOT user-defined permissions.

- Permissions can now be renamed via a new "replaces" property. Perm-users (Assignments) will be updated automatically.
- When permissions that once appeared in a module descriptor are removed in a newer version of the module descriptor, they will be marked deprecated.
    - Initially the displayName of these permissions will be prefixed with "(deprecated)", but the permissions will not be filtered out of any API calls.
    - Eventually these permissions remain assigned to users, but will be filtered out of perms API results unless a new query argument (includeDeprecated=true is specified). This will be handled in a separate story which may not make it into R1 2021.
- A new API will be introduced to allow an operator to purge deprecated permissions. Once this is done a module downgrade will not result in those permissions being re-assigned.
- If a permission name collides with an user-defined permission with the same name,
    - Initially the call to enable the module will fail with an appropriate error message.
    - Eventually we may do something like rename the user-defined permissions and adjust assignments as needed.
- Going forward, all system-defined permissions will include context about the module that defined them (new permission fields moduleName /moduleVersion).
- When upgrading mod-permissions, OKAPI will "refresh" all the existing permissions, this is already done when first enabling mod-permissions, but now will be done on upgrades as well.
    - In most cases this will be a no-op, but this allows us to update permissions w/ the module context.

JIRA

Spikes:

- **OKAPI-839** - Getting issue details... STATUS
- **OKAPI-952** - Getting issue details... STATUS
- **OKAPI-953** - Getting issue details... STATUS

Implementation Stories:

- **OKAPI-947** - Getting issue details... STATUS
- **MODPERMS-115** - Getting issue details... STATUS
- **MODPERMS-116** - Getting issue details... STATUS

Changes to OKAPI

ModuleDescriptor

- Add a new field to the permission.json schema
  - `replaces`:
    - Array of string
    - Each item is a name of a permission that this permission replaces.
    - Optional

During module upgrade

1. OKAPI Makes a call to mod-permissions' _tenantPermissions endpoint, providing a lists of permissions (details below).

Upon mod-permissions Install/Upgrade

1. Send all permissions to mod-permissions to "refresh" them. See "Determination of which permissions are new/updated/removed" below.

Changes to mod-permissions

Tenant Permissions API

_tenantPermissions is updated to determine which permissions are new, updated, removed. That information is that used to update not only the permissions table but also the user-permissions table.   So if a user has a permission which is being renamed, updated with fewer/additional sub-permissions, or removed, the user's permissions are updated accordingly.

Determination of which permissions are new/updated/removed

The first time mod-permissions that implements v2.0 of the _tenantPermissions interface is enabled, OKAPI will call that API with the permissions defined in each of the enabled module descriptors.   Special handling exists in mod-permissions that will allow it to add the necessary module context, enabling it to detect/determine which permissions are new/removed/updated in subsequent _tenantPermissions calls.

OkapiPermissionSet.json Schema

See https://github.com/folio-org/mod-permissions/blob/master/ramls/okapiPermissionSet.json

This is the payload of the _tenantPermissions endpoint remains the same, aside from "moduleId" now being a required field.

Example:

```
{
  "moduleId": "mod-foo-2.0.0",
  "perms": [ <list of permission objects> ]
}
```

Permission.json Schema

See https://github.com/folio-org/mod-permissions/blob/master/ramls/permission.json

This is the stored permission schema. It needs to be expanded to include some additional context about who/what defined it.

- `moduleName`
  - String
  - Optional
  - The name of the module (not including version) that defined this permission
  - Read only
- `moduleVersion`

- String
- Optional
- The version of the module that defined this permission
- Read only

Example 1 - Module-defined permission

```
{
  ...
  "moduleName": "mod-foo",
  "moduleVersion": "1.2.3"
}
```

Or, for a user-defined permission these fields would simply be omitted. Permissions created by the _tenantPermissions API will always have moduleName populated, which Permissions created by the Perms API will never populate moduleName or moduleVersion.

