Red Hat Developer Hub 1.8

Understand and visualize Red Hat Developer Hub project health using Scorecards

Setting up, configuring, and managing customizable Project Health Scorecards

Last Updated: 2026-01-29

Red Hat Developer Hub 1.8 Understand and visualize Red Hat Developer Hub project health using Scorecards

Setting up, configuring, and managing customizable Project Health Scorecards

Legal Notice

Abstract

When using Red Hat Developer Hub (RHDH), you can understand and visualize the health, security posture, and compliance of your software components so that you can centrally monitor Key Performance Indicators (KPIs) collected from third-party systems like GitHub and Jira without consulting multiple external tools.

Table of Contents

CHAPTER 1. EVALUATE PROJECT HEALTH USING SCORECARDS

1.1. COMPONENT HEALTH AND COMPLIANCE MONITORING USING SCORECARDS

IMPORTANT

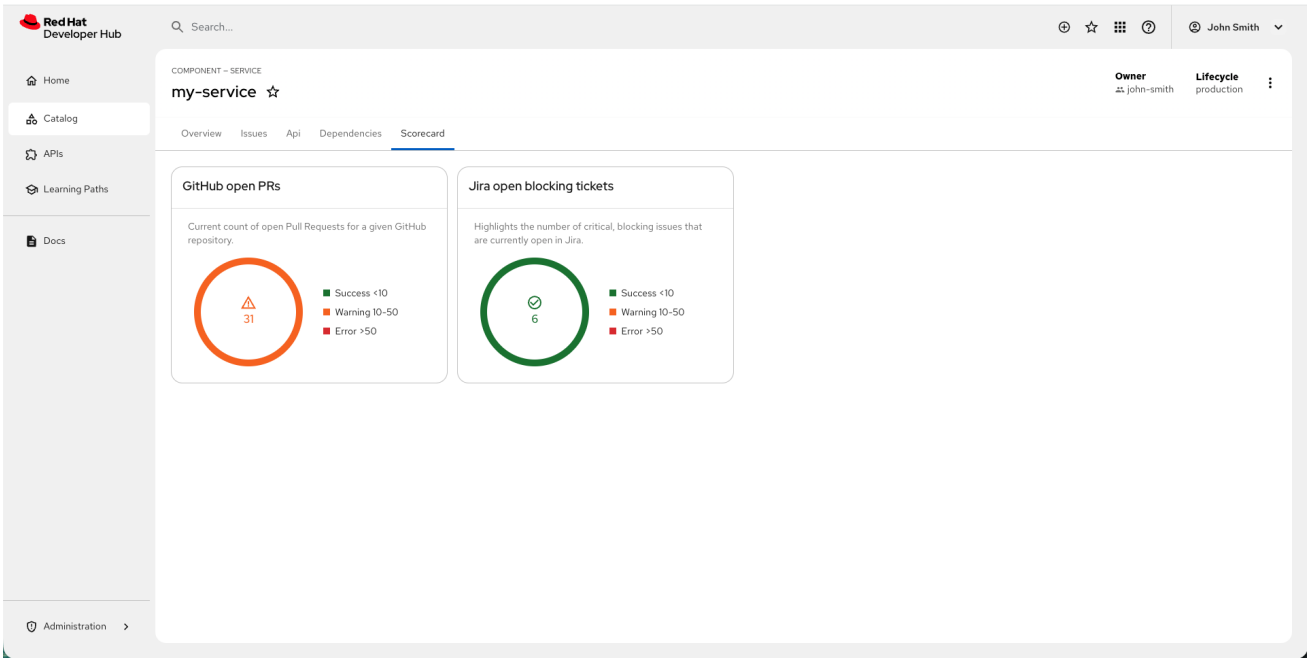
This section describes Developer Preview features in the Scorecard plugin. Developer Preview features are not supported by Red Hat in any way and are not functionally complete or production-ready. Do not use Developer Preview features for production or business-critical workloads. Developer Preview features 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.

You can use the Scorecard plugin in Red Hat Developer Hub RHDH to quickly assess the health, security, and compliance of your services in one place. By bringing key performance indicators together in a single view, you can evaluate component quality and identify issues without jumping between multiple tools or systems.

Use the Scorecard plugin to achieve the following goals:

- Identify and prioritize risks: Use unified health data to make faster remediation decisions.
- Maintain security standards: Automatically surface compliance gaps to enforce best practices.
- Streamline workflows: Access all metrics in RHDH to reduce development overhead.
- Standardize service quality: Define and measure consistent health criteria across the organization.



