Red Hat Developer Hub 0.2

ArgoCD plugin for Backstage

The ArgoCD plugin for Backstage
The ArgoCD plugin for Backstage
Abstract

The ArgoCD plugin represents current status of application.
Table of Contents

CHAPTER 1. ARGOCD PLUGIN FOR BACKSTAGE .................................................. 3
  1.1. FOR ADMINISTRATORS ................................................................. 3
    1.1.1. Installation ................................................................. 3
  1.2. SUPPORT FOR MULTIPLE ARGOCD INSTANCES IN BACKSTAGE .......... 4
    1.2.1. Permissions in ArgoCD plugin ......................................... 7
    1.2.2. Self-signed certificates in ArgoCD plugin .......................... 7
  1.3. FOR USERS .......................................................................... 7
    1.3.1. Using ArgoCD plugin in Backstage .................................. 7
CHAPTER 1. ARGOCD PLUGIN FOR BACKSTAGE

The ArgoCD plugin represents the current status of an application in your Backstage Catalog.

1.1. FOR ADMINISTRATORS

1.1.1. Installation

The Red Hat Plug-ins for Backstage (RHPIB) packages are hosted in a separate NPM registry, which is maintained by Red Hat. To use these packages, you must adjust your NPM configuration to pull the @redhat scoped packages:

```
# update your .npmrc or .yarnrc file
yarn config set "@redhat:registry" https://npm.registry.redhat.com
# then pull a package
yarn add @redhat/backstage-plugin-quay
```

For more information, see npm docs.

Creating a .npmrc file ensures that all the packages are scoped under @redhat and are fetched from Red Hat's NPM registry, while the rest dependencies remain sourced from other registry.

Prerequisites

- Your Backstage application is installed and deployed.

Procedure

1. In the backstage/packages/app project, add the ArgoCD plugin as a package.json dependency as follows:

```
cd packages/app
yarn add @redhat/backstage-plugin-argo-cd
```

2. In the app-config.yaml file available in the root directory, add argo-cd to the proxy object as follows:

```
proxy:
  ...

 '/argocd/api':
  # url to the api of your hosted argoCD instance
  target: https://159.65.209.132/api/v1/
  changeOrigin: true
  # this line is required if your hosted argoCD instance has self-signed certificate
  secure: false
  headers:
    Cookie: $env: ARGOCD_AUTH_TOKEN
```
Also, you can optionally add the base URL for your ArgoCD web UI in the previous code.

3. Add the ArgoCD plugin to the list of plugins using the following code:

```typescript
// packages/app/src/plugins.ts
export { argocdPlugin } from '@redhat/backstage-plugin-argo-cd';
```

4. Add the ArgoCD plugin to the `entityPage.tsx` file as follows:

```typescript
// packages/app/src/components/catalog/EntityPage.tsx
import {
  EntityArgoCDHistoryCard,
  isArgocdAvailable,
} from '@redhat/backstage-plugin-argo-cd';

const overviewContent = (
  <Grid container spacing={3} alignItems="stretch">
    ...
    <EntitySwitch>
      <EntitySwitch.Case if={e => Boolean(isArgocdAvailable(e))}>
        <Grid item sm={6}>
          <EntityArgoCDHistoryCard />
        </Grid>
      </EntitySwitch.Case>
    </EntitySwitch>
    ...
  </Grid>
);```

5. To use the ArgoCD plugin for your component in Backstage, add an annotation to the YAML config file of a component:

   `argocd/app-name: <app-name>`

   To select multiple ArgoCD applications for a component, use labels as follows:

   `argocd/app-selector: <app-selector>`

   **NOTE**

   You can either use annotations or labels for a component.

6. Send a GET HTTP request to ArgoCD’s `/session` endpoint using the username and password to acquire your authentication key.

7. Add the acquired authentication key to the environment variables for your Backstage backend server as follows:

   `ARGOCD_AUTH_TOKEN="argocd.token=<auth-token>"`

### 1.2. SUPPORT FOR MULTIPLE ARGOCD INSTANCES IN BACKSTAGE
There are two options to support multiple ArgoCD instances in Backstage, including:

- Adding proxy configuration for each ArgoCD instance
- Using the ArgoCD backend plugin

**Option 1: Adding proxy configuration for each ArgoCD instance**

1. To create multiple components that fetch data from different ArgoCD instances, add a proxy configuration for each instance as shown in the following example:

   ```yaml
   proxy:
   ...