Removal of Static Module Permissions

In order to support the ability to downgrade a module, the decision has been made to implement a soft delete when permissions are removed from a module descriptor. Here, the OKAPI part stays the same, but on the mod-permissions side, the _tenantPermissions API will not actually delete permissions, nor will it remove the permission assignments. Instead, the permission will be marked as inactive. A separate process for cleaning up inactive permissions will be introduced.

Inactive permissions should be filtered out of responses from the various perms APIs:

`GET /perms/permissions`

- Query the DB for the permissions - automatically/transparently add a clause to filter out inactive permissions
- Query the DB for all of the inactive permissions - here I'm assuming there aren't going to be many of them. we can probably cache this result.
- Filter out the inactive permissions from the subPermissions array in each of the permissions in the results.
- Filter out the inactive permissions from the childOf array in each of the permissions in the results.
- Return the adjusted results
- Add a new query argument - includeInactive
    - Boolean
    - Default: false
    - Indicates whether inactive permissions should be returned or not (including in subPermissions/childOf arrays)

NOTE: In the case where we query for a permission by name... e.g. `GET /perms/permissions?query=permissionName=some.inactive.permission` ? We automatically add a clause to not return inactive permissions, so this would return an empty result set, even though the permission does exist in the system. Or course, if includeInactive=true the results would include the permission specified.

`GET /perms/users/<id>/permissions` (and similar `GET /perms/users/<id>`)

- Query the DB for permission users (assignments)
- Query the DB for all inactive permissions (or consult and use a cached copy if one is available)
- Filter out the inactive permissions from the results
- Update the totalRecord count in the results
- Return the adjusted results
- Add a new query argument - includeInactive
    - Boolean
    - Default: false
    - Indicates whether inactive permissions should be returned or not (including in subPermissions/childOf arrays)

`GET /perms/permissions/<id>`

Leave that as-is.

Notes

Several ideas have been briefly discussed for extending this API's functionality

- Allow finer granularity control, e.g. specify which modules to purge the inactive permissions for
- Introduce a way to purge permissions which have been inactive for longer than a specified amount of time.

Another idea which was discussed (and considered an implementation detail) was to store inactive permissions off to the side, e.g. in a separate table. The idea being that it might help simplify/minimize the amount of filtering needed, specifically we wouldn't need to add clauses to provided CQL queries to omit inactive permissions. We would however still need to filter out permissions from the subPermissions/childOf fields.

Schema Changes

Permission.json Schema

See https://github.com/folio-org/mod-permissions/blob/master/ramls/permission.json

This is the stored permission schema. It needs to be expanded to include some additional context about who/what defined it.

- `inactive`
    - Boolean
    - Optional
    - Default: `false`
    - Indicates whether this permissions has been marked for deletion (soft deleted)

API Changes

A new API will be introduced for removing inactive permissions

Interface: `permissions`

Endpoint: `POST /perms/permissions/purge-inactive`

Request: No body

Responses:

- 200 - application/json with the following fields:
    - `"removed"`: array of permission names which were removed - may be empty
    - `"totalRemoved"`: integer indicating the number of permissions that were removed, may be zero if there are no inactive permissions
- 500 - text/plain with an appropriate message

Required Permissions: `perms.permissions.purge-inactive.post`

Behavior:

- Start transaction
- Query the permissions table for all permissions where `inactive == true`
- For each
    - Query the perm-users table for all records w/ permissions contains the inactive permission
    - Remove the inactive permission assignment and update the record
- Delete from the permissions table where `inactive == true`
- End transaction
- Form a response containing the list of removed permission's names and the total count of removed permissions
- Upon error, rollback transaction and return a 500 and plain text error w/ appropriate message

NOTE: A call to `GET /perms/permissions?query=inactive==true` could be used to retrieve the list of permissions that will be purged prior to using this new endpoint.

