Red Hat Plug-ins for Backstage 2.0

Keycloak backend plugin for Backstage

The Keycloak backend plugin for Backstage
The Keycloak backend plugin for Backstage
Abstract

The Keycloak backend plugin integrates Keycloak into Backstage.
# Table of Contents

CHAPTER 1. KEYCLOAK BACKEND PLUGIN FOR BACKSTAGE .............................................. 3

1.1. CAPABILITIES ................................................................................. 3

1.2. FOR ADMINISTRATORS ................................................................ 3
   1.2.1. Installation ........................................................................ 3
   1.2.2. Configuration .................................................................. 3
   1.2.3. Limitations ...................................................................... 6

1.3. FOR USERS ................................................................................... 7
   1.3.1. Imported users and groups in Backstage using Keycloak plugin 7
CHAPTER 1. KEYCLOAK BACKEND PLUGIN FOR BACKSTAGE

The Keycloak backend plugin integrates Keycloak into Backstage.

1.1. CAPABILITIES

The Keycloak backend plugin has the following capabilities:

- Synchronization of Keycloak users in a realm
- Synchronization of Keycloak groups and their users in a realm

1.2. FOR ADMINISTRATORS

1.2.1. Installation

The Red Hat Plug-ins for Backstage (RHPIB) packages are hosted in a separate NPM registry, which is maintained by Red Hat. To use these packages, you must adjust your NPM configuration to pull the @redhat scoped packages:

```bash
# update your .npmrc or .yarnrc file
yarn config set "@redhat:registry" https://npm.registry.redhat.com
# then pull a package
yarn add @redhat/backstage-plugin-quay
```

For more information, see npm docs.

Creating a .npmrc file ensures that all the packages are scoped under @redhat and are fetched from Red Hat's NPM registry, while the rest dependencies remain sourced from other registry.

You can now install the Backstage package into the backend. When not integrating with a published package, clone the repository locally and add Backstage as follows:

```bash
yarn workspace backend add @redhat/backstage-plugin-keycloak-backend
```

1.2.2. Configuration

1. Add the following configuration to the app-config.yaml file:

```yaml
```
```
```
```
```
```
```
```
```yaml
```
2. Register the plugin in the `packages/backend/src/plugins/catalog.ts` file. You can also configure a schedule in this step. However, there are possible ways of configuration, such as:

- Configure a schedule inside the `app-config.yaml` file:

  ```yaml
  title="app-config.yaml"
  catalog:
    providers:
      keycloakOrg:
        default:
          # ...
          # highlight-add-start
          schedule: # optional; same options as in TaskScheduleDefinition
          # supports cron, ISO duration, "human duration" as used in code
          frequency: { minutes: 1 }
          # supports ISO duration, "human duration" as used in code
          timeout: { minutes: 1 }
          initialDelay: { seconds: 15 }
          # highlight-add-end
  ...
  ```

  Use the configured scheduler inside the `packages/backend/src/plugins/catalog.ts` as follows:

  ```ts
  title="packages/backend/src/plugins/catalog.ts"
  /* highlight-add-start */
  import { KeycloakOrgEntityProvider } from '@redhat/backstage-plugin-keycloak-backend';

  /* highlight-add-end */

  export default async function createPlugin(
    env: PluginEnvironment,
  ): Promise<Router> {
    const builder = await CatalogBuilder.create(env);

    /* ... other processors and/or providers ... */
    /* highlight-add-start */
    builder.addEntityProvider(
      KeycloakOrgEntityProvider.fromConfig(env.config, {
        id: 'development',
        logger: env.logger,
        scheduler: env.scheduler,
      }),
    );
    /* highlight-add-end */

    const { processingEngine, router } = await builder.build();
    await processingEngine.start();
    return router;
  }
  