   '/argocd/api':
   target: https://<someAddress>/api/v1/
   changeOrigin: true
   secure: false
   headers:
   Cookie:
   $env: ARGOCD_AUTH_TOKEN

   '/argocd/api2':
   target: https://<otherAddress>/api/v1/
   changeOrigin: true
   secure: false
   headers:
   Cookie:
   $env: ARGOCD_AUTH_TOKEN2
   
   
   2. Add the required authentication tokens to environment variables, such as `ARGOCD_AUTH_TOKEN2`.
   
   After adding the authentication tokens, add the URL to the desired proxy path in the following component definition annotations:

   ```yaml
   argocd/proxy-url: '/argocd/api2'
   
   The `argocd/proxy-url` annotation defaults to `'/argocd/api'`. Therefore, `argocd/proxy-url` is not required if there is a single proxy configuration.

**Option 2: Using the ArgoCD backend plugin**

1. To create multiple components that fetch data from different ArgoCD instances, set the ArgoCD instance URL dynamically by adding the following configuration to `app-config.yaml` file:

   ```yaml
   argocd:
   username: ${ARGOCD_USERNAME}
   password: ${ARGOCD_PASSWORD}
   waitCycles: 25
   appLocatorMethods:
   - type: 'config'
     instances:
     - name: argoInstance1
       url: https://argoInstance1.com
   ```
A loop is created between deleting ArgoCD project and application to check the deletion of application occurring before the deletion of the project. If waitCycles is set to 25, then the loop can last for 75 seconds before erroring out.

With the previous configuration, the ArgoCD plugin fetches the ArgoCD instances an application is deployed to and use the ArgoCD backend plugin (backstage-plugin-argo-cd-backend) to reach out to each ArgoCD instance based on the following mapping:

- Add the required authentication tokens to environment variables, such as ARGOCD_USERNAME and ARGOCD_PASSWORD inside the argocd object. These authentication tokens are used as credentials for all ArgoCD instances by default.

```
argocd:
  username: ${ARGOCD_USERNAME}
  password: ${ARGOCD_PASSWORD}
  appLocatorMethods:
    - type: 'config'
      instances:
        - name: argoInstance1
          url: https://argoInstance1.com
        - name: argoInstance2
          url: https://argoInstance2.com
```

- Define a username and password for each ArgoCD instance. This mapping has higher priority than the previous option.

```
argocd:
  username: ${ARGOCD_USERNAME}
  password: ${ARGOCD_PASSWORD}
  appLocatorMethods:
    - type: 'config'
      instances:
        - name: argoInstance1
          url: https://argoInstance1.com
        - name: argoInstance2
          url: https://argoInstance2.com
        # dedicated username/password for this instance
        username: ${ARGOCD_USERNAME_INSTANCE_2}
        password: ${ARGOCD_PASSWORD_INSTANCE_2}
```

- Define a token for each instance. This mapping has higher priority than both options mentioned previously.

```
argocd:
  username: ${ARGOCD_USERNAME}
  password: ${ARGOCD_PASSWORD}
  appLocatorMethods:
    - type: 'config'
      token: `${ARGOCD_AUTH_TOKEN}` # optional
      name: argoInstance2
      url: https://argoInstance2.com
      # dedicated username/password for this instance
      username: `${ARGOCD_USERNAME_INSTANCE_2}` # optional
      password: `${ARGOCD_PASSWORD_INSTANCE_2}` # optional
```
instances:
- name: argoInstance1
  url: https://argoInstance1.com
  token: ${ARGOCD_AUTH_TOKEN} # Token to use to instance 1

1.2.1. Permissions in ArgoCD plugin

Setting permissions for ArgoCD user account not only reduces the scope but also reduces the functionality of the backend. If you scope the permissions for read-only, then actions including creating, deleting, and resyncing the application will not be available.

The error handling in ArgoCD is designed to alert you when proper permissions are not in place.

1.2.2. Self-signed certificates in ArgoCD plugin

By default, the ArgoCD server generates a self-signed certificate. Once you install the ArgoCD plugin, the deployment of argocd-server can be patched to be insecure using the following command:

```
kubectl patch deployment argocd-server --type "json" -p '[{"op":"add","path":"/spec/template/spec/containers/0/command/-","value":"--insecure"}]'
```

Also, you can use Helm to install the ArgoCD plugin and be insecure by default:

```
helm upgrade --install argocd argo/argo-cd
  --version 3.33.5
  --set 'server.extraArgs=--insecure'
```

1.3. FOR USERS

1.3.1. Using ArgoCD plugin in Backstage

The ArgoCD plugin displays the current state of an application in Backstage Catalog.

Prerequisites

- Your Backstage application is installed and running.
- You have installed the ArgoCD plugin. For installation steps, see Section 1.1.1, "Installation".

Procedure

1. Open your Backstage application and go to the Catalog page.
2. In the OVERVIEW tab, you can see the ArgoCD overview card at the bottom.
The ArgoCD overview card displays application information including **NAME**, **INSTANCE**, **SYNC STATUS**, **HEALTH STATUS**, and **LAST SYNCED**.

3. To view the detailed application information, select an application from the ArgoCD overview card.
   A pop-up containing detailed information about the application appears, along with a link to ArgoCD dashboard.

The **ArgoCD history** page contains history of application instances along with other information, such as deploy details, author name, message that author added, and revision ID.