Red Hat Developer Hub 1.8

Authorization in Red Hat Developer Hub

Configuring authorization by using role based access control (RBAC) in Red Hat Developer Hub

Last Updated: 2026-01-22

Red Hat Developer Hub 1.8 Authorization in Red Hat Developer Hub

Configuring authorization by using role based access control (RBAC) in Red Hat Developer Hub

Legal Notice

Abstract

Red Hat Developer Hub (RHDH) administrators can use role-based access control (RBAC) to manage authorizations of other users.

Table of Contents

PREFACE

Administrators can authorize users to perform actions and define what users can do in Developer Hub.

Role-based access control (RBAC) is a security concept that defines how to control access to resources in a system by specifying a mapping between users of the system and the actions that those users can perform on resources in the system. You can use RBAC to define roles with specific permissions and then assign the roles to users and groups.

RBAC on Developer Hub is built on top of the Permissions framework, which defines RBAC policies in code. Rather than defining policies in code, you can use the Developer Hub RBAC feature to define policies in a declarative fashion by using a simple CSV based format. You can define the policies by using Developer Hub web interface or REST API instead of editing the CSV directly.

An administrator can define authorizations in Developer Hub by taking the following steps:

1. Enable the RBAC feature and give authorized users access to the feature.
2. Define roles and policies by combining the following methods:
    - The Developer Hub policy administrator uses the Developer Hub web interface or REST API.
    - The Developer Hub administrator edits the main Developer Hub configuration file.
    - The Developer Hub administrator edits external files.

CHAPTER 1. ENABLING AND GIVING ACCESS TO THE ROLE-BASED ACCESS CONTROL (RBAC) FEATURE

The Role-Based Access Control (RBAC) feature is disabled by default. Enable the RBAC plugin and declare policy administrators to start using RBAC features.

The permission policies for users and groups in the Developer Hub are managed by permission policy administrators. Only permission policy administrators can access the Role-Based Access Control REST API.

Prerequisites

- You have added a custom Developer Hub application configuration , and have necessary permissions to modify it.
- You have enabled an authentication provider.

Procedure

1. The RBAC plugin is installed but disabled by default. To enable the **./dynamic-plugins/dist/backstage-community-plugin-rbac** plugin, edit your **dynamic-plugins.yaml** with the following content.

   dynamic-plugins.yaml fragment

   ```
   plugins:
     - package: ./dynamic-plugins/dist/backstage-community-plugin-rbac
       disabled: false
   ```

   See Installing and viewing plugins in Red Hat Developer Hub .

2. Declare policy administrators to enable a select number of authenticated users to configure RBAC policies through the REST API or Web UI, instead of modifying the CSV file directly. The permissions can be specified in a separate CSV file referenced in your **my-rhdh-app-config** config map, or permissions can be created using the REST API or Web UI.
   To declare users such as *<your_policy_administrator_name>* as policy administrators, edit your custom Developer Hub ConfigMap, such as **app-config-rhdh**, and add following code to the **app-config.yaml** content:

   app-config.yaml fragment

   ```
   permission:
     enabled: true
     rbac:
       admin:
         users:
           - name: user:default/<your_policy_administrator_name>
   ```

3. In order to display the available permissions provided by installed plugins in the Developer Hub UI, you must supply the corresponding list of plugin IDs. There are two ways to do this, by updating your application configuration or by using the RBAC REST API permissions endpoint.

a. To provide plugins by updating your application configuration, you can specify the plugins with permissions in your **app-config.yaml** file as follows:

app-config.yaml fragment

```
permission:
  enabled: true
  rbac:
    admin:
      users:
        - name: user:default/<your_policy_administrator_name>
    pluginsWithPermission:
      - catalog
      - scaffolder
      - permission
```

b. To specify the plugins with permissions by using the RBAC REST API permissions endpoint, see the RBAC REST API permissions endpoint .

Verification

1. Sign out from the existing Red Hat Developer Hub session and log in again using the declared policy administrator account.
2. With RBAC enabled, most features are disabled by default.
   a. Navigate to the **Catalog** page in RHDH. The **Create** button is not visible. You cannot create new components.
   b. Navigate to the API page. The **Register** button is not visible.

Next steps

- Explicitly enable permissions to resources in Developer Hub.

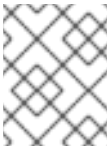
CHAPTER 2. DETERMINING PERMISSION POLICY AND ROLE CONFIGURATION SOURCE

You can configure Red Hat Developer Hub policy and roles by using different sources. To maintain data consistency, Developer Hub associates each permission policy and role with one unique source. You can only use this source to change the resource.

The available sources are:

Configuration file

Configure roles and policies in the Developer Hub **app-config.yaml** configuration file, for instance to declare your policy administrators .
The Configuration file pertains to the default **role:default/rbac_admin** role provided by the RBAC plugin. The default role has limited permissions to create, read, update, delete permission policies or roles, and to read catalog entities.



NOTE

In case the default permissions are insufficient for your administrative requirements, you can create a custom admin role with the required permission policies.

REST API

Configure roles and policies by using the Developer Hub Web UI or by using the REST API.

CSV file

Configure roles and policies by using external CSV files.

Legacy

The legacy source applies to policies and roles defined before RBAC backend plugin version **2.1.3**, and is the least restrictive among the source location options.

IMPORTANT

Replace the permissions and roles using the legacy source with the permissions using the REST API or the CSV file sources.

Procedure

- To determine the source of a role or policy, use a **GET** request.

CHAPTER 3. POLICY EFFECT

The policy effect determines whether the access request must be approved if multiple policy rules match the request. In Red Hat Developer Hub, when one rule permits and the other denies, the deny rule takes precedence, and the policy effect is to deny.

CHAPTER 4. MANAGING ROLE-BASED ACCESS CONTROLS (RBAC) USING THE RED HAT DEVELOPER HUB WEB UI

Policy administrators can use the Developer Hub web interface (Web UI) to allocate specific roles and permissions to individual users or groups. Allocating roles ensures that access to resources and functionalities is regulated across the Developer Hub.

With the policy administrator role in Developer Hub, you can assign permissions to users and groups. This role allows you to view, create, modify, and delete the roles using Developer Hub Web UI.

4.1. CREATING A ROLE IN THE RED HAT DEVELOPER HUB WEB UI

You can create a role in the Red Hat Developer Hub using the Web UI.

Prerequisites

- You have enabled RBAC, have a policy administrator role in Developer Hub, and have added plugins with permission.

Procedure

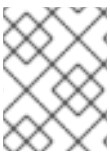
1. Go to **Administration** at the bottom of the sidebar in the Developer Hub.
   The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Click **CREATE** to create a role.
4. Enter the name and description of the role in the given fields and click **NEXT**.
5. Add users and groups using the search field, and click **NEXT**.
6. Select **Plugin** and **Permission** from the drop-downs in the **Add permission policies** section.
7. Select or clear the **Policy** that you want to set in the **Add permission policies** section, and click **NEXT**.
8. Review the added information in the **Review and create** section.
9. Click **CREATE**.

Verification

The created role appears in the list available in the **RBAC** tab.

4.2. EDITING A ROLE IN THE RED HAT DEVELOPER HUB WEB UI

You can edit a role in the Red Hat Developer Hub using the Web UI.



NOTE

The policies generated from a **policy.csv** or ConfigMap file cannot be edited or deleted using the Developer Hub Web UI.

Prerequisites

- You have enabled RBAC, have a policy administrator role in Developer Hub, and have added plugins with permission.
- The role that you want to edit is created in the Developer Hub.

