Red Hat Developer Hub 1.3

Authentication

Configuring authentication to external services in Red Hat Developer Hub

Last Updated: 2024-10-18

Red Hat Developer Hub 1.3 Authentication

Configuring authentication to external services in Red Hat Developer Hub

Legal Notice

Abstract

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

Table of Contents

PREFACE

Depending on your organization's security policies, you might require to identify and authorize users before giving them access to resources, such as Red Hat Developer Hub.

In Developer Hub, authentication and authorization are two separate processes:

1. Authentication defines the user identity, and passes on this information to Developer Hub. Read the following chapters to configure authentication in Developer Hub.
2. Authorization defines what the authenticated identity can access or do in Developer Hub. See Authorization.

NOT RECOMMENDED FOR PRODUCTION

To explore Developer Hub features, you can enable the guest user to skip configuring authentication and authorization, log in as the guest user, and access all the features.

The authentication system in Developer Hub is handled by external authentication providers.

Developer Hub supports following authentication providers:

- Red Hat Single-Sign On (RHSSO)
- GitHub
- Microsoft Azure

To identify users in Developer Hub, configure:

- One (and only one) authentication provider for sign-in and identification.
- Optionally, additional authentication providers for identification, to add more information to the user identity, or enable access to additional external resources.

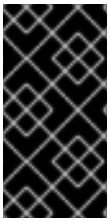
For each authentication provider, set up the shared secret that the authentication provider and Developer Hub require to communicate, first in the authentication provider, then in Developer Hub.

Developer Hub stores user identity information in the Developer Hub software catalog.

NOT RECOMMENDED FOR PRODUCTION

To explore the authentication system and use Developer Hub without authorization policies, you can bypass the Developer Hub software catalog and start using Developer Hub without provisioning the Developer Hub software catalog.

To get, store, and update additional user information, such as group or team ownership, with the intention to use this data to define authorization policies, provision users and groups in the Developer Hub software catalog.



IMPORTANT

Developer Hub uses a one-way synchronization system to provision users and groups from your authentication system to the Developer Hub software catalog. Therefore, deleting users and groups by using Developer Hub Web UI or REST API might have unintended consequences.

CHAPTER 1. AUTHENTICATING WITH THE GUEST USER

To explore Developer Hub features, you can skip configuring authentication and authorization. You can configure Developer Hub to log in as a Guest user and access Developer Hub features.

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

After an Operator-based installation, you can configure Developer Hub to log in as a Guest user and access Developer Hub features.

Prerequisites

- You installed Developer Hub by using the Operator .
- You added a custom Developer Hub application configuration , and have sufficient permissions to modify it.

Procedure

- To enable the guest user in your Developer Hub custom configuration, edit your Developer Hub application configuration with following content:

  app-config-rhdh.yaml fragment

```
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.

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

On a Helm-based installation, you can configure Developer Hub to log in as a Guest user and access Developer Hub features.

Prerequisites

- You Installed Developer Hub by using the Helm Chart .

Procedure

- To enable the guest user in your Developer Hub custom configuration, configure your Red Hat Developer Hub Helm Chart with following content:

  Red Hat Developer Hub Helm Chart configuration fragment

```
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 2. AUTHENTICATING WITH RED HAT SINGLE SIGN-ON (RHSSO)

To authenticate users with Red Hat Single Sign-On (RHSSO):

1. Enable the OpenID Connect (OIDC) authentication provider in RHDH .
2. Provision users from Red Hat Single-Sign On (RHSSO) to the software catalog .

2.1. ENABLING AUTHENTICATION WITH RED HAT SINGLE-SIGN ON (RHSSO)

To authenticate users with Red Hat Single Sign-On (RHSSO), enable the OpenID Connect (OIDC) authentication provider in Red Hat Developer Hub.

Prerequisites

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

Procedure

1. To allow Developer Hub to authenticate with RHSSO, complete the steps in RHSSO, to create a realm and a user and register the Developer Hub application :
   a. Use an existing realm, or create a realm, with a distinctive **Name** such as *<my_realm>*. Save the value for the next step:
      - **RHSSO realm base URL**, such as: *<your_rhsso_URL>*/auth/realms/*<your_realm>*.
   b. To register your Developer Hub in RHSSO, in the created realm, create a Client ID , with:
      i. **Client ID**: A distinctive client ID, such as *<RHDH>*.
      ii. **Valid redirect URIs**: Set to the OIDC handler URL: **https://*<RHDH_URL>*/api/auth/oidc/handler/frame**.
      iii. Navigate 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. To add your RHSSO credentials to your Developer Hub secrets, edit your Developer Hub secrets, such as **secrets-rhdh**, and add the following key/value pairs:

   AUTH_OIDC_CLIENT_ID

Enter the saved **Client ID**.

AUTH_OIDC_CLIENT_SECRET

Enter the saved **Client Secret**.

AUTH_OIDC_METADATA_URL

Enter the saved **RHSSO realm base URL**

3. To set up the RHSSO authentication provider in your Developer Hub custom configuration, edit your custom Developer Hub ConfigMap such as **app-config-rhdh**, and add the following lines to the **app-config-rhdh.yaml** content:

app-config-rhdh.yaml fragment with mandatory fields to enable authentication with RHSSO

```
auth:
  environment: production
  providers:
    oidc:
      production:
        metadataUrl: ${AUTH_OIDC_METADATA_URL}
        clientId: ${AUTH_OIDC_CLIENT_ID}
        clientSecret: ${AUTH_OIDC_CLIENT_SECRET}
signInPage: oidc
```

environment: production

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

metadataUrl, clientId, clientSecret

To configure the OIDC provider with your secrets.

signInPage: oidc

To enable the OIDC provider as default sign-in provider.

Optional: Consider adding the following optional fields:

dangerouslyAllowSignInWithoutUserInCatalog: true

To enable authentication without requiring to provision users in the Developer Hub software catalog.

WARNING

Use this option to explore Developer Hub features, but do not use it in production.

app-config-rhdh.yaml fragment with optional field to allow authenticating users absent from the software catalog

```
auth:
  environment: production
  providers:
    oidc:
      production:
        metadataUrl: ${AUTH_OIDC_METADATA_URL}
        clientId: ${AUTH_OIDC_CLIENT_ID}
        clientSecret: ${AUTH_OIDC_CLIENT_SECRET}
signInPage: oidc
dangerouslyAllowSignInWithoutUserInCatalog: true
```

callbackUrl

RHSSO callback URL.

app-config-rhdh.yaml fragment with optional callbackURL field

```
auth:
  providers:
    oidc:
      production:
        callbackUrl: ${AUTH_OIDC_CALLBACK_URL}
```

tokenEndpointAuthMethod

Token endpoint authentication method.

app-config-rhdh.yaml fragment with optional tokenEndpointAuthMethod field

```
auth:
  providers:
    oidc:
      production:
        tokenEndpointAuthMethod: ${AUTH_OIDC_TOKEN_ENDPOINT_METHOD}
```

tokenSignedResponseAlg

Token signed response algorithm.

app-config-rhdh.yaml fragment with optional tokenSignedResponseAlg field

```
auth:
  providers:
    oidc:
      production:
        tokenSignedResponseAlg: ${AUTH_OIDC_SIGNED_RESPONSE_ALG}
```

scope

RHSSO scope.

app-config-rhdh.yaml fragment with optional scope field

```
auth:
  providers:
    oidc:
      production:
        scope: ${AUTH_OIDC_SCOPE}
```

signIn.resolvers

Declarative resolvers to override the default resolver: **emailLocalPartMatchingUserEntityName**. The authentication provider tries each sign-in resolver until it succeeds, and fails if none succeed.

app-config-rhdh.yaml fragment with optional callbackURL field

```
auth:
  providers:
    oidc:
      production:
        signIn:
          resolvers:
            - resolver: preferredUsernameMatchingUserEntityName
            - resolver: emailMatchingUserEntityProfileEmail
            - resolver: emailLocalPartMatchingUserEntityName
```

Verification

1. Go to the Developer Hub login page.
2. Your Developer Hub sign-in page displays **Sign in using OIDC** and the Guest user sign-in is disabled.
3. Log in with OIDC by using the saved **Username** and **Password** values.

2.2. PROVISIONING USERS FROM RED HAT SINGLE-SIGN ON (RHSSO) TO THE SOFTWARE CATALOG

Prerequisites

- You enabled authentication with RHSSO.

Procedure

- To enable RHSSO member discovery, edit your custom Developer Hub ConfigMap, such as **app-config-rhdh**, and add the following lines to the **app-config-rhdh.yaml** content:

  app-config.yaml fragment with mandatory keycloakOrg fields

  ```
  dangerouslyAllowSignInWithoutUserInCatalog: false
  catalog:
    providers:
      keycloakOrg:
        default:
  ```

```
      baseUrl: ${AUTH_OIDC_METADATA_URL}
      clientId: ${AUTH_OIDC_CLIENT_ID}
      clientSecret: ${AUTH_OIDC_CLIENT_SECRET}
```

dangerouslyAllowSignInWithoutUserInCatalog: false

Allow authentication only for users present in the Developer Hub software catalog.

baseUrl

Your RHSSO server URL, defined when enabling authentication with RHSSO.

clientId

Your Developer Hub application client ID in RHSSO, defined when enabling authentication with RHSSO.

clientSecret

Your Developer Hub application client secret in RHSSO, defined when enabling authentication with RHSSO.

Optional: Consider adding the following optional fields:

realm

Realm to synchronize. Default value: **master**.

app-config.yaml fragment with optional realm field

```
catalog:
  providers:
    keycloakOrg:
      default:
        realm: master
```

loginRealm

Realm used to authenticate. Default value: **master**.

app-config.yaml fragment with optional loginRealm field

```
catalog:
  providers:
    keycloakOrg:
      default:
        loginRealm: master
```

userQuerySize

User number to query simultaneously. Default value: **100**.

app-config.yaml fragment with optional userQuerySize field

```
catalog:
  providers:
    keycloakOrg:
      default:
        userQuerySize: 100
```

groupQuerySize

Group number to query simultaneously. Default value: **100**.

app-config.yaml fragment with optional groupQuerySize field

```
catalog:
  providers:
    keycloakOrg:
      default:
        groupQuerySize: 100
```

schedule.frequency

To specify custom schedule frequency. Supports cron, ISO duration, and "human duration" as used in code.

app-config.yaml fragment with optional schedule.frequency field

```
catalog:
  providers:
    keycloakOrg:
      default:
        schedule:
          frequency: { hours: 1 }
```

schedule.timeout

To specify custom timeout. Supports ISO duration and "human duration" as used in code.

app-config.yaml fragment with optional schedule.timeout field

```
catalog:
  providers:
    keycloakOrg:
      default:
        schedule:
          timeout: { minutes: 50 }
```

schedule.initialDelay

To specify custom initial delay. Supports ISO duration and "human duration" as used in code.

app-config.yaml fragment with optional schedule.initialDelay field

```
catalog:
  providers:
    keycloakOrg:
      default:
        schedule:
          initialDelay: { seconds: 15}
```

Verification

1. Check the console logs to verify that the synchronization is completed.

   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"}

2. Log in with an RHSSO account.

2.3. CREATING A CUSTOM TRANSFORMER TO PROVISION USERS FROM RED HAT SINGLE-SIGN ON (RHSSO) TO THE SOFTWARE CATALOG

To customize how RHSSO users and groups are mapped to Red Hat Developer Hub entities, you can create a backend module that uses the **keycloakTransformerExtensionPoint** to provide custom user and group transformers for the Keycloak backend.

Prerequisites

- You have enabled provisioning users from Red Hat Single-Sign On (RHSSO) 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 of how the backend module can be defined:

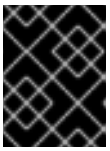
   plugins/*<module-name>*/src/module.ts

```
import {
  GroupTransformer,
  keycloakTransformerExtensionPoint,
  UserTransformer,
} from '@janus-idp/backstage-plugin-keycloak-backend';

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

The module's **pluginId** must be set 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 that the synchronization is completed.

  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, navigate to the **Catalog** page and select **User** to view the list of users.
- When you select a user, you see the information imported from RHSSO.
- You can select a group, view the list, and access or review the information imported from RHSSO.
- You can log in with an RHSSO account.

CHAPTER 3. ENABLING THE GITHUB AUTHENTICATION PROVIDER

To authenticate users with GitHub or GitHub Enterprise:

1. Enable the GitHub authentication provider in Developer Hub .
2. Provision users from GitHub to the software catalog .

3.1. ENABLING AUTHENTICATION WITH GITHUB

To authenticate users with GitHub, enable the GitHub authentication provider in Red Hat Developer Hub.

Prerequisites

- You have added a custom Developer Hub application configuration , and have sufficient permissions to modify it.
- You have sufficient permissions in GitHub to create and manage a GitHub App.

Procedure

1. To allow Developer Hub to authenticate with GitHub, create a GitHub App. Opt for a GitHub App instead of an OAuth app to use fine-grained permissions, gain more control over which repositories the application can access, and use short-lived tokens.
   a. Register a GitHub App with the following configuration:
      - **GitHub App name**: Enter a unique name identifying your GitHub App, such as *<Red Hat Developer Hub>-<GUID>*.
      - **Homepage URL**: Your Developer Hub URL: **https://*<my_developer_hub_url>***.
      - **Authorization callback URL**: Your Developer Hub authentication backend URL: **https://*<my_developer_hub_url>*/api/auth/github/handler/frame**.
      - **Webhook URL**: Your Developer Hub URL: **https://*<my_developer_hub_url>***.
      - **Webhook secret**: Provide a strong secret.
      - **Repository permissions**:
        - Enable **Read-only** access to:
          - **Administration**
          - **Commit statuses**
          - **Contents**
          - **Dependabot alerts**
          - **Deployments**
          - **Pull Requests**

- **Webhooks**

  TIP

  If you plan to make changes using the GitHub API, ensure that **Read and write** permissions are enabled instead of **Read-only**.

- Toggle other permissions as per your needs.

- **Organization permissions**:
  - Enable **Read-only** access to **Members**.
- For **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 **General → Private keys** section, click **Generate a private key**.

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

e. Save the following values for the next step:

- **App ID**
- **Client ID**
- **Client secret**
- **Private key**
- **Webhook secret**

2. To add your GitHub credentials to your Developer Hub secrets, edit your Developer Hub secrets, such as **secrets-rhdh**, and add the following key/value pairs:

AUTH_GITHUB_APP_ID

Enter the saved **App ID**.

AUTH_GITHUB_CLIENT_ID

Enter the saved **Client ID**.

GITHUB_HOST_DOMAIN

Enter your GitHub host domain: **https://github.com** unless you are using GitHub Enterprise.

GITHUB_ORGANIZATION

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

GITHUB_ORG_URL

Enter **$GITHUB_HOST_DOMAIN/$GITHUB_ORGANIZATION**.

GITHUB_CLIENT_SECRET

Enter the saved **Client Secret**.

GITHUB_PRIVATE_KEY_FILE

Enter the saved **Private key**.

GITHUB_WEBHOOK_URL

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

GITHUB_WEBHOOK_SECRET

Enter the saved **Webhook secret**.

3. To set up the GitHub authentication provider and enable integration with the GitHub API in your Developer Hub custom configuration, edit your custom Developer Hub ConfigMap such as **app-config-rhdh**, and add the following lines to the **app-config-rhdh.yaml** content:

app-config-rhdh.yaml fragment with mandatory fields to enable authentication with GitHub

```
auth:
  environment: production
  providers:
    github:
      production:
        clientId: ${AUTH_GITHUB_CLIENT_ID}
        clientSecret: ${AUTH_GITHUB_CLIENT_SECRET}
integrations:
  github:
    - host: ${GITHUB_HOST_DOMAIN}
      apps:
        - appId: ${AUTH_GITHUB_APP_ID}
          clientId: ${AUTH_GITHUB_CLIENT_ID}
          clientSecret: ${GITHUB_CLIENT_SECRET}
          webhookUrl: ${GITHUB_WEBHOOK_URL}
          webhookSecret: ${GITHUB_WEBHOOK_SECRET}
          privateKey: |
            ${GITHUB_PRIVATE_KEY_FILE}
signInPage: github
```

environment: production

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

clientId, clientSecret, host, appId, webhookUrl, webhookSecret, privateKey

Use the Developer Hub application information that you have created in GitHub and configured in OpenShift as secrets.

sigInPage: github

To enable the GitHub provider as default sign-in provider.

Optional: Consider adding the following optional fields:

dangerouslyAllowSignInWithoutUserInCatalog: true

To enable authentication without requiring to provision users in the Developer Hub software catalog.

WARNING

Use **dangerouslyAllowSignInWithoutUserInCatalog** to explore Developer Hub features, but do not use it in production.

app-config-rhdh.yaml fragment with optional field to allow authenticating users absent from the software catalog

```
auth:
  environment: production
  providers:
    github:
      production:
        clientId: ${AUTH_GITHUB_CLIENT_ID}
        clientSecret: ${AUTH_GITHUB_CLIENT_SECRET}
integrations:
  github:
    - host: ${GITHUB_HOST_DOMAIN}
      apps:
        - appId: ${AUTH_GITHUB_APP_ID}
          clientId: ${AUTH_GITHUB_CLIENT_ID}
          clientSecret: ${GITHUB_CLIENT_SECRET}
          webhookUrl: ${GITHUB_WEBHOOK_URL}
          webhookSecret: ${GITHUB_WEBHOOK_SECRET}
          privateKey: |
            ${GITHUB_PRIVATE_KEY_FILE}
signInPage: github
dangerouslyAllowSignInWithoutUserInCatalog: true
```

callbackUrl

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.

app-config-rhdh.yaml fragment with optional enterpriseInstanceUrl field

```
auth:
  providers:
    github:
      production:
        callbackUrl: <your_intermediate_service_url/handler>
```

enterpriseInstanceUrl

Your GitHub Enterprise URL. Requires you defined the **GITHUB_HOST_DOMAIN** secret in the previous step.

app-config-rhdh.yaml fragment with optional enterpriseInstanceUrl field

```
auth:
  providers:
    github:
      production:
        enterpriseInstanceUrl: ${GITHUB_HOST_DOMAIN}
```

Verification

1. Go to the Developer Hub login page.
2. Your Developer Hub sign-in page displays **Sign in using GitHub** and the Guest user sign-in is disabled.
3. Log in with GitHub.

3.2. PROVISIONING USERS FROM GITHUB TO THE SOFTWARE CATALOG

To authenticate users, Red Hat Developer Hub requires their presence in the software catalog. Consider configuring Developer Hub to provision users from GitHub to the software catalog on schedule, rather than provisioning the users manually.

Prerequisites

- You have enabled authentication with GitHub, including the following secrets:
  - **GITHUB_HOST_DOMAIN**
  - **GITHUB_ORGANIZATION**

Procedure

- To enable GitHub member discovery, edit your custom Developer Hub ConfigMap, such as **app-config-rhdh**, and add the following lines to the **app-config-rhdh.yaml** content:

  app-config.yaml fragment with mandatory github fields

```
dangerouslyAllowSignInWithoutUserInCatalog: false
catalog:
  providers:
    github:
      providerId:
        organization: "${GITHUB_ORGANIZATION}"
        schedule:
          frequency:
            minutes: 30
          initialDelay:
            seconds: 15
          timeout:
            minutes: 15
    githubOrg:
      githubUrl: "${GITHUB_HOST_DOMAIN}"
      orgs: [ "${GITHUB_ORGANIZATION}" ]
```

```
    schedule:
      frequency:
        minutes: 30
      initialDelay:
        seconds: 15
      timeout:
        minutes: 15
```

dangerouslyAllowSignInWithoutUserInCatalog: false

Allow authentication only for users present in the Developer Hub software catalog.

organization, githubUrl, and orgs

Use the Developer Hub application information that you have created in GitHub and configured in OpenShift as secrets.

schedule.frequency

To specify custom schedule frequency. Supports cron, ISO duration, and "human duration" as used in code.

schedule.timeout

To specify custom timeout. Supports ISO duration and "human duration" as used in code.

schedule.initialDelay

To specify custom initial delay. Supports ISO duration and "human duration" as used in code.

Verification

1. Check the console logs to verify that the synchronization is completed.

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:production: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:production:refresh","taskInstanceId":"801b3c6c-167f-473b-b43e-e0b4b780c384","timestamp":"2024-09-09 23:55:59"}
```


2. Log in with a GitHub account.

CHAPTER 4. AUTHENTICATION WITH MICROSOFT AZURE

To authenticate users with Microsoft Azure:

1. Enable authentication with Microsoft Azure.
2. Provision users from Microsoft Azure to the software catalog .

4.1. ENABLING AUTHENTICATION WITH MICROSOFT AZURE

Red Hat Developer Hub includes a Microsoft Azure authentication provider that can authenticate users by using OAuth.

Prerequisites

1. You have the permission to register an application in Microsoft Azure.
2. You added a custom Developer Hub application configuration .

Procedure

1. To allow Developer Hub to authenticate with Microsoft Azure, create an OAuth application in Microsoft Azure.
   a. In the Azure portal go to **App registrations**, create a **New registration** with the configuration:

      Name

      The application name in Azure, such as *<My Developer Hub>*.

   b. On the **Home > App registrations >*<My Developer Hub>*> Manage > Authentication** page, **Add a platform**, with the following configuration:

      Redirect URI

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

      Front-channel logout URL

      Leave blank.

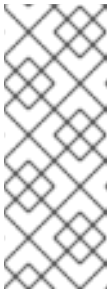
      Implicit grant and hybrid flows

      Leave all checkboxes cleared.

   c. On the **Home > App registrations >*<My Developer Hub>*> Manage > API permissions** page, **Add a Permission**, then add the following **Delegated permission** for the **Microsoft Graph API**:

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

- Optional custom scopes for the Microsoft Graph API that you define both in this section and in the Developer Hub configuration (**app-config-rhdh.yaml**).



NOTE

Your company might require you to grant admin consent for these permissions. Even if your company does not require admin consent, you might do so as it means users do not need to individually consent the first time they access backstage. To grant administrator consent, a directory administrator must go to the admin consent page and click **Grant admin consent for COMPANY NAME**.

d. On the **Home > App registrations ><*My Developer Hub*>> Manage > Certificates & Secrets** page, in the **Client secrets** tab, create a **New client secret**.

e. Save for the next step:

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

2. To add your Microsoft Azure credentials to Developer Hub, add the following key/value pairs to your Developer Hub secrets, such as **secrets-rhdh**:

AUTH_AZURE_TENANT_ID

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

AUTH_AZURE_CLIENT_ID

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

AUTH_AZURE_CLIENT_SECRET

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

3. Set up the Microsoft Azure authentication provider in your Developer Hub custom configuration, such as **app-config-rhdh**:

app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AUTH_AZURE_CLIENT_ID}
        clientSecret: ${AUTH_AZURE_CLIENT_SECRET}
        tenantId: ${AUTH_AZURE_TENANT_ID}
signInPage: microsoft
```

environment: production

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

clientId, clientSecret and tenantId

Use the Developer Hub application information that you have created in Microsoft Azure and configured in OpenShift as secrets.

signInPage: microsoft

Enable the Microsoft Azure provider as default sign-in provider.

Optional: Consider adding following optional fields:

dangerouslyAllowSignInWithoutUserInCatalog: true

To enable authentication without requiring to provision users in the Developer Hub software catalog.

WARNING

Use **dangerouslyAllowSignInWithoutUserInCatalog** to explore Developer Hub features, but do not use it in production.

app-config-rhdh.yaml fragment with optional field to allow authenticating users absent from the software catalog

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AUTH_AZURE_CLIENT_ID}
        clientSecret: ${AUTH_AZURE_CLIENT_SECRET}
        tenantId: ${AUTH_AZURE_TENANT_ID}
signInPage: microsoft
dangerouslyAllowSignInWithoutUserInCatalog: true
```

domainHint

Optional for single-tenant applications. You can reduce login friction for users with accounts in multiple tenants by automatically filtering out accounts from other tenants. If you want to use this parameter for a single-tenant application, uncomment and enter the tenant ID. If your application registration is multi-tenant, leave this parameter blank. For more information, see Home Realm Discovery.

