Developer Preview declarative configuration for AuthN/Z Resources

Red Hat Advanced Cluster Security

May 16, 2023

Contents

Introduction

The declarative configuration feature for RHACS allows the creation of resources in a declarative manner.

This preview is a Developer Preview, for more information on the conditions and warranties of dev preview, please refer to this document.

Within the preview, the following resources can be created declaratively:

Auth providers & their rules

- Roles
- Permission Sets
- Access Scopes

The configurations will be stored in YAML format within Config Maps / Secrets, which will be mounted within the Central deployment.

Currently, the preview allows the complete flow of:

- Creating declarative configurations in YAML format
- Applying those declarative configuration within a Config Map / Secret within Central
- Allowing to modify (update / delete) declarative configurations.

However, some specific things are not yet covered within the preview, namely:

Health status for declarative configurations

Currently, all errors that may occur during the reconciliation of declarative configuration (e.g invalid values, wrong format etc.) are only surfaced within the logs of Central (the logs are at INFO level, so no adjustment required w.r.t the default log level).

Within the GA release of this feature, the health status and potential errors are surfaced within the UI's System Health page.

Allowing declarative configurations to reference system resources

Currently, it is only allowed for declarative configurations to reference system roles (e.g. roles that are added out-of-the-box to your RHACS installation such as Admin, Analyst etc.). However, it's currently not possible to reference system permission sets and access scopes.

Within the GA release of this feature, it will be possible to reference all available system resources.

Handling references during deletion of declarative configuration

Some of the resources that can be created via declarative configuration reference other resources (i.e. a role references a permission set and access scope). As an example, if a permission set that is created via declarative configuration is deleted whilst being referenced by a role, this currently leads to an error.

Within the GA release of this feature, the resource will be kept but made modifiable via UI / API to correct the reference.

Creating the declarative configuration YAML has to be done "by hand"

The creation of the YAML format for each resource currently has to be done "by hand", as well as applying the YAML configurations to a Config Map / Secret.

Within the GA release of this feature, roxctl commands will be available that support generating the YAML configurations, including linting of the configurations.

Enabling the declarative config feature for your RHACS installation

The feature is guarded behind the environment variable `ROX_DECLARATIVE_CONFIGURATION`, which needs to be enabled within Central, which is dependent on the installation method of RHACS.

Installation via roxctl

Adjust the YAML template within the bundle generated by roxctl to include setting the environment variable:

```
...
    env:
        - name: "ROX_DECLARATIVE_CONFIGURAITON"
          value: "true"
...
```

Afterwards, update the installation following the documentation.

Installation via Helm

Create a new **`declarative-config-values.yaml`** file:

```
customize:
  envVars:
    ROX_DECLARATIVE_CONFIGURATION: true
```

Afterwards, apply the configuration changes by following the documentation and including the **`declarative-config-values.yaml`** file.

Installation via Operator

Adjust the Central CR to include the environment variable:

```
...
spec:
  customize:
    envVars:
      - name: ROX_DECLARATIVE_CONFIGURATION
        value: 'true'
...
```

Adding declarative configuration mounts for your RHACS installation

The declarative configurations will be added via a mount point to the Central instance. The configuration itself can either reside in Config Maps or Secrets, depending on whether they contain sensitive information.

Generally, the recommendation for AuthN/Z resources is:
- Configurations for Auth Providers should reside within a Secret.
- For any other configurations, a Config Map should be sufficient.

Note: that a single Config Map / Secret may contain multiple configurations, which is also recommended to limit the number of volume mounts for the Central instance.

Red Hat

Adding the mount point for declarative configuration to your Central instance is dependent on your installation method.
Note: The config map / secret does not have to exist prior to adding the mount point to your installation.

Based on your installation method, follow the documentation below on adding the mount points for your declarative configurations.

Installation via roxctl

Provide the list of config maps / secrets that should be added as declarative config mounts via the flags during the central bundle generation:

```
roxctl central generate k8s/openshift \
--declarative-config-config-maps="declarative-configs" \
--declarative-config-secrets="sensitive-declarative-configs"
```

Afterwards, update the installation following the documentation.

