Red Hat Developer Hub 1.2

Authentication

Configuring authentication to external services in Red Hat Developer Hub

Last Updated: 2024-10-02

Red Hat Developer Hub 1.2 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

Authentication within Red Hat Developer Hub facilitates user sign-in, identification, and access to external resources. It supports multiple authentication providers.

Authentication providers are typically used in the following ways:

- One provider for sign-in and identification.
- Additional providers for accessing external resources.

The Red Hat Developer Hub supports the following authentication providers:

Microsoft Azure
microsoft
GitHub
github
Keycloak
oidc

For each provider that you want to use, follow the dedicated procedure to complete the following tasks:

1. Set up the shared secret that the authentication provider and Red Hat Developer Hub require to communicate.
2. Configure Red Hat Developer Hub to use the authentication provider.

CHAPTER 1. ENABLING THE MICROSOFT AZURE AUTHENTICATION PROVIDER

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

Procedure

1. To allow Developer Hub to authenticate with Microsoft Azure, create an OAuth Application in Microsoft Azure.
   a. Go to Azure Portal > App registrations, and create an **App Registration** for Developer Hub.
   b. On your **App registration** overview page, add a new **Web platform configuration**, with the configuration:

      Redirect URI

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

      Front-channel logout URL

      Leave blank.

      Implicit grant and hybrid flows

      Leave all checkboxes cleared.

   c. On the **API permissions** tab, click **Add Permission**, then add the following **Delegated permission** for the **Microsoft Graph API**:

      - **email**
      - **offline_access**
      - **openid**
      - **profile**
      - **User.Read**
      - Optional custom scopes of the Microsoft Graph API that you define both here 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 admin consent, a directory admin must go to the admin consent page and click **Grant admin consent for COMPANY NAME**.

   d. Go to the **Certificates & Secrets** page, then the **Client secrets** tab, and create a new client secret. Save the **Client secret** for the next step.
2. Add your Microsoft Azure credentials in your Developer Hub secrets.

a. Edit your Developer Hub secrets, such as **secrets-rhdh**.

b. Add the following key/value pairs:

- **AUTH_AZURE_CLIENT_ID**: Enter the **Application ID** that you generated on Microsoft Azure.
- **AUTH_AZURE_CLIENT_SECRET**: Enter the **Client secret** that you generated on Microsoft Azure.
- **AUTH_AZURE_TENANT_ID**: Enter your **Tenant ID** on Microsoft Azure.

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

In the **app-config-rhdh.yaml** content, add the **microsoft** provider configuration under the root **auth** configuration, and enable the **microsoft** provider for sign-in:

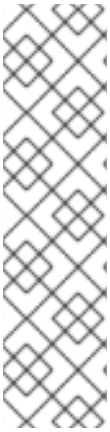
app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    microsoft:
      production:
        clientId: ${AUTH_AZURE_CLIENT_ID}
        clientSecret: ${AUTH_AZURE_CLIENT_SECRET}
        tenantId: ${AUTH_AZURE_TENANT_ID}
        # domainHint: ${AUTH_AZURE_TENANT_ID} 1
        # additionalScopes: 2
          # - Mail.Send
signInPage: microsoft 3
```

1 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.

2 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 is **['user.read']**.

3 To enable the Microsoft Azure provider as default sign-in provider.



NOTE

Optional for environments with restrictions on outgoing access, such as firewall rules. If your environment has outgoing access restrictions make sure your Backstage backend has access to the following hosts:

- **login.microsoftonline.com**: To get and exchange authorization codes and access tokens.
- **graph.microsoft.com**: To fetch user profile information (as seen in this source code). If this host is unreachable, users might see an *Authentication failed, failed to fetch user profile* error when they attempt to log in.

CHAPTER 2. ENABLING THE GITHUB AUTHENTICATION PROVIDER

Red Hat Developer Hub uses a built-in GitHub authentication provider to authenticate users in GitHub or GitHub Enterprise.

2.1. GITHUB APP OVERVIEW

GitHub Apps are generally preferred to OAuth apps because they use fine-grained permissions, give more control over which repositories the application can access, and use short-lived tokens. For more information, see GitHub Apps overview in the GitHub documentation.

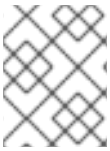
2.2. REGISTERING A GITHUB APP

In a GitHub App, you configure the allowed scopes as part of that application, therefore, you must verify the scope that your plugins require. The scope information is available in the plugin README files.

To add GitHub authentication, complete the steps in Registering a GitHub App on the GitHub website.

Use the following examples to enter the information about your production environment into the required fields on the **Register new GitHub App** page:

- Application name: Red Hat Developer Hub
- Homepage URL: **https://developer-hub-<NAMESPACE_NAME>.<KUBERNETES_ROUTE_HOST>**
- Authorization callback URL: **https://developer-hub-<NAMESPACE_NAME>.<KUBERNETES_ROUTE_HOST>/api/auth/github/handler/frame**



NOTE

The Homepage URL points to the Developer Hub front end, while the authorization callback URL points to the authentication provider backend.

2.3. CONFIGURING A GITHUB APP IN DEVELOPER HUB

To add GitHub authentication for Developer Hub, you must configure the GitHub App in your **app-config.yaml** file.

The GitHub authentication provider uses the following configuration keys:

- **clientId**: the client ID that you generated on GitHub. For example: b59241722e3c3b4816e2
- **clientSecret**: the client secret tied to the generated client ID.
- **enterpriseInstanceUrl** (optional): the base URL for a GitHub Enterprise instance. For example: **https://ghe.<company>.com**. The **enterpriseInstanceUrl** is only needed for GitHub Enterprise.
- **callbackUrl** (optional): the callback URL that GitHub uses when initiating an OAuth flow. For example: https://your-intermediate-service.com/handler. The **callbackUrl** is only needed if Developer Hub is not the immediate receiver, such as in cases when you use one OAuth app for many Developer Hub instances.

To configure the GitHub App, add the provider configuration to your **app-config.yaml** file under the root auth configuration. For example:

```
auth:
 environment: production
 providers:
   github:
     production:
       clientId: ${GITHUB_APP_CLIENT_ID}
       clientSecret: ${GITHUB_APP_CLIENT_SECRET}
       ## uncomment if using GitHub Enterprise
       # enterpriseInstanceUrl: ${GITHUB_URL}
```

2.4. ADDING THE GITHUB PROVIDER TO THE DEVELOPER HUB FRONT END

To add the provider to the front end, add the sign in configuration to your **app-config.yaml** file. For example:

```
signInPage: github
```

Additional resources

- For information about authenticating Backstage access with GitHub, see GitHub Authentication Provider in the community documentation.
- For information about adding the provider to the Backstage front end, see Enabling authentication in Showcase in the community documentation.

CHAPTER 3. ENABLING THE OPENID CONNECT AUTHENTICATION PROVIDER

Red Hat Developer Hub uses the OpenID Connect (OIDC) authentication provider to authenticate with third-party services that support the OIDC protocol.

3.1. OVERVIEW OF USING THE OIDC AUTHENTICATION PROVIDER IN DEVELOPER HUB

You can configure the OIDC authentication provider in Developer Hub by updating your **app-config.yaml** file under the root auth configuration. For example:

```
auth:
  environment: production
  # Providing an auth.session.secret will enable session support in the auth-backend
  session:
    secret: ${SESSION_SECRET}
  providers:
    oidc:
      production:
        metadataUrl: ${AUTH_OIDC_METADATA_URL}
        clientId: ${AUTH_OIDC_CLIENT_ID}
        clientSecret: ${AUTH_OIDC_CLIENT_SECRET}
        prompt: ${AUTH_OIDC_PROMPT} # Recommended to use auto
        ## Uncomment for additional configuration options
        # callbackUrl: ${AUTH_OIDC_CALLBACK_URL}
        # tokenEndpointAuthMethod: ${AUTH_OIDC_TOKEN_ENDPOINT_METHOD}
        # tokenSignedResponseAlg: ${AUTH_OIDC_SIGNED_RESPONSE_ALG}
        # scope: ${AUTH_OIDC_SCOPE}
        ## Declarative resolvers to override the default resolver:
`emailLocalPartMatchingUserEntityName`
        ## The authentication provider tries each sign-in resolver until it succeeds, and fails if none
succeed. Uncomment the resolvers that you want to use.
        # signIn:
        #   resolvers:
        #     - resolver: preferredUsernameMatchingUserEntityName
        #     - resolver: emailMatchingUserEntityProfileEmail
        #     - resolver: emailLocalPartMatchingUserEntityName
signInPage: oidc
```

3.2. CONFIGURING KEYCLOAK WITH THE OIDC AUTHENTICATION PROVIDER

Red Hat Developer Hub includes an OIDC authentication provider that can authenticate users by using Keycloak.

IMPORTANT

The user that you create in Keycloak must also be available in the Developer Hub catalog.

Procedure

1. In Keycloak, create a new realm, for example **RHDH**.
2. Add a new user.

   Username

   Username for the user, for example: **rhdhuser**

   Email

   Email address of the user.

   First name

   First name of the user.

   Last name

   Last name of the user.

   Email verified

   Toggle to **On**.
3. Click **Create**.
4. Navigate to the **Credentials** tab.
5. Click **Set password**.
6. Enter the **Password** for the user account and toggle **Temporary** to **Off**.
7. Create a new Client ID, for example, **RHDH**.

   Client authentication

   Toggle to **On**.

   Valid redirect URIs

   Set to the OIDC handler URL, for example, **https://<RHDH_URL>/api/auth/oidc/handler/frame**.
8. Navigate to the **Credentials** tab and copy the **Client secret**.
9. Save the Client ID and the Client Secret for the next step.
10. In Developer Hub, add your Keycloak credentials in your Developer Hub secrets.
    a. Edit your Developer Hub secrets, such as secrets-rhdh.
    b. Add the following key/value pairs:

       AUTH_KEYCLOAK_CLIENT_ID

       Enter the Client ID that you generated in Keycloak, such as **RHDH**.

       AUTH_KEYCLOAK_CLIENT_SECRET

       Enter the Client Secret that you generated in Keycloak.
11. Set up the OIDC authentication provider in your Developer Hub custom configuration.
    a. Edit your custom Developer Hub ConfigMap, such as **app-config-rhdh**.
    b. In the **app-config-rhdh.yaml** content, add the **oidc** provider configuration under the root **auth** configuration, and enable the **oidc** provider for sign-in:

app-config-rhdh.yaml fragment

```
auth:
  environment: production
  providers:
    oidc:
      production:
      clientId: ${AUTH_KEYCLOAK_CLIENT_ID}
      clientSecret: ${AUTH_KEYCLOAK_CLIENT_SECRET}
      metadataUrl: ${KEYCLOAK_BASE_URL}/auth/realms/${KEYCLOAK_REALM}
      prompt: ${KEYCLOAK_PROMPT} # recommended to use auto
       Uncomment for additional configuration options #callbackUrl:
${KEYCLOAK_CALLBACK_URL} #tokenEndpointAuthMethod:
${KEYCLOAK_TOKEN_ENDPOINT_METHOD} #tokenSignedResponseAlg:
${KEYCLOAK_SIGNED_RESPONSE_ALG} #scope: ${KEYCLOAK_SCOPE}  If you are
using the keycloak-backend plugin, use the
preferredUsernameMatchingUserEntityName resolver to avoid a login error.
      signIn:
        resolvers:
          - resolver: preferredUsernameMatchingUserEntityName
signInPage: oidc
```

Verification

1. Restart your **backstage-developer-hub** application to apply the changes.
2. Your Developer Hub sign-in page displays **Sign in using OIDC**.

3.3. MIGRATING FROM OAUTH2 PROXY WITH KEYCLOAK TO OIDC IN DEVELOPER HUB

If you are using OAuth2 Proxy as an authentication provider with Keycloak, and you want to migrate to OIDC, you can update your authentication provider configuration to use OIDC.

Procedure

1. In Keycloak, update the valid redirect URI to **https://<rhdh_url>/api/auth/oidc/handler/frame**. Make sure to replace **<rhdh_url>** with your Developer Hub application URL, such as, **my.rhdh.example.com**.
2. Replace the **oauth2Proxy** configuration values in the **auth** section of your **app-config.yaml** file with the **oidc** configuration values.
3. Update the **signInPage** configuration value from **oauth2Proxy** to **oidc**.
   The following example shows the **auth.providers** and **signInPage** configuration for **oauth2Proxy** prior to migrating the authentication provider to **oidc**:

```
auth:
  environment: production
  session:
    secret: ${SESSION_SECRET}
  providers:
    oauth2Proxy: {}
signInPage: oauth2Proxy
```

The following example shows the **auth.providers** and **signInPage** configuration after migrating the authentication provider to **oidc**:

```
auth:
  environment: production
  session:
    secret: ${SESSION_SECRET}
  providers:
    oidc:
      production:
        metadataUrl: ${KEYCLOAK_METADATA_URL}
        clientId: ${KEYCLOAK_CLIENT_ID}
        clientSecret: ${KEYCLOAK_CLIENT_SECRET}
        prompt: ${KEYCLOAK_PROMPT} # recommended to use auto
signInPage: oidc
```

4. Remove the OAuth2 Proxy sidecar container and update the **upstream.service** section of your Helm chart's **values.yaml** file as follows:

   - **service.ports.backend**: **7007**
   - **service.ports.targetPort**: **backend**
     The following example shows the **service** configuration for **oauth2Proxy** prior to migrating the authentication provider to **oidc**:

     ```
     service:
       ports:
         name: http-backend
         backend: 4180
         targetPort: oauth2Proxy
     ```

     The following example shows the **service** configuration after migrating the authentication provider to **oidc**:

     ```
     service:
       ports:
         name: http-backend
         backend: 7007
         targetPort: backend
     ```

5. Upgrade the Developer Hub Helm chart.