Red Hat Developer Hub 1.8

Interacting with Model Context Protocol tools for Red Hat Developer Hub

Leveraging the Model Context Protocol (MCP) server to integrate Red Hat Developer Hub (RHDH) with AI clients

Last Updated: 2026-01-22

Red Hat Developer Hub 1.8 Interacting with Model Context Protocol tools for Red Hat Developer Hub

Leveraging the Model Context Protocol (MCP) server to integrate Red Hat Developer Hub (RHDH) with AI clients

Legal Notice

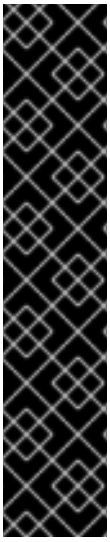
Abstract

Leverage the Model Context Protocol (MCP) server to integrate Red Hat Developer Hub (RHDH) with AI clients. This connection provides a standardized method for AI applications to access RHDH information and workflows through defined MCP tools.

Table of Contents

CHAPTER 1. INTERACTING WITH MODEL CONTEXT PROTOCOL TOOLS FOR RED HAT DEVELOPER HUB

1.1. UNDERSTANDING MODEL CONTEXT PROTOCOL



IMPORTANT

This section describes Developer Preview features in the Model Context Protocol plugin. Developer Preview features are not supported by Red Hat in any way and are not functionally complete or production-ready. Do not use Developer Preview features for production or business-critical workloads. Developer Preview features provide early access to functionality in advance of possible inclusion in a Red Hat product offering. Customers can use these features to test functionality and provide feedback during the development process. Developer Preview features might not have any documentation, are subject to change or removal at any time, and have received limited testing. Red Hat might provide ways to submit feedback on Developer Preview features without an associated SLA.

For more information about the support scope of Red Hat Developer Preview features, see Developer Preview Support Scope.

Model Context Protocol (**MCP**) offers a standardized method for linking AI models and applications (**MCP clients**) with external systems (**MCP servers**). This connection facilitates access to information and workflows residing on those systems. **MCP servers** are responsible for defining the tools that MCP clients can interact with.

Red Hat Developer Hub supports running MCP tools through the **mcp-actions-backend** plugin available in Backstage 1.40 or later.

1.2. INSTALLING THE MCP SERVER AND TOOL PLUGINS IN RED HAT DEVELOPER HUB

To enable MCP support in Red Hat Developer Hub, you need to install the following components:

Backend MCP server plugin

Responsible for running MCP tools.

MCP tool plugins

Individual MCP plugins that expose capabilities relating to the Software Catalog and TechDocs features in RHDH.

Prerequisites

- Your RHDH instance is installed and running.

Procedure

1. Install the backend MCP server plugin: In your dynamic plugins ConfigMap (for example, **dynamic-plugins-rhdh.yaml**), add the MCP server plugin as shown in the following example:

```
plugins:
  - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/backstage-plugin-
```

```
mcp-actions-backend:bs_1.42.5__0.1.2!backstage-plugin-mcp-actions-backend
    disabled: false
```

2. Install any of the following MCP tools that you would like to use:

- To install the Software Catalog MCP tool, in your dynamic plugins ConfigMap (for example, **dynamic-plugins-rhdh.yaml**), add the Software Catalog MCP tool plugin as shown in the following example:

```
  - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool:bs_1.42.5__0.2.3!red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool
    disabled: false
```

- To install the TechDocs MCP tool, in your dynamic plugins ConfigMap (for example, **dynamic-plugins-rhdh.yaml**), add the TechDocs MCP tool plugin as shown in the following example:

```
- package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-techdocs-mcp-tool:bs_1.42.5__0.3.0!red-hat-developer-hub-backstage-plugin-techdocs-mcp-tool
  disabled: false
```

1.3. CONFIGURING MODEL CONTEXT PROTOCOL IN RED HAT DEVELOPER HUB

You can enable your MCP client applications to use the MCP protocol to access RHDH information and workflows. This configuration is a prerequisite for MCP clients to use the defined MCP tools and access the exposed capabilities of RHDH.

Prerequisite

- You have installed the MCP server and tool plugins in Red Hat Developer Hub .

Procedure

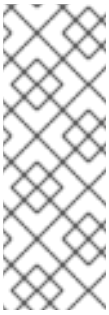
1. In your Red Hat Developer Hub app-config.yaml` file, configure a static token for authentication against the MCP server endpoint. MCP clients (such as **Cursor**, **Continue**, or **Lightspeed Core**) use these tokens to authenticate against the Backstage MCP server. For example:

```
backend:
  auth:
    externalAccess:
      - type: static
        options:
          token: ${MCP_TOKEN}
          subject: mcp-clients
```

where:

${MCP_TOKEN}

Set the token value that you generate manually and share with your MCP clients



NOTE

Tokens must be long and complex strings without whitespace to prevent brute-force guessing.

To generate a sample token, use the following command:

```
$ node -p `require("crypto").randomBytes(24).toString("base64")`
```

2. Register the MCP tools that you install as a plugin source, as shown in the following example:

```
backend:
  actions:
    pluginSources:
      - software-catalog-mcp-tool
      - techdocs-mcp-tool
```

Full app-config.yaml file example with MCP configuration

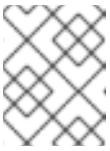
```
app:
  title: AI Dev Developer Hub
  baseUrl: "${RHDH_BASE_URL}"
auth:
  environment: development
  session:
    secret: "${BACKEND_SECRET}"
  providers:
    guest:
      dangerouslyAllowOutsideDevelopment: true
backend:
  actions:
    pluginSources:
      - 'software-catalog-mcp-tool'
      - 'techdocs-mcp-tool'
  auth:
    externalAccess:
      - type: static
        options:
          token: ${MCP_TOKEN}
          subject: mcp-clients
    keys:
      - secret: "${BACKEND_SECRET}"
  baseUrl: "${RHDH_BASE_URL}"
  cors:
    origin: "${RHDH_BASE_URL}"
signInPage: oidc
```

1.3.1. Configuring MCP clients to access the RHDH server

You must configure an MCP client before it can interact with the MCP server. For more information on the list of clients and their respective configurations, see Example Clients.

Prerequisites

- You have configured one of the following endpoints as the server URL, where ***<my_developer_hub_domain>*** is the hostname of your RHDH instance.
  - Streamable: **https://*<my_developer_hub_domain>*/api/mcp-actions/v1**
  - SSE (Legacy): **https://*<my_developer_hub_domain>*/api/mcp-actions/v1/sse**



NOTE

Some clients do not yet support the Streamable endpoint, and you might need to use the SSE endpoint instead.

- You have set the **${MCP_TOKEN}** environment variable in your MCP server configuration as the bearer token when authenticating with the MCP server.

Procedure

1. Configure **Cursor** as a client.
   a. From your Desktop app, navigate to **Cursor Settings** and select **MCP Tools > New MCP Server**.
   b. Add the following configuration:

```
{
  "mcpServers": {
    "backstage-actions": {
      "url": "https://<my_developer_hub_domain>/api/mcp-actions/v1",
      "headers": {
        "Authorization": "Bearer <MCP_TOKEN>"
      }
    }
  }
}
```

   where:

   <MCP_TOKEN>
   Enter the previously configured static token

   <my_developer_hub_domain>
   Enter the hostname of your RHDH instance

2. Configure **Continue** as a client.
   a. In your agent yaml configuration file, add the following configuration:

```
mcpServers:
  - name: backstage-actions
    type: sse
    url: https://<my_developer_hub_domain>/api/mcp-actions/v1/sse
```

```
    requestOptions:
      headers:
        Authorization: "Bearer <MCP_TOKEN>"
```

where:

<MCP_TOKEN>

Enter the previously configured static token

<my_developer_hub_domain>

Enter the hostname of your RHDH instance

3. Configure **Developer Lightspeed for RHDH** as a client. For more details, see Red Hat Developer Lightspeed for Red Hat Developer Hub.

   a. In the **lightspeed-stack.yaml** configuration, add the following configuration for **mcp_servers**:

```
mcp_servers:
  - name: mcp::backstage
    provider_id: model-context-protocol
    url: https://<my_developer_hub_domain>/api/mcp-actions/v1
```

   where:

   model-context-protocol

   This is the tool runtime provider defined and configured in the llama-stack **run.yaml** configuration for use in LCS.

   b. Optional: If you want to use your own Llama Stack configuration, add the following code to your Llama Stack configuration file (**run.yaml**).

```
providers:
  tool_runtime:
  - provider_id: model-context-protocol
    provider_type: remote::model-context-protocol
    config: {}
```

   c. To authorize requests to the MCP endpoint using **<MCP_TOKEN>**, add it in the Developer Lightspeed for RHDH **app-config.yaml** file, to make POST requests to LCS, as shown in the following code:

```
lightspeed:
  mcpServers:
  - name: mcp::backstage
    token: ${MCP_TOKEN}
```

   d. Optional: You can query the LCS **/v1/streaming_query** endpoint directly by providing the **MCP_TOKEN** in the header, as shown in the following code:

```
curl -X POST \
  -H `Content-Type: application/json` \
  -H `MCP-HEADERS: {"mcp::backstage": {"Authorization": "Bearer <MCP_TOKEN>"}}` \
```

```
  -d `{"query": "Can you give me all catalog templates of type 'service', "model": "gpt-4o-
mini", "provider": "openai"}` \
  _<url>_/v1/streaming_query
```

where:

<url>

Enter the LCS endpoint. You can use localhost(pass:c,a,q:[**<RHDH_servicename>.my-rhdh-project.svc.cluster.local:8080**]) or the service name for this field if you are inside the Backstage container.

1.4. USING THE MCP TOOLS TO ACCESS DATA FROM RED HAT DEVELOPER HUB

MCP tool plugins enable seamless integration with the Software Catalog and TechDocs.

1.4.1. Retrieving Software Catalog data through the MCP tool

The Software Catalog MCP lists Backstage entities, such as **Components**, **Systems**, **Resources**, **APIs**, **Locations**, **Users**, and **Groups**.

By default, the tool returns results in a JSON array format. Each entry in the JSON array contains an entity with the following fields: **name**, **description**,**type**, **owner**, **tags**, **dependsOn**, and **kind**.

The optional **verbose** parameter returns the entire Backstage entity object(s).

The following examples show common queries:

- “Fetch all ai-model resources in the Backstage catalog”
- “Fetch the API definition for the beneficiary-management-api API”
- “Construct a curl command based on the API definition for the “insert beneficiary” endpoint in the beneficiary-management-api”

Procedure

- Use the parameters as shown in the following table to configure your Software Catalog MCP tool plugin.

Name	Description	Example
kind	Filters entities by their Backstage kind.	"Component"
type	Filter entities by their Backstage type. (Requires the **kind** parameter)	"model-server"
name	Specifies the name of the Backstage entity to retrieve.	"vllm-model-server"

Name	Description	Example
owner	Filters entities by their owner.	"test-platform"
lifecycle	Filters entities by their lifecycle.	"development"
tags	Filters entities by their tags.	["genai, "ibm", "llm", ...]
verbose	If set to **true**, the system returns the full Backstage entity object instead of the shortened output.	**true**

1.4.2. Accessing and analyzing documentation using the TechDocs MCP tools

The TechDocs MCP tool enables MCP clients to search and retrieve documentation directly from TechDocs registered in your RHDH instance. Use this tool to query documentation content and integrate it as context into your AI applications.

The following TechDocs MCP tools are supported: * **fetch-techdocs** * **analyze-techdocs-coverage** * **retrieve-techdocs-content**

1.4.2.1. Retrieving TechDocs URLs and metadata using fetch-techdocs

The **fetch-techdocs** TechDocs MCP tool lists all Backstage entities with TechDocs. By default, the tool returns results in a JSON array format. Each entry includes entity details and TechDocs metadata, like last update timestamp and build information.

By default, each entry in the JSON array is an entity with the following fields: **name**, **title**, **tags**, **description**, **owner**, **lifecycle**, **namespace**, **kind**, **techDocsUrl**, **matadataUrl** and **metadata**.

The following examples show common queries:

- “Fetch all techdocs from the Backstage server”
- “Fetch all techdocs of the default namespace”
- “Fetch all techdocs created by user:john.doe”

Procedure

- Use the parameters as shown in the following table to configure your **fetch-techdocs** TechDocs MCP tool.

Name	Description	Example
entityType	Filters entities by their type.	"Component"

Name	Description	Example
namespace	Filter entities by their namespace.	"default"
owner	Filters entities by owner.	"user:john.doe"
lifecycle	Filters entities by their lifecycle.	"development"
tags	Filters entities by their tags.	["genai, "ibm", "llm", "granite", "conversational", "task-text-generation"]

1.4.2.2. Measuring documentation gaps using **analyze-techdocs-coverage**

The **analyze-techdocs-coverage** TechDocs MCP tool calculates the percentage of entities that have TechDocs configured. This tool identifies documentation gaps and improve overall documentation coverage. This tool supports filtering by entity type, namespace, owner, lifecycle, and tags to analyze coverage for specific subsets of entities. By default, it returns a JSON entity that includes the fields **totalEntities**, **entitiesWithDocs**, and **coveragePercentage**.

The following examples show common queries:

- "What is the coverage of techdocs in the backstage server"
- "What is the coverage of techdocs in the default namespace"

Procedure

- Use the parameters as shown in the following table to configure your **analyze-techdocs-coverage** TechDocs MCP tool.

Name	Description	Example
entityType	Filters entities by their type.	"Component"
namespace	Filter entities by their namespace.	"default"
owner	Filters entities by owner.	"user:john.doe"
lifecycle	Filters entities by their lifecycle.	"development"
tags	Filters entities by their tags.	["genai, "ibm", "llm", "granite", "conversational", "task-text-generation"]

1.4.2.3. Finding a specific TechDoc using **retrieve-techdocs-content**

The **retrieve-techdocs-content** TechDocs MCP tool retrieves the content of a TechDocs resource,

enabling AI clients to access documentation content for specific Software Catalog entities. By default, the tool returns a JSON entity with the following fields: **entityRef**, **name**, **title**, **kind**, **namespace**, **content**, **path**, **contentType**, **lastModified**, and **metadata**.

The following examples show common queries:

- "Fetch techdoc with reference component:default/my-service"
- "Fetch page about.html from techdoc with reference component:default/my-service"

Procedure

- Use the parameters as shown in the following table to configure your **retrieve-techdocs-content** TechDocs MCP tool.

Name	Description	Example
entityRef	Specifies the entity to retrieve using the **kind:namespace/name** format.	"component:default/my-service"
pagePath	Specifies the path to a specific documentation page. Defaults to **index.html**	"index.html"

1.5. TROUBLESHOOTING MCP SERVER AND CLIENT PROBLEMS

The following procedures guide you through resolving common issues when using a Model Communication Protocol (MCP) server and tool.

1.5.1. Verifying successful installation of MCP plugins

Procedure

1. Log in to the OCP cluster running RHDH and go to your RHDH project using the following code:

```
$ oc project {my-product-namespace}
```

2. Inspect the logs for the installation of the RHDH dynamic plugins using the following code:

```
$ oc logs -c install-dynamic-plugins deployment/<my-product-deployment>
```

Verification

1. You must see an entry for the MCP backend server plugin as shown in the following code:

```
..... prior logs ....
======= Installing dynamic plugin oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/backstage-plugin-mcp-actions-backend:bs_1.42.5__0.1.2!backstage-plugin-mcp-actions-backend
```

```
 ==> Copying image oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/backstage-plugin-mcp-actions-backend:next__0.2.0 to local filesystem
 ==> Successfully installed dynamic plugin oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/backstage-plugin-mcp-actions-backend:bs_1.42.5__0.1.2!backstage-plugin-mcp-actions-backend
```

2. You must see entries for any of the MCP tool plugins you installed as shown in the following code:

```
..... prior logs ....
======= Installing dynamic plugin oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool:bs_1.42.5__0.2.3!red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool
 ==> Copying image oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool:bs_1.42.5__0.2.3!red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool to local filesystem
 ==> Successfully installed dynamic plugin oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool:bs_1.42.5__0.2.3!red-hat-developer-hub-backstage-plugin-software-catalog-mcp-tool
```

1.5.2. Checking MCP tool logs for status and errors

The Backstage **LoggerService** target name starts with the name of the MCP tool (either **software-catalog-mcp-tool** or **techdocs-mcp-tool**). The MCP tools generate a log by default. For example:

```
`[backend]: 2025-09-25T16:24:22.660Z software-catalog-mcp-tool info fetch-catalog-entities: Fetching catalog entities with options: kind="Component"`
```

If any errors occur in the MCP tools, check the logs.

1.5.3. Understand and respond to MCP tool error messages

The MCP tools response provides an optional error message that communicates any issues encountered during the use of the tool, including potential input validation errors.

1.5.4. Resolving the **Model does not support tool calling** error

This error indicates that the model configured in your MCP client lacks the required functionality to handle tool calls. The error message appears similar to: **Invalid request: model gemma3:27b does not support tool calls**.

Procedure

1. Consult your model documentation to confirm its support for tool calling.
2. If the current model does not support tool calling, change the model that your MCP client uses to a tool-calling compatible model.

1.5.5. Resolving authentication issues when tools are not found

If your MCP client connects to the server but cannot find deployed tools, an authentication or configuration issue is likely.

Procedure

1. Verify the authentication token.
    a. Configure a static token for the RHDH MCP server.
    b. Set this token as the bearer token in your MCP client and make sure the authorization header is the configuration that specifies **Bearer** before the token, for example, **Bearer <mcp_token>**.
2. Check the MCP endpoint configuration.
    a. Confirm that the MCP server URL properly resolves, particularly if you are using a desktop client.
    b. Use **legacy SSE endpoint**if your MCP client requires it instead of the Streamable endpoint. (For more details, see the **Configuration** topic).
3. Verify your RHDH **app-config.yaml** file for duplicate backend entries and make sure that the indentation is accurate.
    a. Make sure that the configuration for the static token and MCP plugin sources go under an existing backend field, if one is present.
    b. If you are unsure, see the sample app configuration provided in the reference procedure .

1.5.6. Resolve nonsensical MCP tool output

Nonsensical output often occurs when smaller models or models with smaller context sizes cannot effectively manage repeated tool calls within the same context window.

Procedure

To improve the quality of the tool output, take the following actions:

1. Select an appropriate model for tool calling.
    a. Verify that the model has good support for tool calling.
    b. Make sure your model is not too small. We recommend a model with at least 7 billion parameters and a context window of 32k.
2. Refine your queries.
    a. Use more well-defined queries that limit the amount of data returned in the response from the tool.
3. If possible, increase the context window size of the model. We recommend at least 32k for these MCP tools.