Permission Name Conflict Resolution

OKAPI-952 - Getting issue details... STATUS explores options for handling conflicting permission names. Three potential solutions were considered:

✗ Fail the install/upgrade

This it probably the safest thing to do, but is undesirable given separate conversations about reducing the number of things that would cause an upgrade to completely stop.

✗ Overwrite the existing permission

While this is the current behavior, it constitutes a security vulnerability in that it could lead to permission escalation. For this reason, this solution is essentially off the table.

✓ Rename the existing user-defined permission

For example by adding a numeric suffix. Permissions defined by the system (modules) take precedence.

1. Renaming a user-defined permission and replacing it with a system-defined permission could get confusing.
2. This might be the least disruptive solution being considered.

✗ Prefix/scope the permissions

Here, permissions would be scoped. Module-defined permissions would get a prefix of something like "system", "static" or "module", whereas permissions defined via the perms API would be scoped with something different, such as "custom", "dynamic" or "user-defined".

The main concern with this solution is that it's disruptive and more complicated than the other solutions.

- If we implement this to be implicit, that is module descriptors still define permissions like: inventory-storage.items.collection.get, mod-permissions would add the scope for you.
    - Presumably the other mod-permissions APIs would need to be updated as well to support searching for permissions w/o having to specify the scope. e.g. ?query=permissionName=inventory-storage.items.collection.get.
    - Assuming we migrate to scoped permissions when upgrading to a new version of mod-permissions, the tenant API would need to be extended to add the scope to existing permissions, and to remove the scope upon downgrade.
    - The question then becomes, how does mod-permissions determine the appropriate scope for each of the permissions.
    - I'm not sure if this is a big deal or not, but the UUIDs of the permissions would change. If system operators aren't expecting this it may lead to problems.
    - This approach is much more work compared to other solutions.
    - There will always be this disconnect between what's defined in the system and what's actually in the mod-permissions database. This is confusing.
- If we make this explicit, that is force module developers to add the scope to their permissions in module descriptors, and in search queries, etc.
    - This is clearly much more disruptive in the sense that module developers need to do something (or many things in some cases) in order to keep their modules working as desired/expected.
    - Another problem with this is that you'd need to take precautions to avoid a similar permission escalation vulnerability... i.e. user's shouldn't be able to specify the scope, or at least the scope needs to be validated.

NOTE: There are currently no restrictions on display name uniqueness... So this means that multiple permissions could have the same "display name", but behind the scenes one might be defined by a module and another might be user-defined.

End-to-End Examples

Example 1 - New, modified, and removed permissions

Given the following permissions:

```
{
  "permissionName": "foo",
  "subPermissions": [ ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}, {
  "permissionName": "bar",
  "subPermissions": [ "bar.get", "bar.post", "bar.delete" ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}, {
  "permissionName": "baz",
  "subPermissions": [ ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}
```

And following user:

```
{
  "id": "c183b277-9d16-4687-b22a-2f181529d018",
  "username": "bob",
  "permissions": [ "foo", "bar", "baz", "bar.get", "bar.post", "bar.delete" ]
  ...
}
```

If _tenantPermissions was called with:

```
{
  "ModuleId": "mod-foo-2.0.0",
  "fromModuleId": "mod-foo-1.2.3",
  "toPerms": [
    {
      "permissionName": "zip",
      ...
    }, {
      "permissionName": "zap",
      "subPermissions": [ "zap.get", "zap.post", "zap.delete" ],
      ...
    }, {
      "permissionName": "foo.config",
      "renamedFrom": [ "foo" ],
      ...
    }, {
      "permissionName": "bar",
      "subPermissions": [ "bar.get", "bar.put", "bar.post", "bar.delete" ],
      ...
    }
  ],
  "fromPerms": [
    {
      "permissionName": "foo",
      ...
    }, {
      "permissionName": "bar",
      "subPermissions": [ "bar.get", "bar.post", "bar.delete" ],
      ...
    }, {
      "permissionName": "baz",
      ...
    }
  ]
}
```