Installation via Helm

Within the **`declarative-config-values.yaml`** file created during `Enabling the declarative config feature for your RHACS installation` step, add the declarative config mounts:

```
customize:
  envVars:
    ROX_DECLARATIVE_CONFIGURATION: true
central:
  declarativeConfiguration:
    mounts:
      configMaps:
        - declarative-configs
      secrets:
        - sensitive-declarative-configs
```

Afterwards, apply the configuration changes by following the documentation and including the **`declarative-config-values.yaml`** file.

Installation via Operator

Adjust the Central CR to include the declarative config mounts:

```
...
spec:
  central:
    declarativeConfiguration:
      configMaps:
      - name: "declarative-configs"
      secrets:
      - name: "sensitive-declarative-configs"
...
```

Restrictions for resources created from declarative configurations

Since resources may reference other resources (i.e. a role references both permission set and access scope), there are some restrictions for references:

1. A declarative configuration can only reference a resource that is either also created declaratively or a "system" resource, i.e. a resource that is provided out-of-the-box with RHACS (i.e. system roles / permission sets / access scopes).
2. All references between resources are done via names, meaning all names within the same resource type have to be unique.

Additionally, the following applies to all resources created from declarative configurations:

1. Resources can only be modified (updated / deleted) by altering the declarative configuration. It is not possible to change resources via API / UI.

Creating declarative configurations

Here's a walkthrough of creating a permission set and role via declarative configuration:
Note: this walkthrough assumes that you already added the Config Map "declarative-configs" to your central installation's declarative config mounts within the `Adding declarative configuration mounts for your RHACS installation` step

1. **Creating a YAML configuration for a permission set:**

```
name: restricted
description: Restricted permission set which only allows access to
Administration
resources:
- resource: Administration
  access: READ_WRITE_ACCESS
```

2. **Creating a YAML configuration for an access scope:**

```
name: restricted-remote
description: Access scope including only cluster remote
  rules:
    included:
    - cluster: remote
```

3. **Creating a YAML configuration for a role:**

```
name: restricted
description: Restricted role which only allows access to Administration for
all resources
permissionSet: restricted
accessScope: restricted-remote
```

4. **Define a config map containing all previously defined YAML configurations:**

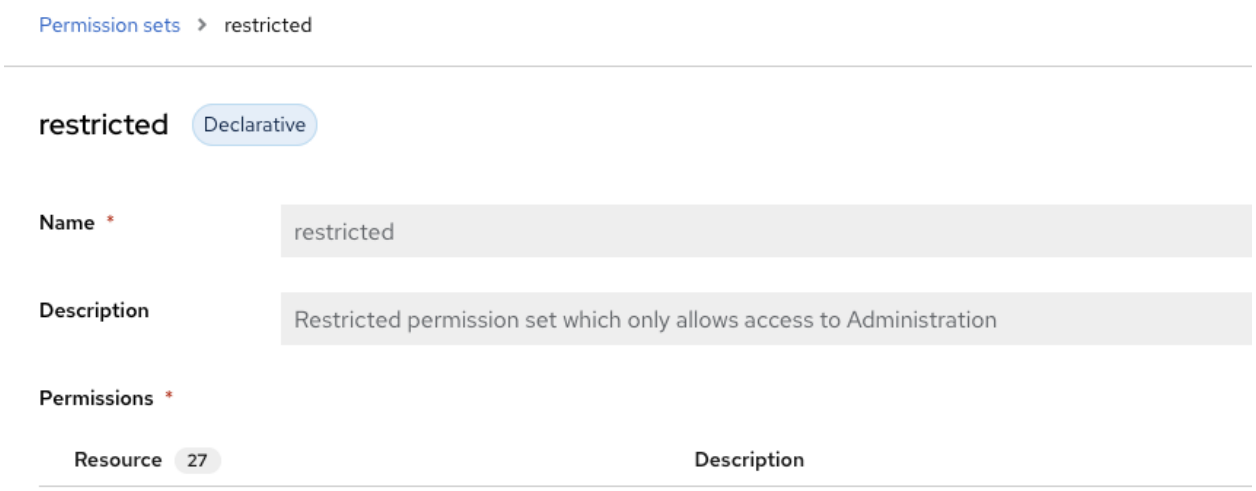
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: declarative-configs
  namespace: stackrox # This needs to match the namespace of your RHACS
installation
data:
  configs: |
    name: restricted
    description: Restricted role which only allows access to Administration
for all resources
    permissionSet: restricted
    accessScope: restricted-remote
    ---
```

```
    name: restricted
    description: Restricted permission set which only allows access to
Administration
    resources:
    - resource: Administration
    access: READ_WRITE_ACCESS
    ---
    name: restricted-remote
    description: Access scope including only cluster remote
    rules:
      included:
      - cluster: remote
```

5. **Create the config map:**

```
kubectl create -f configmap.yaml
```

6. **After the config map is created, central will pick up the changes. This may take some time once reconciliation happens. Afterwards, you will be able to view the created resources in the UI, as an example the permission set:**



YAML examples for all authN/Z declarative configurations

Permission Set

For more information on Permission Sets, visit creating custom permission sets in RHACS.