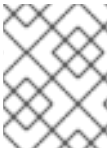
Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub.
   The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Select the edit icon for the role that you want to edit.
4. Edit the details of the role, such as name, description, users and groups, and permission policies, and click **NEXT**.
5. Review the edited details of the role and click **SAVE**.

After editing a role, you can view the edited details of a role on the **OVERVIEW** page of a role. You can also edit a role's users and groups or permissions by using the edit icon on the respective cards on the **OVERVIEW** page.

4.3. DELETING A ROLE IN THE RED HAT DEVELOPER HUB WEB UI

You can delete a role in the Red Hat Developer Hub using the Web UI.



NOTE

The policies generated from a **policy.csv** or ConfigMap file cannot be edited or deleted using the Developer Hub Web UI.

Prerequisites

- You have enabled RBAC and have a policy administrator role in Developer Hub .
- The role that you want to delete is created in the Developer Hub.

Procedure

1. Go to **Administration** at the bottom of the sidebar in the Developer Hub.
   The **RBAC** tab appears, displaying all the created roles in the Developer Hub.
2. (Optional) Click any role to view the role information on the **OVERVIEW** page.
3. Select the delete icon from the **Actions** column for the role that you want to delete.
   Delete this role? pop-up appears on the screen.
4. Click **DELETE**.

CHAPTER 5. MANAGING AUTHORIZATIONS BY USING THE REST API

To automate the maintenance of Red Hat Developer Hub permission policies and roles, you can use Developer Hub role-based access control (RBAC) REST API.

You can perform the following actions with the REST API:

- Retrieve information about:
  - All permission policies
  - Specific permission policies
  - Specific roles
  - Static plugins permission policies
- Create, update, or delete:
  - Permission policy
  - Role

5.1. SENDING REQUESTS TO THE RBAC REST API BY USING THE CURL UTILITY

You can send RBAC REST API requests by using the curl utility.

Prerequisites

- You have access to the RBAC feature .

Procedure

1. Find your Bearer token to authenticate to the REST API.
   a. In your browser, open the web console **Network** tab.
   b. In the main screen, reload the Developer Hub **Homepage**.
   c. In the web console **Network** tab, search for the **query?term=** network call.
   d. Save the **token** in the response JSON for the next steps.
2. In a terminal, run the curl command and review the response:

   GET or DELETE request

   ```
   $ curl -v \
     -H "Authorization: Bearer <token>" \
     -X <method> "https://<my_developer_hub_domain>/<endpoint>" \
   ```

   POST or PUT request requiring JSON body data

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer <token>" \
  -X POST "https://<my_developer_hub_domain>/<endpoint>" \
  -d <body>
```

<token>

Enter your saved authorization token.

<method>

Enter the HTTP method for your API endpoint.

- **GET**: To retrieve specified information from a specified resource endpoint.
- **POST**: To create or update a resource.
- **PUT**: To update a resource.
- **DELETE**: To delete a resource.

https://*<my_developer_hub_domain>*

Enter your Developer Hub URL.

<endpoint>

Enter the API endpoint to which you want to send a request, such as **/api/permission/policies**.

<body>

Enter the JSON body with data that your API endpoint might need with the HTTP **POST** or **PUT** request.

Example request to create a role

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer <token>" \
  -X POST "https://<my_developer_hub_domain>/api/permission/roles" \
  -d '{
     "memberReferences": ["group:default/example"],
     "name": "role:default/test",
     "metadata": { "description": "This is a test role" }
   }'
```

Example request to update a role

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer <token>" \
  -X PUT "https://<my_developer_hub_domain>/api/permission/roles/role/default/test" \
  -d '{
        "oldRole": {
          "memberReferences":  [ "group:default/example" ],
          "name": "role:default/test"
        },
        "newRole": {
          "memberReferences": [ "group:default/example", "user:default/test" ],
          "name": "role:default/test"
        }
```

```
        }'
```

Example request to create a permission policy

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer $token" \
  -X POST "https://<my_developer_hub_domain>/api/permission/policies" \
  -d '[{
     "entityReference":"role:default/test",
     "permission": "catalog-entity",
     "policy": "read", "effect":"allow"
   }]'
```

Example request to update a permission policy

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer $token" \
  -X PUT "https://<my_developer_hub_domain>/api/permission/policies/role/default/test" \
  -d '{
        "oldPolicy": [
          {
            "permission": "catalog-entity", "policy": "read", "effect": "allow"
          }
        ],
        "newPolicy":
          [
            {
              "permission": "policy-entity", "policy": "read", "effect": "allow"
            }
          ]
      }'
```

Example request to create a condition

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer $token" \
  -X POST "https://<my_developer_hub_domain>/api/permission/roles/conditions" \
  -d '{
     "result": "CONDITIONAL",
     "roleEntityRef": "role:default/test",
     "pluginId": "catalog",
     "resourceType": "catalog-entity",
     "permissionMapping": ["read"],
     "conditions": {
       "rule": "IS_ENTITY_OWNER",
       "resourceType": "catalog-entity",
       "params": {"claims": ["group:default/janus-authors"]}
     }
   }'
```

Example request to update a condition

```
$ curl -v -H "Content-Type: application/json" \
  -H "Authorization: Bearer $token" \
```

```
-X PUT "https://<my_developer_hub_domain>/api/permission/roles/conditions/1" \
-d '{
      "result":"CONDITIONAL",
      "roleEntityRef":"role:default/test",
      "pluginId":"catalog",
      "resourceType":"catalog-entity",
      "permissionMapping": ["read",  "update", "delete"],
      "conditions": {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {"claims": ["group:default/janus-authors"]}
      }
    }'
```

Verification

- Review the returned HTTP status code:

  200 OK

  The request was successful.

  201 Created

  The request resulted in a new resource being successfully created.

  204 No Content

  The request was successful, and the response payload has no more content.

  400 Bad Request

  Input error with the request.

  401 Unauthorized

  Lacks valid authentication for the requested resource.

  403 Forbidden

  Refusal to authorize request.

  404 Not Found

  Could not find requested resource.

  409 Conflict

  Request conflict with the current state and the target resource.

5.2. SENDING REQUESTS TO THE RBAC REST API BY USING A REST CLIENT

You can send RBAC REST API requests using any REST client.

Prerequisites

- You have access to the RBAC feature .

Procedure

1. Find your Bearer token to authenticate to the REST API.
   a. In your browser, open the web console **Network** tab.

b. In the main screen, reload the Developer Hub **Homepage**.

c. In the web console **Network** tab, search for the **query?term=** network call.

d. Save the **token** in the response JSON for the next steps.

2. In your REST client, run a command with the following parameters and review the response:

Authorization

Enter your saved authorization token.

HTTP method

Enter the HTTP method for your API endpoint.

- **GET**: To retrieve specified information from a specified resource endpoint.
- **POST**: To create or update a resource.
- **PUT**: To update a resource.
- **DELETE**: To delete a resource.

URL

Enter your Developer Hub URL and API endpoint: https://*<my_developer_hub_domain>*/*<endpoint>*, such as **https://*<my_developer_hub_domain>*/api/permission/policies**.

Body

Enter the JSON body with data that your API endpoint might need with the HTTP **POST** request.

5.3. SUPPORTED RBAC REST API ENDPOINTS

The RBAC REST API provides endpoints for managing roles, permissions, and conditional policies in the Developer Hub and for retrieving information about the roles and policies.

5.3.1. Roles

The RBAC REST API supports the following endpoints for managing roles in the Red Hat Developer Hub.

[GET] /api/permission/roles

Returns all roles in Developer Hub.