app-config-rhdh.yaml fragment with optional domainHint field

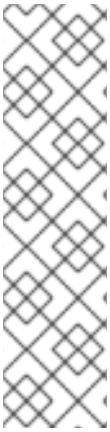
```
auth:
  environment: production
  providers:
    microsoft:
      production:
        domainHint: ${AUTH_AZURE_TENANT_ID}
```

additionalScopes

Optional for additional scopes. To add scopes for the application registration, uncomment and enter the list of scopes that you want to add. The default and mandatory value lists: **'openid', 'offline_access', 'profile', 'email', 'User.Read'**.

app-config-rhdh.yaml fragment with optional additionalScopes field

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        additionalScopes:
          - Mail.Send
```



NOTE

This step is optional for environments with outgoing access restrictions, such as firewall rules. If your environment has such restrictions, ensure that your RHDH backend can access the following hosts:

- **login.microsoftonline.com**: For obtaining and exchanging authorization codes and access tokens.
- **graph.microsoft.com**: For retrieving user profile information (as referenced in the source code). If this host is unreachable, you might see an *Authentication failed, failed to fetch user profile* error when attempting to log in.

4.2. PROVISIONING USERS FROM MICROSOFT AZURE TO THE SOFTWARE CATALOG

To authenticate users with Microsoft Azure, after Enabling authentication with Microsoft Azure, provision users from Microsoft Azure to the Developer Hub software catalog.

Prerequisites

- You have enabled authentication with Microsoft Azure.

Procedure

- To enable Microsoft Azure member discovery, edit your custom Developer Hub ConfigMap, such as **app-config-rhdh**, and add following lines to the **app-config-rhdh.yaml** content:

  app-config.yaml fragment with mandatory microsoftGraphOrg fields

```
dangerouslyAllowSignInWithoutUserInCatalog: false
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        target: https://graph.microsoft.com/v1.0
        tenantId: ${AUTH_AZURE_TENANT_ID}
        clientId: ${AUTH_AZURE_CLIENT_ID}
        clientSecret: ${AUTH_AZURE_CLIENT_SECRET}
```

- **dangerouslyAllowSignInWithoutUserInCatalog: false**

  Allow authentication only for users in the Developer Hub software catalog.

- **target: https://graph.microsoft.com/v1.0**

  Defines 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, clientId** and **clientSecret**

  Use the Developer Hub application information you created in Microsoft Azure and configured in OpenShift as secrets.

Optional: Consider adding the following optional **microsoftGraphOrg.providerId** fields:

- **authority: https://login.microsoftonline.com**

  Defines the authority used. Change the value to use a different authority, such as Azure US government. Default value: **https://login.microsoftonline.com**.

  app-config.yaml fragment with optional queryMode field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        authority: https://login.microsoftonline.com/
```

- **queryMode: basic | advanced**

  By default, the Microsoft Graph API only provides the **basic** feature set for querying. Certain features require **advanced** querying capabilities. See Microsoft Azure Advanced queries.

  app-config.yaml fragment with optional queryMode field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        queryMode: advanced
```

- **user.expand**

  To include the expanded resource or collection referenced by a single relationship (navigation property) in your results. Only one relationship can be expanded in a single request. See Microsoft Graph query expand parameter. This parameter can be combined with] or xref:userFilter[.

  app-config.yaml fragment with optional user.expand field

```
catalog:
  providers:
    microsoftGraphOrg:
```

```
      providerId:
        user:
          expand: manager
```

user.filter

To filter users. See Microsoft Graph API and Microsoft Graph API query filter parameters syntax. This parameter and ???TITLE??? are mutually exclusive, only one can be specified.

app-config.yaml fragment with optional user.filter field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        user:
          filter: accountEnabled eq true and userType eq 'member'
```

user.loadPhotos: true | false

Load photos by default. Set to **false** to not load user photos.

app-config.yaml fragment with optional user.loadPhotos field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        user:
          loadPhotos: true
```

user.select

Define the Microsoft Graph resource types to retrieve.

app-config.yaml fragment with optional user.select field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        user:
          select: ['id', 'displayName', 'description']
```

userGroupMember.filter

To use group membership to get users. To filter groups and fetch their members. This parameter and ???TITLE??? are mutually exclusive, only one can be specified.

app-config.yaml fragment with optional userGroupMember.filter field

```
catalog:
```

```
  providers:
    microsoftGraphOrg:
      providerId:
        userGroupMember:
          filter: "displayName eq 'Backstage Users'"
```

userGroupMember.search

To use group membership to get users. To search for groups and fetch their members. This parameter and ???TITLE??? are mutually exclusive, only one can be specified.

app-config.yaml fragment with optional userGroupMember.search field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        userGroupMember:
          search: '"description:One" AND ("displayName:Video" OR "displayName:Drive")'
```

group.expand

Optional parameter to include the expanded resource or collection referenced by a single relationship (navigation property) in your results. Only one relationship can be expanded in a single request. See https://docs.microsoft.com/en-us/graph/query-parameters#expand-parameter This parameter can be combined with] instead of xref:userFilter[.

app-config.yaml fragment with optional group.expand field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        group:
          expand: member
```

group.filter

To filter groups. See Microsoft Graph API query group syntax .

app-config.yaml fragment with optional group.filter field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        group:
          filter: securityEnabled eq false and mailEnabled eq true and
groupTypes/any(c:c+eq+'Unified')
```

group.search

To search for groups. See Microsoft Graph API query search parameter .

app-config.yaml fragment with optional group.search field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        group:
          search: '"description:One" AND ("displayName:Video" OR "displayName:Drive")'
```

group.select

To define the Microsoft Graph resource types to retrieve.

app-config.yaml fragment with optional group.select field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        group:
          select: ['id', 'displayName', 'description']
```

schedule.frequency

To specify custom schedule frequency. Supports cron, ISO duration, and "human duration" as used in code.

app-config.yaml fragment with optional schedule.frequency field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        schedule:
          frequency: { hours: 1 }
```

schedule.timeout

To specify custom timeout. Supports ISO duration and "human duration" as used in code.

app-config.yaml fragment with optional schedule.timeout field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        schedule:
          timeout: { minutes: 50 }
```

schedule.initialDelay

To specify custom initial delay. Supports ISO duration and "human duration" as used in code.

app-config.yaml fragment with optional schedule.initialDelay field

```
catalog:
  providers:
    microsoftGraphOrg:
      providerId:
        schedule:
          initialDelay: { seconds: 15}
```

Verification

1. Check the console logs to verify that the synchronization is completed.

   Successful synchronization example:

   ```
   backend:start: {"class":"MicrosoftGraphOrgEntityProvider$1","level":"info","message":"Read 1 msgraph users and 1 msgraph groups in 2.2 seconds. Committing...","plugin":"catalog","service":"backstage","taskId":"MicrosoftGraphOrgEntityProvider:default:refresh","taskInstanceId":"88a67ce1-c466-41a4-9760-825e16b946be","timestamp":"2024-06-26 12:23:42"}
   backend:start: {"class":"MicrosoftGraphOrgEntityProvider$1","level":"info","message":"Committed 1 msgraph users and 1 msgraph groups in 0.0 seconds.","plugin":"catalog","service":"backstage","taskId":"MicrosoftGraphOrgEntityProvider:default:refresh","taskInstanceId":"88a67ce1-c466-41a4-9760-825e16b946be","timestamp":"2024-06-26 12:23:42"}
   ```

2. Log in with a Microsoft Azure account.