```
name: A sample permission set
description: A sample permission set created declaratively
resources:
- resource: Integration     # For a full list of supported resources, visit
`Access Control -> Permission Sets`
  access: READ_ACCESS       # Access can be either READ_ACCESS,
READ_WRITE_ACCESS
- resource: Administration
  access: READ_WRITE_ACCESS
```

Access Scope

For more information on Access Scopes, visit creating custom access scopes in RHACS.

```
name: A sample access scope
description: A sample access scope created declaratively
rules:
    included:
        - cluster: secured-cluster-A # Cluster where only specific
namespaces should be included within the access scope.
          namespaces:
            - namespaceA
        - cluster: secured-cluster-B # Cluster where _all_ namespaces
should be included within the access scope.
    clusterLabelSelectors:
        - requirements:
            - key: kubernetes.io/metadata.name
              operator: IN # Operator to use for the label selection. Can
be IN, NOT_IN, EQUAL, EXISTS, NOT_EXISTS.
              values:
                - worker-1
                - worker-2
```

```
                - worker-3
```

Role

For more information on Roles, visit creating custom roles in RHACS.

```
name: A sample role
description: A sample role created declaratively
permissionSet: A sample permission set              # Name of the
declarative permission set.
accessScope: Unrestricted                           # Name of the
declarative access scope.
```

Auth provider

RHACS provides support for multiple types of auth providers:

- Google IAP
- OpenID connect
- SAML 2.0
- User Certificates (PKI)
- OpenShift Auth

The configuration for auth provider consists of a generic configuration shared across auth providers as well as specific configuration for each auth provider type.
Based on the type of auth provider, below are full examples for each supported auth provider.

OIDC

```
name: A sample auth provider
minimumRole: Analyst                          # The minimum role which will
be assigned by default to any user logging in. If left empty, this will be
`None`.
uiEndpoint: central.custom-domain.com:443     # The UI endpoint of your
Central instance.
extraUIEndpoints:                             # In case your Central instance
is exposed to different endpoints, you need to specify them here.
    - central-alt.custom-domain.com:443
groups:                                       # Groups provide a mapping for
users to a specific role, based on their attributes
    - key: email                              # The key can be any claim
returned from the auth provider.
```

```
      value: example@example.com
      role: Admin                          # The role which the users
should be given. This can either be a system role, or a declaratively
created one.
    - key: group
      value: reviewers
      role: Analyst
requiredAttributes:                        # In case attributes returned
from the auth provider should be required. This can be helpful if the
audience should be limited to either a specific organization or group.
    - key: org_id
      value: "12345"
oidc:
  issuer: sample.issuer.com        # Expected issuer for the token
  mode: auto                       # OIDC callback mode. Possible values
are: auto, post, query, fragment. `auto` should be the preferred one.
  clientID: CLIENT_ID
  clientSecret: CLIENT_SECRET
```

Google IAP

```
name: A sample auth provider
minimumRole: Analyst                       # The minimum role which will
be assigned by default to any user logging in. If left empty, this will be
`None`.
uiEndpoint: central.custom-domain.com:443   # The UI endpoint of your
Central instance.
extraUIEndpoints:                          # In case your Central instance
is exposed to different endpoints, you need to specify them here.
    - central-alt.custom-domain.com:443
groups:                                    # Groups provide a mapping for
users to a specific role, based on their attributes
    - key: email                           # The key can be any claim
returned from the auth provider.
      value: example@example.com
      role: Admin                          # The role which the users
should be given. This can either be a system role, or a declaratively
created one.
    - key: group
      value: reviewers
```