Example response (JSON)

```
[
  {
    "memberReferences": ["user:default/username"],
    "name": "role:default/guests"
  },
  {
    "memberReferences": [
      "group:default/groupname",
      "user:default/username"
```

```
    ],
    "name": "role:default/rbac_admin"
  }
]
```

[GET] /api/permission/roles/*<kind>*/*<namespace>*/*<name>*

Returns information for a single role in Developer Hub.

Example response (JSON)

```
[
  {
    "memberReferences": [
      "group:default/groupname",
      "user:default/username"
    ],
    "name": "role:default/rbac_admin"
  }
]
```

[POST] /api/permission/roles/*<kind>*/*<namespace>*/*<name>*

Creates a role in Developer Hub.

Table 5.1. Request parameters

Name	Description	Type	Presence
body	The **memberReferences**, **group**, **namespace**, and **name** the new role to be created.	Request body	Required

Example request body (JSON)

```
{
  "memberReferences": ["group:default/test"],
  "name": "role:default/test_admin"
}
```

Example response

```
201 Created
```

[PUT] /api/permission/roles/*<kind>*/*<namespace>*/*<name>*

Updates **memberReferences**, **kind**, **namespace**, or **name** for a role in Developer Hub.

Request parameters

The request body contains the **oldRole** and **newRole** objects:

Name	Description	Type	Presence
body	The **memberReferences**, **group**, **namespace**, and **name** the new role to be created.	Request body	Required

Example request body (JSON)

```
{
  "oldRole": {
    "memberReferences": ["group:default/test"],
    "name": "role:default/test_admin"
  },
  "newRole": {
    "memberReferences": ["group:default/test", "user:default/test2"],
    "name": "role:default/test_admin"
  }
}
```

Example response

```
200 OK
```

[DELETE] /api/permission/roles/*<kind>*/*<namespace>*/*<name>*?memberReferences=<VALUE>

Deletes the specified user or group from a role in Developer Hub.

Table 5.2. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name of the entity	String	Required
memberReferences	Associated group information	String	Required

Example response

```
204
```

[DELETE] /api/permission/roles/*<kind>*/*<namespace>*/*<name>*

Deletes a specified role from Developer Hub.

Table 5.3. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name of the entity	String	Required

Example response

```
204
```

5.3.2. Permission policies

The RBAC REST API supports the following endpoints for managing permission policies in the Red Hat Developer Hub.

[GET] /api/permission/policies

Returns permission policies list for all users.

Example response (JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
]
```

[GET] /api/permission/policies/*<kind>*/*<namespace>*/*<name>*

Returns permission policies related to the specified entity reference.

Table 5.4. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name related to the entity	String	Required

Example response (JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  },
  {
    "entityReference": "role:default/test",
    "permission": "catalog.entity.create",
    "policy": "use",
    "effect": "allow",
    "metadata": {
      "source": "csv-file"
    }
  }
]
```

[POST] /api/permission/policies

Creates a permission policy for a specified entity.

Table 5.5. Request parameters

Name	Description	Type	Presence
entityReference	Reference values of an entity including **kind**, **namespace**, and **name**	String	Required
permission	Permission from a specific plugin, resource type, or name	String	Required
policy	Policy action for the permission, such as **create**, **read**, **update**, **delete**, or **use**	String	Required

Name	Description	Type	Presence
effect	Indication of allowing or not allowing the policy	String	Required

Example request body (JSON)

```
[
  {
    "entityReference": "role:default/test",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow"
  }
]
```

Example response

```
201 Created
```

[PUT] /api/permission/policies/*<kind>*/*<namespace>*/*<name>*

Updates a permission policy for a specified entity.

Request parameters

The request body contains the **oldPolicy** and **newPolicy** objects:

Name	Description	Type	Presence
permission	Permission from a specific plugin, resource type, or name	String	Required
policy	Policy action for the permission, such as **create**, **read**, **update**, **delete**, or **use**	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example request body (JSON)

```
{
  "oldPolicy": [
    {
      "permission": "catalog-entity",
      "policy": "read",
      "effect": "allow"
    },
    {
      "permission": "catalog.entity.create",
```

```
      "policy": "create",
      "effect": "allow"
    }
  ],
  "newPolicy": [
    {
      "permission": "catalog-entity",
      "policy": "read",
      "effect": "deny"
    },
    {
      "permission": "policy-entity",
      "policy": "read",
      "effect": "allow"
    }
  ]
}
```

Example response

```
200
```

[DELETE] /api/permission/policies/*<kind>*/*<namespace>*/*<name>*?permission={value1}&policy={value2}&effect={value3}

Deletes a permission policy added to the specified entity.

Table 5.6. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name related to the entity	String	Required
permission	Permission from a specific plugin, resource type, or name	String	Required
policy	Policy action for the permission, such as **create**, **read**, **update**, **delete**, or **use**	String	Required
effect	Indication of allowing or not allowing the policy	String	Required

Example response

```
204 No Content
```

[DELETE] /api/permission/policies/*<kind>*/*<namespace>*/*<name>*

Deletes all permission policies added to the specified entity.

Table 5.7. Request parameters

Name	Description	Type	Presence
kind	Kind of the entity	String	Required
namespace	Namespace of the entity	String	Required
name	Name related to the entity	String	Required

Example response

```
204 No Content
```

[GET] /api/permission/plugins/policies

Returns permission policies for all static plugins.

Example response (JSON)

```
[
  {
    "pluginId": "catalog",
    "policies": [
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "read"
      },
      {
        "isResourced": false,
        "permission": "catalog.entity.create",
        "policy": "create"
      },
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "delete"
      },
      {
        "isResourced": true,
        "permission": "catalog-entity",
        "policy": "update"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.read",
        "policy": "read"
      },
```

```
      {
        "isResourced": false,
        "permission": "catalog.location.create",
        "policy": "create"
      },
      {
        "isResourced": false,
        "permission": "catalog.location.delete",
        "policy": "delete"
      }
    ]
  },
  ...
]
```

[GET] /api/permission/plugins/id

Returns object with list plugin IDs:

Example response (JSON)

```
[
  {
    "ids": ["catalog", "permission"]
  }
]
```

[POST] /api/permission/plugins/id

Add more plugins IDs defined in the request object.
Request Parameters: object in JSON format.

Example request body (JSON)

```
[
  {
    "ids": ["scaffolder"]
  }
]
```

Returns a status code of 200 and JSON with actual object stored in the server:

Example response (JSON)

```
[
  {
    "ids": ["catalog", "permission", "scaffolder"]
  }
]
```

[DELETE] /api/permission/plugins/id

Delete plugins IDs defined in the request object.
Request Parameters: object in JSON format.

Example request body (JSON)

```
[
  {
    "ids": ["scaffolder"]
  }
]
```

Returns a status code of 200 and JSON with actual object stored in the server:

Example response (JSON)

```
[
  {
    "ids": ["catalog", "permission"]
  }
]
```

NOTE

In order to prevent an inconsistent state after a deployment restart, the REST API does not allow deletion of plugin IDs that were provided by using the application configuration. These ID values can only be removed through the configuration file.

5.3.3. Conditional policies

The RBAC REST API supports the following endpoints for managing conditional policies in the Red Hat Developer Hub.

[GET] /api/permission/plugins/condition-rules

Returns available conditional rule parameter schemas for the available plugins that are enabled in Developer Hub.

Example response (JSON)

```
[
  {
    "pluginId": "catalog",
    "rules": [
      {
        "name": "HAS_ANNOTATION",
        "description": "Allow entities with the specified annotation",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "annotation": {
              "type": "string",
              "description": "Name of the annotation to match on"
            },
            "value": {
              "type": "string",
              "description": "Value of the annotation to match on"
```

```
            }
          },
          "required": [
            "annotation"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "HAS_LABEL",
        "description": "Allow entities with the specified label",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "label": {
              "type": "string",
              "description": "Name of the label to match on"
            }
          },
          "required": [
            "label"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "HAS_METADATA",
        "description": "Allow entities with the specified metadata subfield",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "key": {
              "type": "string",
              "description": "Property within the entities metadata to match on"
            },
            "value": {
              "type": "string",
              "description": "Value of the given property to match on"
            }
          },
          "required": [
            "key"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "HAS_SPEC",
        "description": "Allow entities with the specified spec subfield",
        "resourceType": "catalog-entity",
        "paramsSchema": {
```

```
      "type": "object",
      "properties": {
        "key": {
          "type": "string",
          "description": "Property within the entities spec to match on"
        },
        "value": {
          "type": "string",
          "description": "Value of the given property to match on"
        }
      },
      "required": [
        "key"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "IS_ENTITY_KIND",
    "description": "Allow entities matching a specified kind",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "kinds": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of kinds to match at least one of"
        }
      },
      "required": [
        "kinds"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "IS_ENTITY_OWNER",
    "description": "Allow entities owned by a specified claim",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "claims": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of claims to match at least one on within ownedBy"
        }
      },
      "required": [
```

```
            "claims"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      }
    ]
  }
  ... <another plugin condition parameter schemas>
]
```

[GET] /api/permission/roles/conditions/:id

Returns conditions for the specified ID.

Example response (JSON)

```
{
  "id": 1,
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

[GET] /api/permission/roles/conditions

Returns list of all conditions for all roles.

Example response (JSON)

```
[
  {
    "id": 1,
    "result": "CONDITIONAL",
    "roleEntityRef": "role:default/test",
    "pluginId": "catalog",
```

```
    "resourceType": "catalog-entity",
    "permissionMapping": ["read"],
    "conditions": {
      "anyOf": [
        {
          "rule": "IS_ENTITY_OWNER",
          "resourceType": "catalog-entity",
          "params": {
            "claims": ["group:default/team-a"]
          }
        },
        {
          "rule": "IS_ENTITY_KIND",
          "resourceType": "catalog-entity",
          "params": {
            "kinds": ["Group"]
          }
        }
      ]
    }
  }
]
```

[POST] /api/permission/roles/conditions

Creates a conditional policy for the specified role.

Table 5.8. Request parameters

Name	Description	Type	Presence
result	Always has the value **CONDITIONAL**	String	Required
roleEntityRef	String entity reference to the RBAC role, such as **role:default/dev**	String	Required
pluginId	Corresponding plugin ID, such as **catalog**	String	Required
permissionMapping	Array permission action, such as **['read', 'update', 'delete']**	String array	Required
resourceType	Resource type provided by the plugin, such as **catalog-entity**	String	Required
conditions	Condition JSON with parameters or array parameters joined by criteria	JSON	Required
name	Name of the role	String	Required
metadata.description	The description of the role	String	Optional

Example request body (JSON)

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["group:default/team-a"]
    }
  }
}
```

Example response (JSON)

```
{
  "id": 1
}
```

[PUT] /permission/roles/conditions/:id

Updates a condition policy for a specified ID.

Table 5.9. Request parameters

Name	Description	Type	Presence
result	Always has the value **CONDITIONAL**	String	Required
roleEntityRef	String entity reference to the RBAC role, such as **role:default/dev**	String	Required
pluginId	Corresponding plugin ID, such as **catalog**	String	Required
permissionMapping	Array permission action, such as **['read', 'update', 'delete']**	String array	Required
resourceType	Resource type provided by the plugin, such as **catalog-entity**	String	Required
conditions	Condition JSON with parameters or array parameters joined by criteria	JSON	Required
name	Name of the role	String	Required

Name	Description	Type	Presence
metadata.description	The description of the role	String	Optional

Example request body (JSON)

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

Example response

```
200
```

[DELETE] /api/permission/roles/conditions/:id

Deletes a conditional policy for the specified ID.

Example response

```
204
```

5.3.4. User statistics

The **licensed-users-info-backend** plugin exposes various REST API endpoints to retrieve data related to logged-in users.

No additional configuration is required for the **licensed-users-info-backend** plugin. If the RBAC backend plugin is enabled, then an administrator role must be assigned to access the endpoints, as the endpoints are protected by the **policy.entity.read** permission.

The base URL for user statistics endpoints is **http://SERVER:PORT/api/licensed-users-info**, such as **http://localhost:7007/api/licensed-users-info**.

[GET] /users/quantity

Returns the total number of logged-in users.

Example request

```
curl -X GET "http://localhost:7007/api/licensed-users-info/users/quantity" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $token"
```

Example response

```
{ "quantity": "2" }
```

[GET] /users

Returns a list of logged-in users with their details.

Example request

```
curl -X GET "http://localhost:7007/api/licensed-users-info/users" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $token"
```

Example response

```
[
  {
    "userEntityRef": "user:default/dev",
    "lastTimeLogin": "Thu, 22 Aug 2024 16:27:41 GMT",
    "displayName": "John Leavy",
    "email": "dev@redhat.com"
  }
]
```

[GET] /users

Returns a list of logged-in users in CSV format.

Example request

```
curl -X GET "http://localhost:7007/api/licensed-users-info/users" \
-H "Content-Type: text/csv" \
-H "Authorization: Bearer $token"
```

Example response

```
userEntityRef,displayName,email,lastTimeLogin
user:default/dev,John Leavy,dev@redhat.com,"Thu, 22 Aug 2024 16:27:41 GMT"
```

CHAPTER 6. MANAGING AUTHORIZATIONS BY USING EXTERNAL FILES

To automate Red Hat Developer Hub maintenance, you can configure permissions and roles in external files, before starting Developer Hub.

6.1. DEFINING AUTHORIZATIONS IN EXTERNAL FILES BY USING THE OPERATOR

To automate Red Hat Developer Hub maintenance, you can define permissions and roles in external files, before starting Developer Hub. You need to prepare your files, upload them to your OpenShift Container Platform project, and configure Developer Hub to use the external files.

Prerequisites

- You enabled the RBAC feature.

Procedure

1. Define your policies in a **rbac-policies.csv** CSV file by using the following format:

   a. Define role permissions:

   ```
   p, <role_entity_reference>, <permission>, <action>, <allow_or_deny>
   ```

   <role_entity_reference>

   Role entity reference, such as: **role:default/guest**.

   <permission>

   Permission, such as: **bulk.import**, **catalog.entity.read**, or **catalog.entity.refresh**, or permission resource type, such as: **bulk-import** or **catalog-entity**.
   See: Permission policies reference.

   <action>

   Action type, such as: **use**, **read**, **create**, **update**, **delete**.

   <allow_or_deny>

   Access granted: **allow** or **deny**.

   b. Assign the role to a group or a user:

   ```
   g, <group_or_user>, <role_entity_reference>
   ```

   <group_or_user>

   Group, such as: **user:default/mygroup**, or user, such as: **user:default/myuser**.

   Sample rbac-policies.csv

   ```
   p, role:default/guests, catalog-entity, read, allow
   p, role:default/guests, catalog.entity.create, create, allow
   g, user:default/my-user, role:default/guests
   g, group:default/my-group, role:default/guests
   ```

-

2. Define your conditional policies in a **rbac-conditional-policies.yaml** YAML file by using the following format:

```
result: CONDITIONAL
roleEntityRef: <role_entity_reference>
pluginId: <plugin_id>
permissionMapping:
  - read
  - update
  - delete
conditions: <conditions>
```

See: Conditional policies reference .

3. Upload your **rbac-policies.csv** and **rbac-conditional-policies.yaml** files to a **rbac-policies** config map in your OpenShift Container Platform project containing Developer Hub.

```
$ oc create configmap rbac-policies \
    --from-file=rbac-policies.csv \
    --from-file=rbac-conditional-policies.yaml
```

4. Update your **Backstage** custom resource to mount in the Developer Hub filesystem your files from the **rbac-policies** config map:

Backstage custom resource fragment

```
apiVersion: rhdh.redhat.com/v1alpha3
kind: Backstage
spec:
  application:
    extraFiles:
      mountPath: /opt/app-root/src
      configMaps:
        - name: rbac-policies
```

5. Update your Developer Hub **app-config.yaml** configuration file to use the **rbac-policies.csv** and **rbac-conditional-policies.yaml** external files:

app-config.yaml file fragment

```
permission:
  enabled: true
  rbac:
    conditionalPoliciesFile: /opt/app-root/src/rbac-conditional-policies.yaml
    policies-csv-file: /opt/app-root/src/rbac-policies.csv
    policyFileReload: true
```

6.2. DEFINING AUTHORIZATIONS IN EXTERNAL FILES BY USING HELM

To automate Red Hat Developer Hub maintenance, you can define permissions and roles in external files, before starting Developer Hub. You need to prepare your files, upload them to your OpenShift Container Platform project, and configure Developer Hub to use the external files.

Prerequisites

- You enabled the RBAC feature.

Procedure

1. Define your policies in a **rbac-policies.csv** CSV file by using the following format:

   a. Define role permissions:

   ```
   p, <role_entity_reference>, <permission>, <action>, <allow_or_deny>
   ```

   <role_entity_reference>

   Role entity reference, such as: **role:default/guest**.

   <permission>

   Permission, such as: **bulk.import**, **catalog.entity.read**, or **catalog.entity.refresh**, or permission resource type, such as: **bulk-import** or **catalog-entity**.
   See: Permission policies reference.

   <action>

   Action type, such as: **use**, **read**, **create**, **update**, **delete**.

   <allow_or_deny>

   Access granted: **allow** or **deny**.

   b. Assign the role to a group or a user:

   ```
   g, <group_or_user>, <role_entity_reference>
   ```

   <group_or_user>

   Group, such as: **user:default/mygroup**, or user, such as: **user:default/myuser**.

   Sample rbac-policies.csv

   ```
   p, role:default/guests, catalog-entity, read, allow
   p, role:default/guests, catalog.entity.create, create, allow
   g, user:default/my-user, role:default/guests
   g, group:default/my-group, role:default/guests
   ```

2. Define your conditional policies in a **rbac-conditional-policies.yaml** YAML file by using the following format:

   ```
   result: CONDITIONAL
   roleEntityRef: <role_entity_reference>
   pluginId: <plugin_id>
   permissionMapping:
     - read
     - update
   ```

```
  - delete
conditions: <conditions>
```

See: Conditional policies reference .

3. Upload your **rbac-policies.csv** and **rbac-conditional-policies.yaml** files to a **rbac-policies** config map in your OpenShift Container Platform project containing Developer Hub.

```
$ oc create configmap rbac-policies \
    --from-file=rbac-policies.csv \
    --from-file=rbac-conditional-policies.yaml
```

4. Update your Developer Hub **Backstage** Helm chart to mount in the Developer Hub filesystem your files from the **rbac-policies** config map:

   a. In the Developer Hub Helm Chart, go to **Root Schema → Backstage chart schema → Backstage parameters → Backstage container additional volume mounts**.

   b. Select **Add Backstage container additional volume mounts**and add the following values:

   mountPath

   /opt/app-root/src/rbac

   Name

   rbac-policies

   c. Add the RBAC policy to the **Backstage container additional volumes**in the Developer Hub Helm Chart:

   name

   rbac-policies

   configMap

   defaultMode

   420

   name

   rbac-policies

5. Update your Developer Hub **app-config.yaml** configuration file to use the **rbac-policies.csv** and **rbac-conditional-policies.yaml** external files:

app-config.yaml file fragment

```
permission:
  enabled: true
  rbac:
    conditionalPoliciesFile: /opt/app-root/src/rbac-conditional-policies.yaml
    policies-csv-file: /opt/app-root/src/rbac-policies.csv
    policyFileReload: true
```

CHAPTER 7. CONFIGURING GUEST ACCESS WITH RBAC UI

Use guest access with the role-based access control (RBAC) front-end plugin to allow a user to test role and policy creation without the need to set up and configure an authentication provider.

NOTE

Guest access is not recommended for production.

7.1. CONFIGURING THE RBAC BACKEND PLUGIN

You can configure the RBAC backend plugin by updating the **app-config.yaml** file to enable the permission framework.

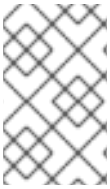
Prerequisites

- You have installed the **@backstage-community/plugin-rbac** plugin in Developer Hub. For more information, see Configuring dynamic plugins.

Procedure

- Update the **app-config.yaml** file to enable the permission framework as shown:

```
permission
  enabled: true
  rbac:
    admin:
      users:
        - name: user:default/guest
    pluginsWithPermission:
      - catalog
      - permission
      - scaffolder
```



NOTE

The **pluginsWithPermission** section of the **app-config.yaml** file includes only three plugins by default. Update the section as needed to include any additional plugins that also incorporate permissions.

7.2. SETTING UP THE GUEST AUTHENTICATION PROVIDER

You can enable guest authentication and use it alongside the RBAC frontend plugin.

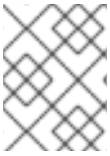
Prerequisites

- You have installed the **@backstage-community/plugin-rbac** plugin in Developer Hub. For more information, see Configuring dynamic plugins.

Procedure

- In the **app-config.yaml** file, add the user entity reference to resolve and enable the **dangerouslyAllowOutsideDevelopment** option, as shown in the following example:

```
auth:
  environment: development
  providers:
    guest:
      userEntityRef: user:default/guest
      dangerouslyAllowOutsideDevelopment: true
```



NOTE

You can use **user:default/guest** as the user entity reference to match the added user under the **permission.rbac.admin.users** section of the **app-config.yaml** file.

CHAPTER 8. DELEGATING ROLE-BASED ACCESS CONTROLS (RBAC) ACCESS IN RED HAT DEVELOPER HUB

An enterprise customer requires the ability to delegate role-based access control (RBAC) responsibilities to other individuals in the organization. In this scenario, you, as the administrator, can provide access to the RBAC plugin specifically to designated users, such as team leads. Each team lead is then able to manage permissions exclusively for users within their respective team or department, without visibility into or control over permissions outside their assigned scope. This approach allows team leads to manage access and permissions for their own teams independently, while administrators maintain global oversight.

In Red Hat Developer Hub, you can delegate RBAC access using the multitenancy feature of the RBAC plugin, specifically the **IS_OWNER** conditional rule. You can either use the web UI or the RBAC backend API, depending on your preferred workflow and level of automation:

- Use the web UI to create roles, assign users or groups, define permissions, and apply ownership conditions through an intuitive interface.
- Use the API for a more flexible and automatable approach, where you can programmatically manage roles, permissions, and ownership conditions using authenticated curl requests.

By delegating RBAC access through either method, you can expect the following outcomes:

- Team leads can manage RBAC settings for their teams independently.
- Visibility of other users' or teams' permissions is restricted.
- Administrators retain overarching control while delegating team-specific access.

Prerequisites

- Your RHDH instance is running with the RBAC plugin installed and configured.
- You have administrative access to RHDH.

8.1. DELEGATING RBAC ACCESS IN RED HAT DEVELOPER HUB BY USING THE WEB UI

You can delegate the RBAC access in Red Hat Developer Hub by using the web UI.

Procedure

1. Log in to your RHDH instance with administrator credentials.
2. Navigate to **Administration → RBAC**.
3. Click **Create Role** and define a new role for team leads, such as **role:default/team_lead**.
4. In the **Members** section, add the user or group, such as **user:default/team_lead**.
5. Grant permissions required by team leads, such as:
    - **policy.entity.create** to allow policy creation.
    - **catalog-entity:read** to allow catalog access.

6. Apply **conditions** to limit access as follows:
   - Use the **IS_OWNER** rule to ensure team leads can only manage resources they own.
7. Click **Save** to create the role and apply changes.

Verification

- Log in as a team lead.
- Verify the following:
  - RBAC UI is accessible.
  - Only users or roles related to their team are visible.
  - No access to roles or permissions outside their scope is granted.

8.2. DELEGATING RBAC ACCESS IN RED HAT DEVELOPER HUB BY USING API

You can delegate the RBAC access in Red Hat Developer Hub by using the RBAC backend API.

Prerequisites

- You have API access using **curl** or another tool.

Procedure

1. Create a new role designated for team leads using the RBAC backend API:

   Example of creating a new role for the team lead using the RBAC backend API

   ```
   $ curl -X POST 'http://localhost:7007/api/permission/roles' \
   --header "Authorization: Bearer $ADMIN_TOKEN" \
   --header "Content-Type: application/json" \
   --data '{
     "memberReferences": ["user:default/team_lead"],
     "name": "role:default/team_lead",
     "metadata": {
       "description": "This is an example team lead role"
     }
   }'
   ```

2. Allow team leads to read catalog entities and create permissions in the RBAC plugin using the following API request:

   Example of granting the team lead role permission to create RBAC policies and read catalog entities

   ```
   $ curl -X POST 'http://localhost:7007/api/permission/policies' \
   --header "Authorization: Bearer $ADMIN_TOKEN" \
   --header "Content-Type: application/json" \
   --data '[
   ```

```
  {
    "entityReference": "role:default/team_lead",
    "permission": "policy.entity.create",
    "policy": "create",
    "effect": "allow"
  },
  {
    "entityReference": "role:default/team_lead",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow"
  }
]'
```

3. To ensure team leads can only manage what they own, use the **IS_OWNER** conditional rule as follows:

Example curl of applying a conditional access policy using the IS_OWNER rule for the team lead role

```
$ curl -X POST 'http://localhost:7007/api/permission/roles/conditions' \
--header "Authorization: Bearer $ADMIN_TOKEN" \
--header "Content-Type: application/json" \
--data '{
 "result": "CONDITIONAL",
 "pluginId": "permission",
 "resourceType": "policy-entity",
 "conditions": {
   "rule": "IS_OWNER",
   "resourceType": "policy-entity",
   "params": {
     "owners": [
       "user:default/team_lead"
     ]
   }
 },
 "roleEntityRef": "role:default/team_lead",
 "permissionMapping": [
   "read",
   "update",
   "delete"
 ]
}'
```

The previous example of conditional policy limits visibility and control to only owned roles and policies.

4. Log in to RHDH as team lead and verify the following:

   a. Use the following request and verify that you do not see any roles:

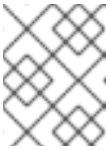
Example curl to retrieve roles visible to the team lead

```
$ curl -X GET 'http://localhost:7007/api/permission/roles' \
--header "Authorization: Bearer $TEAM_LEAD_TOKEN"
```

b. Use the following request to create a new role for their team:

Example curl of team lead creating a new role for their team with ownership assigned

```
$ curl -X POST 'http://localhost:7007/api/permission/roles' \
--header "Authorization: Bearer $TEAM_LEAD_TOKEN" \
--header "Content-Type: application/json" \
--data '{
  "memberReferences": ["user:default/team_member"],
  "name": "role:default/team_a",
  "metadata": {
    "description": "This is an example team_a role",
    "owner": "user:default/team_lead"
  }
}'
```



NOTE

You can set the ownership during creation, but you can also update the ownership at any time.

c. Use the following request to assign a permission policy to the new role:

Example curl for granting read access to catalog entities for the new role

```
$ curl -X POST 'http://localhost:7007/api/permission/policies' \
--header "Authorization: Bearer $ADMIN_TOKEN" \
--header "Content-Type: application/json" \
--data '[
  {
    "entityReference": "role:default/team_a",
    "permission": "catalog-entity",
    "policy": "read",
    "effect": "allow"
  }
]'
```

d. Use the following request to verify that only team-owned roles and policies are visible:

Example curl to retrieve roles and permission policies visible to the team lead

```
$ curl -X GET 'http://localhost:7007/api/permission/roles' \
--header "Authorization: Bearer $TEAM_LEAD_TOKEN"

$ curl -X GET 'http://localhost:7007/api/permission/policies' \
--header "Authorization: Bearer $TEAM_LEAD_TOKEN"
```

Verification

- Log in as a team lead and verify the following:
  - The RBAC UI is accessible.

    - Only the assigned users or group is visible.
    - Permissions outside the scoped team are not viewable or editable.
- Log in as an administrator and verify that you retain full visibility and control.

CHAPTER 9. PERMISSION POLICIES REFERENCE

Permission policies in Red Hat Developer Hub are a set of rules to govern access to resources or functionalities. These policies state the authorization level that is granted to users based on their roles. The permission policies are implemented to maintain security and confidentiality within a given environment.

You can define the following types of permissions in Developer Hub:

- resource type
- basic

The distinction between the two permission types depends on whether a permission includes a defined resource type.

You can define the resource type permission using either the associated resource type or the permission name as shown in the following example:

Example resource type permission definition

```
p, role:default/myrole, catalog.entity.read, read, allow
g, user:default/myuser, role:default/myrole

p, role:default/another-role, catalog-entity, read, allow
g, user:default/another-user, role:default/another-role
```

You can define the basic permission in Developer Hub using the permission name as shown in the following example:

Example basic permission definition

```
p, role:default/myrole, catalog.entity.create, create, allow
g, user:default/myuser, role:default/myrole
```

Developer Hub supports following permission policies:

Catalog permissions

Name	Resource type	Policy	Description
catalog.entity.read	**catalog-entity**	**read**	Enables a user or role to read from the catalog
catalog.entity.create		**create**	Enables a user or role to create catalog entities, including registering an existing component in the catalog
catalog.entity.refresh	**catalog-entity**	**update**	Enables a user or role to refresh a single or multiple entities from the catalog

Name	Resource type	Policy	Description
catalog.entity.delete	**catalog-entity**	**delete**	Enables a user or role to delete a single or multiple entities from the catalog
catalog.location.read		**read**	Enables a user or role to read a single or multiple locations from the catalog
catalog.location.create		**create**	Enables a user or role to create locations within the catalog
catalog.location.delete		**delete**	Enables a user or role to delete locations from the catalog

Bulk import permission

Name	Resource type	Policy	Description
bulk.import	**bulk-import**	**use**	Enables the user to access the bulk import endpoints, such as listing all repositories and organizations accessible by all GitHub integrations and managing the import requests

Scaffolder permissions

Name	Resource type	Policy	Description
scaffolder.action.execute	**scaffolder-action**	**use**	Enables the execution of an action from a template
scaffolder.template.parameter.read	**scaffolder-template**	**read**	Enables a user or role to read a single or multiple one parameters from a template
scaffolder.template.step.read	**scaffolder-template**	**read**	Enables a user or role to read a single or multiple steps from a template
scaffolder.task.create		**create**	Enables a user or role to trigger software templates which create new scaffolder tasks

Name	Resource type	Policy	Description
scaffolder.task.cancel		**use**	Enables a user or role to cancel currently running scaffolder tasks
scaffolder.task.read		**read**	Enables a user or role to read all scaffolder tasks and their associated events and logs
scaffolder.template.management		**use**	Enables a user or role to access frontend template management features, including editing, previewing, and trying templates, forms, and custom fields.

RBAC permissions

Name	Resource type	Policy	Description
policy.entity.read	**policy-entity**	**read**	Enables a user or role to read permission policies and roles
policy.entity.create		**create**	Enables a user or role to create a single or multiple permission policies and roles
policy.entity.update	**policy-entity**	**update**	Enables a user or role to update a single or multiple permission policies and roles
policy.entity.delete	**policy-entity**	**delete**	Enables a user or role to delete a single or multiple permission policies and roles

Kubernetes permissions

Name	Resource type	Policy	Description
kubernetes.clusters.read		**read**	Enables a user to read Kubernetes cluster details under the **/clusters** path
kubernetes.resources.read		**read**	Enables a user to read information about Kubernetes resources located at **/services/:serviceId** and **/resources**
kubernetes.proxy		**use**	Enables a user or role to access the proxy endpoint

OCM permissions

Basic OCM permissions only restrict access to the cluster view, but they do not prevent access to the Kubernetes clusters in the resource view. For more effective permissions, consider applying a conditional policy to restrict access to catalog entities that are of type **kubernetes-cluster**. Access restriction is dependent on the set of permissions granted to a role. For example, if the role had full permissions (**read**, **update**, and **delete**), then you must specify all its permissions in the **permissionMapping** field.

Example permissionMapping definition

```
result: CONDITIONAL
roleEntityRef: 'role:default/<YOUR_ROLE>'
pluginId: catalog
resourceType: catalog-entity
permissionMapping:
  - read
  - update
  - delete
conditions:
  not:
    rule: HAS_SPEC
    resourceType: catalog-entity
    params:
      key: type
      value: kubernetes-cluster
```

Name	Resource type	Policy	Description
ocm.entity.read		**read**	Enables a user or role to read from the OCM plugin
ocm.cluster.read		**read**	Enables a user or role to read the cluster information in the OCM plugin

Topology permissions

Name	Resource type	Policy	Description
kubernetes.clusters.read		**read**	Enables a user to read Kubernetes cluster details under the **/clusters** path
kubernetes.resources.read		**read**	Enables a user to read information about Kubernetes resources located at **/services/:serviceId** and **/resources**
kubernetes.proxy		**use**	Enables a user or role to access the proxy endpoint, allowing the user or role to read pod logs and events within RHDH

Tekton permissions

Name	Resource type	Policy	Description
kubernetes.clusters.read		**read**	Enables a user to read Kubernetes cluster details under the **/clusters** path
kubernetes.resources.read		**read**	Enables a user to read information about Kubernetes resources located at **/services/:serviceId** and **/resources**
kubernetes.proxy		**use**	Enables a user or role to access the proxy endpoint, allowing the user or role to read pod logs and events within RHDH

ArgoCD permissions

Name	Resource type	Policy	Description
argocd.view.read		**read**	Enables a user to read from the ArgoCD plugin

Quay permissions

Name	Resource type	Policy	Description
quay.view.read		**read**	Enables a user to read from the Quay plugin

CHAPTER 10. CONDITIONAL POLICIES IN RED HAT DEVELOPER HUB

The permission framework in Red Hat Developer Hub provides conditions, supported by the RBAC backend plugin (**backstage-plugin-rbac-backend**). The conditions work as content filters for the Developer Hub resources that are provided by the RBAC backend plugin.

The RBAC backend API stores conditions assigned to roles in the database. When you request to access the frontend resources, the RBAC backend API searches for the corresponding conditions and delegates them to the appropriate plugin using its plugin ID. If you are assigned to multiple roles with different conditions, then the RBAC backend merges the conditions using the **anyOf** criteria.

Conditional criteria

A condition in Developer Hub is a simple condition with a rule and parameters. However, a condition can also contain a parameter or an array of parameters combined by conditional criteria. The supported conditional criteria includes:

- **allOf**: Ensures that all conditions within the array must be true for the combined condition to be satisfied.
- **anyOf**: Ensures that at least one of the conditions within the array must be true for the combined condition to be satisfied.
- **not**: Ensures that the condition within it must not be true for the combined condition to be satisfied.

Conditional object

The plugin specifies the parameters supported for conditions. You can access the conditional object schema from the RBAC API endpoint to understand how to construct a conditional JSON object, which is then used by the RBAC backend plugin API.
A conditional object contains the following parameters:

Parameter	Type	Description
result	String	Always has the value **CONDITIONAL**
roleEntityRef	String	String entity reference to the RBAC role, such as **role:default/dev**
pluginId	String	Corresponding plugin ID, such as **catalog**
permissionMapping	String array	Array permission actions, such as **['read', 'update', 'delete']**
resourceType	String	Resource type provided by the plugin, such as **catalog-entity**

Parameter	Type	Description
conditions	JSON	Condition JSON with parameters or array parameters joined by criteria

Conditional policy aliases

The RBAC backend plugin (**backstage-plugin-rbac-backend**) supports the use of aliases in conditional policy rule parameters. The conditional policy aliases are dynamically replaced with the corresponding values during policy evaluation. Each alias in conditional policy is prefixed with a **$** sign indicating its special function.
The supported conditional aliases include:

- **$currentUser**: This alias is replaced with the user entity reference for the user who requests access to the resource. For example, if user Tom from the default namespace requests access, **$currentUser** becomes **user:default/tom**.
  Example conditional policy object with **$currentUser** alias:

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["delete"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["$currentUser"]
    }
  }
}
```

- **$ownerRefs**: This alias is replaced with ownership references, usually as an array that includes the user entity reference and the user's parent group entity reference. For example, for user Tom from team-a, **$ownerRefs** becomes **['user:default/tom', 'group:default/team-a']**.
  Example conditional policy object with **$ownerRefs** alias:

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["delete"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["$ownerRefs"]
```

```
    }
  }
}
```

10.1. ENABLING TRANSITIVE PARENT GROUPS

By default, Red Hat Developer Hub does not resolve indirect parent groups during authentication. In this case, with the following group hierarchy, the **user_alice** user is only a member of the **group_developers** group:

```
group_admin
  └─── group_developers
    └─── user_alice
```

To support multi-level group hierarchies when using the $ownerRefs alias, you can configure Developer Hub to include indirect parent groups in the user's ownership entities. In that case the **user_alice** user is a member of both **group_developers** and **group_admin** groups.

Procedure

- Enable the **includeTransitiveGroupOwnership** option in your **app-config.yaml** file.

```
includeTransitiveGroupOwnership: true
```

10.2. CONDITIONAL POLICIES REFERENCE

You can access API endpoints for conditional policies in Red Hat Developer Hub. For example, to retrieve the available conditional rules, which can help you define these policies, you can access the **GET [api/plugins/condition-rules]** endpoint.