1.1.1. Supported Scorecard metrics providers

When you need to understand what service metrics are available, the Scorecard plugin gathers data from third-party systems through metric providers. The following table helps you identify which metrics are available for each supported provider, so you can decide what data to use for evaluating your services.

The following metric providers are supported:

Provider	Metric ID	Title	Description	Type
GitHub	**github.open_prs**	GitHub open PRs	Number of open pull requests in GitHub.	number
Jira	**jira.open_issues**	Jira open issues	Number of open issues in Jira.	number

1.2. SETTING UP SCORECARDS TO MONITOR YOUR RED HAT DEVELOPER HUB PROJECT HEALTH

1.2.1. Enabling Scorecards

To monitor component health and quality in RHDH, you must enable the Scorecard plugin in your configuration.

Prerequisites

- You have installed your RHDH instance.
- You have provisioned your custom dynamic plugins config map.

Procedure

- Add the following configuration in your RHDH **dynamic-plugin-config.yaml** file:

```
plugins:
  - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-scorecard:bs_1.42.5__1.0.0!red-hat-developer-hub-backstage-plugin-scorecard
    disabled: false
    pluginConfig:
      dynamicPlugins:
        frontend:
          red-hat-developer-hub.backstage-plugin-scorecard:
            entityTabs:
              - path: '/scorecard'
                title: Scorecard
                mountPoint: entity.page.scorecard
            mountPoints:
              - mountPoint: entity.page.scorecard/cards
                importName: EntityScorecardContent
                config:
                  layout:
                    gridColumn: 1 / -1
                  if:
                    allOf:
                      - isKind: component
  - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-scorecard-backend:bs_1.42.5__1.0.0!red-hat-developer-hub-backstage-plugin-scorecard-backend
    disabled: false
```

1.2.2. Installing and configuring Scorecard to view metrics

To enable users to view Scorecard metrics, you need to grant read access using Role-Based Access Control (RBAC). You can configure these permissions either through the RBAC CSV file or the RBAC UI, depending on how you manage access in your environment.

Prerequisite

- You have enabled RBAC, have a policy administrator role in RHDH, and have added **scorecard** to plugins with permission.

Procedure

Grant the required permissions by using one of the following methods:

- To use the RBAC CSV file, add the following policy to your CSV file to allow users to view metrics:

```
g, user:default/<YOUR_USERNAME>, role:default/scorecard-viewer
p, role:default/scorecard-viewer, scorecard.metric.read, read, allow
p, role:default/scorecard-viewer, catalog.entity.read, read, allow
```

  See Permission policies reference.

- To use the RBAC Web UI, complete the following steps:

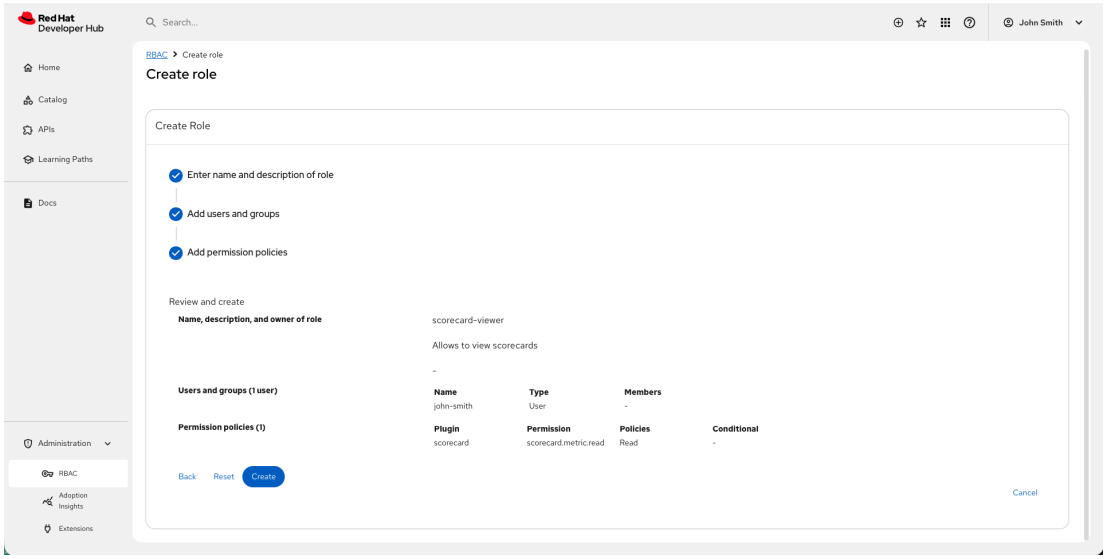
a. In the Red Hat Developer Hub menu, navigate to **Administration > RBAC**.

b. Select or create the **Role** that requires Scorecard access.

c. In the **Add permission policies** section, select **Scorecard** from the plugins dropdown.

d. Expand the **Scorecard** entry, select **policy** with the following details, and click **Next**:

- Name: **scorecard.metric.read**
- Permission: **read**



1.3. INSTALLING AND CONFIGURING SCORECARDS TO VIEW METRICS IN YOUR RED HAT DEVELOPER HUB INSTANCE

You can install and configure the following Scorecards to display metrics and visual health status for software components in the RHDH catalog:

- GitHub Scorecards
- Jira Scorecards

1.3.1. Integrating GitHub health metrics using Scorecards

You can configure the GitHub Scorecard plugin to display repository metrics in your RHDH catalog. This integration allows you to monitor component health and security risks directly from Red Hat Developer Hub.

You can grant RHDH access to the GitHub API by using a GitHub App or a GitHub token.

IMPORTANT

For long-lived integrations or organizational access, you must use a GitHub App.

Prerequisites

- You have installed your RHDH instance.
- You have installed the Scorecard images.

- Optional: If you choose the GitHub App option, you must have permissions in GitHub to create and manage a GitHub App.
- You must have added a custom RHDH application configuration and have enough permissions to change it.

Procedure

To install and configure GitHub Scorecards in your RHDH instance, complete the following steps:

1. Grant GitHub API access: Create one of the following authentication methods:
   - Configure using a GitHub App.
     i. Create a GitHub App with the required permissions (Read-only for Contents to allow reading repositories).

NOTE

You must install the GitHub App on the organization (or user account) that owns repositories you want access to, granting it the necessary repository access permissions.

     A. In the **General > Clients secrets** section, click **Generate a new client secret**
     B. In the **General > Private keys** section, click **Generate a private key**.
     C. In the **Install App** tab, choose an account to install your GitHub App on.
     D. Record the **App ID**, **Client ID**, **Client Secret**, and **Private key** values.

     ii. Add secrets to RHDH by adding the following key/value pairs to your RHDH secrets. You can use these secrets in the RHDH configuration files by using their respective environment variable names.
     - **GITHUB_INTEGRATION_APP_ID**:: The saved **App ID**.
     - **GITHUB_INTEGRATION_CLIENT_ID**:: The saved **Client ID**.
     - **GITHUB_INTEGRATION_CLIENT_SECRET**:: The saved **Client Secret**.
     - **GITHUB_INTEGRATION_HOST_DOMAIN**:: The GitHub host domain: **github.com**.
     - **GITHUB_INTEGRATION_ORGANIZATION**:: Your GitHub organization name, such as ***<your_github_organization_name>***.
     - **GITHUB_INTEGRATION_PRIVATE_KEY_FILE**:: The saved **Private key** content.

     iii. Configure the GitHub integration in your RHDH **app-config.yaml** file by adding the authentication details to the **integrations.github** section:

```
integrations:
  github:
    - host: ${GITHUB_INTEGRATION_HOST_DOMAIN}
      apps:
        - appId: ${GITHUB_INTEGRATION_APP_ID}
          clientId: ${GITHUB_INTEGRATION_CLIENT_ID}
```

```
    clientSecret: ${GITHUB_INTEGRATION_CLIENT_SECRET}
    privateKey: |
      ${GITHUB_INTEGRATION_PRIVATE_KEY_FILE}
```

- Configure using a GitHub token.

  i. Create a GitHub token with the following permissions:

  - Classic Personal Access Token (PAT): Select the **repo** scope for read/write access to private repositories.
  - Fine-Grained Personal Access Token (PAT):
    - Choose the specific repositories that RHDH must access.
    - Grant the token a **Read** permission for **Contents**.

  ii. Add the token to RHDH secrets by adding the following key/value pair to your RHDH secrets.
  where:

  GITHUB_TOKEN

  The generated GitHub token.

  iii. Configure the GitHub integration in your RHDH **app-config.yaml** file by adding the authentication details to the **integrations.github** section:

```
integrations:
  github:
    - host: github.com
      token: ${GITHUB_TOKEN}
```

2. Enable the GitHub Scorecard plugin: Add the GitHub Scorecard module to your RHDH **dynamic-plugins-config.yaml** file:

```
plugins:
  - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-scorecard-backend-module-github:bs_1.42.5__1.0.0!red-hat-developer-hub-backstage-plugin-scorecard-backend-module-github
    disabled: false
```

3. Annotate catalog entities: Link a component to the GitHub data source by editing the **catalog-info.yaml** file for your RHDH entity and adding the required annotations as shown in the following code:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: my-service
  annotations:
    # Required: GitHub project slug in format "owner/repository"
    github.com/project-slug: myorg/my-service
    # Required: Entity source location
    backstage.io/source-location: url:https://github.com/myorg/my-service
spec:
```

```
type: service
lifecycle: production
owner: _<your_team_name>_
```

where:

annotations:github.com/project-slug

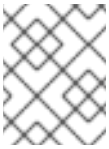
The GitHub repository format, for example, **owner/repository**.

annotations:backstage.io/source-location

The entity source location format, for example, **url:https://github.com/owner/repository**.

spec:owner

Your team name.



NOTE

You must add the team entity to the Catalog to ensure the provided permissions are applicable.

4. Ingest the catalog entity: Add the location of your **catalog-info.yaml** to the **catalog.locations** section in your RHDH **app-config.yaml** file:

```
catalog:
  locations:
    - type: url
      target: https://github.com/<owner>/<repository>/catalog-info.yaml
```

5. Optional: Customize thresholds: Define custom roles for the **GitHub Open Pull Requests** (**github.open_prs**) metric in your RHDH **app-config.yaml** file:

```
scorecard:
  plugins:
    github:
      open_prs:
        thresholds:
          rules:
            - key: success
              expression: '<10'
            - key: warning
              expression: '10-50'
            - key: error
              expression: '>50'
```

where:

scorecard:plugins:github:open_prs:thresholds

Lists the default threshold values for the GitHub open PRs metric.

1.3.2. Integrating Jira health metrics using Scorecards

You can configure the Jira Scorecard plugin to display project tracking and delivery velocity data in your RHDH instance. This integration centralizes development status and facilitates the evaluation of component readiness.

The plugin supports Jira Cloud (API v3) and Jira Data Center (API v2).

Prerequisites

- You must have administrator privileges for Jira and RHDH.
- You have installed your RHDH instance.
- You have installed the Scorecard images.
- You must have added a custom RHDH application configuration and have sufficient permissions to change it.

Procedure

To install and configure Jira Scorecards in your RHDH instance, complete the following steps:

NOTE

You must use the proxy setup to ensure configuration compatibility if you also use the Roadie Jira Frontend Plugin.

1. Generate a Jira configuration token using one of the following methods, depending on your Jira product:
   - Jira Cloud: Create a personal token. You must create a **Base64-encoded** string using the following plain text format: **your-atlassian-email:your-jira-api-token**.

     ```
     $ echo -n 'your-atlassian-email:your-jira-api-token' | base64
     ```

   - Jira datacenter: Create a Personal Access Token (PAT) in your Jira datacenter account.
2. Add Jira secrets: Define the following key/value pairs in your RHDH secrets:

   JIRA_TOKEN

   Enter your generated Jira token.

   JIRA_BASE_URL

   Enter your Jira base URL.
3. Configure the plugin: In your RHDH **dynamic-plugins-config.yaml** file, enable the plugin using either a direct setup or a proxy setup.
   - Use a direct setup:
     i. Add the following code to your RHDH **dynamic-plugins-config.yaml** file:

     ```
     plugins:
       - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-scorecard-backend-module-
     ```

```
jira:pr_1499__0.1.0!red-hat-developer-hub-backstage-plugin-scorecard-backend-module-jira
  disabled: false
```

ii. In your RHDH **app-config.yaml** file, add the following direct setup settings:

```
jira:
  baseUrl: ${JIRA_BASE_URL}
  token: ${JIRA_TOKEN}
  product: _<jira_product>_
```

where:

baseUrl

The base URL of your Jira instance, configured under ${JIRA_BASE_URL} in your RHDH secrets.

token

The Jira token (Base64 string for Cloud, PAT for Data Center), configured under ${JIRA_TOKEN} in your RHDH secrets.

product

Enter the supported product: **cloud** or **datacenter**.

- Use a proxy setup:

i. In your RHDH **dynamic-plugins-config.yaml** file, add the following code:

```
plugins:
  - package: oci://ghcr.io/redhat-developer/rhdh-plugin-export-overlays/red-hat-developer-hub-backstage-plugin-scorecard-backend-module-jira:pr_1499__0.1.0!red-hat-developer-hub-backstage-plugin-scorecard-backend-module-jira
    disabled: false
```

ii. In your RHDH **app-config.yaml** file, add the following proxy settings:

```
proxy:
  endpoints:
    '/jira/api':
      target: ${JIRA_BASE_URL}
      headers:
        Accept: 'application/json'
        Content-Type: 'application/json'
        X-Atlassian-Token: 'no-check'
        Authorization: ${JIRA_TOKEN} # Must be configured in your environment
jira:
  proxyPath: /jira/api
  product: cloud # Change to 'datacenter' if using Jira Datacenter
```

where:

target

The base URL of your Jira instance, configured under ${JIRA_BASE_URL} in your RHDH secrets.

Authorization

The Jira token, configured under ${JIRA_TOKEN} in your RHDH secrets. Set the token value as one of the following values:

- For **Cloud**: **Basic YourCreatedCloudToken**
- For **Data Center**: **Bearer YourJiraToken**

4. Annotate catalog entities: Add the Jira project key to the **metadata.annotations** section of the **catalog-info.yaml** file for your component:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: my-service
  annotations:
    jira/project-key: PROJECT
    jira/component: Component
    jira/label: UI
    jira/team: 9d3ea319-fb5b-4621-9dab-05fe502283e
    jira/custom-filter: 'reporter = "abc@xyz.com" AND resolution is not EMPTY'
spec:
  type: website
  lifecycle: experimental
  owner: guests
  system: examples
  providesApis: [example-grpc-api]
```

where:

jira/project-key

Required: Enter the Jira project key.

jira/component

Optional: Enter the Jira component name.

jira/label

Optional: Enter the Jira label.

jira/team

Optional: Enter the Jira team ID (not the team title).

jira/custom-filter

Optional: Enter a custom Jira Query Language (JQL) filter.

5. Ingest the catalog entity: Add the location of your **catalog-info.yaml** to the **catalog.locations** section in the RHDH **app-config.yaml** file:

```
catalog:
  locations:
    - type: url
      target: https://github.com/<owner>/<repository>/catalog-info.yaml
```

6. Optional: Define metric thresholds and filters: To customize health criteria or filter issues, add the **Jira Open Issues** metric thresholds to your RHDH **app-config.yaml** file:

```
scorecard:
  plugins:
    jira:
      open_issues:
        thresholds:
          rules:
            - key: success
              expression: '<10'
            - key: warning
              expression: '10-50'
            - key: error
              expression: '>50'
```

where:

scorecard:plugins:jira:open_issues:thresholds

Lists the default threshold values for the **Jira Open Issues** metric.

7. Optional: Define global or custom mandatory filters that entities can override by adding the following code to your RHDH **app-config.yaml** file:

```
scorecard:
  plugins:
    jira:
      open_issues:
        options:
          mandatoryFilter: Type = Task AND Resolution = Unresolved
          customFilter: priority in ("Critical", "Blocker")
```

where:

mandatoryFilter

Optional: Replaces the default filter (**type = Bug and resolution = Unresolved**).

customFilter

Optional: Specifies a global custom filter. The entity annotation **jira/custom-filter** overrides this value.

For more information about how to customize the threshold values, see Thresholds in Scorecard plugins.

1.4. SCORECARD METRIC THRESHOLDS

You can configure thresholds to turn raw metrics into meaningful health indicators, such as **Success**, **Warning**, or **Error**, helping you identify healthy components and detect issues early.

1.4.1. Scorecard metric threshold categorization rules

A threshold defines a condition or an expression that determines which visual category a metric value belongs to.

- Evaluation Order: Threshold rules are evaluated in the order they are defined. The first matching rule is applied to the metric value.
- Allowed Categories: Only three keys are supported: **success**, **warning**, and **error**.

- Best Practice: Always order rules from the most restrictive to the least restrictive to ensure logical assignment.

1.4.2. Supported Scorecard threshold expression syntax

Metric threshold expressions use mathematical and logical operators to evaluate data. Use the following operators to define health criteria based on the metric data type.

Metric Type	Operator	Example Expression	Job Performed
Number	**>**, **>=**, **<**, **<=**, **==**, **!=**	**>40**	The category applies if the value is greater than 40.
Number	**-** (Range)	**80-100**	The category applies if the value is between 80 and 100 (inclusive).
Boolean	**==**, **!=**	**==true**	The category applies if the value is exactly true.

1.4.3. Scorecard metric threshold precedence

Threshold rules follow a specific order of precedence. Higher-priority configurations override lower-priority settings, allowing you to define global defaults with entity-specific exceptions.

Priority	Configuration Method	Location	Job Performed
1 (Highest)	Entity Annotations	**catalog-info.yaml**	Overrides specific rules for a single component.
2 (Medium)	App Configuration	**app-config.yaml**	Sets global rules that override provider defaults.
3 (Lowest)	Provider Defaults	Backend Plugin Code	Baseline rules defined by the metric source.

1.4.4. Standardize Scorecard metric thresholds across components

You can define global thresholds in your **RHDH app-config.yaml** file to standardize health indicators across all components. Global configurations replace the default thresholds provided by a metric source.

If you omit a threshold category, such as **success**, the Scorecard plugin does not assign that category to the metric.

The following example defines thresholds for the **jira.open_issues** metric. These settings apply to all components using this metric unless an entity annotation overrides them.

```
scorecard:
  plugins:
    jira:
      open_issues:
        thresholds:
          rules:
            - key: success
              expression: '<10'    # fewer than 10 open issues
            - key: warning
              expression: '10-50'    # Between 10 and 50 open issues
            - key: error
              expression: '>50'      # More than 50 open issues
```

1.4.5. Override rules to configure entity-specific thresholds in Scorecards

Use annotations in the **catalog-info.yaml** file of a component to override global threshold rules. These annotations merge with and take precedence over the global rules defined in your RHDH **app-config.yaml** file.

Annotation structure

The annotation key must use the following format: **scorecard.io/{providerId}.thresholds.rules.{thresholdKey}: '{expression}'**

Element	Description	Example Annotation	Example Value
{providerId}	Unique identifier of the metric	**scorecard.io/jira.open_issues...**	**jira.open_issues**
{thresholdKey}	The overridden category	**...rules.warning**	**success**, **warning**, or **error**
{expression}	The new condition for the rule	**...: '>15'**	**>15**

Example: Override global Jira thresholds

In the following example, the component overrides the **warning** and **error** rules for the **jira.open_issues** metric. The **success** rule remains unchanged from the global configuration.

```
# catalog-info.yaml
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: critical-production-service
  annotations:
    # Changes global 'warning' from '11-50' to '10-15'
    scorecard.io/jira.open_issues.thresholds.rules.warning: '10-15'
    # Changes global 'error' from '>50' to '>15'
    scorecard.io/jira.open_issues.thresholds.rules.error: '>15'
spec:
  type: service
```

1.4.6. Verify logical flow in Scorecard threshold rules

<<<<<<< HEAD Threshold rule evaluation is order-dependent. You must follow logical ordering to avoid unintended category assignments, as the system stops evaluation once a value matches the first rule.

1.4.6.1. Follow logical ordering

Since Scorecard evaluates rules in order and stops at the first match, proper sequencing is essential for accurate health classification. >>>>>>> 8c579a8489 (Incorporated Heena's suggestions)

Rules must be sequenced logically. Order rules from the most strict (smallest range) to the least strict (largest range) or ensure all ranges are mutually exclusive.

=== Problematic rule order example

If rules are ordered incorrectly, a less restrictive rule can prevent stricter rules from being evaluated:

1. **warning: <50**: Any value less than 50 triggers the warning rule and stops evaluation.
2. **success: <10**: This rule is not evaluated because all values less than 10 have already matched the preceding warning rule.

=== Correct ordering example

Order the rules from the most restrictive value range to the least restrictive to ensure a logical flow.

```
# Correct Example: Order from most restrictive (Success) to least restrictive (Error)
rules:
  - key: success
    expression: '<10'     # Only values below 10
  - key: warning
    expression: '10-50'   # Only values between 10 and 50
  - key: error
    expression: '>50'     # All remaining values above 50
```

= Monitoring component health and readiness with Scorecard metrics

View Scorecard metrics to evaluate the health and security of your software components directly in the RHDH catalog.

Prerequisites

- You have installed your RHDH instance.
- You have configured the GitHub or Jira Scorecard (or both) plugin.
- You must have permissions to read the Scorecard metrics.

Procedure

1. In your RHDH navigation menu, go to **Catalog**.
2. Select the software component (catalog entity) that has Scorecard metrics configured.
3. On the component **Service** page, click the **Scorecard** tab.

4. Select a metric tile to view detailed data.