  /* highlight-add-start */
  ```

- Add a schedule directly inside the `packages/backend/src/plugins/catalog.ts` file as follows:
```
+ import { KeycloakOrgEntityProvider } from '@redhat/backstage-plugin-keycloak-backend';

export default async function createPlugin(
  env: PluginEnvironment,
): Promise<Router> {
  const builder = await CatalogBuilder.create(env);

  /* ... other processors and/or providers ... */
  builder.addEntityProvider(
    KeycloakOrgEntityProvider.fromConfig(env.config, {
      id: 'development',
      logger: env.logger,
      /* highlight-add-start */
      schedule: env.scheduler.createScheduledTaskRunner({
        frequency: { minutes: 1 },
        timeout: { minutes: 1 },
        initialDelay: { seconds: 15 }
      }),
      /* highlight-add-end */
    }));
  
  const { processingEngine, router } = await builder.build();
  await processingEngine.start();
  return router;
```

3. Optional: override the default Keycloak query parameters. Configure the parameters inside the
   app-config.yaml file:

```
---
yaml title="app-config.yaml"

catalog:
  providers:
    keycloakOrg:
      default:
        # ...
        # highlight-add-start
        userQuerySize: 500 # Optional
        groupQuerySize: 250 # Optional
        # highlight-add-end
```

Communication between Backstage and Keycloak is enabled by using the Keycloak API. Username/password or client credentials are supported authentication methods.

The following table describes the parameters that you can configure to enable the plugin under
   catalog.providers.keycloakOrg.<ENVIRONMENT_NAME> object in the app-config.yaml file:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>userQuerySize</td>
<td></td>
<td>500</td>
<td>Optional</td>
</tr>
<tr>
<td>groupQuerySize</td>
<td></td>
<td>250</td>
<td>Optional</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Default Value</td>
<td>Required</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>baseUrl</strong></td>
<td>Location of the Keycloak server, such as <a href="https://localhost:8443/auth">https://localhost:8443/auth</a>. Note that the newer versions of Keycloak omit the /auth context path.</td>
<td>&quot;&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>realm</strong></td>
<td>Realm to synchronize</td>
<td>master</td>
<td>No</td>
</tr>
<tr>
<td><strong>loginRealm</strong></td>
<td>Realm used to authenticate</td>
<td>master</td>
<td>No</td>
</tr>
<tr>
<td><strong>username</strong></td>
<td>Username to authenticate</td>
<td>&quot;&quot;</td>
<td>Yes if using password based authentication</td>
</tr>
<tr>
<td><strong>password</strong></td>
<td>Password to authenticate</td>
<td>&quot;&quot;</td>
<td>Yes if using password based authentication</td>
</tr>
<tr>
<td><strong>clientId</strong></td>
<td>Client ID to authenticate</td>
<td>&quot;&quot;</td>
<td>Yes if using client credentials based authentication</td>
</tr>
<tr>
<td><strong>clientSecret</strong></td>
<td>Client Secret to authenticate</td>
<td>&quot;&quot;</td>
<td>Yes if using client credentials based authentication</td>
</tr>
<tr>
<td><strong>userQuerySize</strong></td>
<td>Number of users to query at a time</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td><strong>groupQuerySize</strong></td>
<td>Number of groups to query at a time</td>
<td>100</td>
<td>No</td>
</tr>
</tbody>
</table>

When using client credentials, the access type must be set to confidential and service accounts must be enabled. You must also add the following roles from the realm-management client role:

- query-groups
- query-users
- view-users

1.2.3. Limitations

If you have self-signed or corporate certificate issues, you can set the following environment variable before starting Backstage:
NODE_TLS_REJECT_UNAUTHORIZED=0

NOTE
The solution of setting the environment variable is not recommended.

1.3. FOR USERS

1.3.1. Imported users and groups in Backstage using Keycloak plugin

After configuring the plugin successfully, the plugin imports the users and groups each time when started.

After the first import is complete, you can select User to list the users from the catalog page:

You can see the list of users on the page:
When you select a user, you can see the information imported from Keycloak:

You can also select a group, view the list, and select or view the information imported from Keycloak for a group: