Red Hat Developer Hub 1.8

Authentication in Red Hat Developer Hub

Configuring authentication to external services in Red Hat Developer Hub

Last Updated: 2026-01-22

Red Hat Developer Hub 1.8 Authentication in Red Hat Developer Hub

Configuring authentication to external services in Red Hat Developer Hub

Legal Notice

Abstract

As a Red Hat Developer Hub (RHDH) platform engineer, you can manage authentication of other users to meet the specific needs of your organization.

Table of Contents

CHAPTER 1. UNDERSTANDING AUTHENTICATION AND USER PROVISIONING

Learn about the authentication process from creating user and group entities in the software catalog to user sign-in, and how authentication and catalog plugins enable each step. Understanding this process is essential for successfully configuring your Developer Hub instance, securing access through authorization, and enabling features that rely on synchronized user and group data.

To fully enable catalog features, provision user and group data from the Identity Provider to the Developer Hub software catalog. Catalog provider plugins handle this task asynchronously. These plugins query the Identity Provider (IdP) for relevant user and group information, and create or update corresponding entities in the Developer Hub catalog. Scheduled provisioning ensures that the catalog accurately reflects the users and groups in your organization.

When a user attempts to access Developer Hub, Developer Hub redirects them to a configured authentication provider, such as Red Hat Build of Keycloak (RHBK), GitHub, or Microsoft Azure. This external IdP is responsible for authenticating the user.

On successful authentication, the Developer Hub authentication plugin, configured in your **app-config.yaml** file, processes the response from the IdP, resolves the identity in the Developer Hub software catalog, and establishes a user session within Developer Hub.

Configuring authentication and user provisioning is critical for several reasons.

- Securing your Developer Hub instance by ensuring only authenticated users can gain access.
- Enabling authorization by allowing you to define access controls based on user and group memberships synchronized from your IdP.
- Provisioning user and group data to the catalog is necessary for various catalog features that rely on understanding entity ownership and relationships between users, groups, and software components.

IMPORTANT

Without this provisioning step, features such as displaying who owns a catalog entity might not function correctly.

TIP

To explore Developer Hub features in a non-production environment, you can:

- To use Developer Hub without external IdP, enable the guest user to skip configuring authentication and authorization, log in as the guest user, and access all Developer Hub features.
- To use Developer Hub without authorization policies and features relying on the software catalog, you can enable the **dangerouslyAllowSignInWithoutUserInCatalog** resolver option. This setting bypasses the check requiring a user to be in the catalog but still enforces authentication.

IMPORTANT

Developer Hub uses a one-way synchronization model, where user and group data flow from your Identity Provider to the Developer Hub software catalog. As a result, deleting users or groups manually through the Developer Hub Web UI or REST API might be ineffective or cause inconsistencies, since Developer Hub will create those entities again during the next import.

CHAPTER 2. AUTHENTICATING WITH THE GUEST USER

For trial or non-production environments, you can enable guest access to skip configuring authentication and authorization and explore Developer Hub features.

2.1. AUTHENTICATING WITH THE GUEST USER ON AN OPERATOR-BASED INSTALLATION

For trial or non-production environments installed by using the Red Hat Developer Hub Operator, you can enable guest access to skip configuring authentication and authorization and explore Developer Hub features.

Prerequisites

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- You use the Red Hat Developer Hub Operator to run Developer Hub .

Procedure

- Add the following content to the **app-config.yaml** file:

```
auth:
  environment: development
  providers:
    guest:
      dangerouslyAllowOutsideDevelopment: true
```

Verification

1. Go to the Developer Hub login page.
2. To log in with the Guest user account, click **Enter** in the **Guest** tile.
3. In the Developer Hub **Settings** page, your profile name is **Guest**.
4. You can use Developer Hub features.

2.2. AUTHENTICATING WITH THE GUEST USER ON A HELM-BASED INSTALLATION

For trial or non-production environments installed by using the Red Hat Developer Hub Helm chart, you can enable guest access to skip configuring authentication and authorization and explore Developer Hub features.

Prerequisites

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- You use the Red Hat Developer Hub Helm chart to run Developer Hub .

Procedure

- Add following content to your Red Hat Developer Hub Helm Chart:

```
upstream:
  backstage:
    appConfig:
      app:
        baseUrl: 'https://{{- include "janus-idp.hostname" . }}'
      auth:
        environment: development
        providers:
          guest:
            dangerouslyAllowOutsideDevelopment: true
```

Verification

1. Go to the Developer Hub login page.
2. To log in with the Guest user account, click **Enter** in the **Guest** tile.
3. In the Developer Hub **Settings** page, your profile name is **Guest**.
4. You can use Developer Hub features.

CHAPTER 3. AUTHENTICATING WITH RED HAT BUILD OF KEYCLOAK (RHBK)

3.1. ENABLING USER AUTHENTICATION WITH RED HAT BUILD OF KEYCLOAK (RHBK), WITH OPTIONAL STEPS

Authenticate users with Red Hat Build of Keycloak (RHBK), by provisioning the users and groups from RHBK to the Developer Hub software catalog, and configuring the OpenID Connect (OIDC) authentication provider in Red Hat Developer Hub.

Prerequisites

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- You have enough permissions in RHSSO to create and manage a realm and a client.

  TIP

  Alternatively, ask your RHBK administrator to prepare in RHBK the required realm and client.

Procedure

1. Register your Developer Hub app in RHBK:
   a. Use an existing realm, or create a realm, with a distinctive **Name** such as *<my_realm>*. Save the value for the next step:
      - **RHBK realm base URL**, such as: *<your_rhbk_URL>*/realms/*<your_realm>*.
   b. To register your Developer Hub in RHBK, in the created realm, secure the first application, with:
      i. **Client ID**: A distinctive client ID, such as *<RHDH>*.
      ii. **Valid redirect URIs**: Set to the OIDC handler URL: **https://*<my_developer_hub_domain>*/api/auth/oidc/handler/frame**.
      iii. Go to the **Credentials** tab and copy the **Client secret**.
      iv. Save the values for the next step:
         - **Client ID**
         - **Client Secret**
   c. To prepare for the verification steps, in the same realm, get the credential information for an existing user or create a user. Save the user credential information for the verification steps.
2. Add your RHSSO credentials to Developer Hub, by adding the following key/value pairs to your Developer Hub secrets. You can use these secrets in the Developer Hub configuration files by using their environment variable name.

   KEYCLOAK_CLIENT_ID

Enter the saved **Client ID**.

KEYCLOAK_CLIENT_SECRET

Enter the saved **Client Secret**.

KEYCLOAK_BASE_URL

Enter the saved **RHBK realm base URL**

KEYCLOAK_REALM

Enter the realm name to provision users.

KEYCLOAK_LOGIN_REALM

Enter the realm name to authenticate users.

3. Enable the Keycloak catalog provider plugin in your **dynamic-plugins.yaml** file.
The plugin is named after RHBK upstream project.

   This plugin imports RHBK users and groups to the Developer Hub software catalog.

```
plugins:
  - package: './dynamic-plugins/dist/backstage-community-plugin-catalog-backend-module-keycloak-dynamic'
    disabled: false
```

4. Enable provisioning RHBK users and groups to the Developer Hub software catalog, by adding the **catalog.providers.keycloakOrg** section to your **app-config.yaml** file:

```
catalog:
  providers:
    keycloakOrg:
      default:
        baseUrl: ${KEYCLOAK_BASE_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        realm: ${KEYCLOAK_REALM}
        loginRealm: ${KEYCLOAK_LOGIN_REALM}
```

baseUrl

Enter your RHBK server URL, defined earlier.

clientId

Enter your Developer Hub application client ID in RHBK, defined earlier.

clientSecret

Enter your Developer Hub application client secret in RHBK, defined earlier.

realm

Enter the realm name to provision users.

loginRealm

Enter the realm name to authenticate users.

5. Optional: Add optional fields to the **keycloackOrg** catalog provider section in your **app-config.yaml** file:

```
catalog:
```

```
  providers:
    keycloakOrg:
      default:
        baseUrl: ${KEYCLOAK_BASE_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        realm: ${KEYCLOAK_REALM}
        loginRealm: ${KEYCLOAK_LOGIN_REALM}
        userQuerySize: 100
        groupQuerySize: 100
        schedule:
          frequency: { hours: 1 }
          timeout: { minutes: 50 }
          initialDelay: { seconds: 15}
```

userQuerySize

Enter the user count to query simultaneously. Default value: **100**.

groupQuerySize

Enter the group count to query simultaneously. Default value: **100**.

schedule

frequency

Enter the schedule frequency. Supports cron, ISO duration, and "human duration" as used in code.

timeout

Enter the timeout for the user provisioning job. Supports ISO duration and "human duration" as used in code.

initialDelay

Enter the initial delay to wait for before starting the user provisioning job. Supports ISO duration and "human duration" as used in code.

6. Enable the RHBK authentication provider, by adding the OIDC provider section in your **app-config.yaml** file:

```
auth:
  environment: production
  providers:
    oidc:
      production:
        metadataUrl: ${KEYCLOAK_BASE_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        prompt: auto
signInPage: oidc
```

environment: production

Mark the environment as **production** to hide the Guest login in the Developer Hub home page.

metadataUrl, clientId, clientSecret

Configure the OIDC provider with your secrets.

prompt

Enter **auto** to allow the identity provider to automatically determine whether to prompt for credentials or bypass the login redirect if an active RHSSO session exists.
The identity provider defaults to **none**, which assumes that you are already logged in. Sign-in requests without an active session are rejected.

signInPage

Enter **oidc** to enable the OIDC provider as default sign-in provider.

7. Optional: Add optional fields to the OIDC authentication provider section in your **app-config.yaml** file:

```
auth:
  providers:
    oidc:
      production:
        metadataUrl: ${KEYCLOAK_BASE_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        callbackUrl: ${KEYCLOAK_CALLBACK_URL}
        tokenEndpointAuthMethod: ${KEYCLOAK_TOKEN_ENDPOINT_METHOD}
        tokenSignedResponseAlg: ${KEYCLOAK_SIGNED_RESPONSE_ALG}
        additionalScopes: ${KEYCLOAK_SCOPE}
        signIn:
          resolvers:
            - resolver: oidcSubClaimMatchingKeycloakUserId
            - resolver: preferredUsernameMatchingUserEntityName
            - resolver: emailMatchingUserEntityProfileEmail
            - resolver: emailLocalPartMatchingUserEntityName
              dangerouslyAllowSignInWithoutUserInCatalog: true
        sessionDuration: { hours: 24 }
    backstageTokenExpiration: { minutes: _<user_defined_value>_ }
  signInPage: oidc
```

callbackUrl

RHBK callback URL.

tokenEndpointAuthMethod

Enter your token endpoint authentication method.

tokenSignedResponseAlg

Token signed response algorithm.

additionalScopes

Enter additional RHBK scopes to request for during the authentication flow.

signIn

resolvers

After successful authentication, the user signing in must be resolved to an existing user in the Developer Hub catalog. To best match users securely for your use case, consider configuring a specific resolver.
Enter the resolver list to override the default resolver: **oidcSubClaimMatchingKeycloakUserId**.

Available values:

oidcSubClaimMatchingKeycloakUserId

Matches the user with the immutable **sub** parameter from OIDC to the RHBK user ID. Consider using this resolver for enhanced security.

emailLocalPartMatchingUserEntityName

Matches the email local part with the user entity name.

emailMatchingUserEntityProfileEmail

Matches the email with the user entity profile email.

preferredUsernameMatchingUserEntityName

Matches the preferred username with the user entity name.
The authentication provider tries each sign-in resolver in order until it succeeds, and fails if none succeed.

WARNING

In production mode, configure only one resolver to make sure users are securely matched.

dangerouslyAllowSignInWithoutUserInCatalog: true

Configure the sign-in resolver to bypass the user provisioning requirement in the Developer Hub software catalog.

WARNING

In production mode, do not enable the **dangerouslyAllowSignInWithoutUserInCatalog** option.

sessionDuration

Lifespan of the user session. Enter a duration in **ms** library format (such as '24h', '2 days'), ISO duration, or "human duration" as used in code.

backstageTokenExpiration

Enter a value to modify the Developer Hub token expiration from its default value of one hour. It refers to the validity of short-term cryptographic tokens, not to the session duration. The expiration value must be set between 10 minutes and 24 hours.

WARNING

If multiple valid refresh tokens are issued due to frequent refresh token requests, older tokens will remain valid until they expire. Enhance security and prevent potential misuse of older tokens by enabling a refresh token rotation strategy in your RHBK realm.

1. From the **Configure** section of the navigation menu, click **Realm Settings**.
2. From the **Realm Settings** page, click the **Tokens** tab.
3. From the **Refresh tokens** section of the **Tokens** tab, toggle the **Revoke Refresh Token** to the **Enabled** position.

Verification

1. To verify user and group provisioning, check the console logs.
   Successful synchronization example:

```
2025-06-27T16:02:34.647Z catalog info Read 5 Keycloak users and 3 Keycloak groups in 0.4 seconds. Committing... class="KeycloakOrgEntityProvider" taskId="KeycloakOrgEntityProvider:default:refresh" taskInstanceId="db55c34b-46b3-402b-b12f-2fbc48498e82" trace_id="606f80a9ce00d1c86800718c4522f7c6" span_id="7ebc2a254a546e90" trace_flags="01"

2025-06-27T16:02:34.650Z catalog info Committed 5 Keycloak users and 3 Keycloak groups in 0.0 seconds. class="KeycloakOrgEntityProvider" taskId="KeycloakOrgEntityProvider:default:refresh" taskInstanceId="db55c34b-46b3-402b-b12f-2fbc48498e82" trace_id="606f80a9ce00d1c86800718c4522f7c6" span_id="7ebc2a254a546e90" trace_flags="01"
```

2. To verify RHBK user authentication:
   a. Go to the Developer Hub login page.
   b. Your Developer Hub sign-in page displays **Sign in using OIDC** and the Guest user sign-in is disabled.
   c. Log in with OIDC by using the saved **Username** and **Password** values.

3.2. ENABLING USER PROVISIONING WITH LDAP

When Red Hat Build of Keycloak (RHBK) depends on Lightweight Directory Access Protocol (LDAP) to resolve user and group identities, you can opt to provision users and groups from LDAP directly to the Red Hat Developer Hub software catalog, rather than using the RHBK provisioning mechanism.

Prerequisites

- You have configured authentication with Red Hat Build of Keycloak (RHBK) .
- You have collected the required LDAP credentials:

  LDAP URL

  Your LDAP server URL, such as **ldaps://ds.example.net**.

  Bind dn

  Your bind distinguished name, such as **cn=admin,OU=Users,DC=rhdh,DC=test**

  LDAP secret

  Your LDAP secret.

  Recommended: LDAP certificates and keys

  To use a secure LDAP connexion (**ldaps://**): you stored your LDAP certificates and keys respectively in the **ldap_certs.pem** and **ldap_keys.pem** files.

WARNING

In production mode, use a secure LDAP connexion.

Procedure

1. Enter your LDAP credentials to Developer Hub, by adding the **LDAP_SECRET** environment variable to your Developer Hub secrets .

```
$ oc patch secret my-rhdh-secrets --patch '{"stringData": { "LDAP_SECRET": "
<ldap_secret>" }}'
```

   <ldap_secret>

   Enter your LDAP secret.

2. Recommended: To use a secure LDAP connection (**ldaps://**), add your LDAP certificates and keys files to a {a-platform-generic} secret.

```
$ oc create secret generic my-rhdh-ldap-secrets \
    --from-file=./ldap_certs.pem \
    --from-file=./ldap_keys.pem
```

3. Enable the LDAP catalog provider plugin in your **dynamic-plugins.yaml** file.

```
plugins:
  - package: './dynamic-plugins/dist/backstage-plugin-catalog-backend-module-ldap-dynamic'
    disabled: false
```

4. Enable provisioning GitHub users and groups to the Developer Hub software catalog, by adding the LDAP catalog provider section to your **app-config.yaml** file:

   a. Optional: Remove other catalog providers, by removing the other catalog providers section.

b. Enter the mandatory fields:

```
catalog:
  providers:
    ldapOrg:
      default:
        target: ldaps://ds.example.net
        bind:
          dn: cn=admin,ou=Users,dc=rhdh
          secret: ${LDAP_SECRET}
        users:
          - dn: OU=Users,OU=RHDH Local,DC=rhdh,DC=test
            options:
              filter: (uid=*)
        groups:
          - dn: OU=Groups,OU=RHDH Local,DC=rhdh,DC=test
        schedule:
          frequency: PT1H
          timeout: PT15M
```

target

Enter your LDAP server URL, such as **ldaps://ds.example.net**.

bind

Enter your service account information:

dn

Enter your service account distinguished name (DN), such as **cn=admin,OU=Users,DC=rhdh,DC=test**

secret

Enter the name of the variable containing your LDAP secret: **${LDAP_SECRET}**.

users

Enter information about how to find your users:

dn

Enter the DN containing the user information.

options

filter

Enter your filter, such as **(uid=*)** to provision to the RHDH software catalog only users with an existing **uid**.

groups

Enter information about how to find your groups:

dn

Enter the DN containing the group information.

schedule

Enter your schedule information:

frequency

Enter your schedule frequency, in the cron, ISO duration, or "human duration" format.

timeout

Enter your schedule timeout, in the ISO duration or "human duration" format.

initialDelay

Enter your schedule initial delay, in the ISO duration or "human duration" format.

c. Optional: To change how Developer Hub maps LDAP user fields to the software catalog, enter optional **maps** and **set** fields.

```
catalog:
  providers:
    ldapOrg:
      default:
        target: ldaps://ds.example.net
        bind:
          dn: cn=admin,ou=Users,dc=rhdh
          secret: ${LDAP_SECRET}
        users:
          - dn: OU=Users,OU=RHDH Local,DC=rhdh,DC=test
            options:
              filter: (uid=*)
            map:
              rdn: uid
              name: uid
              description: {}
              displayName: cn
              email: mail
              picture: {}
              memberOf: memberOf
            set:
              metadata.customField: 'hello'
        groups:
          - dn: OU=Groups,OU=RHDH Local,DC=rhdh,DC=test
        schedule:
          frequency: PT1H
          timeout: PT15M
```

rdn

To change the default value: **uid**, enter the relative distinguished name of each entry.

name

To change the default value: **uid**, enter the LDAP field to map to the RHDH **metadata.name** field.

description

To set a value, enter the LDAP field to map to the RHDH **metadata.description** field.

displayName

To change the default value: **cn**, enter the LDAP field to map to the RHDH **metadata.displayName** field.

email

To change the default value: **mail**, enter the LDAP field to map to the RHDH **spec.profile.email** field.

picture

To set a value, enter the LDAP field to map to the RHDH **spec.profile.picture** field.

memberOf

To change the default value: **memberOf**, enter the LDAP field to map to the RHDH **spec.memberOf** field.

set

To set a value, enter the hard coded JSON to apply to the entities after ingestion, such as **metadata.customField: 'hello'**.

d. Optional: To change how Developer Hub maps LDAP group fields to the software catalog, enter optional **groups.maps** fields.

```
catalog:
  providers:
    ldapOrg:
      default:
        target: ldaps://ds.example.net
        bind:
          dn: cn=admin,ou=Users,dc=rhdh
          secret: ${LDAP_SECRET}
        users:
          - dn: OU=Users,OU=RHDH Local,DC=rhdh,DC=test
            options:
              filter: (uid=*)
        groups:
          - dn: OU=Groups,OU=RHDH Local,DC=rhdh,DC=test
            map:
              rdn: uid
              name: uid
              description: {}
              displayName: cn
              email: mail
              picture: {}
              memberOf: memberOf
              members: member
              type: groupType
            set:
              metadata.customField: 'hello'
        schedule:
          frequency: PT1H
          timeout: PT15M
```

rdn

To change the default value: **cn**, enter the relative distinguished name of each entry.

name

To change the default value: **cn**, enter the LDAP field to map to the RHDH **metadata.name** field.

description

To set a value, enter the LDAP field to map to the RHDH **metadata.description** field.

displayName

To change the default value: **cn**, enter the LDAP field to map to the RHDH **metadata.displayName** field.

email

To change the default value: **mail**, enter the LDAP field to map to the RHDH **spec.profile.email** field.

picture

To set a value, enter the LDAP field to map to the RHDH **spec.profile.picture** field.

memberOf

To change the default value: **memberOf**, enter the LDAP field to map to the RHDH **spec.memberOf** field.

members

To change the default value: **member**, enter the LDAP field to map to the RHDH **spec.children** field.

type

To change the default value: **groupType**, enter the LDAP field to map to the RHDH **spec.type** field.

set

To set a value, enter the hard coded JSON to apply to the entities after ingestion, such as **metadata.customField: 'hello'**.

e. Recommended: To use a secure LDAP connection (**ldaps://**), enter optional **tls** fields.

Optional tls fields

```
catalog:
  providers:
    ldapOrg:
      default:
        target: ldaps://ds.example.net
        bind:
          dn: cn=admin,ou=Users,dc=rhdh
          secret: ${LDAP_SECRET}
        users:
ldapOrg:
  default:
    tls:
      rejectUnauthorized: true
      keys: '/path/to/keys.pem'
      certs: '/path/to/certs.pem'
```

rejectUnauthorized

Set to **false** to allow self-signed certificates

WARNING

This option is not recommended for production.

keys

Enter a file containing private keys in PEM format

certs

Enter a file containing cert chains in PEM format

f. Optional: Enter configuration for vendor-specific attributes to set custom attribute names for distinguished names (DN) and universally unique identifiers (UUID) in LDAP directories. Default values are defined per supported vendor and automatically detected.

```
catalog:
  providers:
    ldapOrg:
      default:
        vendor:
          dnAttributeName: customDN
          uuidAttributeName: customUUID
```

dnAttributeName

Enter the attribute name that holds the distinguished name (DN) for an entry.

uuidAttributeName

Enter the attribute name that holds a universal unique identifier (UUID) for an entry.

g. Optional: Enter low level users and groups configuration in the **options** subsection.

```
catalog:
  providers:
    ldapOrg:
      default:
        target: ldaps://ds.example.net
        bind:
          dn: cn=admin,ou=Users,dc=rhdh
          secret: ${LDAP_SECRET}
        users:
          options:
            scope: sub
            filter: (uid=*)
            attributes:
              - cn
              - uid
              - description
            paged:
            pageSize: 500
        groups:
          options:
```

```
      scope: sub
      filter: (cn=*)
      attributes:
        - cn
        - uid
        - description
      paged:
        pageSize: 500
        pagePause: true
```

scope

To change the default value: **one**, enter how deep the search should go within the directory tree:

- **base** to search only the base DN.
- **one** to search one level below the base DN.
- **sub** to search all descendant entries.

filter

To change the default value: **(objectclass=*)**, enter your LDAP filter. With the default mapping:

- For users, enter **(uid=*)** to make sure only users with valid uid field is synced, since users without uid will cause error and ingestion fails.
- For groups, enter **(cn=*)**

TIP

When you change the mapping, also update the filter.

attributes

To change the default value: all attributes **['*', '+']**, enter the array of attribute names to import from LDAP.

paged

Enter a value to enable paged results.

pageSize

Enter a value to set the results page size, such as **500**.

pagePause

Enter **true** to tell the client to wait for the asynchronous results of the next page, when the page limit has been reached.

5. Recommended: To use a secure LDAP connection (**ldaps://**), mount your LDAP certificates and keys files in your Developer Hub deployment, by editing your Backstage custom resource.

```
kind: Backstage
spec:
  application:
```

```
    extraFiles:
      mountPath: /opt/ldap-secrets
      secrets:
        - name: my-rhdh-database-database-secrets
          key: ldap-certs.pem, ldap-keys.pem
```

Verification

- To verify user and group provisioning, check the console logs.
  Successful synchronization example:

```
2025-10-15T20:45:49.072Z catalog info Read 4 LDAP users and 6 LDAP groups in 0.3
seconds. Committing... class="LdapOrgEntityProvider"
taskId="LdapOrgEntityProvider:default:refresh" taskInstanceId="9bb48fd5-2f55-4096-9fd0-
61cee6679952" trace_id="6a318e2eadba84e20df773948668aa4c"
span_id="cbec568cb6e64985" trace_flags="01"
2025-10-15T20:45:49.075Z catalog info Committed 4 LDAP users and 6 LDAP groups in 0.0
seconds. class="LdapOrgEntityProvider" taskId="LdapOrgEntityProvider:default:refresh"
taskInstanceId="9bb48fd5-2f55-4096-9fd0-61cee6679952"
trace_id="6a318e2eadba84e20df773948668aa4c" span_id="cbec568cb6e64985"
trace_flags="01"
```

3.3. CREATING A CUSTOM TRANSFORMER TO PROVISION USERS FROM RED HAT BUILD OF KEYCLOAK (RHBK) TO THE SOFTWARE CATALOG

Customize how Red Hat Developer Hub provisions users and groups to Red Hat Developer Hub software catalog entities, by creating a backend module that uses the **keycloakTransformerExtensionPoint** to offer custom user and group transformers for the Keycloak backend.

Prerequisites

- You have enabled provisioning users from Red Hat Build of Keycloak (RHBK) to the software catalog.

Procedure

1. Create a new backend module with the **yarn new** command.
2. Add your custom user and group transformers to the **keycloakTransformerExtensionPoint**. The following is an example **plugins/*<module_name>*/src/module.ts** file defining the backend module:

```
import {
  GroupTransformer,
  keycloakTransformerExtensionPoint,
  UserTransformer,
} from '@backstage-community/plugin-catalog-backend-module-keycloak';

const customGroupTransformer: GroupTransformer = async (
  entity, // entity output from default parser
  realm, // Keycloak realm name
```

```
  groups, // Keycloak group representation
) => {
  /* apply transformations */
  return entity;
};
const customUserTransformer: UserTransformer = async (
  entity, // entity output from default parser
  user, // Keycloak user representation
  realm, // Keycloak realm name
  groups, // Keycloak group representation
) => {
  /* apply transformations */
  return entity;
};

export const keycloakBackendModuleTransformer = createBackendModule({
  pluginId: 'catalog',
  moduleId: 'keycloak-transformer',
  register(reg) {
    reg.registerInit({
      deps: {
        keycloak: keycloakTransformerExtensionPoint,
      },
      async init({ keycloak }) {
        keycloak.setUserTransformer(customUserTransformer);
        keycloak.setGroupTransformer(customGroupTransformer);
        /* highlight-add-end */
      },
    });
  },
});
```

IMPORTANT

Set the module's **pluginId** to **catalog** to match the **pluginId** of the **keycloak-backend**; otherwise, the module fails to initialize.

3. Install this new backend module into your Developer Hub backend.

```
$ backend.add(import(backstage-plugin-catalog-backend-module-keycloak-transformer))
```

Verification

- Developer Hub imports the users and groups each time when started. Check the console logs to verify the synchronization result.
  Successful synchronization example:

```
{"class":"KeycloakOrgEntityProvider","level":"info","message":"Read 3 Keycloak users and 2 Keycloak groups in 1.5 seconds.
Committing...","plugin":"catalog","service":"backstage","taskId":"KeycloakOrgEntityProvider:default:refresh","taskInstanceId":"bf0467ff-8ac4-4702-911c-380270e44dea","timestamp":"2024-09-25 13:58:04"}
{"class":"KeycloakOrgEntityProvider","level":"info","message":"Committed 3 Keycloak users and 2 Keycloak groups in 0.0
```

seconds.","plugin":"catalog","service":"backstage","taskId":"KeycloakOrgEntityProvider:default:refresh","taskInstanceId":"bf0467ff-8ac4-4702-911c-380270e44dea","timestamp":"2024-09-25 13:58:04"}

- After the first import is complete, go to the **Catalog** page and select **User** to view the list of users.
- When you select a user, you see the information imported from RHBK.
- You can select a group, view the list, and access or review the information imported from RHBK.
- You can log in with an RHBK account.

CHAPTER 4. ENABLING AUTHENTICATION WITH GITHUB

4.1. ENABLING USER AUTHENTICATION WITH GITHUB, WITH OPTIONAL STEPS

Authenticate users with GitHub by provisioning the users and groups from GitHub to the Developer Hub software catalog, and configuring the GitHub authentication provider in Red Hat Developer Hub.

Prerequisites

- You have enough permissions in GitHub to create and manage a GitHub App.

  TIP

  Alternatively, ask your GitHub administrator to prepare the required GitHub App.

- You have added a custom Developer Hub application configuration , and have enough permissions to change it.

Procedure

1. Allow Developer Hub to authenticate with GitHub, by creating a GitHub App.

   

   NOTE

   Use a GitHub App instead of an OAuth app to use fine-grained permissions, use short-lived tokens, scale with the number of installations by avoiding rate limits, and have a more transparent integration by avoiding to request user input.

   a. Register a GitHub App with the following configuration:

      GitHub App name

      Enter a unique name identifying your GitHub App, such as **authenticating-with-rhdh-*<GUID>***.

      Homepage URL

      Enter your Developer Hub URL: **https://*<my_developer_hub_domain>***.

      Authorization callback URL

      Enter your Developer Hub authentication backend URL: **https://*<my_developer_hub_domain>*/api/auth/github/handler/frame**.

      Webhook

      Clear "Active".

      Organization permissions

      Enable **Read-only** access to **Members**.

      Where can this GitHub App be installed?

      Select **Only on this account**.

   b. In the **General → Clients secrets** section, click **Generate a new client secret**

   c. In the **Install App** tab, choose an account to install your GitHub App on.

d. Save the following values for the next step:

- **Client ID**
- **Client secret**

2. Add your GitHub credentials to Developer Hub by adding the following key/value pairs to your Developer Hub secrets. You can use these secrets in the Developer Hub configuration files by using their environment variable name.

GITHUB_CLIENT_ID

Enter the saved **Client ID**.

GITHUB_CLIENT_SECRET

Enter the saved **Client Secret**.

GITHUB_URL

Enter the GitHub host domain: **https://github.com**.

GITHUB_ORG

Enter your GitHub organization name, such as ***<your_github_organization_name>***.

3. Enable the GitHub catalog provider plugin in your **dynamic-plugins.yaml** file to import GitHub users and groups to the Developer Hub software catalog.

```
plugins:
  - package: './dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic'
    disabled: false
```

4. Enable provisioning GitHub users and groups to the Developer Hub software catalog by adding the GitHub catalog provider section to your **app-config.yaml** file:

```
catalog:
  providers:
    githubOrg:
      id: githuborg
      githubUrl: "${GITHUB_URL}"
      orgs: [ "${GITHUB_ORG}" ]
      schedule:
        frequency:
          minutes: 30
        initialDelay:
          seconds: 15
        timeout:
          minutes: 15
```

id

Enter a stable identifier for this provider, such as **githuborg**.

Entities from this provider are associated with this identifier. Therefore, do not to change the identifier over time since that might lead to orphaned entities or conflicts.

githubUrl

Enter the configured secret variable name: **${GITHUB_URL}**.

orgs

Enter the configured secret variable name: **${GITHUB_ORG}**.

schedule.frequency

Enter your schedule frequency, in the cron, ISO duration, or "human duration" format.

schedule.timeout

Enter your schedule timeout, in the ISO duration or "human duration" format.

schedule.initialDelay

Enter your schedule initial delay, in the ISO duration or "human duration" format.

1. Enable the GitHub authentication provider, by adding the GitHub authentication provider section to your **app-config.yaml** file:

```
auth:
  environment: production
  providers:
    github:
      production:
        clientId: ${GITHUB_CLIENT_ID}
        clientSecret: ${GITHUB_CLIENT_SECRET}
signInPage: github
```

environment

Enter **production** to disable the Guest login option in the Developer Hub login page.

clientId

Enter the configured secret variable name: **${GITHUB_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${GITHUB_CLIENT_SECRET}**.

signInPage

Enter **github** to enable the GitHub provider as your Developer Hub sign-in provider.

1. Optional: Add optional fields to the GitHub authentication provider section in your **app-config.yaml** file:

```
auth:
  environment: production
  providers:
    github:
      production:
        clientId: ${GITHUB_CLIENT_ID}
        clientSecret: ${GITHUB_CLIENT_SECRET}
        callbackUrl: <your_intermediate_service_url/handler>
        sessionDuration: { hours: 24 }
        signIn:
          resolvers:
            - resolver: usernameMatchingUserEntityName
              dangerouslyAllowSignInWithoutUserInCatalog: true
signInPage: github
```

callbackUrl

Enter the callback URL that GitHub uses when initiating an OAuth flow, such as: *<your_intermediate_service_url/handler>*. Define it when Developer Hub is not the immediate receiver, such as in cases when you use one OAuth app for many Developer Hub instances.

sessionDuration

Enter the user session lifespan, in **ms** library format (such as '24h', '2 days'), ISO duration, or "human duration".

signIn

resolvers

After successful authentication, Developer Hub resolves the user signing in to an existing user in the Developer Hub catalog. Configure a specific resolver to best match users securely for your use case..

Enter the resolver list to override the default resolver: **usernameMatchingUserEntityName**.

The authentication provider tries each sign-in resolver in order until it succeeds. If none of the attempts succeed, the sign-in fails.

WARNING

In production mode, configure only one resolver to make sure users are securely matched.

resolver

Enter the sign-in resolver name. Available resolvers:

- **usernameMatchingUserEntityName**
- **preferredUsernameMatchingUserEntityName**
- **emailMatchingUserEntityProfileEmail**

dangerouslyAllowSignInWithoutUserInCatalog

Enter **true** to configure the sign-in resolver to bypass the user provisioning requirement in the Developer Hub software catalog.

WARNING

In production more, do not enable **dangerouslyAllowSignInWithoutUserInCatalog**.

Verification

1. Verify user and group provisioning by checking the console logs.
   Successful synchronization example:


```
{"class":"GithubMultiOrgEntityProvider","level":"info","message":"Reading GitHub users and teams for org: rhdh-dast","plugin":"catalog","service":"backstage","target":"https://github.com","taskId":"GithubMultiOrgEntityProvider:githuborg:refresh","taskInstanceId":"801b3c6c-167f-473b-b43e-e0b4b780c384","timestamp":"2024-09-09 23:55:58"}
{"class":"GithubMultiOrgEntityProvider","level":"info","message":"Read 7 GitHub users and 2 GitHub groups in 0.4 seconds.
Committing...","plugin":"catalog","service":"backstage","target":"https://github.com","taskId":"GithubMultiOrgEntityProvider:githuborg:refresh","taskInstanceId":"801b3c6c-167f-473b-b43e-e0b4b780c384","timestamp":"2024-09-09 23:55:59"}
```


2. To verify GitHub authentication:

   a. Go to the Developer Hub login page.

   b. Your Developer Hub sign-in page displays **Sign in using GitHub** and the Guest user sign-in is disabled.

   c. Log in with a GitHub account.

Additional resources

- Integrating Red Hat Developer Hub with GitHub

4.2. ENABLING USER AUTHENTICATION WITH GITHUB AS AN AUXILIARY AUTHENTICATION PROVIDER

If your primary authentication provider is not GitHub, users might lack the permissions needed for templates or plugins that require GitHub access. The recommended solution is to configure GitHub as an auxiliary authentication provider. This approach uses the primary provider for user identity management and the auxiliary provider to grant the necessary GitHub permissions, without re-resolving the user's identity.

Give users access to these features by configuring GitHub as an auxiliary authentication provider.

Prerequisites

- You have enough permissions in GitHub to create and manage a GitHub App.

  TIP

  Alternatively, ask your GitHub administrator to prepare the required GitHub App.

- You have added a custom Developer Hub application configuration , and have enough permissions to change it.
- You have configured a primary authentication provider to provision user and group identities to the Red Hat Developer Hub software catalog, and establish Developer Hub user sessions.

Procedure

1. Add the **auth.providers.github** section to your **app-config.yaml** file:

```
auth:
  providers:
    github:
      production:
        clientId: ${GITHUB_CLIENT_ID}
        clientSecret: ${GITHUB_CLIENT_SECRET}
        disableIdentityResolution: true
```

where: **clientId**:: Enter the configured secret variable name: **${GITHUB_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${GITHUB_CLIENT_SECRET}**.

disableIdentityResolution

Enter **true** to skip user identity resolution for this provider to enable sign-in from an auxiliary authentication provider.

WARNING

Do not enable this setting on the primary authentication provider you plan on using for sign-in and identity management.

Verification

1. Go to the Developer Hub login page.
2. Log in with your primary authentication provider account.
3. In the top user menu, go to **Settings** > **Authentication Providers**.
4. In the **GitHub** line, click the **Sign in** button and log in.
5. In the **GitHub** line, the button displays **Sign out**.

Additional resources

- Integrating Red Hat Developer Hub with GitHub

CHAPTER 5. ENABLING USER AUTHENTICATION WITH MICROSOFT AZURE, WITH OPTIONAL STEPS

Authenticate users with Microsoft Azure by provisioning the users and groups from Azure to the Developer Hub software catalog, and configuring the Azure authentication provider in Red Hat Developer Hub.

Prerequisites

- You have the permission to register an application in Azure.

  TIP

  Alternatively, ask your Azure administrator to prepare the required Azure application.

- You added a custom Developer Hub application configuration , and have enough permissions to change it.
- Your Developer Hub backend can access the following hosts:

  login.microsoftonline.com

  The Microsoft Azure authorization server, which enables the authentication flow.

  graph.microsoft.com

  The server for retrieving organization data, including user and group data, to import into the Developer Hub catalog.

Procedure

1. Register your Developer Hub app in Azure, by using the Azure portal.
   a. Sign in to the Microsoft Entra admin center.
   b. Optional: If you have access to multiple tenants, use the **Settings** icon in the top menu to switch to the tenant in which you want to register the application from the **Directories + subscriptions** menu.
   c. Browse to **Applications > App registrations**, and create a **New registration** with the configuration:

      Name

      Enter a name to identify your application in Azure, such as *<Authenticating with Developer Hub>*.

      Supported account types

      Select **Accounts in this organizational directory only**

      Redirect URI

      Select a platform

      Select **Web**.

      URL

      Enter the backend authentication URI set in Developer Hub:
      https://*<my_developer_hub_domain>*/api/auth/microsoft/handler/frame

d. On the **Applications > App registrations >*<Authenticating with Developer Hub>*> Manage > API permissions** page, **Add a Permission**, **Microsoft Graph**, select the following permissions:

Application Permissions

GroupMember.Read.All, User.Read.All

Enter permissions that enable provisioning user and groups to the Developer Hub software catalog.
Optional: **Grant admin consent** for these permissions. Even if your company does not require admin consent, consider doing so as it means users do not need to individually consent the first time they access Developer Hub.

Delegated Permissions

User.Read, email, offline_access, openid, profile

Enter permissions that enable authenticating users.
Optional: Enter optional custom scopes for the Microsoft Graph API that you define both here and in your **app-config.yaml** Developer Hub configuration file.

e. On the **Applications > App registrations >*<Authenticating with Developer Hub>*> Manage > Certificates & secrets** page, in the **Client secrets** tab, create a **New client secret**.

f. Save the following values for the next step:

- **Directory (tenant) ID**
- **Application (client) ID**
- **Application (client) Secret ID**

2. Add your Azure credentials to Developer Hub, by adding the following key/value pairs to your Developer Hub secrets:

MICROSOFT_TENANT_ID

Enter your saved **Directory (tenant) ID**.

MICROSOFT_CLIENT_ID

Enter your saved **Application (client) ID**.

MICROSOFT_CLIENT_SECRET

Enter your saved **Application (client) secret**.

3. Enable the Microsoft Graph catalog provider plugin in your **dynamic-plugins.yaml** file. This plugin imports Azure users and groups to the Developer Hub software catalog.

```
plugins:
  - package: './dynamic-plugins/dist/backstage-plugin-catalog-backend-module-msgraph-dynamic'
    disabled: false
```

4. Enable provisioning Azure users and groups to the Developer Hub software catalog, by adding the Microsoft Graph catalog provider section in your **app-config.yaml** file:

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        target: https://graph.microsoft.com/v1.0
        tenantId: ${MICROSOFT_TENANT_ID}
        clientId: ${MICROSOFT_CLIENT_ID}
        clientSecret: ${MICROSOFT_CLIENT_SECRET}
        schedule:
          frequency:
            hours: 1
          timeout:
            minutes: 50
          initialDelay:
            minutes: 50
```

target

Enter **https://graph.microsoft.com/v1.0** to define the MSGraph API endpoint the provider is connecting to. You might change this parameter to use a different version, such as the beta endpoint.

tenandId

Enter the configured secret variable name: **${MICROSOFT_TENANT_ID}**.

clientId

Enter the configured secret variable name: **${MICROSOFT_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${MICROSOFT_CLIENT_SECRET}**.

schedule

frequency

Enter the schedule frequency in the cron, ISO duration, or human duration format. In a large organization, user provisioning might take a long time, therefore avoid using a low value.

timeout

Enter the schedule timeout in the ISO duration or human duration format. In a large organization, user provisioning might take a long time, therefore avoid using a low value.

initialDelay

Enter the schedule initial delay in the ISO duration or human duration format.

5. Optional: Add optional fields to the Microsoft authentication provider section in your **app-config.yaml** file:

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        authority: https://login.microsoftonline.com/
        queryMode: advanced
        user:
          expand: manager
          filter: accountEnabled eq true and userType eq 'member'
```

```
        loadPhotos: true
        select: ['id', 'displayName', 'description']
      userGroupMember:
        filter: "displayName eq 'Backstage Users'"
        search: '"description:One" AND ("displayName:Video" OR "displayName:Drive")'
      group:
        expand: member
        filter: securityEnabled eq false and mailEnabled eq true and
groupTypes/any(c:c+eq+'Unified')
        search: '"description:One" AND ("displayName:Video" OR "displayName:Drive")'
        select: ['id', 'displayName', 'description']
```

authority

Enter your Azure authority URL if it is different from the default: **https://login.microsoftonline.com**.

queryMode

Enter **advanced** when the default **basic** query mode is insufficient for your queries to the Microsoft Graph API. See Microsoft Azure advanced queries.

user

Add this section to configure optional user query parameters.

expand

Enter your expansion parameter to include the expanded resource or collection referenced by a single relationship (navigation property) in your results. A single request can expand only one relationship. See Microsoft Graph query expand parameter.
You can combine this parameter with **userGroupMember.filter** or **user.filter**.

filter

Enter your user filter. See Microsoft Graph API and Microsoft Graph API query filter parameters syntax.
This parameter and **userGroupMember.filter** are mutually exclusive, specify only one.

loadPhotos

Developer Hub loads photos by default. Enter **false** to avoid loading user photos.

select

Enter the Microsoft Graph resource type list to retrieve.

userGroupMember

Add this section to use group membership to get users.

filter

Enter your filter to filter groups and fetch their members.
This parameter and **user.filter** are mutually exclusive, specify only one.

search

Enter your search query to search for groups and fetch their members.
This parameter and **user.filter** are mutually exclusive, specify only one.

group

Enter your configuration to get groups.

expand

Enter your expansion parameter to include the expanded resource or collection referenced by a single relationship (navigation property) in your results. A single request can expand only one relationship. See Customize Microsoft Graph responses with query parameters.
You can combine this parameter with **user.filter** or **userGroupMember.filter**.

filter

Enter your group filter parameter. See Microsoft Graph API query group syntax .

search

Enter your group search parameter. See Microsoft Graph API query search parameter.

select

Enter the Microsoft Graph resource type list to retrieve.

6. Enable Azure authentication, by adding the Microsoft authentication provider to your **app-config.yaml** file content:

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${MICROSOFT_CLIENT_ID}
        clientSecret: ${MICROSOFT_CLIENT_SECRET}
        tenantId: ${MICROSOFT_TENANT_ID}
signInPage: microsoft
```

environment

Enter **production** to disable the **Guest** login option in the Developer Hub login page.

clientId

Enter the configured secret variable name: **${MICROSOFT_CLIENT_ID}**.

clientSecret

Enter the configured secret variable name: **${MICROSOFT_CLIENT_SECRET}**.

tenantId

Enter the configured secret variable name: **${MICROSOFT_TENANT_ID}**.

signInPage

Enter **microsoft** to set the Azure provider as your Developer Hub sign-in provider.

7. Optional: Add optional fields to the Microsoft authentication provider section in your **app-config.yaml** file:

```
auth:
  environment: production
  providers:
    microsoft:
      production:
```

```
      clientId: ${MICROSOFT_CLIENT_ID}
      clientSecret: ${MICROSOFT_CLIENT_SECRET}
      tenantId: ${MICROSOFT_TENANT_ID}
      domainHint: ${MICROSOFT_TENANT_ID}
      additionalScopes:
        - Mail.Send
      sessionDuration:
        hours: 24
      signIn:
        resolvers:
          - resolver: usernameMatchingUserEntityName
            dangerouslyAllowSignInWithoutUserInCatalog: true
signInPage: microsoft
```

domainHint

- Leave this parameter empty, or enter the tenant ID when your application registration is single-tenant.
- Leave this parameter empty when your application registration is multi-tenant.
- Enter the tenant ID to reduce login friction for users with accounts in multiple tenants, by automatically filtering out accounts from other tenants.
  For more information, see Home Realm Discovery.

additionalScopes

Enter the list of additional scopes to add scopes for the application registration. The default and mandatory value lists following scopes:

- **openid**
- **offline_access**
- **profile**
- **email**
- **User.Read**

sessionDuration

Lifespan of the user session. Enter a duration in **ms** library (such as '24h', '2 days'), ISO duration, or "human duration" format.

signIn.resolvers

After successful authentication, Developer Hub resolves the user signing in to an existing user in the Developer Hub catalog. To best match users securely for your use case, consider configuring a specific resolver.
Enter the resolver list to override the default resolver: **userIdMatchingUserEntityAnnotation**.

The authentication provider tries each sign-in resolver in order until it succeeds, and fails if none succeed.

WARNING

In production mode, configure only one resolver to make sure users are securely matched.

resolver

Enter the sign-in resolver name. Available resolvers:

emailMatchingUserEntityAnnotation

Use this resolver to look up the user by matching their Microsoft email to the email entity annotation.

emailLocalPartMatchingUserEntityName

Use this resolver to look up the user by matching their Microsoft email user name to the user entity name.

emailMatchingUserEntityProfileEmail

Use this resolver to look up the user by matching their Microsoft email to the user entity profile email.

dangerouslyAllowSignInWithoutUserInCatalog

Enter **true** to configure the sign-in resolver to bypass the user provisioning requirement in the Developer Hub software catalog.

WARNING

In production mode, do not enable **dangerouslyAllowSignInWithoutUserInCatalog**.

Verification

1. To verify user and group provisioning, check the console logs for **MicrosoftGraphOrgEntityProvider** events.
   Successful synchronization example:

```
2025-06-23T13:37:55.804Z catalog info Read 9 msgraph users and 3 msgraph groups in 1.5 seconds. Committing... class="MicrosoftGraphOrgEntityProvider" taskId="MicrosoftGraphOrgEntityProvider:providerId:refresh" taskInstanceId="e104a116-6481-4ceb-9bc4-0f8f9581f959" trace_id="e4c633659cffd6b1529afa55a5bfbad7" span_id="76affd0420e8baa6" trace_flags="01"

2025-06-23T13:37:55.811Z catalog info Committed 9 msgraph users and 3 msgraph groups in 0.0 seconds. class="MicrosoftGraphOrgEntityProvider" taskId="MicrosoftGraphOrgEntityProvider:providerId:refresh" taskInstanceId="e104a116-
```

```
6481-4ceb-9bc4-0f8f9581f959" trace_id="e4c633659cffd6b1529afa55a5bfbad7"
span_id="76affd0420e8baa6" trace_flags="01"
```

2. To verify Azure user authentication:

   a. Go to the Developer Hub login page.

   b. Your Developer Hub sign-in page displays **Sign in using Microsoft** and the Guest user sign-in is disabled.

   c. Log in with an Azure account.

CHAPTER 6. TROUBLESHOOTING AUTHENTICATION ISSUES

Learn how to troubleshoot authentication issues.

6.1. REDUCING THE SIZE OF ISSUED TOKENS

By default, the authentication backend issues user identity tokens with ownership references of the user in the **ent** claim of the JSON Web Token (JWT) payload. This makes it easier for consumers of the token to resolve ownership of the user. However, depending on the structure of your organization and how you resolve ownership claims, the tokens can grow large and cause HTTP errors that prevent you from accessing parts of RHDH. Use the **omitIdentityTokenOwnershipClaim** flag to remove the **ent** claim from tokens and reduce their size.

Procedure

1. In the **app-config.yaml** file, set **omitIdentityTokenOwnershipClaim** to **true** as follows:

```
auth:
  omitIdentityTokenOwnershipClaim: true
```