The **api/plugins/condition-rules** returns the condition parameters schemas, for example:

```
[
  {
    "pluginId": "catalog",
    "rules": [
      {
        "name": "HAS_ANNOTATION",
        "description": "Allow entities with the specified annotation",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "annotation": {
              "type": "string",
              "description": "Name of the annotation to match on"
            },
            "value": {
              "type": "string",
              "description": "Value of the annotation to match on"
            }
          },
```

```
      "required": [
        "annotation"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_LABEL",
    "description": "Allow entities with the specified label",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "label": {
          "type": "string",
          "description": "Name of the label to match on"
        }
      },
      "required": [
        "label"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_METADATA",
    "description": "Allow entities with the specified metadata subfield",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
        "key": {
          "type": "string",
          "description": "Property within the entities metadata to match on"
        },
        "value": {
          "type": "string",
          "description": "Value of the given property to match on"
        }
      },
      "required": [
        "key"
      ],
      "additionalProperties": false,
      "$schema": "http://json-schema.org/draft-07/schema#"
    }
  },
  {
    "name": "HAS_SPEC",
    "description": "Allow entities with the specified spec subfield",
    "resourceType": "catalog-entity",
    "paramsSchema": {
      "type": "object",
      "properties": {
```

```
            "key": {
              "type": "string",
              "description": "Property within the entities spec to match on"
            },
            "value": {
              "type": "string",
              "description": "Value of the given property to match on"
            }
          },
          "required": [
            "key"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "IS_ENTITY_KIND",
        "description": "Allow entities matching a specified kind",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "kinds": {
              "type": "array",
              "items": {
                "type": "string"
              },
              "description": "List of kinds to match at least one of"
            }
          },
          "required": [
            "kinds"
          ],
          "additionalProperties": false,
          "$schema": "http://json-schema.org/draft-07/schema#"
        }
      },
      {
        "name": "IS_ENTITY_OWNER",
        "description": "Allow entities owned by a specified claim",
        "resourceType": "catalog-entity",
        "paramsSchema": {
          "type": "object",
          "properties": {
            "claims": {
              "type": "array",
              "items": {
                "type": "string"
              },
              "description": "List of claims to match at least one on within ownedBy"
            }
          },
          "required": [
            "claims"
          ],
```

```
            "additionalProperties": false,
            "$schema": "http://json-schema.org/draft-07/schema#"
          }
        }
      ]
    }
    ... <another plugin condition parameter schemas>
]
```

The RBAC backend API constructs a condition JSON object based on the previous condition schema.

10.2.1. Examples of conditional policies

In Red Hat Developer Hub, you can define conditional policies with or without criteria. You can use the following examples to define the conditions based on your use case:

A condition without criteria

Consider a condition without criteria displaying catalogs only if user is a member of the owner group. To add this condition, you can use the catalog plugin schema **IS_ENTITY_OWNER** as follows:

Example condition without criteria

```
{
  "rule": "IS_ENTITY_OWNER",
  "resourceType": "catalog-entity",
  "params": {
    "claims": ["group:default/team-a"]
  }
}
```

In the previous example, the only conditional parameter used is **claims**, which contains a list of user or group entity references.

You can apply the previous example condition to the RBAC REST API by adding additional parameters as follows:

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "rule": "IS_ENTITY_OWNER",
    "resourceType": "catalog-entity",
    "params": {
      "claims": ["group:default/team-a"]
    }
  }
}
```

A condition with criteria

Consider a condition with criteria, which displays catalogs only if user is a member of owner group OR displays list of all catalog user groups.

To add the criteria, you can add another rule as **IS_ENTITY_KIND** in the condition as follows:

Example condition with criteria

```
{
  "anyOf": [
    {
      "rule": "IS_ENTITY_OWNER",
      "resourceType": "catalog-entity",
      "params": {
        "claims": ["group:default/team-a"]
      }
    },
    {
      "rule": "IS_ENTITY_KIND",
      "resourceType": "catalog-entity",
      "params": {
        "kinds": ["Group"]
      }
    }
  ]
}
```

NOTE

Running conditions in parallel during creation is not supported. Therefore, consider defining nested conditional policies based on the available criteria.

Example of nested conditions

```
{
  "anyOf": [
    {
      "rule": "IS_ENTITY_OWNER",
      "resourceType": "catalog-entity",
      "params": {
        "claims": ["group:default/team-a"]
      }
    },
    {
      "rule": "IS_ENTITY_KIND",
      "resourceType": "catalog-entity",
      "params": {
        "kinds": ["Group"]
      }
    }
  ],
  "not": {
    "rule": "IS_ENTITY_KIND",
    "resourceType": "catalog-entity",
    "params": { "kinds": ["Api"] }
  }
}
```

You can apply the previous example condition to the RBAC REST API by adding additional parameters as follows:

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/test",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["read"],
  "conditions": {
    "anyOf": [
      {
        "rule": "IS_ENTITY_OWNER",
        "resourceType": "catalog-entity",
        "params": {
          "claims": ["group:default/team-a"]
        }
      },
      {
        "rule": "IS_ENTITY_KIND",
        "resourceType": "catalog-entity",
        "params": {
          "kinds": ["Group"]
        }
      }
    ]
  }
}
```

The following examples can be used with Developer Hub plugins. These examples can help you determine how to define conditional policies:

Conditional policy defined for Keycloak plugin

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "catalog",
  "resourceType": "catalog-entity",
  "permissionMapping": ["update", "delete"],
  "conditions": {
    "not": {
      "rule": "HAS_ANNOTATION",
      "resourceType": "catalog-entity",
      "params": { "annotation": "keycloak.org/realm", "value": "<YOUR_REALM>" }
    }
  }
}
```

The previous example of Keycloak plugin prevents users in the **role:default/developer** from updating or deleting users that are ingested into the catalog from the Keycloak plugin.

NOTE

In the previous example, the annotation **keycloak.org/realm** requires the value of **<YOUR_REALM>**.

Conditional policy defined for Quay plugin

```
{
  "result": "CONDITIONAL",
  "roleEntityRef": "role:default/developer",
  "pluginId": "scaffolder",
  "resourceType": "scaffolder-action",
  "permissionMapping": ["use"],
  "conditions": {
    "not": {
      "rule": "HAS_ACTION_ID",
      "resourceType": "scaffolder-action",
      "params": { "actionId": "quay:create-repository" }
    }
  }
}
```

The previous example of Quay plugin prevents the role **role:default/developer** from using the Quay scaffolder action. Note that **permissionMapping** contains **use**, signifying that **scaffolder-action** resource type permission does not have a permission policy.

Additional resources

- Chapter 9, *Permission policies reference*

CHAPTER 11. USER STATISTICS IN RED HAT DEVELOPER HUB

In Red Hat Developer Hub, the **licensed-users-info-backend** plugin provides statistical information about the logged-in users using the Web UI or REST API endpoints.

The **licensed-users-info-backend** plugin enables administrators to monitor the number of active users on Developer Hub. Using this feature, organizations can compare their actual usage with the number of licenses they have purchased. Additionally, you can share the user metrics with Red Hat for transparency and accurate licensing.

The **licensed-users-info-backend** plugin is enabled by default. This plugin enables a **Download User List** link at the bottom of the **Administration → RBAC** tab.

11.1. DOWNLOADING ACTIVE USERS LIST IN RED HAT DEVELOPER HUB

You can download the list of users in CSV format using the Developer Hub web interface.

Prerequisites

- RBAC plugins (**@backstage-community/plugin-rbac** and **@backstage-community/plugin-rbac-backend**) must be enabled in Red Hat Developer Hub.
- An administrator role must be assigned.

Procedure

1. In Red Hat Developer Hub, navigate to **Administration** and select the **RBAC** tab.
2. At the bottom of the **RBAC** page, click **Download User List**
3. Optional: Modify the file name in the **Save as** field and click **Save**.
4. To access the downloaded users list, go to the **Downloads** folder on your local machine and open the CSV file.