```
      role: Analyst
requiredAttributes:                          # In case attributes returned
from the auth provider should be required. This can be helpful if the
audience should be limited to either a specific organization or group.
    - key: org_id
      value: "12345"
iap:
  audience: audience
```

SAML 2.0

SAML provides two configurations: static and dynamic.

For the dynamic configuration, you only need to provide the following:

```
name: A sample auth provider
minimumRole: Analyst                         # The minimum role which will
be assigned by default to any user logging in. If left empty, this will be
`None`.
uiEndpoint: central.custom-domain.com:443    # The UI endpoint of your
Central instance.
extraUIEndpoints:                            # In case your Central instance
is exposed to different endpoints, you need to specify them here.
    - central-alt.custom-domain.com:443
groups:                                      # Groups provide a mapping for
users to a specific role, based on their attributes
    - key: email                             # The key can be any claim
returned from the auth provider.
      value: example@example.com
      role: Admin                            # The role which the users
should be given. This can either be a system role, or a declaratively
created one.
    - key: group
      value: reviewers
      role: Analyst
requiredAttributes:                          # In case attributes returned
from the auth provider should be required. This can be helpful if the
audience should be limited to either a specific organization or group.
    - key: org_id
      value: "12345"
saml:
```

```
  spIssuer: sample.issuer.com
  metadataURL: sampl.provider.com/metadata
```

For the static configuration, you need to provide the following:

```
name: A sample auth provider
minimumRole: Analyst                        # The minimum role which will
be assigned by default to any user logging in. If left empty, this will be
`None`.
uiEndpoint: central.custom-domain.com:443   # The UI endpoint of your
Central instance.
extraUIEndpoints:                           # In case your Central instance
is exposed to different endpoints, you need to specify them here.
    - central-alt.custom-domain.com:443
groups:                                     # Groups provide a mapping for
users to a specific role, based on their attributes
    - key: email                            # The key can be any claim
returned from the auth provider.
      value: example@example.com
      role: Admin                           # The role which the users
should be given. This can either be a system role, or a declaratively
created one.
    - key: group
      value: reviewers
      role: Analyst
requiredAttributes:                         # In case attributes returned
from the auth provider should be required. This can be helpful if the
audience should be limited to either a specific organization or group.
    - key: org_id
      value: "12345"
saml:
  spIssuer: sample.issuer.com
  cert: |
    <cert in PEM format>
  ssoURL: saml.provider.com
  idpIssuer: idp.issuer.com
```

User certificates

```
name: A sample auth provider
minimumRole: Analyst                        # The minimum role which will
be assigned by default to any user logging in. If left empty, this will be
`None`.
uiEndpoint: central.custom-domain.com:443   # The UI endpoint of your
Central instance.
extraUIEndpoints:                           # In case your Central instance
is exposed to different endpoints, you need to specify them here.
    - central-alt.custom-domain.com:443
groups:                                     # Groups provide a mapping for
users to a specific role, based on their attributes
    - key: email                            # The key can be any claim
returned from the auth provider.
      value: example@example.com
      role: Admin                           # The role which the users
should be given. This can either be a system role, or a declaratively
created one.
    - key: group
      value: reviewers
      role: Analyst
requiredAttributes:                         # In case attributes returned
from the auth provider should be required. This can be helpful if the
audience should be limited to either a specific organization or group.
    - key: org_id
      value: "12345"
userpki:
    # List of certificate authorities that should be accepted.
    certificateAuthorities: |
        <cert in PEM format>
```

OpenShift Auth

```
name: A sample auth provider
minimumRole: Analyst                        # The minimum role which will
be assigned by default to any user logging in. If left empty, this will be
`None`.
uiEndpoint: central.custom-domain.com:443   # The UI endpoint of your
```

```
Central instance.
extraUIEndpoints:                       # In case your Central instance
is exposed to different endpoints, you need to specify them here.
    - central-alt.custom-domain.com:443
groups:                                 # Groups provide a mapping for
users to a specific role, based on their attributes
    - key: email                        # The key can be any claim
returned from the auth provider.
      value: example@example.com
      role: Admin                       # The role which the users
should be given. This can either be a system role, or a declaratively
created one.
    - key: group
      value: reviewers
      role: Analyst
requiredAttributes:                     # In case attributes returned
from the auth provider should be required. This can be helpful if the
audience should be limited to either a specific organization or group.
    - key: org_id
      value: "12345"
openshift:
    enable: true
```