We'd end up with:

```
{
  "permissionName": "foo.config",
  "subPermissions": [ ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}, {
  "permissionName": "bar",
  "subPermissions": [ "bar.get", "bar.put", "bar.post", "bar.delete" ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}, {
  "permissionName": "zip",
  "subPermissions": [ ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}, {
  "permissionName": "zap",
  "subPermissions": [ "zap.get", "zap.post", "zap.delete" ],
  "grantedTo": [ "c183b277-9d16-4687-b22a-2f181529d018" ],
  ...
}
```

And

```
{
  "id": "c183b277-9d16-4687-b22a-2f181529d018",
  "username": "bob",
  "permissions": [ "foo.config", "bar", "zip", "zap", "bar.get", "bar.put", "bar.post", "bar.delete", "zap.
get", "zap.post", "zap.delete" ]
  ...
}
```

Example 2 - Sub-permission provided multiple times

Given the following permissions:

```
[ {
  "permissionName": "a",
  "subPermissions": [ "x" ]
}, {
  "permissionName": "b",
  "subPermissions": [ "x" ]
} ]
```

And following user:

```
{
  "username": "foo",
  "permissions": [ "a", "b", "x" ]
}
```

When "b" changes from "x" to "y"

```
[ {
  "permissionName": "a",
  "subPermissions": [ "x" ]
}, {
  "permissionName": "b",
  "subPermissions": [ "y" ]
} ]
```

take care that we don't delete "x" that is still a provided via "a":

```
{
  "username": "foo",
  "permissions": [ "a", "b", "x", "y" ]
}
```

There should be a unit test for this case.

Open Issues

- RESOLOVED How should we handle collisions (i.e. A user-defined permission exists in the system when a module renames or adds a permission with the same name)?
    - See OKAPI-952 - Getting issue details... STATUS
    - See "Permission Name Conflict Resolution" for additional details.
    - ✗ Fail the install/upgrade w/ an appropriate message.
    - ✗ Overwrite the existing permission. Not ideal as this could lead to privilege escalation, and confusion.
        - This is the current behavior... if a permission with the same name as one sent to the _tenantPermissions API, it is overwritten.
    - ✗ One idea being discussed is to prefix/scope permissions with the module name, and disallow users from creating permissions which follow this pattern.

        - Maybe mod-permissions automatically adds the scope based on the API being used; e.g. permissions defined via _tenantPermissions would be scoped with something like "system", "static" or "module", whereas permissions defined via the perms API would be scoped with something different, such as "custom", "dynamic" or "user-defined".
        - NOTE: There are currently no restrictions on display name uniqueness... So this means that multiple permissions could have the same "display name", but behind the scenes one might be defined by a module and another might be user-defined.
    - ✓ Another idea is to automatically rename the existing permission, e.g. if a user-defined permisison named foo.get exists, and a module defines one with the same name, the user-defined permission is renamed to foo.get.1
- RESOLVED The PoC uses the same okapiPermission.json schema for the three arrays (newPermissions, modifiedPermissions, removedPermissions). The current behavior of RMB is to generate one java class for this, named after the first field which references it (newPermissions). This gets confusing when you're populating the a "RemovedPermissions" array with "NewPermission" objects. One sub-optimal workaround is to duplicate the okapiPermission.json schema so RMB generates three distinct pojos w/ more appriopriate names. There's probably a better way to deal with this, but I'm not aware of it.
    - https://github.com/folio-org/raml-module-builder#step-6-design-the-raml-files : Use `"javaType": "org.folio.rest.jaxrs.model.MyEntity"` to set the class name of the generated Java type to prevent a duplicate name where the second class overwrites the first class or to avoid a misleading name. Otherwise the field name in the .json schema file is used as class name.
- RESOLVED How are module downgrades handled?
    - See OKAPI-953 - Getting issue details... STATUS
    - One challenge here is what happens when a module upgrade removes or renames a permission which had was assigned to users. If the upgrade process removes those permissions from the perm user record, we won't be able re-assign those permissions to the various users upon downgrading to the previous version of the module.
    - The prevailing solution ATM is to use a soft delete. There are a couple ways we could do this:
        - ✗ 1) Add a new field to the user permission record - "removedPermissions" - a list of permissions which the user once had, but are now marked for removal. As long as this list isn't purged, it should be possible to back out the user perm changes upon module downgrade.
            - It occurred to me that this "removedPermissions" field would need to be protected somehow - that is a user shouldn't be able to modify it, as that could be another path to abuse and permission escalation. Since this information is solely for system/internal use, it might be as simple as adding a new column to the perm users table, which isn't accessible to API users.... In other words only this can only be changed by the _tenantPermissions API during modules upgrades /downgrades.
        - ✓ 2) Instead of removing the permissions, just mark them as inactive/deleted. The user-perms records (assignments) will not be touched in this case. A separate mechanism for cleaning up inactive/deleted permissions could then be introduced and manually invoked at some later time once the system operators are confident a module downgrade will never need to happen.
            - Would we want to limit this to only system defined permissions, or would we change the behavior of DELETE /perms /permissions/<id>?
                - The team decided to keep this limited to _tenantPermissions, at least for now.
    - Julian Ladisch proposed another solution in the comments
- RESOLVED Is it enough to just add `RenamedFrom` in MD or should we also consider adding the actual version change info when the rename happens? For example `fromVersion=1.1.1, toVersion=1.12` More precise metadata like this gives us more control how to handle perm changes, and flexibility to reuse an old perm name in later versions (if need to)
    - It was determined that this is not needed.
- RESOLVED Who should determine which permissions are new, updated, deleted, OKAPI or mod-permissions? From Adam Dickmeiss 's comments below.
    - ✗ 1) OKAPI compares modules descriptors - current version and new version being enabled, then tells mod-permissions which perms are new, modified and removed.
        - This is how the PoC works. However, in the case where a module is enabled, disabled, and then some time later, a newer version of that module is enabled. Here mod-permissions would (likely) still have the original permissions in the database, but from OKAPI's perspective it's the first time this module has been enabled for the tenant. This means it would send everything in the "newPermissions" list. Given how this is currently implemented, it's not a problem for new and updated permissions (newPermissions is essentially handled as an upsert). However, permissions removed in the new version of the module will remain in the database (and in theory could still be assigned) since OKAPI will not send these in the removedPermissions list. I think this is minor but worth noting. We may be able to detect and remove these "zombie" permissions as part of the "purge-inactive" API described above.
        - Idea: When mod-permissions is upgraded, OKAPI refreshes all the existing module permissions... this would result in the permissions getting the module context. Mod-permissions would need to match the permissions (by name, and other criteria... they should match exactly).
    - ✗ 2) Mod-permissions is responsible for looking at the database and determining which permissions are new, modified, or removed.
        - The catch here is that there will be a transition period where mod-permissions knows only about permissions, it doesn't have the moduleId context. This will make it difficult to determine which perms are new/modified/removed. One idea is to have OKAPI also provide not only the moduleId (the id of the new module version), but also fromModuleId, the id of the current module version. This would allow mod-permissions to ask OKAPI for the module descriptor and use that to help determine which which perms are new/modified/removed.
        - IMO it feels a little wrong to force mod-permissions to understand module descriptors, and to a lesser extent to have a dependency on an OKAPI interface. It should only need to know about permissions, possibly with some context about which module the permission was defined in.
    - ✓ 3) OKAPI sends both from and to permission lists. See "Determination of which permissions are new/updated/removed"

Decisions

Status	DONE
Stakeholders	Jakub Skoczen Julian Ladisch Adam Dickmeiss Hongwei Ji Mikhail Fokanov
Outcome	Static permission removal will use a soft delete to accommodate module downgrades

Created date	02 Dec 2020
Owner	Craig McNally